Large-Scale Storage of Whole Slide Images and Fast Retrieval of Tiles Using DRAM

Daniel E. Lopez Barron[a], Praveen Rao[b], Deepthi Rao[b], Ossama Tawfik[c], and Arun Zachariah[b]

[a]University of Missouri-Kansas City, Kansas City, MO, USA
[b]University of Missouri-Columbia, Columbia, MO, USA*
[c]Saint Luke's Hospital of Kansas City, MO, USA

ABSTRACT

The U.S. Food and Drug Administration (FDA) has approved two digital pathology systems for primary diagnosis. These systems produce and consume whole slide images (WSIs) constructed from glass slides using advanced digital slide scanners. WSIs can greatly improve the workflow of pathologists through the development of novel image analytics software for automatic detection of cellular and morphological features and disease diagnosis using histopathology slides. However, the gigabyte size of a WSI poses a serious challenge for storage and retrieval of millions of WSIs. In this paper, we propose a system for scalable storage of WSIs and fast retrieval of image tiles using DRAM. A WSI is partitioned into tiles and sub-tiles using a combination of a space-filling curve, recursive partitioning, and Dewey numbering. They are then stored as a collection of key-value pairs in DRAM. During retrieval, a tile is fetched using key-value lookups from DRAM. Through performance evaluation on a 24-node cluster using 100 WSIs, we observed that, compared to Apache Spark, our system was three times faster to store the 100 WSIs and 1,000 times faster to access a single tile achieving millisecond latency. Such fast access to tiles is highly desirable when developing deep learning-based image analytics solutions on millions of WSIs.

Keywords: Whole slide imaging, scalable storage, fast retrieval, DRAM, image analytics

1. INTRODUCTION AND OBJECTIVE

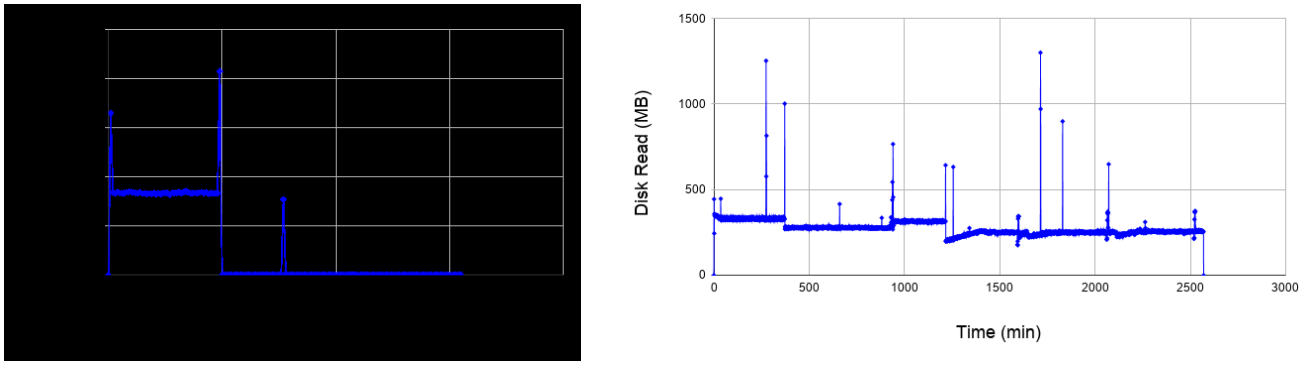
Whole slide imaging is touted as a disruptive technology in digital pathology and can produce a gigapixel image (or WSI) in a few minutes by scanning a glass slide with near-optical resolution.[1] There continues to be growing interest in using WSIs for primary diagnosis and consultation[2] as they can greatly improve the workflow of pathologists. From a regulatory perspective, Philips and Leica Biosystems have received FDA clearance to market their digital pathology systems for primary diagnosis in the US.[3,4] This is a major step forward to enable widespread adoption of WSIs for primary diagnosis; however, the technical barriers due to gigabyte-sized WSIs pose a serious challenge for large-scale storage and retrieval of WSIs. In fact, the Memorial Sloan Kettering Cancer Center plans to scan 40,000 slides per month and expects to produce millions of WSIs in the next few years;[5] this will create several petabytes of WSI data. Fast access to small portions of WSIs is highly desirable for next-generation image analytics on histopathology slides using deep learning. Thus, developing an effective solution for large-scale storage and retrieval of WSIs is a critical challenge.

