CC-Log: Drastically Reducing Storage Requirements for Robots Using Classification and Compression

Santiago Gonzalez Vijay Chidambaram Jivko Sinapov Peter Stone

Department of Computer Science, University of Texas at Austin

Abstract

Modern robots collect a wealth of rich sensor data during their operation. While such data allows interesting analysis and sophisticated algorithms, it is simply infeasible to store all the data that is generated. However, collecting only samples of the data greatly minimizes the usefulness of the data. We present CC-LOG, a new logging system built on top of the widely-used Robot Operating System that uses a combination of classification and compression techniques to reduce storage requirements. Experiments using the Building-Wide Intelligence Robot, a mobile autonomous mobile platform capable of operating for long periods of time in human-inhabited environments, showed that our proposed system can reduce storage requirements by more than an order of magnitude. Our results indicate that there is significant unrealized potential in optimizing infrastructure commonly used in robotics applications and research.

1 Introduction

In recent times, we have witnessed an unprecedented surge in sensors and IoT devices. This has been accompanied by an exponential increase in the amount of data being generated. Sensors often generate small amounts of data, but at high frequency (*e.g.,* at 100 Hz). Prior research has looked at how to efficiently store and process these streams of data in sensor networks [1–3].

However, one such source of sensor data has been largely unexplored: robotics. Modern robotic platforms seek to understand natural language commands and perform a wide variety of tasks. To achieve this, robots include a large number of sensors. For example, structured infrared depth sensors (such as the Microsoft Kinect) and LIDAR scanners are used for localization and navigation in a number of robots [4]. Other sensors used include rotary encoders, inertial measurement sensors, and standard cameras. Such sensors produce data frequently (*e.g.,* depth sensors produce 30 GB in 15 seconds), and logging the sensor data allows post-hoc analysis and debugging [5, 6].

However, the dizzying rate of data production poses a problem for limited storage and short battery life of robots. Due to these restrictions, logging all data on local storage or streaming data as it is produced to a remote server are not feasible solutions. Sub-sampling the sensor data at a fixed reduced rate runs the risk of missing data essential for debugging or analysis.

We tackle this problem in the context of the Building-Wide Intelligence (BWI) project at the University of Texas at Austin [4]. The robot associated with the project, the BWIBot, is used for research for human-robot interaction, multi-robot interaction, and planning in cohabited environments. The BWIBot has sensors such as the Kinect and the odometer which produce ten to hundreds of MB of data per minute.

We present a system, CC-LOG, that combines event classification and compression to drastically reduce the storage used for logging (§3). First, instead of logging all events, we use a Support Vector Machine (SVM) to classify events as anomalous or not, and only log anomalous events (along with a few events preceding and succeeding the anomalous event). Second, we leverage the high inherent redundancy of the JSON format of the logs, using well-known loseless compression schemes such as LZMA [7] to reduce the size of the logs significantly. We use the Gazebo robot simulation framework [8] to collect data from several runs of the robot (as it is logistically difficult to collect enough data on the shared physical robot to evaluate our system). Compression reduces the logged data to 10% of its original size, and our classification scheme our event classification scheme further reduces the amount of data logged (§4). In our evaluation, the classifier does not produce any false negatives (*i.e.,* missing data we want to log). CC-LOG was developed as a package in the widely-used Robot Operating System [5] to facilitate integration into different robot platforms.

We believe our work on CC-LOG is a strong indicator of the untapped opportunities in the field of robotics. Analyzing robotics infrastructure from a systems viewpoint can yield significant improvements in efficiency (§6). Sadly, the different areas of research in computer science have become silos, with very little cross-over of ideas. By presenting this work in HotStorage, we hope to alleviate this problem a little by showing there are interesting systems challenges and opportunities for fruitful collaboration in robotics.

