Big Data for Medical Image Analysis: A Performance Study

Rui Zhang, Hongzhi Wang, Renu Tewari, Gero Schmidt and Deepika Kakrania
IBM Research - Almaden
650 Harry Rd, San Jose, CA 95120, USA
Contact author email: ruiz@us.ibm.com

***Abstract*—Big data systems can be used to facilitate powerful medical image analysis at scale. Understanding their behaviors in this context can lead to many benefits, ranging from superior infrastructure configurations to optimized parallel algorithm implementations. This paper is, to our knowledge, a first step towards developing such an understanding for state-of-the-art big data platforms. We characterize a representative medical image segmentation pipeline, detailing the per-stage CPU, memory, I/O reads and writes, and execution time patterns. This characterization has already helped us overcome a bottleneck persistently causing analysis to crash unexpectedly, and avoid poor architecture choices on storage and parallel execution.**

I. Introduction

Big data analytics is coming to healthcare. Around 75% of U.S. non-federal acute care hospitals are adopting basic electronic healthcare record systems [1]. Emerging internet-of-things technologies, such as wearable fitness trackers and health apps on smart phones, are making it easier for personal health data to be continuously collected from millions of consumers in a pervasive fashion. A single institution could host medical imaging data in tens and hundreds of TBs [2]. It is not merely a matter of Volume. The remainders of the big data 5Vs – Volume, Velocity, Variety, Veracity and Value [3] – are also increasingly evident in analytics for healthcare. For instance, the need for Velocity can be found in medical image analysis where sophisticated computation is performed on images arriving at high rates [4], while Variety arises when images are further combined with structured patient records.

Medical image analysis, as an important type of healthcare analytics, can rely on highly specialized pipelines that compare and aggregate 3D images in unique ways. The behaviors of this relatively new class of analytics, from a distributed systems perspective (e.g. IO bandwidth consumption, execution time), have yet to be understood. Such an understanding could yield great benefits for medical image analysis in particular, given its high computational costs. To our knowledge, machine learning algorithms can also be used for medical image analysis and there exist systems performance studies of machine learning algorithms [5], [6] that have provided useful insights of their behaviors for performance optimizations. Nonetheless, machine learning algorithms are generally iterative and can have high I/O utilization. By contrast, specialized medical image analysis pipelines such as the one discussed in this paper are often not iterative and can have dramatically different resource consumption patterns. Due to the fundamental differences in acquisition devices and imaging conditions, steps crucial for medical image analysis, such as image registration and intensity bias correction, are generally not applicable to regular images. Furthermore, compared to regular 2D images (e.g. photos), the 3D format exacerbates performance bottlenecks. Hence, known system behaviors of regular image analysis would not apply to medical image analysis.

This paper aims to provide a first characterization of the unique systems performance and resource behaviors of medical image analysis in a big data setup. This is contrary to one research [7] that primarily aimed to build models to make CPU and memory consumption predictions for a medical imaging tool used in high performance computing (HPC). In particular, we focus our study on the analytics pipeline for anatomy segmentation, which identifies anatomical structures in medical images as an essential building block for various further analysis. This pipeline can be widely applied to many medical imaging problems and is not limited to any specific imaging modality. The research contributions made by this paper are as follows:

- A detailed systems-level characterization (including CPU, memory, I/O reads and writes, and execution time) for each stage in parallel anatomy segmentation using state-of-the-art big data frameworks such as IBM® Platform LSF® [8], Apache Spark [9] and IBM® Spectrum Scale™ [10]. The study has revealed surprising findings, including one bottleneck that causes jobs to crash unexpectedly.
- A set of practical performance recommendations for medical image analysis in a big data setting, including how to make best use of storage, memory and Spark.
- A list of research directions with regard to both algorithms and system platform optimization.

The following section introduces anatomy segmentation and its big data implementations. Section III presents the performance

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

study. Important insights are further discussed in Section IV before we conclude in Section V with future topics.

II. Background

This section briefly introduces the multi-atlas analysis and the big data framework used in the performance study.