Recently, Yildirim et.al.[6] studied the use of parallel file systems, Apache Hadoop,[7] and Amazon Web Services (AWS) cloud storage to enable scalable data access of WSIs. Their focus, however, was on retrieving the full (tiled) image of a particular resolution in a WSI. Motivated by the need for fast random access to small portions of a WSI, our previous work[8] showed that using a space-filling curve for data partitioning and Apache Spark[9]–a

*Part of this work was done when P.Rao and A. Zachariah were employed by University of Missouri-Kansas City.

Further author information: (Send correspondence to P. Rao)
D. Lopez Barron.: Email: dl544@mail.umkc.edu
P. Rao.: E-mail: praveen.rao@missouri.edu
D. Rao.: E-mail: raods@health.missouri.edu
O. Tawfik.: Email: ossama.tawfik@gmail.com
A. Zachariah.: Email: azachariah@mail.missouri.edu

To appear in SPIE Defense + Commerical Sensing: Big Data II: Learning, Analytics, and Applications Conference 2020 Conference, Anaheim, CA



(a) Small dataset (b) Large dataset

Figure 1. Disk read pattern during the training phase

popular in-memory cluster computing framework–can enable fast retrieval of (small) tiles in WSIs. Our system's latency to retrieve a title was in seconds. This is because data may need to be fetched from secondary storage at times and move through several software layers. *Given the large amount of DRAM in a commodity cluster, can we reduce the tile access latency if data are always kept in and fetched from DRAM?* Our work precisely attempts to answer this question.

2. MOTIVATION

To understand the IO access pattern during deep learning, we conducted a simple experiment. We trained an autoencoder[10] on the MS COCO dataset[11] containing 123K images of common everyday scenes. We ran the training on CloudLab[12] using a machine with Intel Core i9-9980XE CPU (3.00 GHz), 64 GB RAM, NVIDIA GeForce RTX 2080 Ti GPU (11 GB), and 2 TB hard disk. The original dataset was 19 GB; we created a larger dataset by replicating the original dataset 6 times (119 GB). We used the `dstat` command in Linux to measure the IO per second and then aggregated the IO per minute. In Figure 1, we report the read cost in MB per minute during the training of the autoencoder for 3 epochs on the two datasets. Figure 1(a) shows the amount of disk read for the 19 GB dataset. As expected after initially loading the dataset into main memory, the training phase used the cached files resulting in a total disk read of 20 GB. However, on the large dataset as shown in Figure 1(b), disk reads continued to occur during the training phase as the entire dataset could not be cached. The total disk read was 680 GB – 5.7 times the size of the large dataset. Thus, disk IO can become a bottleneck when training deep learning models on large datasets. Faster access to image tiles in the data can therefore reduce the training time leading to an improved image analytics pipeline.

3. BACKGROUND ON RAMCLOUD

Ousterhout et al. proposed RAMCloud,[13] a storage system that utilizes DRAM to store large datasets with low-latency access. RAMCloud provides strong consistency and organizes data in the form of a key-value store on thousands of servers. Data are kept in DRAM at all times. RAMCloud requires microsecond latency for performing reads (e.g., 5 μs) and writes of small objects. It uses a log-structured approach to manage the DRAM of the servers and employs secondary storage for backup and fault-tolerance. It relies on high-speed networking (e.g., Infiniband) that is becoming common in data centers, and uses polling and kernel bypass to achieve low latency and high throughput. In terms of performance, RAMCloud has outperformed state-of-the-art in-memory key-value stores and main-memory databases.[13]

Next, we describe RAMCloud's data model and processing primitives. RAMCloud organizes data in tables and is a key-value store. Each key can be up to 64 KB in size. The size of the value can be at most 1 MB. When a table is created with a name, it returns a unique ID. Using the ID, a key-value pair can be written to the table. Given a key, its value can be fetched from the table. RAMCloud supports multi-writes, wherein a set of key-value pairs can be written (into multiple tables) in a single user-invoked operation. Similarly, it supports multi-reads, wherein given a set of keys (and tables), their respective values can be fetched from the

corresponding tables in a single user-invoked operation. All key-value pairs in a table can be enumerated. A table is split into tablets (by hashing the keys in it) and stored across the DRAM of the servers. Data are always served from DRAM leading to low latency accesses.

