Multi-Task Learning for Storage Systems

Giulio Zhou*
Carnegie Mellon University
giuliozhou@cmu.edu

Martin Maas
Google Brain
mmaas@google.com

Abstract

Storage systems rely on predicting future workload behavior for making decisions in components such as caches, block allocators, and prefetchers. However, they are often oblivious to the applications using them, and rely on features such as access frequencies or offsets. This limits the prediction accuracy they can achieve.

We propose a new approach to prediction in storage systems, inspired by recent multi-task learning successes in NLP. Instead of relying on low-level features, we propose to train a machine learning model on unstructured application-level features that are already available in distributed tracing systems, widely deployed in data centers. While these features are predictive, the long-tailed, heterogeneous and dynamic nature of data center workloads means that training such models is expensive and needs to be repeated when workloads shift or the prediction task changes. We address this problem through a multi-task model that jointly learns how to perform different storage-related tasks. The aspiration is that when a new scenario is encountered, the model can be fine-tuned to the new task with a small number of samples instead of data-intensive end-to-end training.

We present an instance of this approach, based on the Census distributed tracing framework and a shared multi-task Transformer model underpinning 4 task-specific models. We demonstrate that this shared model is stable over time, and can be co-trained to perform all four tasks. We believe that this approach represents a new strategy for ML in storage systems and shows that storage prediction tasks that have traditionally been treated as separate can share information.

1 Introduction

Storage systems account for a significant portion of resources in data centers. This includes storage servers with flash or spinning disks [1], caches [2], distributed file systems [3, 4], and higher-level data management systems [5, 6]. Oftentimes, these services build upon one another and applications access a wide array of different storage services, resulting in complex interactions between them. In addition, warehouse-scale computers have a long tail of applications with a diverse range of storage access behaviors. This complexity makes it challenging to maximize the efficiency of any particular storage service, as doing so requires the system to reason about the application's behavior.

Storage services have long relied on hand-tuned heuristics to make decisions, such as when to admit or evict an entry from a cache [7], how to place data on different storage tiers based on access patterns [8] or predicting future access patterns for prefetching [9, 10]. These decisions are often made by policies that use information available at the storage level, such as sequences of access offsets and interarrival times. These policies are commonly tuned to achieve high efficiency for the set of workloads the service encounters, and may be specifically tuned for particularly important applications (e.g., databases). However, hand-optimization is infeasible for the long tail of data center workloads, which necessitates an automated approach that incorporates application-level information.

*Work done while at Google Brain

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

One solution is for the user to supply application-level hints, such as explicit prefetches [11]. Adding such hints offers users fine-grained control over system behavior, but suffers from two shortcomings: First, they add complexity to the software stack, requiring highly system-specific interfaces for their propagation and execution, which translates into technical debt and abstraction violations. Second, wrong or outdated policies can lead to regressions in system behavior. An alternative approach is to instead use application-level information that is already available in the system. Modern warehouse-scale computers often run distributed tracing frameworks [12, 13, 14] designed to attribute resource usage, diagnose bugs in distributed systems or collect traces to be analyzed and replayed at a later point. Data attached to storage requests by these systems captures a large amount of application-level context such as the services that a request travelled through before it reached the storage layer.

We focus on one such framework, Census (and its open-source variant OpenCensus [15]). Census enables profiling and resource accounting for distributed systems. It provides a tagging API that allows services to generate key-value pairs (*Census Tags*), which are automatically propagated with all downstream requests and can be used to understand complex interactions. A side-effect of this instrumentation is that by the time a request reaches a storage system, its accumulated Census tags encode rich context, including the path it took through the distributed system. However, exploiting this information is challenging. First, the data itself is schemaless and unstructured, and can contain any string the programmer chooses. Second, data centers run a diverse and dynamic set of applications, which means that models need to learn a long tail of different application behaviors and be robust to changes in the system. This indicates that periodic retraining is required but that a large amount of data is needed for the model to cover the full set of applications and general behaviors.

This problem is reminiscent of natural language processing: While a large amount of data is required to learn general knowledge about all concepts in the world, multi-task learning can be used to only perform this full training once and then fine-tune the model for different prediction tasks, using a much smaller amount of data. We adapt the same approach for storage problems: We propose a method for training a single, shared model that is trained on a long time period of storage traces. This type of model could then be rapidly fine-tuned to different tasks, such as to make caching decisions with different cache sizes or under shifting workload patterns. We show the promise of this approach by demonstrating a model that generalizes to unseen examples and is therefore more robust over time than a static lookup table, and that works across four different decision tasks in storage systems.

These results suggest that there is information that can be shared between different storage predictions problems that have traditionally been treated as separate. This work is a starting point for integrating more powerful ML techniques into storage systems, by jointly learning a single model underpinning multiple storage systems. We believe that further research in this area is warranted.

