# Technology Enablers for Big Data, Multi-Stage Analysis in Medical Image Processing

Shunxing Bao*, Prasanna Parvarthaneni*, Yuankai Huo*, Yogesh Barve*, Andrew J. Plassard*,
Yuang Yao*, Hongyang Sun*, Ilwoo Lyu*, David H. Zald†, Bennett A. Landman* and Aniruddha Gokhale*
*Dept. of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37235, USA
†Dept. of Psychology and Psychiatry, Vanderbilt University, Nashville, TN 37235, USA

***Abstract*—Big data medical image processing applications involving multi-stage analysis often exhibit significant variability in processing times ranging from a few seconds to several days. Moreover, due to the sequential nature of executing the analysis stages enforced by traditional software technologies and platforms, any errors in the pipeline are only detected at the later stages despite the sources of errors predominantly being the highly compute-intensive first stage. This wastes precious computing resources and incurs prohibitively higher costs for re-executing the application. The medical image processing community to date remains largely unaware of these issues and continues to use traditional high-performance computing clusters, which incur a high operating cost due to the use of dedicated resources and expensive centralized file systems. To overcome these challenges, this paper proposes an alternative approach for multi-stage analysis in medical image processing by using the Apache Hadoop ecosystem and offering it as a service in the cloud. We make the following contributions. First, we propose a concurrent pipeline execution framework and an associated semi-automatic, real-time monitoring and checkpointing framework that can detect outliers and achieve quality assurance without having to completely execute the expensive first stage of processing thereby expediting the entire multi-stage analysis. Second, we present a simulator to rapidly estimate the execution time for a given multi-stage analysis, which can aid the users in deciding the appropriate approach for their use cases. We conduct empirical evaluation of our framework and show that it requires 76.75% lesser wall time and 29.22% lesser resource time compared to the traditional approach that lacks such a quality assurance mechanism.**

***Keywords*—Hadoop, Medical image processing, Big data multi-stage analysis, Simulator.**

## I. Introduction

The research presented in this paper addresses the execution time variability incurred by big data, multi-stage analysis applications found in medical image processing and reduces the cost of hosting these applications in the cloud. A typical medical image processing job involves multiple stages of analysis activities as shown in Figure 1. For instance, to conduct a group image analysis, the voxels[1] of raw individual images are processed and quantified in the first stage to provide a filtered matrix or a mathematical value. When all the first stage jobs are completed, the collected quantified data for the group are fed to a second stage or subsequent higher stages to perform a group-based statistical analysis. The processing of such multi-level analysis jobs is, however, complicated by the fact that it often must deal with heterogeneous magnetic resonance imaging (MRI) modalities acquired by different protocols, e.g., Diffusion weighted imaging (DWI), T1-weighted (T1W) or T2-weighted (T2W). Hence, non-unified image resolution and intensity may lead to different processing speeds even in the same medical image processing pipeline or algorithm [18], [27]. To our knowledge, there has been no study to date to address these performance issues. Moreover, the medical image processing community has traditionally utilized expensive high performance clusters for their multi-stage analysis, which when combined with the performance issues makes executing these jobs prohibitively expensive. Thus, a key issue in medical image processing is dealing with the significant variance in the processing times incurred by such multi-stage analysis.

To support our claim, we collected anonymous trace statistics logs of completed jobs from the XNAT system [19] for 4 years worth of multi-stage analysis without accessing any Protected Health Information (PHI) or imaging data. These jobs were executed in Vanderbilt's high performance computing cluster called Advanced Computing Center for Research and Education (ACCRE), which is representative of modern high performance clusters. The ACCRE cluster is deployed with a 10 Gigabit Ethernet on 12 racks with IBM's General Parallel Filesystem (GPFS). Each rack has 1 Gigabit Ethernet for its computation nodes. The total CPU cores is over 6,000. The anonymized statistics logs involved a total of 96,103 jobs whose completion times ranged from 15 seconds to 9 days.

Although these jobs were single image-based with first stage analysis (e.g., performing multi-atlas segmentation or surface mapping) and the output image(s) varied based on the type of processing, the data illustrates the compute-intensive nature and execution time variability of the first stage. One key reason for the long execution times was that within the ACCRE cluster, the input data for the jobs are transferred from GPFS through a high speed switch to the available cores of each computation node, and the resulting output is returned to GPFS over high speed networks. However, the computation nodes within the same rack share the low-speed network bandwidth, which often leads to network saturation and impacts I/O intensive jobs. Thus, researchers and practitioners who are routinely involved in the execution of such pipelines will incur a high price for using these clusters besides paying the high maintenance fees for using GPFS and the high speed networks.

Our study also revealed a critical insight about the multi-stage analysis pipeline. We observed that in comparison to the first stage processing times, the second and subsequent stages are usually substantially faster. The first stage analysis deals with much more data cleaning and preprocessing in preparation

[1]A medical image is usually collected as a series of two-dimensional image slices, and combined or stacked to get a three-dimensional image. The intensity value of the image, represented in the units of "voxel," is stored either as a three-dimensional cube or a three-dimensional matrix.

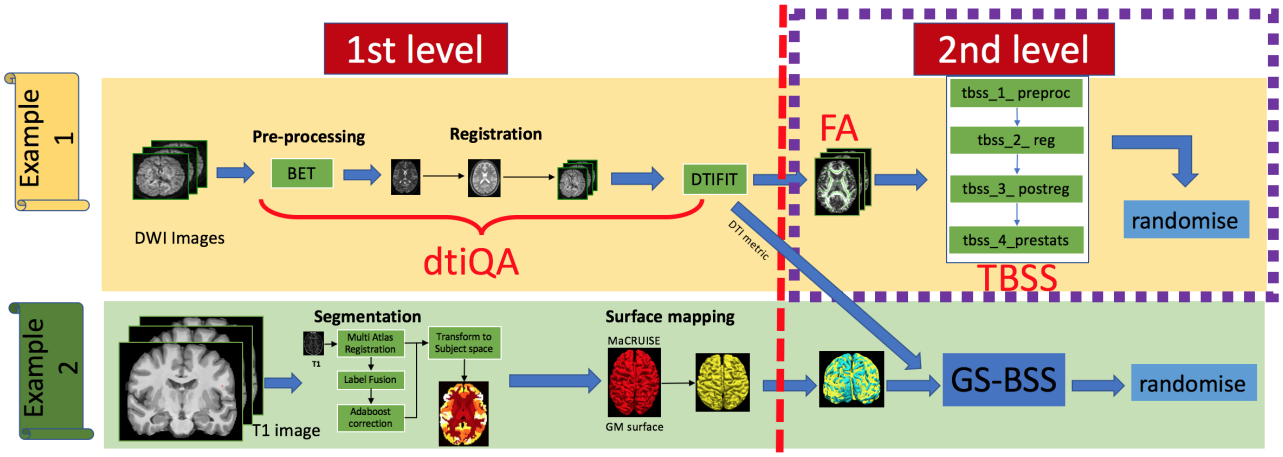



Fig. 1. A classical medical image processing multi-level analysis Tract-Based Spatial Statistics and Gray Matter Surface-based Spatial statistics. Our target pipeline for this work is the first example dtiQA & TBSS pipeline. The purple box indicates our second contribution focus area.

for cross subject analysis. It is thus very common for the first stage to take on the order of days to even months to process a group of images. Existing platforms to execute the pipeline force it to execute the stages in a serial order, which means that the second and subsequent stages of analysis cannot start executing until the first stage is entirely complete. Consequently, any outliers or anomalies observed in the later stages of the analysis pipeline, which are often a result of anomalies in the first stage (e.g., due to wrong parameter, incorrect pipeline design, mistakes in input data collection) are detected only very late in the processing. This wastes significant amount of resources, time and money, and requires re-executing the first stage after correcting the errors, which again requires very long running times. Unfortunately, there is no existing capability that can detect outliers in the first stage in a timely manner such that early detection and corrective action can be taken without unduly wasting resources and time. Thus, a key requirement to deal with is to identify errors as early as possible and to promote concurrent execution of the pipeline thereby speeding up execution time and saving the cost of executing these jobs.