4. OUR PROPOSED SYSTEM

Our system is designed to enable scalable storage of WSIs and fast retrieval of tiles in them. It uses a set of techniques to partition a WSI so that it can stored in a distributed manner across of cluster of machines. As a result, small portions of a WSI can be efficiently retrieved with millisecond latency. Our system is built atop RAMCloud and uses its key-value data model. It is illustrated in Figure 2.

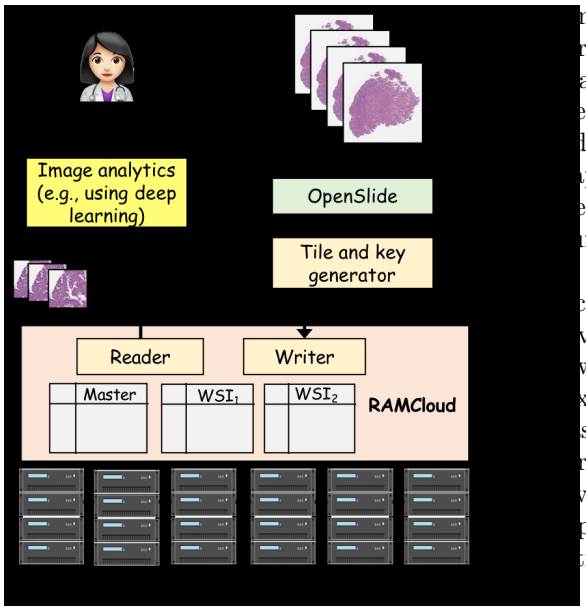
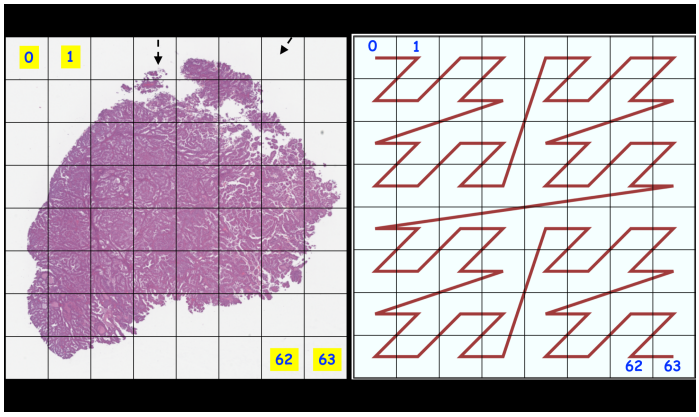


Figure 2. System architecture

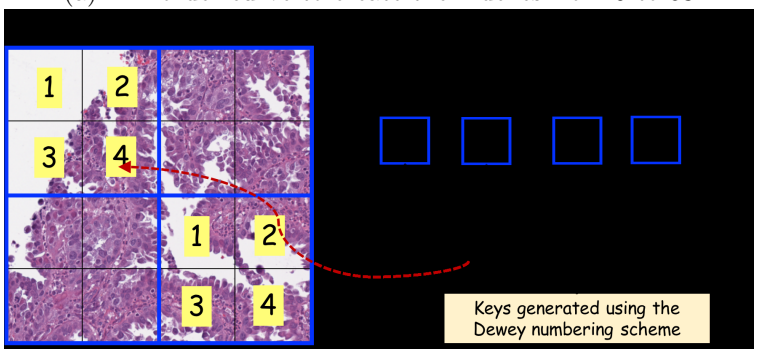
Storing WSIs: A WSI contains an image pyramid with different resolutions. The highest resolution is represented by a Level mage (e.g., 400X magnification) and is the largest in size com- red to lower resolution images at higher levels. As a Level 0 age can be several GBs in size, our system partitions it into s using a space-filling curve.[14] A space-filling curve (e.g., Z- ler curve, Hilbert curve) provides a mathematical function to p points in a higher dimensional space (e.g., 2D) to 1D, while serving spatial proximity of points in the higher dimension. r system is designed for fast image analytics where rectangu- regions of WSIs (containing one or more tiles) are randomly cessed. Thus, based on the space-filling curve, each tile in a vel 0 image is assigned an index from 1 to 2^r, where r is a wer of 2. Neighboring tiles tend be closer in terms of their in- x values, which is ideal when nearby tiles are to be retrieved. sed on the desired maximum size of a tile, a Level 0 image is rtitioned using different iterations of the space-filling curve. (A examples are shown later in Section 5.) A similar partitioning proach can be applied to a Level 1 image. A tile of a WSI is racted using OpenSlide.[15]

