Algorithmic Enhancements to Data Colocation Grid Frameworks for Big Data Medical Image Processing

By

Shunxing Bao

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University in

partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

December 15th, 2018

Nashville, Tennessee

Approved:

Bennett A. Landman, Ph.D.

Aniruddha Gokhale, Ph.D.

Douglas C. Schmidt, Ph.D.

Alan Tackett, Ph.D.

Hongyang Sun, Ph.D.

This dissertation is dedicated to my mom and dad, Nawen and Jianguo.

ACKNOWLEDGMENTS

I'd like to express my deepest respect and gratitude to my parents, Nawen Ge, Jianguo Bao, and my grandmother Ami Chen. I feel soooooo proud and fortunate to be your child. Home is always the source of happiness. Home is always the shelter whenever I feel down. And home is always home. You support me physically, spiritually, financially, endlessly and let me do whatever I want to do. Love you all, forever.

For the four-year Ph.D. study, I'd like to express my extreme thanks to my advisor Prof. Bennett A. Landman. Without your support and mentorship, I cannot achieve so much for three years. Although life to work with you is incredible hard, but it's 120% worth it! You can always easily catch up any novel research points and views. I am not flattering, you are the smartest person I've ever met. Your affirmation and encouragement are my best gift during my Ph.D. student life. I'd like to thank my co-advisor Prof. Aniruddha Gokhale. We've known each other since 2012. Your generous guidance helps me build the connection among multidisciplinary research topics. You are also my old friend that can share life experience. For my committee member Prof. Alan Tackett, I feel so lucky to work with. You are a hands-on person, who help me enlarge my research vision. I really appreciate my committee member Prof. Douglas C. Schmidt for your constructive feedback of how to make research more impactful. Prof. Hongyang Sun, my committee member as well, as a friend and mentor, you selflessly to share your research experience influence me a lot. I also want to thank my master's mentor, Dr. Joe Porter. You convince me and let me start the road to be Ph.D.

I want to acknowledge to my lab mates during my Ph.D. student life. For people in MASI lab, Dr. Zhoubing Xu, you are so calm and easy going, and you let me know how to make Ph.D. life more efficient. Prof. Yuankai Huo, you are my best friend and big brother, you give me a huge amount of help both in life and research. I feel so comfortable to sit right next to you. Prasanna Parveanii and Dr. Ilwoo Lyu, thank you for being so patient

and helpful whenever I ask you for help. Thank you my colleagues in MASI, Shikha Chaganti, Stephen Damon, Andrew Plassard, Justin Blaber, Camilo Bermudez, Robert Harrigan, Vishwesh Nath, Colin Hansen, Allison Hainline, Hyeonsoo Moon, Sandra González-Villà, Roza Bayrak, Frederick Weitendorf, Katrina Nelson, Meg Bobo, Peijun Hu, Jiaqi Liu, Jiachen Wang, Yunxi Xiong, Yucheng Tang, Riqiang Gao, Cailey Cline, Anupam Kumar, Sam Remedios, Sahil Panjwani, Karthik Ramadass. Wow, I can't believe that I've worked with so many nice folks in MASI!

For people in DOC group, Kyoungho An, Faruk Caglar, Prithvi Patil, Subhav Pradhan, Kuroda Takayuki, Shashank Shekhar, Anirban Bhattacharjee, Yogesh Barve, Shweta Khare, Ziran Min, Zhuangwei Kang, Travis Brummett, Robert Canady, I really appreciate you all for always being so supportive.

I'd like to thank all of the staffs who have helped me in ACCRE.

I'd like to special acknowledge to my dear - Peng (Dana) Zhang. We've experienced a lot, learned a lot, worked a lot and achieved a lot. It is absolute bliss for me to be with you.

Finally, I want to thank my friends, Yi Li, Jian Lou, Jianing Wang, Zhixiong Chen, Steve Nyemba, Zhiyu Wan, Zhijun Yin, Rich Zeigler and my best man Jianing Song, who have given me tremendous support during my graduate career.

ABSTRACT

Large-scale medical imaging studies to date have predominantly leveraged in-house, laboratory-based or traditional grid computing resources for their computing needs, where the applications often use hierarchical data structures (e.g., Network file system file stores) or databases (e.g., COINS, XNAT) for storage and retrieval. The resulting performance for laboratory-based approaches reveal that performance is impeded by standard network switches since typical processing can saturate network bandwidth during transfer from storage to processing nodes for even moderate-sized studies. On the other hand, the grid may be costly to use due to the dedicated resources used to execute the tasks and lack of elasticity. With increasing availability of cloud-based big data frameworks, such as Apache Hadoop, cloud-based services for executing medical imaging studies have shown promise.