To address these challenges, we propose a concurrently executing medical image processing pipeline with early error/anomaly detection via periodic monitoring of intermediate results in the first stage. Our approach is based on the well-known Apache Hadoop ecosystem [1], which has been shown to be effective in other application domains (e.g., fraud detection, web search). Since the Hadoop ecosystem integrates storage with the computation node, medical image processing jobs can be completed faster with minimal data transfer. Moreover, the Hadoop ecosystem is amenable to be hosted in the cloud and does not require expensive systems and high maintenance fees for GPFS. By using the Hadoop ecosystem, we monitor intermediate results of the compute-intensive first stage and catch anomalies early on before passing the intermediate processed data to the next stage, thus enabling concurrent processing of the pipeline stages.

Despite this promise, however, before deploying medical image processing jobs in the cloud and using our new approach, researchers and practitioners will require some way to estimate the potential performance gains (and lowering of cost) in using our cloud-based solution. Since such an estimate cannot be obtained by actually executing the jobs on the cluster or cloud resources, we also present a simulation framework that can estimate the performance requirements of medical image processing jobs under different large-scale cloud resource allocations and running mixed types of workloads.

The proposed innovations are empirically evaluated in a private, research cloud comprising a Gigabit network and 188 CPU cores. For larger scalability estimation, we use the historical trace data and test it on the simulation engine. To evaluate the concurrent pipeline with early error detection methodology, we use a real-world, two-stage medical image processing job. The evaluation results show that our proposed framework requires 76.75% less wall time and 29.22% less resource time compared to the traditional approach that lacks the quality assurance mechanism.

The rest of the paper is organized as follows: Section II compares our work with related work; Section III describes the contributions that realize the medical image processing-as-a-service; Section IV describes our evaluation approach and presents experimental results; and finally Section V presents concluding remarks alluding to ongoing and future work, and discusses the broader applicability of our approach.

## II. Related Work

In this section we summarize a sampling of prior efforts. Specifically, we focus on related research along the following dimensions: cloud-based medical image processing, and efforts that focus on quality assurance of these jobs.

### A. Literature Survey

*1) Cloud-based Deployment:* Wang et al. [24] develop a novel ultrafast, scalable and reliable image reconstruction technique for four-dimensional CT (4DCT) and cone beam CT (CBCT) using MapReduce. They show the utility of MapReduce for solving large-scale medical physics imaging problems in a cloud computing environment. The Java Image Science Toolkit (JIST) is a tool that integrates with Amazon Web Service (AWS) EC2 and Amazon S3 storage to perform medical image processing, which submits local first stage analysis to AWS to utilize the pay as you go feature of the cloud [6]. Huo et al. [21] provide a dockerizing approach for deploying large-scale image

processing to High Performance Computing environment and potential affordable cloud.

Zhang et al. [34] implement a distributed computing framework using Apache Spark cloud for automatic segmentation of skeletal muscle cell image. The paper aims for load balancing on available resources of the Spark cluster for the muscle cell segmentation, and propose a parallelized hierarchical tree-based region selection algorithm to efficiently segment muscle cells. Roychowdhury et al. [29] proposed the Azure-based Generalized Flow for medical image classification. The flow automates generalized workflow by combining the efficacy of cloud-computing frameworks with machine learning algorithms for medical image analysis.

The above cloud-based services are either for clinical usage, hosting the application for experiencing the benefit of using cloud, or to ease the interaction and deployment cost of using cloud. However, they do not aim to deal with cost, rapid execution, or design for multi-stage medical image processing of group-based analysis, which illustrates a different set of challenges. Moreover, none of them solve the performance improvement and quality assurance of intermediate results via early error detection as we propose in this paper.

*2) Quality Assurance in Medical Image Processing:* In [10], a quality assessment framework is presented that leverages MapReduce to find and assess errors in large medical datasets. It presents an approach to finding any errors in a medical dataset by formulating a SPARKL super-query having common-join elements. Their technique avoids redundant computation of joins, which enables them to complete over 80 join operations using two Map/Reduce iterations. Unlike our work which focuses on the medical image processing of datasets, their work focuses on finding errors in medical datasets themselves.

There are several medical image processing systems and pipeline toolboxes, e.g. FSL [31] and JIST. These tools focus on providing a medical image processing pipeline, however, without a focus on the quality assurance. Although our proposed work on early error detection works in the context of the Hadoop ecosystem, our ultimate goal is to carry over our ideas to these existing frameworks.

Many prior efforts exist that deal with performance optimization and fault detection strategies in HPC and cloud environment, however, we have not found any effort that addresses the challenges in Big Data medical image processing multi-stage pipelines. Naksinehaboon et al. [25] built a model that aims to reduce full checkpointing overhead by performing a set of incremental checkpoints between two consecutive full checkpoints. Moreover, they demonstrated a method to find the number of those incremental checkpoints and empirically verified that the total wasted time in the incremental checkpoint model is significantly smaller than the wasted time in the full checkpoint model. Di et al. [13] presented a novel multi-level periodic checkpoint model based on various types of locations (software RAID, local SSD, remote file system) to deal with unpredictable types of failures. The authors also proposed an iterative method to find optimal checkpoint intervals for each level and how to optimize the selection of checkpoint levels. Recently, they constructed a two-level checkpoint model online solution without requiring knowledge of the job length in advance [14], where the first level checkpoint deals with errors with low checkpoint/recovery overheads such as transient memory errors, while checkpoint level two deals with hardware crashes such as node failures. The model shows that periodic patterns are optimal and determines the best pattern. It is possible that our approach may benefit from these more general-purpose reliability efforts, which forms part of our future work.

Nicolae et al. [26] presented a series of design principles that facilitate checkpoint-restart on IaaS clouds and show how they can be applied in IaaS cloud architectures that involves the ability to roll back I/O operations performed by the application. The principles were implemented on BlobSeer, a versioning storage service specifically designed for high throughput under concurrency. Camarasu [11] proposed an end-to-end Monte Carlo simulation framework combining dynamic simulation load-balancing with parallel, incremental merge of checkpointed results. The framework is designed for heterogeneous, non-reliable distributed systems deployed in production and focused on improved usage of existing infrastructures rather than on the design or tuning of their services.

In summary, the above works focus on creating models to deal with different types (heterogeneous software applications and hardware) of failure and multiple levels of failures for HPC and cloud environments. Our proposed checkpointing mechanism aims to solve multi-level analysis in medical image processing, where the first-level takes a long time. Moreover, our checkpointing tool utilizes output to perform reverse engineering to identify wrong input or first-level pipeline design.

Our earlier effort on realizing a cloud-hosted "medical image processing-as-a-service" approach focused on improving the performance of just one concrete and single stage medical image processing transformation process: *dicom2nifti* and supported it on the Apache Hadoop ecosystem [8]. We named it as Hadoop & HBase Toolkit for Medical Image Processing (HadoopBase-MIP) system. Specifically, we made modifications and optimizations to HBase, which is a NoSQL database that is built upon Hadoop. Our work treated the dicom2nifti transformation process as a MapReduce job where the reduce phase was a no-op while providing a modified row key design and a new region split policy for the map phase that maximizes the colocation of the computation task (i.e., dicom2nifti) with the stored data. Although our results were promising, this work only handles fast processing type jobs and does not consider different types of medical image processing jobs and multi-stage analysis, which is a completely different problem that we tackle in this work.