2 Background

Related Work: There is a large amount of work on predictions in storage systems, including cache policies [16], placement decisions [8], garbage collection scheduling [17] and predictive prefetching [18]. While most approaches only use storage-level information such as access patterns, prior work has incorporated application-level information. For example, speculative execution has been used to generate prefetch hints [19, 20]. Other work uses decision trees to generate file storage policies from file metadata [21]. These approaches work for individual applications but cannot support thousands of interacting services in a data center, with a long tail of behaviors.

Neural approaches address long-tail behavior by being able to handle exponentially many possible unseen examples while being compact. Jozefowicz et al. [22] find that while Kneser-Ney smoothed 5-gram models [23] are competitive with LSTM models on common words, LSTM models are significantly better at modeling uncommon words in the tail. Neural branch predictors [24] are able to exploit much longer histories using resources that grow linearly with respect to history length. Recent work on neural prefetching [25] makes the observation that the same properties can be exploited to learn memory access patterns (but does not use application-level features).

We build on work in multi-task learning, which is similar to transfer learning and trains a single model on multiple related tasks to learn a more general representation. These techniques are important for building a single model that can adapt to workload shifts and new prediction tasks with little data, and have driven recent successes in natural language processing such as BERT [26], a Transformer [27] model that is first pre-trained on unsupervised text and then fine-tuned to multiple tasks.

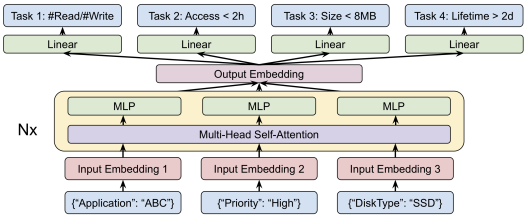

Figure 1: Model Architecture with three Census key-value pairs and four tasks.

Census/OpenCensus: Data center applications are often composed of a large number of services that communicate via message passing [28]. Services often have multiple clients using them (e.g., different webservices accessing a database) and rely on multiple different downstream services (e.g., the database service might connect to storage servers and an authentication service). This makes analysis and resource accounting challenging: When we encounter a request from a database, should it be attributed to the database or one of the upstream services using it?

(Open)Census is a distributed tracing library that provides insights into such systems. It traces requests as they travel through the system. Census allows applications to attach key-value pairs (Census Tags) that are automatically propagated through all systems and allows a service to determine the context of a request that it receives (e.g., for resource accounting). One of the underlying insights of our paper is that this same information represents powerful features for reasoning about the distributed system: Existing tags already capture properties of the workload that a programmer deemed important, and programmers could select new tags based on what they believe would be predictive features.

3 Prediction Tasks and Model Architecture

We use sampled traces from a production cluster and target 4 storage prediction tasks. These tasks represents common prediction problems that different types of storage systems need to solve.

- **Read-Write Ratio:** Predicting the ratio of read vs. write operations is helpful for placing data. Read-only files may be candidates for replication while write-only files may be best stored in a log structure. We assign each file a category based on the quintile of $reads/(reads + writes)$. Files without reads or writes are assigned a separate class.
- **Cacheability:** Caches exist in storage systems throughout all layers and often operate at the granularity of fixed-size blocks. One cache-related prediction task is whether to admit a given block into the cache, based on whether it is cacheable. We assign each (128 KB) block access request a binary cacheability label, based on whether this particular block will be re-accessed within some fixed time window (chosen to be 2 hours in our case).
- **File Lifetime:** File lifetime predictions can help storage systems reduce fragmentation and garbage collection work [29]. We predict lifetimes at a per-file basis and group lifetimes into exponentially-spaced buckets: $[0, 1s)$, $[1s, 4s)$, ..., $[4096s, \infty)$.
- **Final File Size:** Knowing the final size of a file at the time it is allocated can improve allocation decisions. We sum together the number of bytes written to a file, and group them into buckets $[0, 1KB)$, $[1KB, 4KB)$, ..., $[8MB, \infty)$.

Our multi-task model is based on the Transformer [27] architecture. Its self-attention mechanism makes it highly suited to processing sets [30] such as unordered collections of Census Tags. The model input is a Census Tag collection, a set of Census Tag key-value pairs associated with each storage request. We train a single model that maps a tokenized Census Tag collection to multiple classification outputs, one for each task. Given a Census Tag collection $X = \{x_1, x_2, \ldots, x_n\}$ where x_i is the ith (unordered) key-value string pair, we pass each x_i through a single embedding layer $\phi : \mathbb{N} \to \mathbb{R}^m$ to create a set of encoded Census Tags $Y = \{\phi(x_1), \phi(x_2), \ldots, \phi(x_n)\}$. Y is then passed into the Transformer encoder $M : \mathbb{R}^{n \times m} \to \mathbb{R}^{n \times m}$ and its output averaged to produce the shared output embedding $S = \sum_{i=1}^{n} M(Y)_i$ where $S \in R^m$. For each task, we pass S through a task-specific multi-layer perceptron to produce classification outputs. During training, we optimize the sum of cross entropy losses across all tasks.