Despite this promise, our studies have revealed that existing big data frameworks illustrate different performance limitations for medical imaging applications, which calls for new algorithms that optimize their performance and suitability for medical imaging. For instance, Apache HBases data distribution strategy of region split and merge is detrimental to the hierarchical organization of imaging data (e.g., project, subject, session, scan, slice). 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. Due to the sequential nature of executing the analysis stages 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. To address these challenges, this research propose a framework - Hadoop & HBase for Medical Image Processing (HadoopBase-MIP) - which develops a range of performance optimization algorithms and employs a number of system behaviors modeling for data storage, data access and data processing. We also introduce how to build

up prototypes to help empirical system behaviors verification. Furthermore, we introduce a discovery with the development of HadoopBase-MIP about a new type of contrast for medical imaging deep brain structure enhancement. And finally we show how to move forward the Hadoop based framework design into a commercialized big data / High performance computing cluster with cheap, scalable and geographically distributed file system.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

Big data in medical imaging offers an opportunity to study specific control populations (age/sex/demographics/genetics) and identify substantive homogeneous sub-cohorts so that one may understand the role that individual factors play in treatment response. To support this cause, existing medical imaging studies have often leveraged either in-house, laboratory-sized computing resources or grid computing resources. Both these existing approaches, however, incur various limitations. Consider, for example, a medical imaging application that converts Digital Imaging and Communications in Medicine (DICOM) files to NiFTI (a research file format). If converting a 50 MB volume takes 15 seconds, an ideal gigabit network about 100 MB/s saturates with slightly less than 30 simultaneous processes.

These constraints are significant for medical imaging applications. For instance, consider how contemporary grid computing approaches separate data storage from computation. To analyze data, each dataset must be copied from a storage archive, submitted to an execution node, processed, synthesized to a result, and results returned to a storage archive. When imaging datasets become massive, which is clearly the trend, the bottleneck associated with copying and ensuring consistency overwhelms the benefits of increasing the number of computational nodes, i.e., performance gains no longer scale with the number of computational nodes, but are limited by the network.

The network issues are not the only challenges. Despite the presence of vast repositories of magnetic resonance imaging (MRI) and computed tomography (CT) images, which are accumulating at a rate of nearly 100 million examinations per year in the U.S. [5], today we lack the image processing, statistical, and informatics tools for large-scale analysis and integration with other clinical information (e.g., genetics and medical histories).

What is needed is an efficient mechanism for query, retrieval, and analysis of all patient data (including imaging) which would enable clinicians, statisticians, image scientists, and engineers to better design, optimize, and translate systems for personalized care into practice. All of this must be affordable and perform well. These needs preclude both in-house and grid-based approaches.

Consequently, cloud-based services to address these growing needs hold promise. Specifically, Big Data processing frameworks such as Apache Hadoop ecosystem are promising in this context. The Hadoop framework provides flexible, distributed, scalable and fault tolerant storage and parallel computational modules, and the associated HBase, which is a NoSQL database built atop Hadoops distributed file system, has the potential for building up a big data platform for medical imaging.

In this work, we mainly present a "medical image processing-as-a-service" grid framework, Hadoop & HBase for Medical Image Processing (HadoopBase-MIP), which integrates the Hadoop and HBase (a database built upon Hadoop) for data colocation by moving computation close to medical image storage to reduce network saturation for big data processing.

1.1 Background

We start with the medical image format that we routinely deal with and then introduce the overview of Apache HBase.

1.1.1 Traditional medical image format Overview

DICOM (Digital Imaging and Communications in Medicine) is the international standard for medical images and related information (ISO 12052). It defines the formats for medical images that can be exchanged with the data and quality necessary for clinical use (http://dicom.nema.org/). It has a hybrid structure that contains regular data (patient/clinical information), multimedia data (images, video). Data inside a DICOM file is formed as

a group of attributes [6].