2 Background

We first provide some background on the BWIBot and the Robot Operating System. The BWIBot version 2 (also called the Segbot) contains a desktop computer powered by an Intel i7-4790T/i7-6700T processor, placed in HD-Plex H1.S fanless case, with 6 Gigabit Ethernet Network Interfaces, along with 4 USB3 and 2 USB2 interfaces. A 20 inch touchscreen is mounted at a human-operable height to serve as the primary user interface with the robot. The BWIBot is equipped with several sensors including a PointGrey BlackFly GigE camera and the Kinect. The BWIBot is built on top of the Segway RMP 110 mobile base which, coupled with Simultaneous Localization and Mapping (SLAM), provides sufficiently accurate odometry estimates for robust navigation.

The robot is controlled and programmed using the Robot Operating System (ROS) [5]. ROS is structured as a group of communicating processes called *nodes*. While nodes need to communicate with each other using a common format, each node can be written in a different language. This allows developers to easily extend ROS in the language of their choice. Nodes and sensors can send messages to a *topic* that other nodes can subscribe to. Typically, ROS processes the raw data from sensors and publishes a topic with formatted sensor data, which other nodes consume. For example, the odometry sensor topic publishes several KB of data every 10 milliseconds (100 Hz). The raw odometry data is processed by ROS and published in JSON format. The JSON format has a lot of redundancy that increases the storage requirements.

3 Design

Goal. In this work, we focus on efficiently logging the BWIBot odometry data (but our techniques are general and will work for other sensor data as well). We consider that the logging data is used mainly for debugging and analysis that considers anomalous events to be of interest. We note that sometimes the log data is used to obtain statistics about the robot (*e.g.,* how many kilometers has the robot covered over the past year?). For such cases, other techniques (such as statistical sampling) can be used so that samples are logged to allow statistics aggregation with enough accuracy; we do not handle it in this work.

We present CC-LOG, a new logging module in the Robot Operating System [5] that drastically reduces the storage requirements for logging sensor data. At a high level, CC-LOG processes data from subscribed topics, decides whether they are classified as anomalous, and, if so, they are compressed and logged to storage. CC-LOG accomplishes this through the use of a unique sliding window structure and a classifier.

Anomalous events typically include the sudden and extreme acceleration or sudden twisting to a large degree. Note that both acceleration and twisting are not anomalous in the general case; only when they deviate from normal behavior. Since anomalous events typically are not instantaneous, CC-LOG takes recent history into account when classifying data. Additionally, while anomalous events are interesting in and of themselves, it may be desirable to save data for a period of time following the event. Such data can help roboticists see how, and if the robot recovers from anomalies.

We first describe how anomalies are detected, how we use windows to log anomalous events, and how log events are compressed.

Anomaly Detection. To classify feature vectors as anomalous or nominal, we elected to use a one-class Support Vector Machine [9] (SVM) with a nonlinear radial basis function (RBF) kernel [10]. SVMs function by attempting to find the maximally-separating hyperplane between two sets of data using linear programming techniques. Since most datasets are not linearly separable, kernel functions can be used to map data into a high-dimensional feature space. In our case, we have chosen to use a Gaussian RBF kernel. One-class SVMs, specifically, are a popular distribution density estimation technique; largely due to their simplicity and wide applicability to different domains. Within the CC-LOG system, we use the Scikit-learn library [11] (built atop libsvm) to provide SVM functionality.

Machine learning classifiers require a *feature vector*: a vector of values that have been a meaningful dependence on the set to be classified. For example, for classification based on odometry data, the x-component of the robot's velocity is a useful feature. Feature selection is important in many types of machine learning. A sub-optimal selection of features can result in an inaccurate classifier. Feature vectors are constructed as a concatenated array of recently seen features.

CC-LOG uses pose position, pose orientation, linear twist, and angular twist data for our feature vectors. We deliberately limited the size of our feature set to reduce the size of the training and testing datasets. Selecting feature vectors happens at setup time, even before the first event has been logged.