Our system maintains a table (in RAMCloud) called `Master` to store metadata required to retrieve tiles and sub-tiles at different levels of a WSI. Suppose we use Z-order curves for partitioning. For each WSI identified by `fileID`, a key-value pair (k, v) is stored in `Master`, where k is `fileID.imageLevel` and v is `(ZIteration, DeweyLevel, imageHeight, imageWidth)`, where `ZIteration` is the iteration of the Z-order curve, and `DeweyLevel` is based on the Dewey numbering scheme.[16] A `DeweyLevel` is 0 if the tiles created by the space-filling curve are less than 1 MB in size; hence, they are not recursively partitioned. Otherwise, a tile is partitioned into create sub-tiles until each sub-tile is less than 1 MB.

For each WSI, a table is created (in RAMCloud) with `fileID` as the name. This table will store the actual tiles and sub-tiles of a WSI. Given an image level, each tile is denoted by a key-value pair (k, v): The key k is `imageLevel.tileIndex`, where `imageLevel` is the image level and `tileIndex` is the index based on the Z-order curve. An example of creating tile indexes from 0 to 63 is shown in Figure 3(a). The value v is a tuple (`tileWidth`, `tileHeight`, `tileBytes`), where `tileWidth` is the width of the tile, `tileHeight` is the height of the tile, and `tileBytes` is actual tile content (e.g., in TIFF). If a tile does not contain any tissue section (see Figure 3(a)), then it's value is a magic string to indicate a blank tile. We employ a nuclei detection technique[17] to identify tiles with zero cell count. As long as a tile is under 1 MB in size, it can be directly stored in a RAMCloud table.



(a) A Z-order curve to create tile indexes from 0 to 63



(b) Dewey numbering for sub-tiles

Figure 3. (a) WSI partitioning (b) Recursive partitioning into sub-tiles (Sources: https://portal.gdc.cancer.gov, https://en.wikipedia.org/wiki/File:Four-level_Z.svg)

If a non-blank tile is greater than 1 MB in size, it must be partitioned recursively into sub-tiles. Consider the tile with `tileIndex` of 15 as shown in Figure 3(b). Suppose the size of this tile is greater than 1 MB. We first partition the tile into 4 equal quadrants. If each quadrant is still more than 1 MB, it must be partitioned again into 4 quadrants. Suppose each sub-tile is now less than 1 MB. Thus, the tile is partitioned into 16 sub-tiles. Each sub-tile is identified by a Dewey number as shown in Figure 3(b). For example, the top left sub-tile is denoted by `15.1.1`. The bottom right sub-tile is denoted by `15.4.4`. This approach enables us to generate keys deterministically once we keep track of the number of levels of recursive partitioning, which we refer to as the Dewey level. Recall that `DeweyLevel` is stored in `Master` for each WSI's image level. Each sub-tile is stored using `imageLevel.DeweyID` as the key, where `DeweyID` is the Dewey number of the sub-tile. To enable fast storage of tiles and sub-tiles in a WSI, we use RAMCloud's *multi-write* operation.