When a patient gets a computed tomography (CT) or magnetic resonance imaging (MRI) scan, for example a patient's brain image, a group of 2-dimensional DICOM images are generated slice by slice (or a sequence of 3-dimension images acquisition). A non-exhaustive set of medical imaging DICOM attributes for the slices include: project, subject, session and scan, where a project is a particular study, a subject is a participant within the study, a session is a single imaging event for the subject, and a scan is a single result from the event.

In order to study the entire brain, all 2-dimensional DICOM images should be collected together. Even though medical imaging data is stored as DICOM images, a substantial amount of medical image analysis software are NiFTI-aware (e.g.FSL (http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/), AFNI (https://afni.nimh.nih.gov/afni/), SPM (http://www.fil.ion.ucl.ac.uk/spm/) and Freesurfer (http://freesurfer.net/)). NiFTI is a medical image data format, which is termed as a "short-term measure to facilitate inter-operation of functional MRI data analysis software package," developed and founded by the NiFTI Data Format Working Group (http://nifti.nimh.nih.gov/).

Converting a large group of slices of DICOM images belonging to one patient into a small number of NiFTI format images (many-to-many relationship) is a significant step in any medical imaging study. Any processing of DICOM datasets will need to determine which CT/MRI **scan** that slice belongs to and using the **Session** attribute which records when the CT/MRI scan volume is carried out. However, finding a **Session** needs to first know the attribute **Subject** to which it belongs. Finally, the **Project** attribute collects all subjects together. Thus, for medical imaging applications involving DICOM, the following attributes are necessary: $project \rightarrow subject \rightarrow session \rightarrow scan \rightarrow slice$.

1.1.2 Overview Apache HBase

HBase [7] uses the Hadoop Distributed File System (HDFS) to provide distributed and replicated access to data. We chose HBase since it can group data logically and physically based on row-key value. However, HDFS can only provide logical order. HBase also provides a flexible translation layer (region-split policy) for dealing with data collocalization statically or dynamically.

The key concepts from the HBase architecture are summarized in Table 1.1. Briefly, HBase maintains tables, which have a row key that is commonly used as an index, and where data columns are stored with the row key. All data in HBase is "type free," which are essentially in the format of a byte array. The table is sorted and stored based on the row key.

Table 1.1: HBase architecture key concepts summary

Concept	**Comment**
Table / HTable	A collection of related data with a column-based format within HBase.
Region	HBase Tables are divided horizontally by row key range into "Regions." A region contains all rows in the table between the region's start key and end key.
Store	Data storage unit of HBase region.
HFile / Storefiles	The unit of Store, which is collocated with a Hadoop datanode and stored on HDFS.
memStore	When write data is uploaded to a HTable, it is initially saved in a cache as memStore. Once the cache size exceeds a pre-defined threshold, the memStore is flushed to HDFS and saved as HFile.
HMaster	HBase cluster master to monitor a RegionServer's behavior for load balancing.Table operator. e.g., create,delete and update a table.
Regionserver	Serves read/write I/O of all regions in a cluster node. When Regionservers collocate with Hadoop datanode, it can achieve data locality. Subsequently, most reads are served by the RegionServer from the local disk and memory cache, and short circuit reads are enabled.
Rowkey	A unique identifier of a row record in table.
Column family	Columns in Apache HBase are grouped into column families.
Column identifier	The member in column family, also called as column qualifier. Multiple column identifiers can be used within one column family.

HBase tables are divided into "regions" for distributed storage such that each region contains a continuous set of row keys from the overall table. The data in a region is physically collocated with an HDFS data node to provide data locality, which is performed by an operation called major compaction. When the region size grows above a pre-set physical size threshold, a "RegionSplitPolicy" takes effect and divides the region into smaller pieces. The newly created regions are automatically moved to different nodes for load balancing of the entire cluster. The row key and RegionSplitPolicy are integral to the performance and data retrieval of HBase and Hadoop.