### B. Novelty and Generality of Our Work

The novelty of the proposed work is that we provide a new way of thinking about the second and later stages of multi-level analysis to boost the quality assurance of the entire process rather than just the intermediate results. Although we use the Hadoop ecosystem for the subsequent stages of the pipeline, the proposed incremental real time learning monitor for early error detection is not tightly coupled to it. Moreover, our second stage analysis can exploit our prior work - HadoopBase-MIP - on hierarchical key optimizations [8].

While most of the studies [3], [15], [32] incorporate some form of quality checking (though not cloud based) at individual scan level or outlier detection level (which are all at the first stage) where data were checked for extreme outliers (relative to expected outcome for given study), these checks are not yet carried over into the second stage of analysis. All the data is first

processed and then based on the study design fed into second stage analysis in a serial order. While this approach works for smaller datasets, for studies involving large-scale data as mentioned above where a number of hypothesis are tested, it is beneficial to have intermediary checks on second level or higher level while detecting any faulty outliers or draw conclusion at early stages as data is processed instead of waiting until the end. This is a capability we provide in our work.

## III. Quality Assurance in Medical Image Processing Multi-Stage Analysis Pipeline: Design and Estimating Performance Improvements

In this section we provide details on our technical contributions. First, we shed more light on the challenges faced in handling medical image processing (MIP) jobs using two case studies. We then describe our contributions.

### A. Eliciting Challenges using Two Case Studies

Figure 1 depicts two examples of multi-stage analysis jobs in medical image processing. The first example pertains to the analysis for understanding the brain structure (white matter skeleton [23]) differences within a group of people. In this case, the first-stage pipeline is dtiQA, which determines how usable the diffusion data is and where a diffusion of water molecules is used to generate contrast in MRI images. The input is a group of raw brain DWI images that are collected by MRI scanners. The first-stage pre-processing conducts brain extraction of images (i.e., removing head skull and other structures except the human brain), and then performs registration. The registration (especially non-rigid one) for medical images is time-consuming, because it generally does not have an analytic solution [4], [5]. Therefore, finding the optimal deformation in non-rigid registration requires an optimization process at each iteration where each deformation updates all voxels. If the size of a voxel is $1mm^3$, and the size of the entire brain is $1600cc = 1.6 \times 10^6 mm^3$, then there are $1.6 \times 10^6$ updates at each iteration of the optimization process. Thus, non-rigid registration requires significant computation. Assuming that we have $N$ images, we could let all images do registration[2] to one target image, where each image would take an hour, or perform an all-subjects-to-all-subjects registration which takes even more time. For each pre-processed and registered image, DTIFIT is the last step in the first-stage analysis of *dtiQA*. It generates a quantified image called fractional anisotropy (FA), which can be viewed as an image with all its data stored in a 3D matrix.

The FA images, which are the quantified results generated from the first-stage, are then fed to a second-level statistical analysis, e.g., Tract-Based Spatial Statistics (TBSS) [30] or Gray Matter Surface-based Spatial statistics (GS-BSS) [28], to ensure that all FA images adhere to a common template. Specifically, TBSS contains 4 steps[3] and also involves registration. However, the registration time here is significantly less than that in the first stage because an FA image does not need as much data cleaning and preprosessing as the original raw image. Finally, we run a randomized statistical analysis (randomise), and subsequently observe the spatial distribution in order to find differences in the brain over the entire cortical region or regions of interest set by the researchers.

The second example in Figure 1 shows the first-stage processing comprising image segmentation for cortical parcellation of raw T1W images (or tissue segmentation of raw T1W images). Most segmentation methods are also very time consuming by embedding a registration process simultaneously [16], [22]. All input images need to first register together or register to a known segmented structure template image, and a tissue class is then determined for each image voxel (e.g., white matter, gray matter, cerebrospinal fluid). From the tissue segmentation, we can further extract brain structural (or morphological) representations like gray matter surface. The cortical surfaces could better support quantitative information than 3D MRI images in terms of a topological preservation. This information is then used by the second-stage analysis to conduct cortical-based statistical analysis such as GS-BSS. Again, since the input to the second-stage has already been preprocessed, the next stages of analysis are faster than the first-stage processing.

As evident from the two case studies above, in most multi-stage medical image processing analysis, the first-stage takes a long time, and the second and subsequent stages cannot start until all first-stage results are generated. Further, due to a lack of capabilities to monitor the progress of the first-stage and detect any anomalies therein, errors can propagate to the subsequent stages and are eventually detected only in the later stages, which is wasteful of resources, time and money. Thus, addressing these problems calls for a new pipeline execution and error detection mechanism, which is the first contribution of this paper. At the same time, migrating existing pipelines from cluster-based deployment to the new cloud-based approach requires the users to have a rough estimate of when the new approach will pay dividends. Thus, a capability that can provide such estimates is needed, which is another contribution of this paper.

### B. Contribution 1: Semi-automated, Real-time Quality Assurance Framework for Medical Image Processing Pipeline

To speed up the execution of the medical image processing pipeline, we propose concurrent execution of the stages and early detection of anomalies. To that end, as we accumulate results in the first-stage, the second-stage analyses can start executing based on all collected first-stage results up to that point, and its analyses model can be incrementally updated when more first-stage results show up. In the meantime, if periodic monitoring of the first-stage results indicates an anomaly or outlier, these errors can be flagged right away and either the results can be discarded or the pipeline can be aborted instead of letting it run for a significantly longer time for the errors to be eventually detected. The same idea can be carried over to each stage of a multi-stage pipeline. The rationale for this approach is that after kicking off a multi-month task, users would not want to wait until all the data processing is done. Incrementally updating second (and later) stage analysis results would help the users detect unusual findings and outliers in the first-stage analysis significantly faster and earlier. It would also help users to draw preliminary conclusions before finishing all data processing.

Figure 2 illustrates our proposed real-time monitoring and intermediate checkpointing framework for multi-stage analysis.

[2]Registration means a process of transforming/warping a moving image into a target image space for further analysis.

[3]Specifically, tbss_1_preproc helps prepare all FA data in the right folder and format; tbss_2_reg applies non-rigid registration of all FA images into standard template space; tbss_3_postreg creates the mean FA image and skeletonizes it; tbss_4_prestats projects all subjects' FA data onto the mean FA skeleton and creates 4D images for later statistic comparison usage.

Except for the human quality assurance that involves manual checking, the entire pipeline is automated. In our approach we assume that all the first-stage jobs are HBase MapReduce map tasks (e.g., each map task is a dtiQA processing phase). The pipeline can be scripted using tools such as Matlab. In effect, this script runs on a single input image, and the output, such as an FA image, is uploaded to HBase. At this point our results monitoring job kicks in. The monitor can be configured to run periodically, say, every one hour, to include any new results generated in the first-stage. Anomaly detection is then carried out when intermediate results are available during the execution of the monitor.