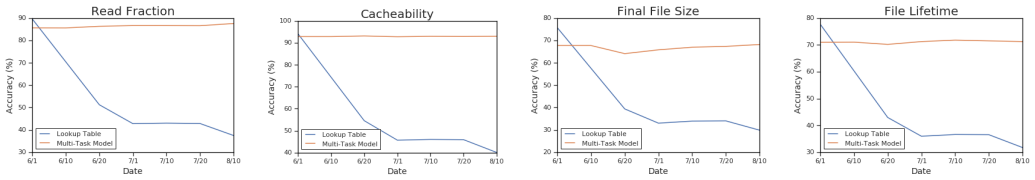

Figure 2: Stability of the lookup table vs. the ML model (trained on 10 days, evaluated on 2 months). Accuracy counts the percentage of requests that are in the lookup table and have the same label.

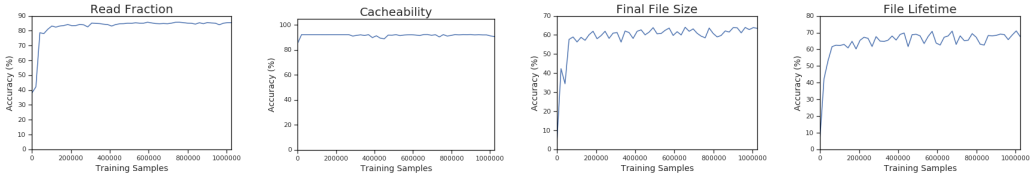

Figure 3: Training curves for simultaneously co-training our multi-task model on four storage tasks (trained on 10 days, evaluated on 10 days starting 2 months later).

4 Evaluation and Discussion

Static Lookup Table Baseline and Filtered Training: Census Tag collections follow a long-tailed distribution. The most common Census Tag collections are stable over a long period of time and comprise a sizable fraction of all examples. The tail is more diverse and fast-changing, with many Census Tag collections only appearing rarely. It is therefore possible to store the stable "head" in a small static table. The table maps Census Tag collections to the most commonly observed class for that particular collection. This approach allows for a training setup and deployment scenario where the most common Census Tag collections are handled using a static table and the long tail is handled by training an ML model. We simulate this approach by filtering out Census Tag collections that occur more than 2,000 times in the dataset. We also consider a simple baseline using a static lookup table as a stand-in for the model, and show how the model generalizes far better to the tail.

Transformer model generalizes much better than a static lookup table: We used an initial 10-day training period to build a static lookup table and train the Transformer model, and then evaluated the accuracy on subsequent 10-day periods over two months. Figure 2 illustrates how the static lookup table degrades considerably on all 4 tasks. The main source of prediction error in the static table approach is its inability to predict on unseen combinations of Census tags, i.e. shifts in $P(X)$. Another (less common) source of error are changes in the conditional label distribution $P(Y|X)$. While nearest neighbor techniques could form a stronger baseline, they require storing many training examples and querying them on-the-fly, which leads to significant storage and querying overheads.

Multi-Task models can simultaneously learn different storage tasks: Figure 3 shows that our Transformer model can simultaneously learn the different storage tasks. The ability to train the model on multiple tasks simultaneously represents the foundation of fine-tuning the model to unseen tasks. Being able to do so in few steps is important since workloads shift frequently. We can envision using this in an online training approach that incrementally fine-tunes a pre-trained model using a few gradient updates on the most recent data to allow adaptation at a finer granularity.

Future Work: Our tokenization approach, while simple, could be further improved through subtokenization of Census tag values (i.e., segmentation into common vs. unique parts). Further, one of the main practical challenges of deploying ML in production systems is the computational cost and inference latency. ML techniques such as distillation [31] and quantization [32] can create significantly cheaper models with minimal accuracy loss. We speculate that future multi-task learning models will achieve higher accuracy than single-task models, and that multi-task pre-training will speed up model retraining and the training of new tasks.

Conclusion: While our research is preliminary, we think that it presents a new approach to applying ML to storage problems and future work should investigate opportunities to jointly learn different storage tasks that are currently treated separately.

Acknowledgements: We would like to thank Colin Raffel for answering our questions related to NLP and model architectures, Homer Wolfmeister for helping us shape the problem formulation, Ramki Gummadi for detailed feedback and code reviews, and Richard McDougall for sharing technical insights across the entire storage stack. We also want to thank David Andersen, Eugene Brevdo, Andrew Bunner, Maya Gupta, Erez Louidor, Petros Maniatis, Azalia Mirhoseini, Seth Pollen, Yundi Qian, Nikhil Sarda, Mustafa Uysal, Tzu-Wei Yang, and Wangyuan Zhang for helpful discussions and feedback.