1.1.3 Multi-level medical image processing

Figure-1.1 depicts two examples of multi-level analysis jobs in medical image processing. The first example pertains to the analysis needed in understanding the brain structure differences within a group of people. In this case, the first level pipeline shown is *dtiQA*, whose purpose is to determine how usable diffusion data is and where a diffusion of water molecules is used to generate contrast in MR images. The input to the first example pipeline is a group of raw brain DWI images that are collected by MRI scanners. The pre-processing does brain extraction of images (i.e., remove head skull and other structures except the human brain), and then performs registration. The registration (especially non-rigid registration) for medical images is time-consuming, because non-rigid registration generally does not have an analytic solution [8, 9]. Therefore, finding the optimal deformation in non-rigid registration requires an optimization process at each iteration of which a deformation is to update all voxels. If the size of a voxel is $1mm^3$, and the entire brain image's size $1,600,000mm^3$, then there are 1,600,000 iterations of the optimization process. Thus, non-rigid registration requires a significant computation. Assuming that we have N images, we could register all images to one target image, where each image would take 10 minutes, or perform an all-subjects-to-all-subjects which takes more time. For each pre-processed and registered image, DTIFIT is the last step in the first level analysis of

dtiQA, which generates a quantified image called fractional anisotropy (FA), which can be viewed as an image and all its data are stored in a 3D matrix.

The fractional anisotropy (FA) images, which are the quantified results generated from the first level, are fed to a second level statistical analysis, e.g., Tract-Based Spatial Statistics (TBSS) [10] or Gray Matter Surface-based Spatial statistics (GS-BSS) [11], to ensure that all output FA images adhere to a common template. Specifically, TBSS contains 4 steps [1] also involving registration. However, the registration time in the second level is much smaller than the first level because the FA image does not need to do much data cleaning and prepossessing like in original raw image. Finally, we run randomized statistical analysis, and subsequently observe the spatial distribution in order to find differences in the brain over the entire cortical region or partial regions of interest using parameters set by the researcher.

The second example in the Figure- 1.1 shows the first level processing comprising image segmentation for cortical parcellation of raw T1W images (or tissue segmentation of raw T1W images). Most segmentation methods are also a very time consuming process, which commonly embed a registration process [12, 13, 2, 1, 14, 15]. All input images need to first be registered together or registered 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 the gray matter surface. The cortical surfaces could better support quantitative information than 3D MRI in terms of a topological preservation. This information is then used by the second level analysis to conduct cortical-based statistical analysis such as GS-BSS. Again, once the second level data has already been preprocessed, the further analysis analysis is faster than the first level processing.

[1] tbss_1_preproc helps prepare all FA data in the right folder and format; tbss_2_reg would apply non-rigid registration of all FA images into standard template space; tbss_3_postreg creates the mean FA image and skeletonize it; tbss_4_prestats would project all the subjects' FA data onto the mean FA skeleton and create 4D images for later statistic comparison usage.

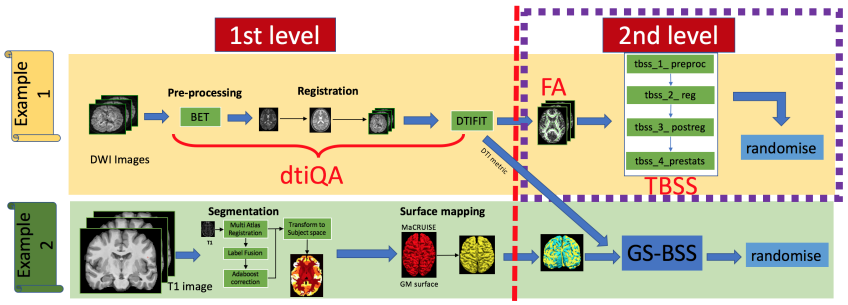


Figure 1.1: A classical medical image processing multi-level analysis Tract-Based Spatial Statistics and Gray Matter Surface-based Spatial statistics. Our target pipeline of this work is the first example dtiQA & TBSS pipeline. The purple box indicates our second contribution focus area.

1.1.4 Overview of LStore

In Vanderbilt's high performance computing (HPC) cluster called Advanced Computing Center for Research and Education, General Parallel File System (GPFS) [16] is the current solution. Meanwhile, ACCRE also supports a cheaper solution, LStore, a distributed and scalable storage cross geography. LStore aims to provide a peta level storage, e.g. for the compact muon solenoid (CMS) experiment [17]. When moving forward our design strategies to HPC environment, seeking a feasible, cheap storage and geographic distributed storage as backend rather than costly in house Lustre / GPFS for HadoopBase-MIP based computation is important. LStore provides a flexible logistical storage framework for distributed and scalable access to data for a wide spectrum of users. LStore is designed to provide: virtually unlimited scalability in raw storage; support for arbitrary metadata associated with each file; user controlled fault tolerance and data reliability on a file and directory level; scalable performance in raw data movement; a virtual file system interface with a native mount in Linux (exportable via NFS and CIFS to other platforms) and a high performance command line interface; and LStore supports the geographical distribution and migration of data to facilitate quick access. These features are accomplished by segregating directory and metadata services from data transfer. Logistical Networking is used as the block-level data abstraction for storage with LStore clients directly performing

all data I/O without going through an intermediate server as is done for NFS and CIFS. This means adding storage adds both capacity but also bandwidth." [18].

