M3: Scaling Up Machine Learning via Memory Mapping

Dezhi Fang
College of Computing
Georgia Institute of Technology
dezhifang@gatech.edu

Duen Horng Chau
College of Computing
Georgia Institute of Technology
polo@gatech.edu

ABSTRACT

To process data that do not fit in RAM, conventional wisdom would suggest using distributed approaches. However, recent research has demonstrated virtual memory's strong potential in scaling up graph mining algorithms on a single machine. We propose to use a similar approach for general machine learning. We contribute: (1) our latest finding that memory mapping is also a feasible technique for scaling up general machine learning algorithms like logistic regression and k-means, when data fits in or exceeds RAM (we tested datasets up to 190GB); (2) an approach, called M3, that enables existing machine learning algorithms to work with out-of-core datasets through memory mapping, achieving a speed that is significantly faster than a 4-instance Spark cluster, and comparable to an 8-instance cluster.

CCS Concepts

•**Software and its engineering** → **Virtual memory;** •**Computing methodologies** → **Machine learning;**

1. INTRODUCTION

Leveraging virtual memory to extend algorithms for out-of-core data has received increasing attention in data analytics communities. Recent research demonstrated virtual memory's strong potential to scale up graph algorithms on a single PC [4, 3]. Available on almost all modern platforms, virtual memory based approaches are straight forward to implement and to use, and can handle graphs with as many as 6 billion edges [3]. Some single-thread implementations on a PC can even outperform popular distributed systems like Spark (128 cores) [4]. Memory mapping a dataset into a machine's virtual memory space allows the dataset to be treated identically as an in-memory dataset. The algorithm developer no longer needs to explicitly determine how to partition the (large) dataset, nor manage which partitions should be loaded into RAM, or unloaded from it. The OS performs similar actions on the developer's behalf, through paging the dataset in and out of RAM, via highly optimized OS-level operations.

SIGMOD/PODS'16 June 26 - July 01, 2016, San Francisco, CA, USA
© 2016 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-3531-7/16/06.
DOI: http://dx.doi.org/10.1145/2882903.2914830

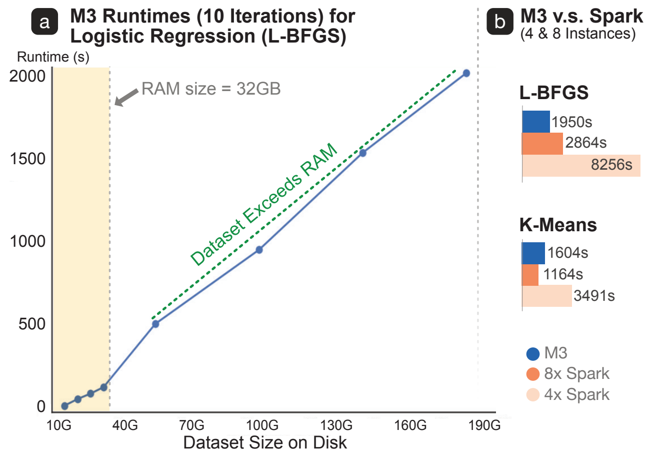

Figure 1: a: M3 runtime scales linearly with data size, when data fits in or exceeds RAM. b: M3's speed (one PC) comparable to 8-instance Spark (orange), and significantly faster than 4-instance Spark (light orange).

2. SCALING UP USING M3

As existing works focused on graph algorithms such as PageRank and finding connected components, we are investigating whether a similar virtual memory based approach can generalize to machine learning algorithms at large.

Inspired by prior works on graph computation, our M3[1] approach uses memory mapping to amplify a single machine's capability to process large amounts of data for machine learning algorithms. As memory mapping a dataset allows it to be treated identically as an in-memory dataset, M3 is a transparent scale-up strategy that developers can easily apply, requiring minimal modifications to existing code. For example, Table 1 shows that with only minimal code changes and a trivial helper function, existing algorithm implementation can easily handle much larger, memory-mapped datasets.

Modern 64bit machines have address spaces large enough to fit large datasets into (up to 1024PB). Because the operating system has access to a variety of internal statistics on how the mapped data is being used, the access to such data can be further optimized by the operating system via

[1]M3 stands for Machine Learning via Memory Mapping.

Original	M3
`Mat data;`	`double *m = mmapAlloc(file, rows * cols);` `Mat data(m, rows, cols);`

Table 1: M3 introduces minimal changes to code originally using in-memory data structure, enabling it to work with much larger memory-mapped data.

methods including least recent used caching and read-ahead to achieve efficiency [1].

To test the feasibility of M3, we minimally modified mlpack, an efficient machine learning library written in C++ [2]. Memory mapping can be easily applied to other languages and algorithm libraries.

3. EXPERIMENTS

Our current evaluation focuses on: (1) understanding of how M3 scales with increasing data sizes; and (2) how M3 compares with distributed systems such as Spark, as prior work suggested the possibility that a single machine can outperform a computer cluster [4].