A. Medical image analysis and multi-atlas segmentation

Medical image analysis aims to quantitatively extract clinical information from medical images. In this domain, anatomy segmentation is a crucial building block, where the goal is to locate and outline the boundaries of distinct anatomical structures from images. Label fusion based multi-atlas segmentation is a highly competitive, widely applicable segmentation technique [11]. This technique, as depicted in Figure 1, applies deformable image registration to establish one-to-one correspondence between each pre-labeled training image, called an atlas, and a novel target image. Then, the segmentation label is transferred to the target image by warping each atlas based on the correspondence (Stage 1, Image Registration). Each warped atlas provides one candidate segmentation for the target image. Due to anatomy variation across subjects and errors produced in image registration, each candidate segmentation may contain errors. To reduce segmentation errors produced by registration-based label transfer, label fusion is applied to integrate all candidate segmentations into a consensus segmentation (Stage 2, Label Fusion). Multi-atlas segmentation has been successful in addressing a broad range of segmentation problems.

The image-based registration in this study was computed using the Advanced Normalization Tools (ANTS) software [12]. The registration sequentially optimized an affine transform and a deformable transform (Syn) between the pair of images, using the Mattes mutual information metric. The gradient step was set to 0.1. Three resolution levels, with maximum 200 iterations at the coarse, 100 iterations at the middle levels and 50 iterations at the finest level were applied. We applied image similarity based local weighted voting for label fusion. The voting weights were computed using joint label fusion [13]. We applied the joint label fusion software distributed from ANTS with the default parameters.

B. Parallel segmentation via LSF, Spark and Spectrum Scale

This paper focuses on the parallel execution of the pipeline introduced in Section II-A in a computing cluster over a distributed storage layer that supports data sharing across nodes and between pipeline stages. A data parallel approach was taken, where multiple jobs were launched each processing a subset of the input data. In a realistic setting, a fixed set of n training images would be used for the segmentation of incoming novel target images. For each new target image, n Image Registration jobs are launched, each conducting the warping between the target image and a training image. n candidate segmentations are produced from Image Registration jobs (on different nodes) and written to the distributed storage layer. These outputs are subsequently read from the storage

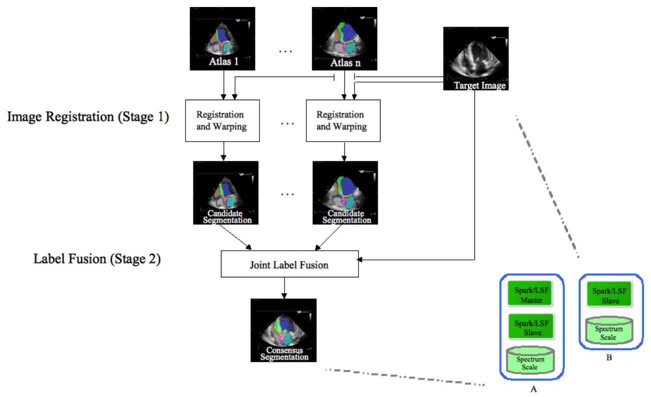

Fig. 1. Multi-atlas segmentation pipeline on a big data cluster.

layer by a subsequent Label Fusion job and merged according to Section II-A. Note that as a whole the environment is effectively handling $n \times m$ Image Registration jobs and m Label Fusion jobs, where m is the number of target images being processed at any time. These distributed file I/Os facilitated by the storage layer are the only form of network communication required by parallel segmentation. Since these I/Os are captured in our performance study, we do not discuss network communications separately.

Both LSF and Spark are considered for parallel execution. At the highest level, both feature a centralized job manager that works with a scheduler local to each cluster node to dispatch concurrent jobs, typically one per CPU core. LSF has its roots in HPC but is applicable to big data, whereas Spark is a new big data framework centered around caching data in resilient distributed datasets (RDDs) [9] for repeated access by iterative algorithms. We have provided a full implementation of the multi-atlas segmentation pipeline for LSF and completed an initial prototype of Image Registration in Spark.