1.2 Key Research Challenge

HDFS aims to deal with large file, and it has very poor performance with small files [19]. Storing too many small in HDFS will explode the metadata's RAM requirements. Moreover, searching files on HDFS is sequential read, and too many files will lead to more streaming and degrade the file read / write performance. Finally, HDFS does not support random read / write. The main big data problem for medical imaging is high volume, the image itself is relatively small compared with traditional "large" file definitions. As above background presented, HBase has potential to use for its capability of logically and physically colocating relevant data with common index. Now we start introducing challenge of why this colocation matters medical image analysis, and how it can be optimized for implementing and discovering more big data usage.

1.2.1 Challenge 1: How to move medical image processing to the cloud

Adopting high performance cloud computing for medical image processing is a popular trend given the pressing needs of large studies. Amazon Web Services (AWS) provide reliable, on-demand, and inexpensive cloud computing services [20, 21]. Our research objective is to implement an affordable, scalable and easy-to-use AWS framework for the Java Image Science Toolkit (JIST) [22, 23]. JIST is a plugin for Medical-Image Processing, Analysis, and Visualization (MIPAV) that provides a graphical pipeline implementation allowing users to quickly test and develop pipelines [24]. JIST is DRMAA-compliant allowing it to run on portable batch system grids. However, as new processing methods are implemented and developed, memory may often be a bottleneck for not only lab computers, but also possibly some local grids. Integrating JIST with the AWS cloud alleviates these possible restrictions and does not require users to have deep knowledge of programming in

Java. Workflow definition/management and cloud configurations are two key challenges in this research.

1.2.2 Challenge 2: Why the data colocation based approach matters

The first challenge explores utilizing the cloud with pay-as-go fashion with the pipeline depends on the software JIST. However, in order to deploy big data based analysis with more generic approach, a better solution using a local private grid / cloud is an alternative approach. Although HBase/HDFS is widely used in practice, multiple challenges manifest in the context of medical imaging applications. First, there is no standard for the default row key design. Intuitively, the data should be placed as sparse as possible and distributed evenly across various points of the regions in the table. Such a strategy can avoid data congestion in a single region, which otherwise could give rise to read/write hot-spots and lower the speed of data updates. Because row keys are sorted in HBase, using randomly generated keys when input as data to HBase would help leverage the data distribution in the table. As shown later, however, such an approach incurs performance penalties for medical imaging applications.

Second, since the original DICOM file name is a unique identifier called Global Unique identifier [25], if the task of interest is storing slice-wise DICOM data within HBase, then a naïve approach would be to use the DICOM GUID. Since the GUID is a hash of the data, it will not collocate data together and thus will saturate the network at the time of retrieving all DICOM images of a scan volume. Further, the standard RegionSplitPolicy will randomly assign files with hashed DICOM GUID file names as row keys to regions based on the key and a convenient split point based on region size, which may not be efficient as we show later.

1.2.3 Challenge 3: How to identify the limits of Apache Hadoop for Medical Imaging compared with traditional cluster

The potential solution from challenge 1 gives a promise that a data colocation framework may help a specific medical image processing, namely "DICOM to NiFTI". This creates new questions, when does that novel Hadoop/HBase framework perform better than traditional clusters like Sun grid engine (SGE) [26]? In this case, there are many parameters of concern, such as the cluster size, machine cores, node memory, distribution of resources, image processing job, etc [27, 28, 29, 30]. A theoretical way to understand the trade off of using data colocation approach is needed, and if the theoretical models translate practical things are needed to answer.

1.2.4 Challenge 4: System optimization and enabler for boosting Big Data applications

Three optimization opportunities apply with the research and development of HadoopBase-MIP.

- Heterogeneous hardware: Hadoop/HBase uses an approximately balanced data allocation strategy by default. In previous work, we observe that the throughput of a cluster is low especially when it combines with different types and / or different number of cores per machine. Thus, performance aware data collocation models are needed.