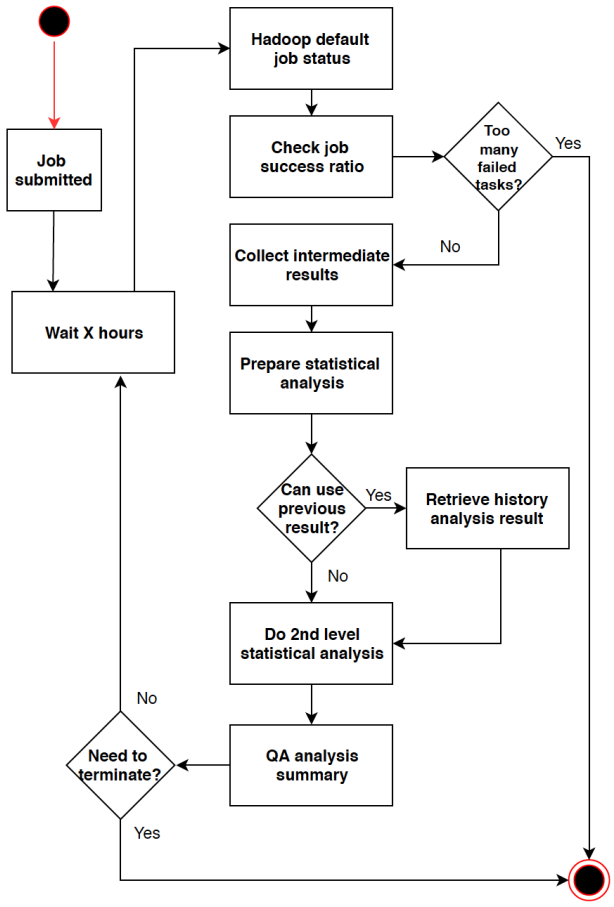



Fig. 2. Real-time monitoring and intermediate checkpointing framework for multi-stage analysis (in this case 2-stage).

In terms of realizing this approach, the pipeline starts from the Hadoop YARN default job status monitor, which can find the basic job running status such as File System Counters (number of bytes read and written on machine file system and HDFS), Job Counters (total number of launched data-local or rack-local map tasks, total time spent on map tasks), MapReduce Framework (map input and output record, GC time elapsed, CPU, physical & virtual memory usage), etc. We exploit a feature of the Hadoop system for anomaly detection. For each MapReduce job, the Hadoop system maintains a file that records map-task success ratio. Our monitor can capture exceptions raised in the job such as map task failed or the job stuck at one point. Anything that is anomalous would then be reported to the system administrator. An example of such a log is shown below which shows progress. A potential indicator of anomaly is when a task does not make any progress over the periodic interval of our monitor task or if it has made progress and then the percentage drops from its previous value:

```
17/09/12 13:39:08 INFO mapreduce.Job:  map 0% reduce 0%
17/09/12 19:20:03 INFO mapreduce.Job:  map 1% reduce 0%
17/09/12 19:29:26 INFO mapreduce.Job:  map 2% reduce 0%
17/09/12 19:34:59 INFO mapreduce.Job:  map 3% reduce 0%
```

Note that the above approach is only for detecting anomalies in the execution of MapReduce tasks but does not provide any quality assurance about the correctness of results produced by these tasks, which requires analyzing the medical image processing results. Therefore, in the intermediate second-stage analysis, the "check success ratio" function performs the following: (1) scans HadoopBase-MIP's result column and issues fast query to find all intermediate results; (2) retrieves value to the designated folder, does command-line based QA to check file format and content (in our case, we embedded a script to check medical image dimension); (3) decides based on a pre-set analysis type if the second stage analysis can utilize previous historical analysis results, and if so starts the second-stage analysis and merges the result in the summary folder. If the entire MapReduce job is not completed, then the monitor would wait a pre-set number of hours before running the next analysis. If the analysis results meet an evaluation metric requirement, the monitor then reports a message to the administrator to recommend terminating the processing.

As a concrete example, consider the second-stage analysis like TBSS in Figure 1, which is an embedded pipeline in itself. The first two steps of TBSS (tbss_1_preproc and tbss_2_reg) incur a one-time cost for each image if we register each image to the same target template. In the subsequent incremental second-stage analysis, these two steps can be skipped for those images that have already been processed in the previous second-stage analysis, so we just need to store those intermediate results. The last two steps of TBSS (tbss_3_postreg and tbss_4_prestats) need the entire dataset of current second-stage analysis and hence the results of these two steps cannot be used for future second-level analysis.

In essence, the performance benefit of adding a monitor at the first-stage is due to the following a reason: while the first-stage analysis utilizes all the cluster resources, the second-stage analysis often involves a "summary" process that only runs on a single machine with single core. Compared to the whole cluster resource usage by the first-stage, the second-level analysis consumes significantly less resources. Thus, an effective monitor on the first-level analysis is promising and can greatly help resource conservation of the whole cluster.

### *C. Contribution 2: Performance Improvements Estimation via Simulation Engine*

Recall that a user in medical image processing may want to first estimate whether using our approach can provide substantial gains in their use cases or not. We now present the design and implementation of two simulators for that purpose: one for the traditional cluster using shared network storage with simple linux utility for resource management (SLURM [33], for job dispatching) called SLURM-simulator, and the other for the Hadoop ecosystem using distributed storage called HadoopBase-MIP-simulator.

In a traditional cluster, a job does not care about the placement of data, which is always retrieved from the network storage, gets processed, with the results sent back to the storage. All simultaneously running jobs on the same rack will compete for network bandwidth during retrieval and uploading since a rack usually has access to a slower network. For HadoopBase-MIP, the jobs are dispatched by a MapReduce computation module [12]. data-locality is one of the best features that Hadoop MapReduce provides, i.e., the computation code gets dispatched to where data is. Since we leverage HBase, which is a NoSQL database built on top of Hadoop's distributed file system (HDFS), we focus on HBase–based MapReduce instead of the traditional Hadoop MapReduce.

In HBase MapReduce, the map task is dispatched to the region server, a single host on which the data block resides [2]. Thus, the computation is local to where the data is. If, however, all the cores on that node are occupied, then the data is moved to another node on the same rack. Only in such a rack-local case, the map tasks would compete for the network bandwidth as in the traditional cluster case. For HadoopBase-MIP, we use default YARN to control MapReduce tasks. We switch scheduling methods by using two popular built-in schedulers, namely, capacity scheduler and fair scheduler, for I/O-intensive and compute-intensive applications, respectively. The capacity scheduler can maximize job dispatching with more data-local map tasks while fair scheduler schedules jobs to better utilize all CPU slots and memory resources.

Before introducing the workflow in our simulator design, we outline the following key assumptions to simplify the design of our simulator engine:

- An identical one Gigabit bandwidth is connected to all of machines under the same rack. Each rack is connected to the GPFS with high speed fiber. To ease network contention on each rack, we balance the total number of machines per rack.
- For each job on the same rack, if data retrieval or upload request occurs while the network is available, then the entire bandwidth is consumed for an interval of 50 milliseconds, and the rest of the data transfer needs to wait until its next turn. Moreover, we do not consider any priority between retrieval and upload.
- Job dispatching time does not include queuing delay. In traditional cluster, a simple First Come First Service (FCFS) scheduler is used. Jobs are ordered based on submission time and scheduled by a single entity. For fair comparison with SLURM, in our HadoopBase-MIP simulator we also mimic a single scheduler entity and introduce a single MapReduce job at a time so that the job can occupy the entire cluster resource as in the case of the traditional cluster. All jobs are dispatched to maintain data-locality first; if there is an available CPU slot without local data, the resource manager finds a node that has most pending jobs.
- We only consider the Map Phase, i.e., all the necessary processing pipelines are embedded in the map tasks and the reduce tasks are no-op, which has been shown to maximize data locality in our prior work [7], [8].