Querying Tiles of a WSI: A query is defined by a rectangular bounding box $[x_1, y_1, x_2, y_2]$ on a WSI with a given `fileID` and `imageLevel`. We map this bounding box into a set of tile indexes. We first fetch the metadata (such as `DeweyLevel`) from `Master` using `fileID.imageLevel` as the key. For each tile, we generate the Dewey IDs based on `DeweyLevel`. Using RAMCloud's *multi-read* operation, we retrieve all sub-tiles from the table `fileID` in a single call. The sub-tiles are then written to local storage.

5. PRELIMINARY RESULTS

We implemented our system using Scala, Python, and C++ APIs of RAMCloud. We conducted the performance evaluation of our system using a 24-node cluster on CloudLab,[12] a testbed for cloud computing research. Each node was installed with Ubuntu Linux 16.04.10 and had a 10-core Intel processor, 480 GB local SSD storage, 250 GB block storage, and 64 GB RAM. For the experiments, we used 100 WSIs (in SVS format) from The Cancer Genome Atlas (TCGA).[18] The total size of the WSIs was 273 GB. The SVS file sizes ranged from 2.4 GB to 3.8 GB. In Figure 4, we report the average tile size (in TIFF format) for a WSI's Level 0 image based on partitioning with different iterations of the Z-order curve. For our evaluation on 100 WSIs, we chose 7 iterations of the Z-order curve to ensure that the sub-tiles were under 1 MB with `DeweyLevel` of 1.

5.1 Storage Performance:

For each SVS file, we partitioned the Level 0 image data into 16,384 tiles. On an average, there were 5056 blank tiles per Level 0 image. These blank tiles were stored in an optimized way as discussed in Section 4. A non-blank tile was partitioned into 4 sub-tiles using DeweyLevel of 1. Both the tiles and sub-tiles were in TIFF format, and

Z-order iteration	# of tiles	Avg. tile size
3	64	616 MB
4	256	154 MB
5	1,024	39 MB
6	4,096	9.7 MB
7	16,384	2.5 MB

Figure 4. Partitioning into tiles

Figure 5. Retrieval performance

# of tiles retrieved	Avg. size of data retrieved
1	0.93 MB
3	2.85 MB
5	4.76 MB
7	6.66 MB

Figure 6. Avg. retrieval size

hence, required more storage than the original SVS files. (In fact, JPEG would reduce the tile size.) The total DRAM storage consumed by the 100 WSIs was 1.18 TB. It took an average of 18.22 secs to store each SVS file in our system. On the other hand, storing a single SVS file using Apache Spark took over 70 secs on an average. *Our system built atop RAMCloud was three times faster than using Apache Spark for storing WSIs.*[8]

5.2 Tile Retrieval Performance:

To understand the tile retrieval performance of our system, we generated queries to retrieve 1, 3, 5, or 7 tiles from each of the 100 WSIs. The multiread operation of RAMCloud was used to retrieve tiles/sub-tiles. Figure 5 shows the average elapsed time to retrieve different number of tiles. The average amount of data retrieved by a query is shown in Figure 6. The retrieval cost increased linearly with increase in the number of tiles. *Clearly, our system build atop RAMCloud achieved millisecond latency to fetch tiles from WSIs.* Compared to using Apache Spark and Spark SQL, our system was 1,000 times faster to retrieve a single tile. Such fast access to WSI tiles can enable fast training of deep learning algorithms on large-scale WSI data.

6. CONCLUSIONS

We presented a system for scalable storage of WSIs and fast retrieval of tiles using DRAM. Our system partitions WSIs into tiles and sub-tiles using a combination of space-filling curves, recursive partitioning, and Dewey numbering. These are stored as key-value pairs using RAMCloud, which is a cluster-based system that uses DRAM to store large datasets with low-latency access. Through performance evaluation on a 24-node cluster, we observed that our system can achieve millisecond latency to fetch single tiles. Such fast access to WSI tiles will be essential for large-scale deep learning on millions of WSIs for next generation diagnostic pathology.

7. ACKNOWLEDGMENTS

The first author was supported by the Mexican Council for Science and Technology (CONACyT) under scholarship 264829. The second author would like to acknowledge the partial support of NSF Grant No. 1841752. We thank CloudLab for providing computing resources to conduct the experiments.