Experiment Setup. All tests with M3 are conducted on a desktop computer with Intel i7-4770K quad-core CPU at 3.50GHz (8 hyperthreads), 4×8GB RAM, 1TB SSD of OCZ RevoDrive 350. We used Amazon EC2 `m3.2xlarge` instances for Spark experiments. Each instance has 8 vCPUs (hyperthreads of an Intel Xeon core) with 30GB memory and 2×80GB SSDs. The Spark clusters are created by Amazon Elastic MapReduce and the datasets are stored on the cluster's HDFS.

Dataset. We used the *Infimnist*[2] dataset, an infinite supply of digit images (0–9) derived from the well-known MNIST dataset using pseudo-random deformations and translations. Each image is 28×28 pixel grayscale image (784 features; each image is 6272 bytes). We generated up to 32M images, whose dense data matrix representation contains 23.5 billion entries, amounting to 190GB. Smaller datasets are subsets of the full 32M images.

Algorithms Evaluated. We selected *logistic regression* (L-BFGS for optimization) and k-means, since they are commonly used classification and clustering algorithms.[3]

3.1 Key Findings & Implications

1. M3 scales linearly when data fits in RAM and when out-of-core, for logistic regression (Figure 1a). The dotted vertical line in the figure indicates RAM size (32GB). M3's runtime scales linearly both when the dataset fits in RAM (yellow region in Fig. 1a), and when it exceeds RAM (green dotted line), at a higher scaling constant, as expected.

Looking at M3's resource utilization, we saw that M3 is I/O bound: disk I/O was 100% utilized while CPU was only utilized at around 13%. This suggests strong potential for M3 reaching even higher speed if we use faster disks, or configurations such as RAID 0.

2. M3's speed (one PC) is comparable to 8-instance Spark and significantly faster than 4-instance Spark for logistic regression (L-BFGS) and k-means (Figure 1b). This result echoes prior works focusing on graph computation that suggests cluster may not be necessary for moderately-sized datasets [4, 3]. Our result extends those findings to two general machine learning methods.

For *logistic regression* (with 10 iterations of L-BFGS), M3 is about 30% faster than 8-instance Spark. 4-instance Spark's runtime was 4.2X that of M3. For *k-means* (10 iterations, 5 clusters), M3 ran at a speed comparable to 8-instance Spark (1.37X), and more than twice as fast as 4-instance Spark.

Certainly, using more Spark instances will increase speed, but that may also incur additional overhead (e.g., communication between nodes). Here, we showed that for moderately-sized datasets, single-machine approaches like M3 can be attractive alternatives to distributed approaches.

[2]http://leon.bottou.org/projects/infimnist

[3]We are primarily interested in runtimes, so we did not perform image pre-processing.

4. CONCLUSIONS & ONGOING WORK

We are taking a first major step in assessing the feasibility of using virtual memory as a fundamental, alternative way to scale up machine learning algorithms. M3 adds an interesting perspective to existing solutions primarily based on distributed systems.

We contribute: (1) our latest finding that memory mapping could become a feasible technique for scaling up general machine learning algorithms when the dataset exceeds RAM; (2) M3, an easy-to-apply approach that enables existing machine learning implementations to work with out-of-core datasets; (3) our observations that M3 on a PC can achieve a speed that is significantly faster than a 4-instance Spark cluster, and comparable to an 8-instance cluster.

We will extend our M3 approach to a wide range of machine learning (including online learning) and data mining algorithms. We plan to extensively study the memory access patterns and locality of algorithms (e.g., sequential scans vs random access) to better understand how they they affect performance. We plan to develop mathematical models and systematic approaches to profile and predict algorithm performance and energy usage based on extensive evaluations across platforms, datasets, and languages.

5. ACKNOWLEDGMENTS

We thank Dr. Ryan Curtin and Minsuk (Brian) Kahng for their feedback. This work was supported by NSF grants IIS-1217559, TWC-1526254, IIS-1563816, and by gifts from Google, Symantec, Yahoo, eBay, Intel, Amazon, LogicBlox.

6. REFERENCES

[1] S. Boyd-Wickizer, A. T. Clements, Y. Mao, A. Pesterev, M. F. Kaashoek, R. Morris, N. Zeldovich, et al. An analysis of linux scalability to many cores. In *OSDI*, volume 10, pages 86–93, 2010.

[2] R. R. Curtin, J. R. Cline, N. P. Slagle, W. B. March, P. Ram, N. A. Mehta, and A. G. Gray. mlpack: A scalable C++ machine learning library. *Journal of Machine Learning Research*, 14:801–805, 2013.

[3] Z. Lin, M. Kahng, K. M. Sabrin, D. H. P. Chau, H. Lee, and U. Kang. Mmap: Fast billion-scale graph computation on a pc via memory mapping. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 159–164. IEEE, 2014.

[4] F. McSherry, M. Isard, and D. G. Murray. Scalability! but at what cost? In *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*, 2015.