- Large dataset analysis: When large summary statistical analyses are requested, huge volumes of data can saturate memory of one machine. Thus, one needs to split a dataset into small chunks. Most chunks would be sent to where their data are located and run in parallel on different machines, but few chunks would need to move data via network. This process is called Map phase in the MapReduce computation paradigm [19, 3]. Once all intermediate results of all chunks are collected on one machine, they are further processed and final result is generated, this is depicted as Reduce phase.

Integrated processing of large datasets introduces dependencies with the number of jobs (which is based on chunk size) and a way to optimize chunk size is needed.

- Rapid NoSQL query: HBase is a column based NoSQL database, namely all data that are in same column are stored together [3, 31]. Each row of table is a multi-record that is based on total number of columns that are defined with a unique name as rowkey. If we store all medical images with other data like index, age and sex into same column, then when we perform a query, linear search with image traversal is required, which severely decreases the speed of search. On the other hand, for HBase MapReduce, each single map needs to set a start/stop row and a typical column. When doing subset datasets analysis, data is scattered in HBase, and not all record between start/ stop row are needed. It is important to structure data in a manner to skip unnecessary row and image traversal.

1.2.5 Challenge 5: Enhancing big data frameworks for heterogeneous mixed workloads of multi-level based medical imaging analysis

Owing to the high cost of running these analysis jobs in the cluster, alternatives such as hosting in the cloud offer a promise as was validated by our potential solution of first three challenges, where are demonstrated the use of cloud-based Apache Hadoop ecosystem to support a specific transformation process in medical image processing. However, this work focused only on a single job type. Our extensive work in the medical image processing field has revealed that multi-level analysis can comprise a variety of different job types with differing execution times. There is no existing capability to determine the feasibility of hosting such heterogeneous and increasingly large-sized job types found in medical image processing and using technologies beyond traditional cluster software, such as the Apache Hadoop ecosystem.

Furthermore, in most medical image processing multi-level analysis, the first level analysis takes too long, and in many popular medical image processing software, the second

level analysis cannot start until all of the first level results are generated. Due to a lack of capabilities to monitor the progress of the first level and detect any anomalies in that level, these errors are noticed only in the latter stages which is wasteful of resources, time and money. We know of no existing capability that can detect anomalies in real-time and with minimal human involvement.

1.2.6 Challenge 6: Evaluation of big data registration-based Image Enhancement

MRI is an important tool for analysis of deep brain grey matter structures. However, analysis of these structures is limited due to low intensity contrast typically found in whole brain imaging protocols. We empirically discovered a big data registration-enhancement (BDRE) technique to augment the contrast of deep brain structures using an efficient large-scale non-rigid registration strategy. However, direct validation is problematic given a lack of ground truth data. How to verify the proposed image enhancement technique is challenging.

1.2.7 Challenge 7: How to integrate Apache Hadoop into high performance environment using logistical storage framework as back-end

Apache Hadoop ecosystem aims to use commodity hardware and network setup. Adopting HPC resources is an option because of its strong computation capability. Meanwhile, HPC fast network makes Hadoop's separation of computation and storage reasonable. Owing to a HPC based distributed, scalable and geographically dispersed file system, LStore, which uses cheap hard drives, we expect like to see how LStore can be integrated with HDFS and served as a cost efficient storage backend. The main challenges are (1) how to avoid data redundancy on HDFS side once data already available on LStore side (2) how to ease the effort of implementing the plugin integration by choosing LStore fuse client interface or command line based interface.

1.3 Overview of the Proposed Research Goals

This section highlights our research contribution including framework design and theoretical models desgin. And all following innovations are empirically verified in a private research cloud comprising a typical gigabit network.

1.3.1 Addressing Challenge 1: AWS plugin for a toolkit for medical image processing

As an experience with cloud utilization, for this challenge we focus more on system framework design and setup.

- We provide a concrete workflow definition/management for JIST to deploy a local project to be distributed and processed in the cloud.
- We integrate JIST to Amazon AWS (a reliable, on-demand, and inexpensive cloud computing services) to execute high performance computing.
- We demonstrate a cost/benefit analysis of a pipeline that can be easily parallelized and requires memory that would limit the processes to serially execute only on a local lab machine.