REFERENCES

[1] Ghaznavi, F., Evans, A., Madabhushi, A., and Feldman, M., "Digital Imaging in Pathology: Whole-Slide Imaging and Beyond," *Annual Review of Pathology: Mechanisms of Disease* **8**, 339–351 (2013).

[2] Pantanowitz, L., Sinard, J. H., Henricks, W. H., Fatheree, L. A., Carter, A. B., Contis, L., Beckwith, B. A., Evans, A. J., Lal, A., and Parwani, A. V., "Validating whole slide imaging for diagnostic purposes in pathology: guideline from the College of American Pathologists Pathology and Laboratory Quality Center," *Archives of Pathology and Laboratory Medicine* **137**(12), 1710–1722 (2013).

[3] "FDA allows marketing of first whole slide imaging system for digital pathology." https://www.fda.gov/NewsEvents/Newsroom/PressAnnouncements/ucm552742.htm.

[4] "FDA Clears Leica Biosystems Digital Pathology System." https://www.fdanews.com/articles/191523-fda-clears-leica-biosystems-digital-pathology-system.

[5] "Computational Pathology – In the Midst of a Revolution: How Computational Pathology is Transforming Clinical Practice and Biomedical Research." http://on-demand.gputechconf.com/gtc/2017/presentation/s7603-thomas-fuchs-in-the-midst-of-revolution.pdf.

[6] Yildirim, E. and Foran, D. J., "Parallel Versus Distributed Data Access for Gigapixel-Resolution Histology Images: Challenges and Opportunities," *IEEE Journal of Biomedical and Health Informatics* **21**, 1049–1057 (July 2017).

[7] White, T., [*Hadoop: The Definitive Guide*], O'Reilly Media, Inc., 1st ed. (2009).

[8] Barron, D. L., Yarlagadda, D. V. K., Rao, P., Tawfik, O., and Rao, D., "Scalable storage of whole slide images and fast retrieval of tiles using Apache Spark," in [*Proc. SPIE 10581, Medical Imaging 2018: Digital Pathology*], **10581**, 1–6 (2018).

[9] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I., "Spark: Cluster Computing with Working Sets," in [*Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*], *HotCloud'10*, 10–10 (2010).

[10] "Convolutional Autoencoder." https://github.com/OliverEdholm/Convolutional-Autoencoder.

[11] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollar, P., and Zitnick, C. L., "Microsoft COCO: Common objects in context," in [*Computer Vision – ECCV 2014*], 740–755, Springer International Publishing, Cham (2014).

[12] Duplyakin, D., Ricci, R., Maricq, A., Wong, G., Duerig, J., Eide, E., Stoller, L., Hibler, M., Johnson, D., Webb, K., Akella, A., Wang, K., Ricart, G., Landweber, L., Elliott, C., Zink, M., Cecchet, E., Kar, S., and Mishra, P., "The Design and Operation of CloudLab," in [*Proceedings of the USENIX Annual Technical Conference (ATC)*], 1–14 (July 2019).

[13] Ousterhout, J., Gopalan, A., Gupta, A., Kejriwal, A., Lee, C., Montazeri, B., Ongaro, D., Park, S. J., Qin, H., Rosenblum, M., Rumble, S., Stutsman, R., and Yang, S., "The RAMCloud Storage System," *ACM Transactions on Computer Systems* **33**(3), 7:1–7:55 (2015).

[14] Sagan, H., [*Space-Filling Curves*], Springer Verlag (1994).

[15] "OpenSlide." https://openslide.org (2017).

[16] Tatarinov, I., Viglas, S. D., Beyer, K., Shanmugasundaram, J., Shekita, E., and Zhang, C., "Storing and Querying Ordered XML Using a Relational Database System," in [*Proc. of the 2002 ACM SIGMOD Conference*], 204–215 (2002).

[17] "Nuclei Detection with OpenCV." https://www.kaggle.com/javidimail/nuclei-detection-with-opencv (2018).

[18] "The Cancer Genome Atlas." https://www.cancer.gov/tcga (2019).