The full pipelines for both of our SLURM-simulator and HadoopBase-MIP-simulator are shown in Figure 3 while a complete implementation of the simulators is available from our open source repository at https://www.nitrc.org/projects/hadoop_2016/. We briefly describe the two simulators below.

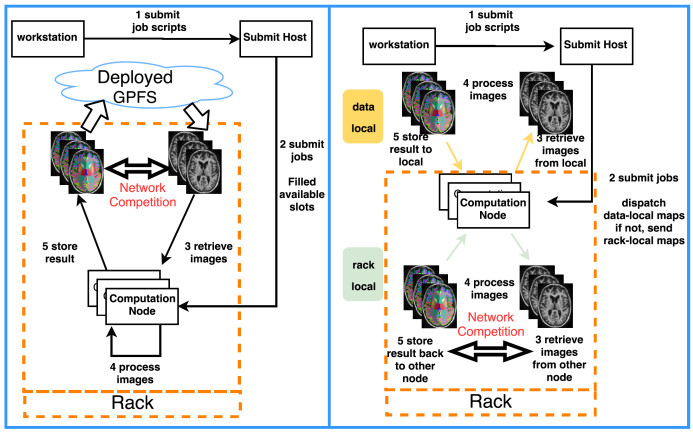



Fig. 3. Full pipelines of SLURM-simulator for the traditional cluster (left) and the HadoopBase-MIP-simulator (right).

*1) SLURM-simulator:* The input for both simulators are text files. The text files for SLURM-simulator contains 3-element-tuples with the format <input_file_size, estimate_processing_time, output_file_size>, and each row represents one job. Jobs are dispatched in a FCFS manner, i.e., according to the submission order. Within the same rack, each job needs to wait for the available network resource to retrieve the input data. Once all the necessary data is retrieved, the job is processed and the results are uploaded. After the job completes and successfully uploads the results, the associated CPU core is freed and assigned to the next pending job. The simulator tracks and summarizes the time usage per rack.

*2) HadoopBase-MIP-simulator:* The HadoopBase-MIP-simulator works in a similar manner but with a few differences. First, a text file for HadoopBase-MIP-simulator contains 4-element-tuples with the format <input_file_size, estimate_processing_time, output_file_size, data_location>. Since uploading original datasets to HBase database incurs a one-time cost, users can upload the data to database and find out relevant node id to fill up the text files. The scheduler can then schedule the available data-local and rack-local jobs. Specifically, the scheduler determines if the job is data-local, otherwise it is scheduled as rack local. It mimics the YARN default fair scheduler to fully utilize the cluster resource. Obviously, data-local jobs use hard disks to do data transfer without network competition.

## IV. Evaluation Methodology and Experimental Results

The goal of this section is to evaluate the efficacy of our simulator in guiding the users to the appropriate solution, and evaluating the performance gains and anomaly detection capabilities of our Hadoop-based concurrent pipeline framework.

### A. Experiment Scenarios

For our evaluations we have defined three experimental scenarios that are described below.

*1) Experiment 1:* The goal of experiment 1 is to empirically verify the prediction accuracy of the simulator for running a mix load of medical image processing jobs (which represents real-world scenarios) in a real, in-house heterogeneous cluster using HadoopBase-MIP and traditional cluster with Sun Grid Engine [17] (SGE, a similar scheduler with SLURM), We use the same experimental design strategy as in [9], which uses *gzip* to compress 1,876 T1 images to the .gz format and adds a sleep function to mimic different processing times. The images are pre-split equally to 47 HBase regions (each region with approximately 40 images), and data allocation ratio of 10 images per CPU core. Each job compresses only one NiFTI image with 2GB memory available and generates one compressed image. The total input size of the images is 29.5GB and the processing generates 20.57GB of compressed files as output.

To explore the impact of different processing times, we artificially increase the processing time by adding a sleep function without any data retrieval to increase the job length by an additional 30 and 90 seconds. The reason we select those two time ranges is because our theoretical model [9] shows that if we run a large number of jobs with similar type, then even a job length as small as 65 seconds[4] would be enough to saturate the network because all jobs would try to access the storage at the same time. Thus, if we only run the 30-second jobs, the jobs will saturate the network and decrease the performance of SGE. Similarly, if we only run the 90-second jobs, the network saturation will be alleviated and the performance of SGE should be similar to that of HadoopBase-MIP. So we mix short and long length jobs to observe the effectiveness of our simulation engine's design.

Two test scenarios are configured as follows: (1) We select 80% of the 1,876 images and configure them as short jobs, the rest 20% are configured as long jobs; (2) We reverse the job length setup from the previous scenario by configuring 80% of the jobs to be long and the rest to be short.

*2) Experiment 2:* The goal of experiment 2 is to use the simulator on the historical trace data we alluded to in Section I to simulate the total execution wall clock time trend for a much broader scale usage and find the performance crossover point between traditional cluster with SLURM and HadoopBase-MIP. This provides an idea of when users should select our technique. Since we do not have any performance data to compare against due to a lack of other simulators and infeasibility of re-running the jobs both due to the computational overhead and because we had access only to the logs that we mentioned in Section I, we use these results simply to illustrate the trends. For machines in the cluster, we assume using homogeneous machines with 10 cores. Each rack maximally can attach 50 machines. The simulation scales the size of cluster cores from 10 to 6000. For simulating the scalability, we start from one machine, and in each next step, we add one additional machine with 10 cores up to a total of 600 homogeneous machines and hence 6000 cores for a total of 12 racks. The experiment is based on our high performance computing cluster ACCRE's design (http://www.accre.vanderbilt.edu/?page_id=63). Note that only 12 racks is very hard to saturate GPFS, but network competition within each racks are where saturation is expected to occur which makes the case for our fair comparison between traditional cluster and HadoopBase-MIP. This allows us to understand the minimum resource allocation needed when moving to using our technique. We use simulation models to schedule the mix types of jobs (IO-intensive and compute-intensive) with actual time order according to the historic record and find the running time of all the jobs from the logs. We separate all 96,103 jobs into 3 categories based on job length: all jobs whose processing time are less than 2 hours (51,496 jobs), 24 hours (73,558 jobs) and all 96,103 jobs taken as a whole, respectively.

*3) Experiment 3:* This experiment is concerned with validating the effectiveness of our monitoring and checkpointing capability in the concurrent multi-stage analysis pipeline by incrementally conducting second stage analysis while checking for errors in the first-stage. We used an existing medical image processing multi-stage analysis where the first-stage does dtiQA as depicted in Figure 1. Specifically, this experiment aims to analyze significant differences related to age-effect of a group of humans' brains in the same study. The age distribution of dataset ranges from 23-31 years old.

Our example has 423 DWI images, whose total input size is 3.5GB, and they would generate 423 FA images as output with a total size around 0.74GB. The estimated time of the first-stage dtiQA pipeline for one image is 7 hours (the total execution time is 35.06 hours estimated by our HadoopBase-MIP simulator). Note that each first-stage dtiQA is a single image-based processing task and would generate an FA image. The FA image would be verified by software FSL (which simply checks the image dimension).

Unlike the traditional multi-stage analysis which runs the two stages in a serial order, using our monitor and checkpoint approach we started both stages concurrently as follows. After every one hour of execution, we check if there are enough new results generated from the first-stage. If not, the monitor waits one more hour to check the output of intermediate results. If the number of newly collected first-stage results meets the threshold (e.g., 10 new images), then the second stage analysis starts (this is the incremental execution of the second stage).