The storage layer is IBM Spectrum Scale, a cluster file system with advanced features such as data striping for parallel access, replication and object storage support. Unlike HDFS [14], Spectrum Scale supports the POSIX protocol, which makes it easy to expose data to a non-Hadoop frameworks like LSF. The single name space provided by Spectrum Scale makes it easy for multi-atlas jobs to access images across the cluster (i.e. on a different node from the one where the job is launched).

III. Performance Study

The study is primarily focused on the LSF-based multi-atlas segmentation implementation, but we also discuss the Spark implementation where appropriate.

A. Experimental setup

The performance study was carried out on a cluster of two high performance nodes running Red Hat Enterprise Linux release 7.1. Each node was equipped with two 4-core processors (2.40 GHz, 2 threads per core), 32 GB of memory, and two 64 GB SATA solid-state drives (SSD). Spectrum Scale

4.2 was distributed across the two nodes and an additional third storage node utilizing a 10 Gbps Ethernet network. Although more nodes could be used, we believe the 2-node setup makes it easier to push resource (e.g. memory) limits and is sufficient to monitor parallelism (up to 32 concurrent jobs). Each launched job could access any data distributed in Spectrum Scale, but only had access to local memory.

The brain magnetic resonance imaging (MRI) dataset from the MICCAI multi-atlas labeling challenge [15] was used in our study. The original dataset contains 20 training images and 15 testing images. Anatomical labels for 140 structures are labeled for each training images. We used a subset of 18-20 MBs for the experiments to keep running time reasonable.

B. *Performance and resource usage characterization*

We submitted $m = 4$ target images to LSF concurrently and each target image was segmented by referencing $n = 3$ training images. This produced $4 \times 3 = 12$ concurrent Image Registration jobs and 4 Label Fusion jobs.

Image Registration jobs were scheduled in a balanced fashion and each node handled 6 jobs. Only 3 jobs were allowed to run concurrently at a time, resulting in 2 waves of 3 concurrent jobs on each node. Figure 2 depicts the memory (left), CPU (center) and I/O (right) utilization of these jobs on cluster node A. Node B had almost identical patterns which are not reported here. Memory was well utilized and there was a gradual utilization increase initially as more images and supporting data structures were imported by the jobs. In fact, memory was the main performance bottleneck and we had to reduce the concurrent jobs on a single node to 3, so as not to trigger out-of-memory failures (more discussions in Section IV-A). With memory utilization limited, the next performance bottleneck manifested itself in the figure, as CPU utilization constantly reached 100%. Against our initial intuition, I/O was not a bottleneck, as there were only sporadic I/Os corresponding to reading images and writing output files.

For Label Fusion, to have a more direct comparison with Image Registration, we manually enforced the same job concurrency on node A (i.e. 3 concurrent Label Fusion jobs) as during Image Registration. As shown in Figure 3, overall Label Fusion is far less resource-intensive than Image Registration, with CPU utilization under 20% and up to 3x lower memory utilization. I/O was similarly sparse. Table I further shows that Label Fusion completes over 10x faster than Image Registration. We focused the remainder of the paper on Image Registration, as it is far more computation and data intensive.

C. *Initial sensitivity analysis*

The observations in Figure 2 remain almost entirely the same when Spark was used instead of LSF. Hence the patterns discussed in this section are insensitive to specific scheduling frameworks and inherent to parallel multi-atlas segmentation.

Initial analysis suggests that the execution time and resource consumption increases nearly linearly with the number of training (or target) images. They have an inverse correlation with the number of CPU cores in the cluster. We plan to conduct proper scalability experiments.