Logging Windows. To help in debugging and analysis, CC-LOG logs the events *around* each detected anomaly (*i.e.,* what happened just before and after the anomaly was detected). To implement this, CC-LOG uses a *logging window*. Once an anomaly is detected, a fixed set of past and future samples are included in the current logging window which will be persisted to storage. If an-

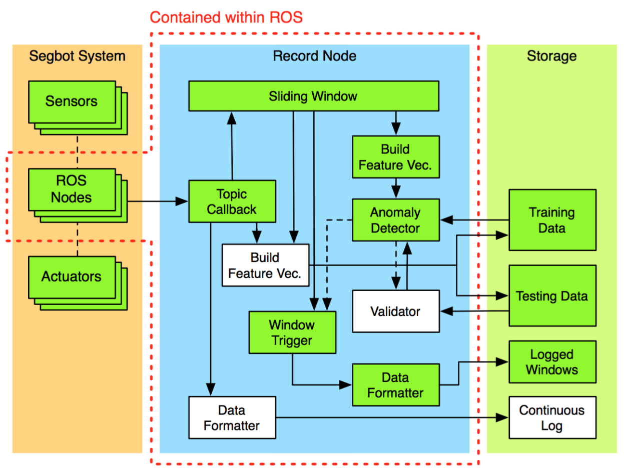

Figure 1: CC-LOG Architecture. Components used during *smart* logging are shown in green. Components such as building feature vectors and validators are used in other phases such as learning.

other anomalous event is detected within an active logging window, the window is extended appropriately, so that events are not redundantly logged in two windows.

Compressing Log Data. Currently, data is stored in an uncompressed JavaScript Object Notation (JSON) format for easy consumption. The JSON format results in unnecessarily large files due to the numerous repeated keys and punctuation. This quality of JSON lends itself very well to numerous compression techniques.

We now describe the architecture of the system, and the different modes in which CC-LOG operates. Figure 1 shows the different components of CC-LOG in the context of the Robot Operating System.

Collecting Data. CC-LOG has a basic logging mode where all subscribed sensor inputs are logged to a file. This mode is useful for collecting training and testing datasets. Whenever a message is received from the sensor topics, it is added to the sliding window and a feature vector is built from its current state. These resultant feature vectors are what compose the classifiers training and testing data. Additionally, the received message itself is filtered and formatted into JSON and saved into a log file, as in a traditional logging system.

Learning Mode. The classifier is trained and tested in the learning mode. Training and testing data are loaded from separate, labeled (i.e., marked as *nominal* or *anomalous*) directories on the file system. Since each feature vector is contained in its own file, it is easy to manually change which data is being used for training and which is being used for testing. Learning happens completely independently from the rest of ROS. Once the classifier is trained, a validation pass occurs, where a disjoint set of testing data is passed through the newly trained classifier, and results are reported.

Total Events	512
True Positives	20
False Positives	183
False Negatives	0
True Negatives	309
Precision	9.8%
Recall	100%

Table 1: Classification Results

Smart Logging. After training, the system is run in the smart logging mode where data is only logged if it is deemed to be part of an anomalous event. Figure 1 illustrates the different components of CC-LOG, along with the relevant parts of ROS. The BWIBot's sensors, actuators, and other ROS nodes are active as in the default configuration. Whenever a message is received from a subscribed topic, a callback inserts it into the current window. A feature vector is composed based on the active window and used to classify the message. If the message is classified as anomalous, the current logging window is formatted, time-stamped, and written to storage.

4 Evaluation

We evaluated CC-LOG using the BWIBot platform [4]. Obtaining training data from physical runs of the BWIBot proved tricky since the robot was shared among several project, and anomalies are rare and difficult to identify. Instead, we used the Gazebo simulation framework [8], a fully-functional simulation environment generating sensor data similar to an actual BWIBot run. Gazebo allows us to easily obtain sensor data from a large number of runs for the BWIBot. In the future, we plan to evaluate our system on actual runs of BWIBot.