For the second stage analysis, we first use TBSS to process all current and previous first-stage analysis results and then run randomise to observe the spatial distribution of age effects with positive correlation or negative correlation. For each intermediate round, each person's age effect is that person's age minus the average age of the group (423 subjects) for fair comparison of each intermediate second stage analysis result. After a specified number of iterations, if we observe that the second stage analysis is producing similar results, we assume that the results are converging and no additional first or second stage processing is needed. Accordingly the pipeline is stopped. Doing so we can execute the entire pipeline faster compared to traditional approaches. At the same time, if any error is observed in the second stage at any checkpoint interval of the monitor, the system flags the error and lets the user remove the outliers in the first-stage, and resume.

Since there is no similar monitoring mechanism implemented on HadoopBase-MIP or similar medical image processing system, we used the baseline performance as the time to run the entire pipeline in the traditional way.

[4]A typical Gigabit network is ideally 128 MB/second, and in our empirical observation, the average speed of the network can reach 80 MB/second. For each processing task, the average data input and output size is about 27.33 MB((29.5GB input + 20.57 output) / 1876 datasets), and if we allocate whole cluster of 188 cores to run those types of jobs, 188=80/(27.33/processing time), and we will get processing time = 65 seconds to rightly saturate network.

### B. Experimental Setup and Metrics

**Metrics**: We are concerned with three metrics in this work: the first one is wall-clock time and the second one is system throughput, which is the number of datasets (or jobs) processed per minute. The last metric is p-value that represents the significant difference for the statistical analysis of a group study. In particular, p-value = 0.05 and 0.01 are two standard significance levels to reject a null-hypothesis. For instance, the null-hypothesis for experiment 3 is that there is no brain structure difference related with age. A lower p-value we get for any brain voxels implies that the brain area has significant differences based on age, so we can reject the null-hypothesis. Experiment 3 is designed to find all high p-value voxels to know the area of brain structure that is affected by age.

**Datasets**: For the experiments used in validation, experiment 1's dataset is retrieved from normal healthy subjects gathered from [20]. Experiment 2 uses historic anonymized statistic logs with no actual medical image processing and data retrieval. Experiment 3 uses 423 subjects' data from The Tennessee Twin Study (TTS) (RDoC Constructs: Neural Substrates, Heritability and Relation to Psychopathology).

**Hardware**: The testbed used for our validation involves 9 physical machines with a total of 188 cores. Specifically, the testbed consists of 5 machines with 12 cores of AMD Operon 4184 processors and 4 machines with 32 cores of Intel Xeon E5-2630 running Ubuntu 16.04.2 LTS (64 bit). At least 2GB RAM was available per core. The storage is allocated to HDFS and a Gigabit network connects all the machines. Each machine is deployed with a local Seagate ST32000542AS disk, and is used as a Hadoop Datanode and HBase RegionServer for data locality. All machines are also configured using the Sun Grid Engine (Ubuntu Package: gridengine-*). NAS was provided via CIFS using a Drobo 5N storage device (www.droboworks.com) with a 12TB RAID6 array. Both SGE and HadoopBase-MIP contain an additional common master node as cluster master. A typical Gigabit network is deployed at the cluster.

### C. Experimental Results

*1) Experiment 1:* For each scenario, we repeat the entire test 20 times on a real cluster. When fast jobs dominate the workloads (80% of jobs are processed in 30 seconds), HadoopBase-MIP's performance on the cluster achieves a mean±standard deviation throughput of 152.5±3.3 dataset/minute, which is 1.5-fold better on average than the traditional cluster using SGE whose mean±standard deviation throughput is 102.2±2.5 dateset/minute. In comparison, the simulator predicts that HadoopBase-MIP is 1.53-fold better than traditional cluster (163.9±4.6 vs. 106.9±5.7 dataset/minute). For the second test case, most jobs are long jobs (80% of jobs need to be processed in 120 seconds). Thus, the difference in throughput is smaller between the two systems compared with the first test case, and HadoopBase-MIP becomes 1.17-fold better than traditional cluster (91.3±2.3 vs. 179.4±2.7 dataset/minute). Our simulator estimates that HadoopBase-MIP would be 1.15-fold better than traditional cluster (94.0±0.6 vs. 81.7±1.7 dataset/minute).

*2) Experiment 2:* Figure 4 presents all simulation scenarios for HadoopBase-MIP and traditional cluster with SLURM using different cluster setups. The execution time is in log10 scale for better illustration and comparison since the range between low and high values is too large. When the cluster is full of relatively short jobs (less than 2 hours as shown in Figure 4(1)), we can see that SLURM suffers from network saturation when the cluster scales to more than 50 machines (or 1 rack).

When processing jobs whose processing times are less than 24 hours, the performance of SLURM is closer to that of HadoopBase-MIP (as shown in Figure 4(2)). However, when we estimate the total time of all 96,103 jobs, we can see that the performance for HadoopBase-MIP fluctuates with a jitter trend. The reason is that the load contains jobs whose lengths are dramatically different (ranging from 15 seconds up to 9 days). Thus, the data location and job running sequence would affect the entire performance, e.g., due to high data locality simulated scheduling, long running jobs co-located on the same machine. In summary, the simulation results help us better understand how HadoopBase-MIP would behave before we deploy it at large scale. In general, HadoopBase-MIP shows benefit when scaling up on cheap network environment and running short-length jobs.

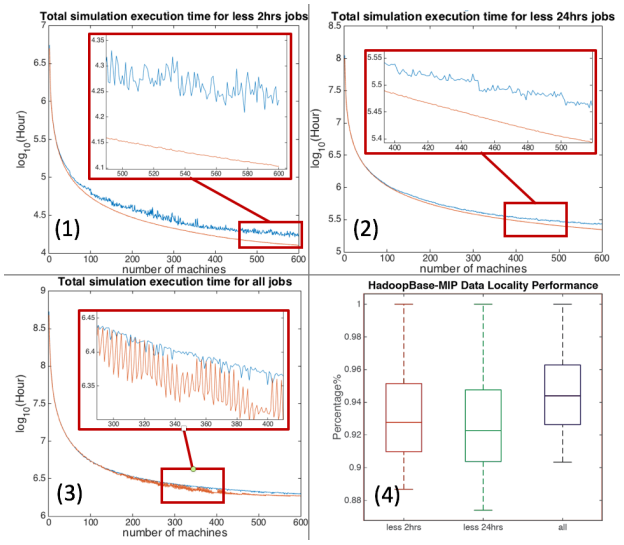



Fig. 4. Simulation result of estimating total execution time (log10 (Hour)) according to historic jobs trace. (1)-(3) show running all jobs whose processing times are less than 2 hours, 24 hours and all 96,103 jobs, respectively. Blue line represents the performance for traditional cluster while orange line represents the performance for HadoopBase-MIP. (4) shows a summary of all HadoopBase-MIP MapReduce jobs' data locality ratio for each scenario, which reveals that most jobs are data-local map.

*3) Experiment 3:* In this experiment, there are 20 intermediate rounds including the last one. Using our approach we first get a ratio of significant trend as shown in Figure 5(1). Here, all results are projected to a common space (mean FA skeleton image). Then, we do t-test based on groups (corresponding to negative age) and find the ratio of voxels that are significant within the common space, using p-value=0.01, 0.05 and 0.1 as three significance levels that we are interested in. However, the trend in Figure 5(1) has a strange behavior: with the number of new outputs generated from the first stage, the trend of the p-values should not fluctuate that dramatically; moreover, the involved significant ratio is too small. Therefore, QA checking should be invoked. As a result, we found some bad examples within the first stage as shown in Figure 5(2). After removing all bad outputs from the first stage and re-running the second stage analysis, Figure 5(3) shows a new trend of ratio of total number of voxels with significant difference in common space based on p-value within different intermediate rounds. The trend

shows after intermediate round 11, the number of activated voxels gradually increased, and the trend is saturated after about 15 rounds. Figure 5(4) presents the qualitative result of the finding from each intermediate second stage analysis, the significance area reveals that FA image has negative correlation with age-based effect. This result has the similar outcome from the work [23], which does TBSS analysis of age-effect on FA image, thus validating our finding.