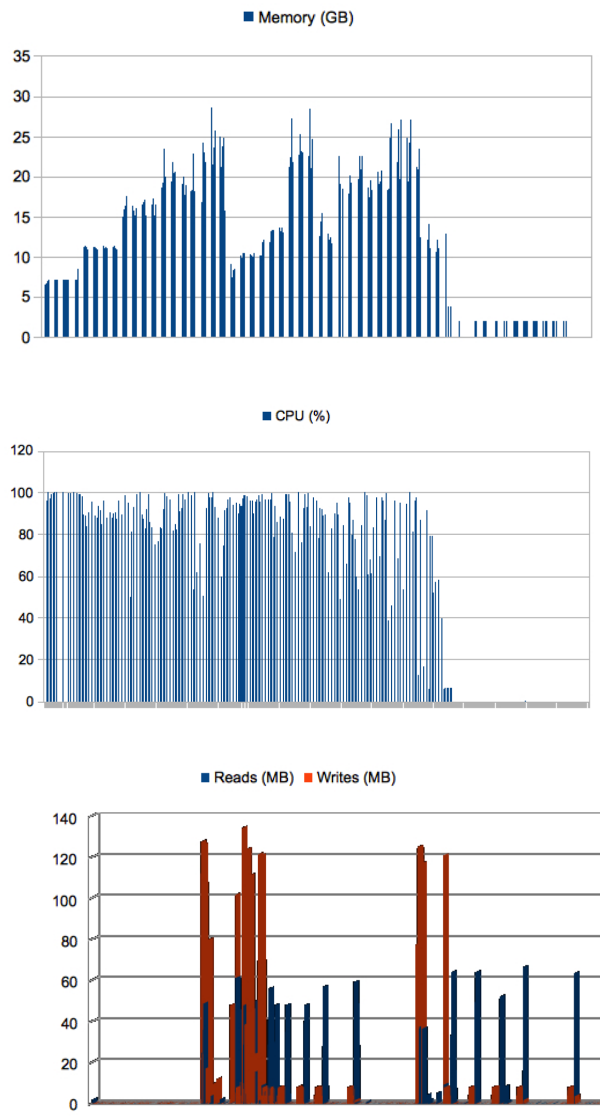

Fig. 2. Image Registration resource usage. X-axis is time in seconds (i.e. I/O read and write throughput is measured in MBs per second).

TABLE I
EXECUTION TIME BREAKDOWN (HOURS)

Image Registration	*Label Fusion*	*Total*
2.68	0.2	2.88

IV. Insights and Recommendations

We present a focused discussion of several insights unearthed in Section III and suggest practical optimizations.

A. *Memory capacity is the main bottleneck*

While it is not entirely surprising that the analysis (particularly Image Registration) is memory-bound, we were not

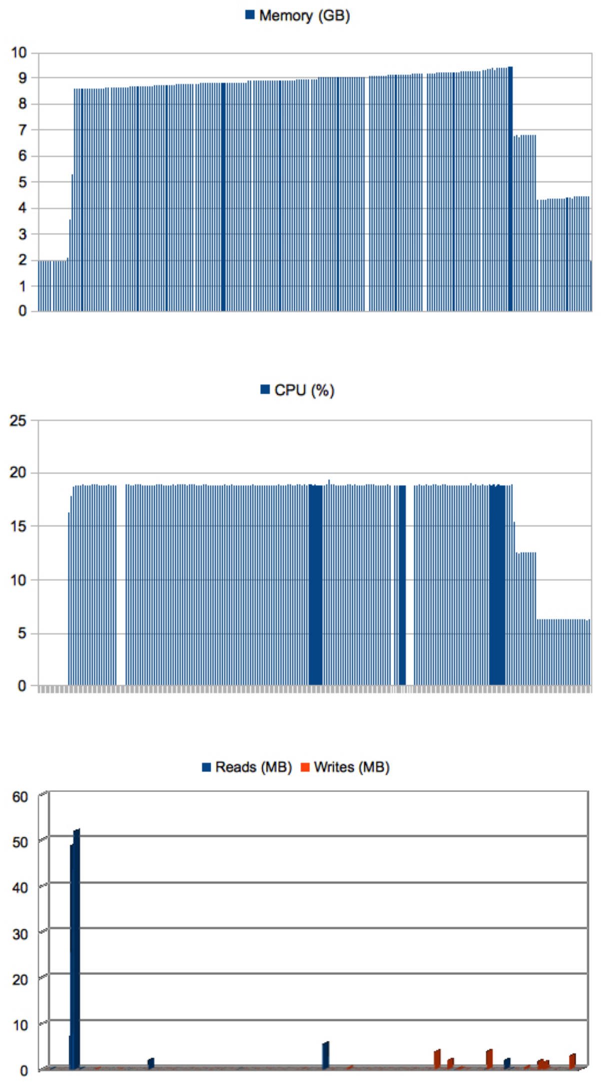

Fig. 3. Label Fusion resource usage. X-axis is time in seconds (i.e. I/O read and write throughput is measured in MBs per second).