We obtained 1495 events in total. We used 983 events for training. Training took 35 s and consumed 500 MB of RAM. There were 492 nominal samples and 20 anomalous samples in the test data. Our testing and training datasets were disjoint. The small number of anomalous samples is a result of the difficulty of achieving anomalous behavior in the simulation. Planning failures were the main anomaly mode reached in simulation.

Classification Accuracy. We used a sliding window size of 500; as a result, classification takes into account the last several seconds. We log 100 events before and after each anomalous event (discarding repeated events). There are 3250 features in our feature vector. We experimented with different settings for the support vector machine that trade-off between generalization and data fitting. The results in the best configuration are presented in Table 1. We note that there are no false negatives, but

Scheme	*File Size*	*Compression Ratio*
None	4.6 MB	100%
LZ4	596 KB	13.0%
LZFSE	479 KB	10.4%
ZIP	463 KB	10.1%
TAR gzip	463 KB	10.1%
LZMA (Level 6)	329 KB	7.2%

Table 2: Evaluating different compression schemes

about 40% false positives. The high rate of false positives arises due to the fact that anomalous events are normal events done in an extreme manner (*e.g.,* too much twisting, abnormal acceleration, *etc.*); the false positives arise due to our sensitivity to potentially anomalous events. We preferred to err on the side of caution and focus on avoiding false negatives. We believe the false positive rate can be reduced with more training.

Compression. We evaluated compressing the log data using a number of different schemes available on Apple platforms [12]. The results are shown in Table 2. We are able to achieve a significant reduction in file size through numerous techniques. Both ZIP and TAR gzip files both use the DEFLATE compression algorithm, thus providing similar results. All tested lossless compression techniques result in files approximately one-tenth the size of the original, uncompressed JSON log file. Notably, LZFSE is 2–3*X* faster and more energy efficient than DEFLATE-based compression schemes while achieving comparable compression [12].

5 Discussion

We briefly discuss our results and opportunities for future work in efficiently logging data on the BWIBot.

Evaluating using real runs from the BWIBot. Based on our promising results from the Gazebo simulation, we hope to achieve similar results on real data from the physical BWIBot. We have collected 730.3 MB of nominal data (represented by 17,512 feature vectors) from the BWIBot. The data was obtained from the robot autonomously navigating throughout a section of the building without human assistance. Notably, the robot was in motion for the entire duration of the data collection. We have also collected runs with anomalies during the robot's motion: this data was collected in a large atrium with many people, frequently rearranged seating, and poor visibility of the surrounding walls. In this environment, the robot has significant trouble localizing itself, leading to peculiar movement. Note that while we have collected the raw data, there is significant work ahead, including sanitizing and labeling the data. We might need to modify our classification approach to deal with such large data. Finding good hyper-parameter combinations will be crucial to getting good classification accuracy.

Lossy Compression. In our current work, we only explored loss-less compression schemes. However, if the analysis performed on the logs could handle errors in the data, we believe lossy compression schemes could compress the data even further [13]. We would like to investigate compressive sampling techniques such as the Randomized Timing Vector (RTV) algorithm [14]. While RTV may provide lossy compression, a desired level of compression is specifiable, and there is minimal memory overhead since compression occurs as data is collected, rather than in large batches.

Logging different kinds of data. In the future, we hope to incorporate many other types of data into the logging system, including point clouds, local cost maps, and visual data. Efficiently logging these types of data will throw up interesting challenges, though we believe our overall approach should still work.

6 Opportunities in Robotics

Logging efficiency is just one of many systems challenges that are posed by the *intelligent robotics* community's ongoing long-term research goal of creating autonomous robots capable of robust, long-term operation in the real world.

Storage for Robotics. Currently, robotics uses general-purpose file systems to store data. As we have shown in this work, not all data that is stored has the same value: for debugging purposes anomalous data is important; for analytics, only aggregate information is important. Furthermore, file systems for robotic platforms not need many features of the general-purpose file system such as directories (flat namespaces would suffice), or strong durability guarantees (losing recent data is acceptable [15]). We believe file systems targeted for robotics platforms can be significantly simpler, and can drastically reduce storage requirements.