1.3.2 Addressing Challenge 2: data colocation based approach really matters

We prototype a medical image processing as as service framework, to address the network saturation of cluster. To address these problems, we present design principles and empirical validation for a new data model for use with cloud-based distributed storage and computation systems that provides practical access to distributed imaging archives, integrates with existing data workflows, and effectively functions with commodity hardware. Our approach makes specific improvements to the Apache Hadoop ecosystem, notably HBase, which is a NoSQL database built atop Hadoop's distributed file system. Specifically, we make the following contributions:

- **A row-key design for Apache HBase:** A hierarchical key structure is proposed as a necessary step to accommodate nested layers of priority for data-collocation.
- **New RegionSplit policy:** A computationally efficient approach is proposed to optimally manage data collocation in the context of the hierarchical key structure.
- **Experimental results:** The proposed innovations are evaluated in the context of a routine image analysis task (file format conversion) in a private research cloud comprising a typical Gigabit network with 12 nodes.

1.3.3 Addressing Challenge 3: theoretical and empirical way to identify the limits of Apache Hadoop for Medical Imaging compared with traditional cluster

For both traditional network file system approach and HadoopBase-MIP, we are able to provide two theoretical models for each scenarios. The models are two metric that user may care about. First, we consider wall-clock time, which represents the total time as experienced by the user. The second part is resource time, which measures elapsed time on each node when a process starts across all nodes.

1.3.4 Addressing Challenge 4: System optimization for boosting Big Data application

Based on above 3 sub challeneges, our target design criteria are (1) improving the HadoopBase-MIP frameworks performance in a heterogeneous cluster, (2) performing population based summary statistics on large datasets, and (3) introducing a table design scheme for rapid NoSQL query. Thus, we present a load balancer for heterogeneous clusters by moving data with the ratio matched cluster computation resource; a dataset summary statistic model is designed and implemented by MapReduce paradigm; and a HBase table scheme is introduced for fast data query to better utilize the MapReduce model.

1.3.5 Addressing Challenge 5: Enhancing big data frameworks for heterogeneous mixed workloads of multi-level based medical imaging analysis

To address the above challenges of heterogeneous mix of and realize the desired capabilities that can be hosted on the cloud, we present the design principles and empirical validation for a cloud-hosted, medical image processing as-a-service, and make the following contributions: (1) Simulation tools: We present a simulation engine suite to estimate the performance of running medical image processing on traditional cluster (centralized storage) versus the Hadoop-based approach (decentralized storage). We name our system as Hadoop & HBase Toolkit for Medical Image Processing (HadoopBase-MIP). (2) Semiautomated, real-time quality assurance (QA) model framework: We present the design of a semiautomatic, real-time monitor and checkpoint framework which aims to optimize the performance of medical image processing by finding anomalies in the first level processing in a timely manner thereby expediting the entire multi-level analysis; for this work, we focus only on two-level analysis jobs.

1.3.6 Addressing Challenge 6: Evaluation of big data registration-based Image Enhancement

Rather than validate each boundarys' correctness in BDRE due to difficulties of direct visualization of our target structures, we investigate the usefulness of BDRE for the multi-atlas (MA) segmentation on those deep brain structures and assume only T1 images are available. The target structures are two deep brain structures: thalamic nuclei and hippocampal subfields.

1.3.7 Addressing Challenge 7: How to integrate Apache Hadoop into high performance environment using logistical storage framework as back-end

We integrated a plugin between HDFS interface and LStore to provide regular input / output stream and other file system operation. The plugin can avoid data copying redundancy between HDFS and LStore. We empirically verified our HDFS LStore plugin on in-house HPC and long distance geographically WAN connected remote HPC cluster. The performance of the plugin was compared with two current LStore client interfaces: LStore command line interface (ideal case) and LStore fuse mounted client interface (baseline). As a preliminary work, two system metrics we care most for now: the throughput of (multiple) read and the throughput (multiple) write.

1.3.8 Dissertation Outline

The remainder of the dissertation is organized as follows: Chapter 2 presents prior work that relates to our research; Chapter 3 describes our approach to integrate AWS with medical image processing software JIST; Chapter 4 describes the state of art of our HadoopBase-MIP data colocation framework; Chapter 5 presents the theoretical bounds for performance trade of using HadoopBase-MIP and traditional cluster; Chaper 6 explores a real-time large volume of image averaging framework by using HadoopBase-MIP; Chapter 7 discusses our online checkpointing monitoring that embedded in HadoopBase-MIP for accelerating the QA for multi-level analysis; Chapter 8 discusses how BDRE image enhancement is verifid on two deep brain structures; Chapter 9 introduces how HDFS is integrated with HPC file system LStore; and Chapter 10 summarizes the goals of our ongoing work on investigating design patterns for designing blockchain-based systems and presents concluding remarks.