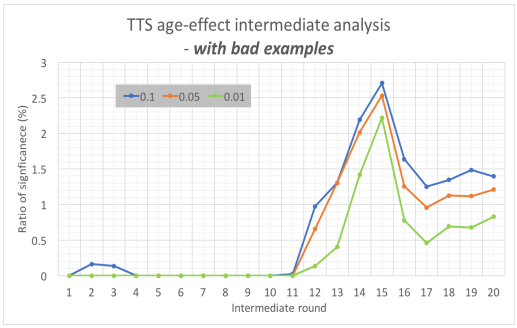
If we only run the second stage analysis after the last round as usual without intermediate QA mechanism, we can still conclude that some part of the brain white matter skeleton structure (around 1500 voxels) is significantly different regarding age-effect. However, the conclusion is wrong since it fails to detect outliers which distort the real result. Again, experiment 3 aims to provide a guidance for users to use this incremental second stage analysis to detect error and draw conclusion, so that they do not need to wait till all data processing is complete. For instance, in real life, by using our proposed monitor, users should stop the pipeline around round 15-16 if they see a weird fluctuation as shown by Figure 5(1), and users could conclude their findings when they see some p-value saturation around round 15-19 as shown by Figure 5(4).

In summary, the total wall-clock time of processing the first stage dtiQA pipeline was 35.93 hours (2.84% slower than HadoopBase-MIP simulator prediction), and the second level analysis took 12.16 hours. Moreover, the time to re-run the second level analysis was 11.58 hours. Thus, the final total wall-clock time using the traditional approach was 59.67 hours with 2773 hours' resource time. On the other hand, using our incremental checkpointing monitor, it would take 22.01 hours' wall-clock time on the first level with 1.25 hours for running the 15th intermediate second level analysis. Once the wrong output from the first level is removed, re-running the whole second level analysis using incremental behavior on a single machine would lead us to draw conclusion at round 15 taking 10.5 hours. Thus, in total 33.76 hours' wall-clock time and 2146 hours' resource time will result from our proposed framework, which is 76.75% less wall-clock time and 29.22% less resource time than the traditional approach.

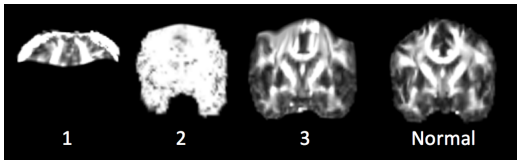
## V. Conclusions

The computation time for medical image processing jobs involving multi-stage analysis often exhibits a significant variance ranging from a few seconds to several days, and most of the processing time is spent on the first-level analysis in the pipeline. Moreover, the cost is very high to process these jobs using traditional clusters, which are deployed with storages like GPFS and high speed networks. The high cost is further exacerbated when errors occurring in the first stages are not detected due to limitations in existing approaches and get propagated to subsequent stages when they are eventually detected. To address these concerns, this paper presented an alternative Hadoop-based approach that utilizes cheap hardware and storage, and enables a concurrent medical imaging pipeline where errors can be detected much earlier in the first stage itself. Additionally, to aid the users in making an informed decision as to when to use our approach versus traditional cluster-based approaches, we present the design of a low-cost simulator.

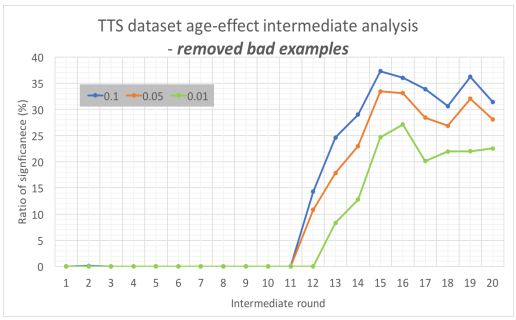
In this work, we did not focus on proving the efficiency of our proposed monitoring mechanism. Instead, we are using it as a prototype to convince the community that one should always examine the outcomes from those time consuming preprocessing stages as early as possible rather than losing valuable computation and time resources. In our approach, the outlier detection was based on "strange p-value behavior" from the second level group analysis via human checking. Meanwhile, the summary trend of intermediate results gave us hint to promote filtering anomality samples in the first level, also based on manual checking. For future work, integrating effective outlier detection in the first level with a quantitative approach for identifying "strange trend" in the second level is necessary. Finally, for facilities that have neither large scale HPC clusters nor large image sets to deal with, our recent work [9] has discussed the trade-offs between using HadoopBase-MIP and traditional in-house clusters.



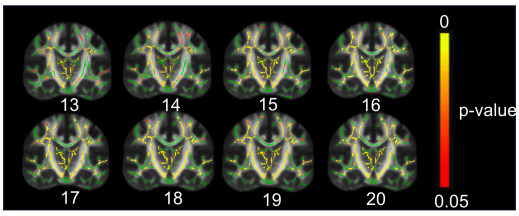

(1) Found weird p-value trend




(2) 3 samples of bad output from 1st level processing




(3) Remove all bad examples and re-run 2nd level analysis




(4) Qualitative result of whole intermediate analysis from round number 13 - 20 basing on negative correlation p-value

Fig. 5. Experiment 3's result of using second level incremental learner monitor architecture for QA. In (2) and (4), 0.1, 0.05 and 0.01 are three different p-value significant level. The ratio of significance means the percentage of significant area that is found in group analysis of whole brain white matter skeleton.

## ACKNOWLEDGMENTS

This work was funded in part by NSF CAREER IIS 1452485 and US Ignite CNS 1531079. This work was conducted in part using the resources of the Advanced Computing Center for Research and Education at Vanderbilt University, Nashville, TN. This project was supported in part by the National Center for Research Resources, Grant UL1 RR024975-01, and is now at the National Center for Advancing Translational Sciences, Grant 2 UL1 TR000445-06. Any opinions, findings, and conclusions expressed in this material are those of the author(s) and do not necessarily reflect the views of the sponsoring agencies.

## REFERENCES

[1] Apache Hadoop Project Team. The Apache Hadoop Ecosystem. http://hadoop.apache.org/.

[2] Apache HBase Team. *Apache hbase reference guide*. Apache, version 2.0.0 edition, Apr. 2016.

[3] A. J. Asman, Y. Huo, A. J. Plassard, and B. A. Landman. Multi-atlas learner fusion: An efficient segmentation approach for large-scale data. *Medical image analysis*, 26(1):82–91, 2015.

[4] B. B. Avants, N. Tustison, and G. Song. Advanced normalization tools (ants). *Insight j*, 2:1–35, 2009.

[5] B. B. Avants, N. J. Tustison, G. Song, P. A. Cook, A. Klein, and J. C. Gee. A reproducible evaluation of ants similarity metric performance in brain image registration. *Neuroimage*, 54(3):2033–2044, 2011.

[6] S. Bao, S. M. Damon, B. A. Landman, and A. Gokhale. Performance management of high performance computing for medical image processing in amazon web services. In *Proceedings of SPIE–the International Society for Optical Engineering*, volume 9789. NIH Public Access, 2016.