Processing streaming data on low-power devices. Robots will need the ability to process increasingly large streams of sensory data in real time, without sacrificing the ability to make quick, reactive decisions, also subject to real-time and power constraints. As a concrete example, consider navigation in drones. Recent work has shown that it is computationally (and in terms of energy) expensive to use vision algorithms to guide the drone, but sending data over the network to be processed has a cost as well [16]. Thus, interesting trade-offs need to be made regarding what computation is performed on the drone, what data is saved, and what is sent over the network.

Storage for Continuous Learning. There is a growing consensus that robots such as the BWIBots will need to continuously learn from their data as they operate for long periods of times in human-inhabited environments. Thus, they would not only need to log data associated with anomalous events, but would also need to log data that is useful for various learning tasks. For example, if a robot is learning to adjust its navigation policy parameters when moving around people, it would not need to store data in situations where people are not present in its surrounding. Furthermore, data for learning would only need to be stored if the classifier or reinforcement learning policy parameters have not yet converged. Therefore, in future work we plan to extend CC-LOG to enable logging data associated with specific learning tasks that the robot needs to solve.

Nuanced Scheduling for ROS Nodes. The Robot Operating System (ROS) currently schedules all nodes similarly. Each data source is represented by a node, and different data sources produce data at different rates. Nodes for sources that produce data more frequently should be scheduled more frequently; there is currently no easy way to do this in ROS. Similarly, the complexity of functions performed by each ROS node also differs widely; some nodes do large computations, while other nodes do simple, quick calculations. The scheduling of ROS nodes need to take such differences into account. We believe a variant of Lottery Scheduling [17], where the admin can dynamically hand out tokens to different ROS nodes, would work well in this setting.

Efficient, Lightweight Processes. We notice that with time the BWIBot becomes sluggish for interactive use. We attribute this to the large number of processes that are running concurrently on the BWIBot. For example, each ROS node handles a different function like obstacle avoidance, localization, mapping, *etc.*; all of these nodes need to be running concurrently for most workloads. We believe ROS nodes (each a process) are too heavyweight for running long-running workloads on the BWIBot.

Finally, many data privacy and security issues arise as well. And as increasingly sophisticated sensors with embedded processors are introduced, we will need increasingly robust, and sophisticated operating systems to manage the computation and data inter-dependencies of a whole host of on-board processors with differing constraints and capabilities.

7 Related Work

We are not aware of prior research which seeks to reduce the storage footprint in robotic platforms. The most closely related work is FlashLogger [18] from Suman Nath. FlashLogger uses two techniques to reduce the energy footprint (and storage footprint) of logging: lossy compression of data, and "aging" old data by recompressing old data in a lower fidelity version. In contrast to FlashLogger, CC-LOG uses both compression and classification to reduce storage footprint, and currently uses only lossless compression techniques.

Prior work in other domains have identified that certain applications do not need completely accurate information. The SEsSAW project [19] at HP identifies similar (but not duplicate) data and exploits the similarity to reduce storage requirements and speed up analytics. The Impression Store [20] exploits the fact that most queries only require aggregate information to reduce storage requirements using compressive sensing. Wagner *et al.* use stratified sampling in their sublog work to accelerate query processing [21].

8 Conclusion

We have presented CC-LOG, a logging system that uses compression and classification of anomalous events to reduce the storage requirements for sensor data in robots. We use a support vector machine to classify odometry data. We show that CC-LOG correctly classifies all anomalous events (no false negatives) and reduces storage requirements by over 10X.

We believe our work is indicative of the interesting systems challenges in robotics. Robot sensors generate a large amount of data at a high frequency. There remain interesting challenges in logging and processing such sensor data, and in making the rest of the software infrastructure for robots (such as the Robot Operating System) more efficient. We hope this paper spurs further collaboration and exchange of ideas between the robotics and systems community.