References

[1] Ana Klimovic, Christos Kozyrakis, Eno Thereska, Binu John, and Sanjeev Kumar. Flash storage disaggregation. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, pages 29:1–29:15, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4240-7. doi: 10.1145/2901318.2901337. URL `http://doi.acm.org/10.1145/2901318.2901337`.

[2] Brad Fitzpatrick. Distributed caching with memcached. *Linux journal*, 2004(124):5, 2004.

[3] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. 2003.

[4] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, et al. The hadoop distributed file system. In *MSST*, volume 10, pages 1–10, 2010.

[5] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[6] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):8, 2013.

[7] Alan Jay Smith. Cache memories. *ACM Computing Surveys (CSUR)*, 14(3):473–530, 1982.

[8] Jorge Guerra, Himabindu Pucha, Joseph S Glider, Wendy Belluomini, and Raju Rangaswami. Cost effective storage using extent based dynamic tiering.

[9] Steven Przybylski. The performance impact of block sizes and fetch strategies. In *[1990] Proceedings. The 17th Annual International Symposium on Computer Architecture*, pages 160–169. IEEE, 1990.

[10] Steven P Vanderwiel and David J Lilja. Data prefetch mechanisms. *ACM Computing Surveys (CSUR)*, 32(2):174–199, 2000.

[11] Russel H Patterson. Informed prefetching and caching. Technical report, Carnegie Mellon University, Pittsburgh, PA, School of Computer Science, 1997.

[12] Paul Barham, Rebecca Isaacs, Richard Mortier, and Dushyanth Narayanan. Magpie: Online modelling and performance-aware systems.

[13] Eno Thereska, Brandon Salmon, John Strunk, Matthew Wachs, Michael Abd-El-Malek, Julio Lopez, and Gregory R Ganger. Stardust: tracking activity in a distributed storage system. In *ACM SIGMETRICS Performance Evaluation Review*, volume 34, pages 3–14. ACM, 2006.

[14] Benjamin H Sigelman, Luiz Andre Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. 2010.

[15] Opencensus. URL `https://opencensus.io/`.

[16] Nathan Beckmann, Haoxian Chen, and Asaf Cidon. LHD: Improving cache hit rate by maximizing hit density. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 389–403, Renton, WA, April 2018. USENIX Association. ISBN 978-1-939133-01-4. URL `https://www.usenix.org/conference/nsdi18/presentation/beckmann`.

[17] W. Shin, M. Kim, K. Kim, and H. Y. Yeom. Providing qos through host controlled flash ssd garbage collection and multiple ssds. In *2015 International Conference on Big Data and Smart Computing (BIGCOMP)*, pages 111–117, Feb 2015. doi: 10.1109/35021BIGCOMP.2015.7072819.

[18] Xiaoning Ding, Song Jiang, Feng Chen, Kei Davis, and Xiaodong Zhang. Diskseen: Exploiting disk layout and access history to enhance i/o prefetch. In *USENIX Annual Technical Conference*, volume 7, pages 261–274, 2007.

[19] Fay Chang and Garth Gibson. Automatic i/o hint generation through speculative execution. USENIX, 1999.

[20] Edmund B Nightingale, Peter M Chen, and Jason Flinn. Speculative execution in a distributed file system. In *ACM SIGOPS operating systems review*, volume 39, pages 191–205. ACM, 2005.

[21] Michael Mesnier, Eno Thereska, Gregory R Ganger, Daniel Ellard, and Margo Seltzer. File classification in self-* storage systems. In *International Conference on Autonomic Computing, 2004. Proceedings.*, pages 44–51. IEEE, 2004.

[22] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[23] Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 181–184. IEEE, 1995.

[24] Daniel A Jiménez and Calvin Lin. Dynamic branch prediction with perceptrons. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pages 197–206. IEEE, 2001.

[25] Milad Hashemi, Kevin Swersky, Jamie Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. Learning memory access patterns. In *International Conference on Machine Learning*, pages 1924–1933, 2018.

[26] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[28] Thomas Erl. *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 1900.

[29] Taejin Kim, Sangwook Shane Hahn, Sungjin Lee, Jooyoung Hwang, Jongyoul Lee, and Jihong Kim. Pcstream: Automatic stream allocation using program contexts. In *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*, Boston, MA, July 2018. USENIX Association. URL `https://www.usenix.org/conference/hotstorage18/presentation/kim-taejin`.

[30] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL `http://proceedings.mlr.press/v97/lee19d.html`.

[31] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[32] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.