prepared for how hungry it is. In Figure 2, memory consumption was kept just under system capacity by limiting the number of concurrent Image Registration jobs to 3 per node. In earlier runs when no limit was in place, memory capacity was easily exceeded and the jobs failed in these runs (often after running for an hour or two), most likely due to unanticipated memory allocation exceptions. Note that, ANTS, a widely used image registration software is used in these jobs. Hence such catastrophic failures may be common place in the current (or even future) generation of medical image analysis. Even if it is feasible to change the source code to properly handle such memory failures (e.g. by using external storage for virtual memory), the performance degradation could still be extremely severe. To address the memory bottleneck, we plan to investigate the possibility of partitioning the jobs into smaller ones that require less memory at a time. However, this is non-trivial given the complexity of image registration algorithms and can potentially involve expensive changes to the ANTS registration code. Meanwhile, some practical measures can be taken to cope.

Recommendation 1: Balanced job scheduling across nodes is helpful, as it fairly distributes the memory burden across cluster nodes. This reduces the possibility of nodes with inadequate memory capacity being given too many jobs and running out of memory. In a homogeneous cluster, jobs may only need be distributed evenly (as in Section III); whereas in a heterogeneous cluster, more intelligence is needed to distribute more jobs and memory demands to nodes with greater resources. In our study, it was obvious that the memory demand per Image Registration job would be around 8 - 10 GB and we chose to limit the number of concurrent jobs to 3 given a 32 GB capacity per node. Scheduling intelligence is needed to distribute jobs and memory demands according to resources on each node. In our study, it was obvious that the memory demand per Image Registration job would be around 8 - 10 GB and we chose to limit the number of concurrent jobs to 3 given a 32 GB capacity per node.

Recommendation 2: A distributed memory-sharing layer may be used, in the absence of balanced scheduling or when it is difficult to forecast the memory demand. As such, free memory from nodes with less work can be leveraged to alleviate a memory hot spot on other nodes.

B. *CPU is the next bottleneck*

Once we manually limited memory consumption to under system capacity, CPU utilization reached and maintained 100%, indicating a bottleneck shift. Although the CPU bottleneck does not have as catastrophic effects as the memory bottleneck, it increases run time and limits throughput.

Recommendation 3: increasing CPU capacity would lower analysis time, assuming that enough memory is provisioned. Although this suggestion may appear obvious, it is not always valid. When CPU utilization is low, any additional processing power provided is likely not leveraged. In our study, however, since CPU was fully utilized, any additional CPU cycles can help reduce execution time. This can be achieved by increasing the number of CPU cores and memory proportionally or by offloading CPU computation to GPUs.

C. *I/O performance is insignificant*

Prior to the study, we were under the illusion that I/O performance would be important, given that the size of medical images easily surpasses many other data types one may encounter in the big data arena: item ratings, tweets, photos etc. To our surprise, there were only sporadic I/O activities which accounted for negligible fractions of job execution time. Obviously slower, less expensive storage devices and networks could now be used without much performance impact, but this realization also led us to an important architecture suggestion.

Recommendation 4: Shared distributed storage should be preferred to shared-nothing storage. Shared storage, which allows all storage to be accessed from any nodes, can be set up as a separate tier of nodes that are easier to manage. The shared-nothing model, by contrast, requires compute and storage to be configured on the same set of nodes and jobs to be scheduled on nodes where the data resides. Such data locality makes little performance difference given our observations and makes data harder to share across nodes.