Acknowledgments

We would like to thank the anonymous reviewers for their feedback and guidance. A portion of this work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CNS-1330072, CNS-1305287, IIS-1637736, IIS-1651089), ONR (21C184-01), AFOSR (FA9550-14-1-0087), Raytheon, Toyota, AT&T, and Lockheed Martin. Peter Stone serves on the Board of Directors of Cogitai, Inc. Vijay Chidambaram was supported in part by generous donations from VMware and Facebook. The terms of this arrangement have been reviewed and approved by UT Austin in accordance with its policy on objectivity in research. Any opinions, findings, and conclusions, or recommendations expressed herein are those of the authors and do not necessarily reflect the views of other institutions.

References

[1] Paris Carbone, Stephan Ewen, Seif Haridi, Asterios Katsifodimos, Volker Markl, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. *Data Engineering*, page 28, 2015.

[2] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[3] Yongrui Qin, Quan Z Sheng, Nickolas JG Falkner, Schahram Dustdar, Hua Wang, and Athanasios V Vasilakos. When things matter: A survey on data-centric internet of things. *Journal of Network and Computer Applications*, 64:137–153, 2016.

[4] Piyush Khandelwal, Shiqi Zhang, Jivko Sinapov, Matteo Leonetti, Jesse Thomason, Fangkai Yang, Ilaria Gori, Maxwell Svetlik, Priyanka Khante, Vladimir Lifschitz, J. K. Aggarwal, Raymond Mooney, and Peter Stone. Bwibots: A platform for bridging the gap between ai and human–robot interaction research. *The International Journal of Robotics Research*, 0(0):0278364916688949, 0.

[5] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, 2009.

[6] Adam Oliner, Archana Ganapathi, and Wei Xu. Advances and challenges in log analysis. *Communications of the ACM*, 55(2):55–61, 2012.

[7] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.

[8] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE, 2004.

[9] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[10] Bernhard Scholkopf, Kah-Kay Sung, Christopher JC Burges, Federico Girosi, Partha Niyogi, Tomaso Poggio, and Vladimir Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE transactions on Signal Processing*, 45(11):2758–2765, 1997.

[11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[12] Apple. Data compression. "`https://developer.apple.com/reference/compression/data_compression`".

[13] Mark Nelson and Jean-Loup Gailly. *The data compression book*, volume 2. M&amp;t Books New York, 1996.

[14] Marc J Rubin. *Efficient and automatic wireless geohazard monitoring*. PhD thesis, Colorado School of Mines. Arthur Lakes Library, 2007.

[15] Vijay Chidambaram, Thanumalayan Sankaranarayana Pillai, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Optimistic Crash Consistency. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP '13)*, Farmington, PA, November 2013.

[16] Hasan Genc, Ting-Wu Chin, Matthew Halpern, and Vijay Janapa Reddi. Optimizing sensor-cloud architectures for real-time autonomous drone operation. Sensors to Cloud Architectures Workshop (SCAW), June 2017.

[17] Carl A Waldspurger and William E Weihl. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, page 1. USENIX Association, 1994.

[18] Suman Nath. Energy efficient sensor data logging with amnesic flash storage. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, pages 157–168. IEEE Computer Society, 2009.

[19] Kalapriya Kannan, Suparna Bhattacharya, Kumar Raj, Muthukumar Murugan, and Doug Voigt. Seesaw-similarity exploiting storage for accelerating analytics workflows. In *8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16)*. USENIX Association, 2016.

[20] Jiaxing Zhang, Ying Yan, Liang Jeff Chen, Minjie Wang, Thomas Moscibroda, and Zheng Zhang. Impression store: Compressive sensing-based storage for big data analytics. In *HotCloud*, 2014.

[21] Tal Wagner, Eric Schkufza, and Udi Wieder. A sampling-based approach to accelerating queries in log management systems. In *Companion Proceedings of the 2016 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity*, SPLASH Companion 2016, pages 37–38, New York, NY, USA, 2016. ACM.