[7] S. Bao, B. Landman, and A. Gokhale. Algorithmic enhancements to big data computing frameworks for medical image processing. In *Cloud Engineering (IC2E), 2017 IEEE International Conference on*, pages 13–16. IEEE, 2017.

[8] S. Bao, A. J. Plassard, B. A. Landman, and A. Gokhale. Cloud engineering principles and technology enablers for medical image processing-as-a-service. In *Cloud Engineering (IC2E), 2017 IEEE International Conference on*, pages 127–137. IEEE, 2017.

[9] S. Baoa, F. D. Weitendorfa, A. J. Plassarda, Y. Huob, A. Gokhalea, and B. A. Landmana. Theoretical and empirical comparison of big data image processing with apache hadoop and sun grid engine. In *SPIE Medical Imaging*, pages 101380B–101380B. International Society for Optics and Photonics, 2017.

[10] S. Bonner, A. S. McGough, I. Kureshi, J. Brennan, G. Theodoropoulos, L. Moss, D. Corsar, and G. Antoniou. Data quality assessment and anomaly detection via map/reduce and linked data: a case study in the medical domain. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 737–746. IEEE, 2015.

[11] S. Camarasu-Pop, T. Glatard, R. F. Da Silva, P. Gueth, D. Sarrut, and H. Benoit-Cattin. Monte carlo simulation on heterogeneous distributed systems: A computing framework with parallel merging and checkpointing strategies. *Future Generation Computer Systems*, 29(3):728–738, 2013.

[12] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[13] S. Di, M. S. Bouguerra, L. Bautista-Gomez, and F. Cappello. Optimization of multi-level checkpoint model for large scale hpc applications. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 1181–1190. IEEE, 2014.

[14] S. Di, Y. Robert, F. Vivien, and F. Cappello. Toward an optimal online checkpoint solution under a two-level hpc checkpoint model. *IEEE Transactions on Parallel and Distributed Systems*, 28(1):244–259, 2017.

[15] A. Di Martino, C.-G. Yan, Q. Li, E. Denio, F. X. Castellanos, K. Alaerts, J. S. Anderson, M. Assaf, S. Y. Bookheimer, M. Dapretto, et al. The autism brain imaging data exchange: towards large-scale evaluation of the intrinsic brain architecture in autism. *Molecular psychiatry*, 19(6):659, 2014.

[16] J. Doshi, G. Erus, Y. Ou, S. M. Resnick, R. C. Gur, R. E. Gur, T. D. Satterthwaite, S. Furth, C. Davatzikos, A. N. Initiative, et al. Muse: Multi-atlas region segmentation utilizing ensembles of registration algorithms and parameters, and locally optimal atlas selection. *NeuroImage*, 127:186–195, 2016.

[17] W. Gentzsch. Sun grid engine: Towards creating a compute power grid. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 35–36. IEEE, 2001.

[18] S. S. Ghosh, S. Kakunoori, J. Augustinack, A. Nieto-Castanon, I. Kovelman, N. Gaab, J. A. Christodoulou, C. Triantafyllou, J. D. Gabrieli, and B. Fischl. Evaluating the validity of volume-based and surface-based brain image registration for developmental cognitive neuroscience studies in children 4 to 11years of age. *Neuroimage*, 53(1):85–93, 2010.

[19] R. L. Harrigan, B. C. Yvernault, B. D. Boyd, S. M. Damon, K. D. Gibney, B. N. Conrad, N. S. Phillips, B. P. Rogers, Y. Gao, and B. A. Landman. Vanderbilt university institute of imaging science center for computational imaging xnat: A multimodal data archive and processing environment. *NeuroImage*, 124:1097–1101, 2016.

[20] Y. Huo, K. Aboud, H. Kang, L. E. Cutting, and B. A. Landman. Mapping lifetime brain volumetry with covariate-adjusted restricted cubic spline regression from cross-sectional multi-site mri. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 81–88. Springer, 2016.

[21] Y. Huo, J. Blaber, S. M. Damon, B. D. Boyd, S. Bao, P. Parvathaneni, C. B. Noguera, S. Chaganti, V. Nath, J. M. Greer, et al. Towards portable large-scale image processing with high-performance computing. *Journal of digital imaging*, 31(3):304–314, 2018.

[22] A. Keshavan, C. Madan, E. Datta, and I. McDonough. Mindcontrol: Organize, quality control, annotate, edit, and collaborate on neuroimaging processing results. *Research Ideas and Outcomes*, 3:e12276, 2017.

[23] C. Kodiweera, A. L. Alexander, J. Harezlak, T. W. McAllister, and Y.-C. Wu. Age effects and sex differences in human brain white matter of young to middle-aged adults: a dti, noddi, and q-space study. *Neuroimage*, 128:180–192, 2016.

[24] B. Meng, G. Pratx, and L. Xing. Ultrafast and scalable cone-beam ct reconstruction using mapreduce in a cloud computing environment. *Medical physics*, 38(12):6603–6609, 2011.

[25] N. Naksinehaboon, Y. Liu, C. Leangsuksun, R. Nassar, M. Paun, S. L. Scott, et al. Reliability-aware approach: An incremental checkpoint/restart model in hpc environments. In *2008 8th International Symposium on Cluster Computing and the Grid (CCGRID*, pages 783–788. IEEE, 2008.

[26] B. Nicolae and F. Cappello. Blobcr: efficient checkpoint-restart for hpc applications on iaas clouds using virtual disk image snapshots. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 34. ACM, 2011.

[27] Y. Ou, H. Akbari, M. Bilello, X. Da, and C. Davatzikos. Comparative evaluation of registration algorithms in different brain databases with varying difficulty: results and insights. *IEEE transactions on medical imaging*, 33(10):2039–2065, 2014.

[28] P. Parvathaneni, B. P. Rogers, Y. Huo, K. G. Schilling, A. E. Hainline, A. W. Anderson, N. D. Woodward, and B. A. Landman. Gray matter surface based spatial statistics (gs-bss) in diffusion microstructure. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 638–646. Springer, 2017.

[29] S. Roychowdhury and M. Bihis. Ag-mic: Azure-based generalized flow for medical image classification. *IEEE Access*, 4:5243–5257, 2016.

[30] S. M. Smith, M. Jenkinson, H. Johansen-Berg, D. Rueckert, T. E. Nichols, C. E. Mackay, K. E. Watkins, O. Ciccarelli, M. Z. Cader, P. M. Matthews, et al. Tract-based spatial statistics: voxelwise analysis of multi-subject diffusion data. *Neuroimage*, 31(4):1487–1505, 2006.

[31] S. M. Smith, M. Jenkinson, M. W. Woolrich, C. F. Beckmann, T. E. Behrens, H. Johansen-Berg, P. R. Bannister, M. De Luca, I. Drobnjak, D. E. Flitney, et al. Advances in functional and structural mr image analysis and implementation as fsl. *Neuroimage*, 23:S208–S219, 2004.

[32] D. C. Van Essen, K. Ugurbil, E. Auerbach, D. Barch, T. Behrens, R. Bucholz, A. Chang, L. Chen, M. Corbetta, S. W. Curtiss, et al. The human connectome project: a data acquisition perspective. *Neuroimage*, 62(4):2222–2231, 2012.

[33] A. B. Yoo, M. A. Jette, and M. Grondona. Slurm: Simple linux utility for resource management. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer, 2003.

[34] Z. Zhang, F. Xing, F. Liu, and L. Yang. High throughput automatic muscle image segmentation using cloud computing and multi-core programming.