D. Multi-atlas segmentation is not naturally Spark-compatible

We have not found Spark easy (vs. LSF) to integrate with multi-atlas segmentation. The key Spark performance feature is its ability to cache data for repeated, parallel access by interactive algorithms. But it is not easy for multi-atlas segmentation to utilize Spark caching.

Recommendation 5: Considerable application rewriting and optimizations are needed to fully leverage certain big data frameworks. Significant re-writing of the segmentation pipeline is needed to fully leverage Spark. RDDs need to be created for both images and intermediate data to truly benefit from Spark parallelism. What is more, although training images are re-used as a new target image arrives, any existing intermediate data remain dependent on the previous target image and not trivially reusable (cache-able). This is because deformed registration fundamentally consists of target-training image pair computations.

V. Conclusions and Future Works

This paper has presented, to our knowledge, some of the first insights on how healthcare analytics like multi-atlas segmentation behave in big data systems. While it is our belief that this initial set of results can already provide great values, we recognize that there is much potential for a further, more comprehensive study. Future works include:

- Exploring the possibility of partitioning the jobs into smaller ones that require less memory at a time.
- Investigating the possibility of (a) swapping data to external storage when it runs out of memory and (b) bringing jobs to the nodes where the image data resides.
- Scaling up the experiments in multiple dimensions: cluster size, training and target image sizes, and the number of training and target images, as well as various more detailed resource performance counters such as those in relation to the CPU cache and memory.

References

[1] D. Charles, M. Gabriel, and T. Searcy, "Adoption of electronic health record systems among u.s. non-federal acute care hospitals: 2008-2014," *ONC Data Brief*, no. 23, 2015.

[2] S. Sarcar, "Data explosion in medical imaging," in *Presentations at the Fifth Elephant - a conference on Big Data in Bangalore*, July 2012.

[3] Y. Demchenko, P. Grosso, C. De Laat, and P. Membrey, "Addressing big data issues in scientific data infrastructure," in *Collaboration Technologies and Systems (CTS), 2013 International Conference on*. IEEE, 2013, pp. 48–55.

[4] K. Mader, "Scaling up fast: Real-time image processing and analytics using spark," in *Spark Summit Presentations*, 2014.

[5] M. Li, J. Tan, Y. Wang, L. Zhang, and V. Salapura, "Sparkbench: A comprehensive benchmarking suite for in memory data analytic platform spark," in *Proceedings of the 12th ACM International Conference on Computing Frontiers*, ser. CF '15. New York, NY, USA: ACM, 2015, pp. 53:1–53:8.

[6] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, 2010, pp. 41–51.

[7] R. Banalagay, K. J. Covington, D. Wilkes, and B. A. Landman, "Resource estimation in high performance medical image computing. neuroinformatics," *Neuroinformatics*, vol. 12, no. 4, 2014.

[8] "Lsf." [Online]. Available: http://www-03.ibm.com/systems/platformcomputing/products/lsf/

[9] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. 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*. USENIX Association, 2012, pp. 2–2.

[10] "Spectrum scale." [Online]. Available: http://www.ibm.com/systems/storage/spectrum/scale/

[11] T. Rohlfing, R. Brandt, R. Menzel, D. B. Russakoff, and C. R. M. Jr, "Quo vadis, atlas-based segmentation?" *The Handbook of Medical Image AnalysisVolume III: Registration Models*, pp. 435–486, 2005.

[12] B. Avants, C. Epstein, M. Grossman, and J. Gee, "Symmetric diffeomorphic image registration with cross-correlation: Evaluating automated labeling of elderly and neurodegenerative brain," *Medical Image Analysis*, vol. 12, no. 1, pp. 26–41, 2008.

[13] H. Wang, J. W. Suh, S. Das, J. Pluta, C. Craige, and P. Yushkevich, "Multi-atlas segmentation with joint label fusion," *IEEE Trans. on PAMI*, vol. 35, no. 3, pp. 611–623, 2013.

[14] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, ser. MSST '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/MSST.2010.5496972

[15] B. Landman and S. Warfield, *eds. MICCAI 2012 Workshop on Multi-Atlas Labeling*. CreateSpace, 2012.