Chapter 2

RELATED WORK

2.1 Overview

Recent trends indicate that there is a substantial interest in adopting the MapReduce paradigm – and thereby the Apache Hadoop ecosystem – for medical image data processing. Several medical image processing studies have encountered one or more of the trio of computation, storage and network bandwidth bottlenecks, and have developed optimizations to overcome these encountered problems. This section compares our work with prior efforts in medical imaging and beyond. Additionally, we also review prior studies that are not necessarily focused on medical imaging. We scope out our comparisons to only those prior works that have leveraged the Apache Hadoop ecosystem.

2.2 Related Work involving Medical Imaging Applications with Medical Image Data

A recent study [32] illustrates how transitioning the medical image processing computations to the MapReduce paradigm and the Apache Hadoop framework pays rich dividends over traditional processing approaches, which often are sequential in nature. Our work differs from this prior work in that not only is our use case different – we focus on mapping DICOM images to NiFTI formats – but more importantly we demonstrate new optimization strategies for the Apache Hadoop ecosystem instead of simply leveraging the default strategies provided by Apache Hadoop, which is the case with most prior efforts. In fact the authors in this related work point out the need to identify opportunities for optimizations, which is precisely the intent of our presented research. Similarly, [33] demonstrates how the Apache Hadoop ecosystem can be used in medical imaging but do not report on any optimizations. A recent prior work [33] has used the Apache Hadoop ecosystem for

content-based image retrieval where the MapReduce paradigm is used to extract vector features of the images. Similar to [32], the authors in this study demonstrate how the Apache Hadoop ecosystem can be used in medical imaging but do not report on any optimizations.

The work reported in [6] is synergistic to our work in that it focuses on the row- versus column-oriented storage issues for DICOM images. The authors highlight the pros and cons of row- versus column-oriented storage policies, and indicate how the complex structure of the DICOM images requires a hybrid mechanism for storage. Specifically, their approach stores frequently used attributes of a DICOM file into row-based layer/store, and optional/private attributes into a column-based store so that it will reduce null values. The motivation stems from the fact that if all DICOM attributes are stored into a row-based store, then a search or joining operation will unnecessarily involve numerous null values thereby adversely impacting efficiency.

The SYSEO project [34] also describes a hybrid row-column data store for DICOM images using similar criteria as in [6] to decide between row- versus column-based storage. Their work was motivated by the need to find alternatives to existing but prohibitively expensive solutions for medical image storage. Moreover, image annotation and query retrieval were additional dimensions that needed improvements in performance.

Our work does not treat DICOM file attributes in as much depth as in [6], i.e., we need not to know the details of the attributes stored in a DICOM file when we store it to HBase; rather we simply store the entire DICOM file to HBase. For our DICOM to NiFTI processing, the processing operation can directly fetch the related attributes from DICOM files and convert them into NiFTI files. For other forms of medical imaging applications and data processing, such as image annotations, we may need to incorporate these hybrid storage mechanisms along with our optimizations, which forms our future work.

2.3 Usefulness of Hadoop & HBase in other Application Domains

Several prior research efforts have proposed different performance optimizations to different elements of the Apache Hadoop ecosystem for domains beyond just medical image processing. The MHBase project [35] describes a distributed real-time query processing mechanism for meteorological data with the intent to provide safe storage and efficiency.

The data in Internet of Things are always large volume, which update frequently and are inherently multi-dimensional. The work in [36] proposes an optimization based on high update throughput and query efficient index framework (UQE-Index) including pre-splitting the HBase region for reducing the cost of data movement. The work in [37] addresses the problem of the HBase multidimensional (up to four-dimension) data queries in Internet of Things with better response time.

Recent work in Internet of Things (IOT) [36] proposes an optimization based on high update throughput and query efficient index framework including pre-splitting the HBase region for reducing the cost of data movement. Likewise, [37] addresses the problem of the HBase multidimensional data queries (up to four-dimension) in IOT with better response time.

A recent work [38] demonstrates an optimized key-value pair schema for speeding up locating data and increase cache hit rate for biological transcriptomic data. The performance is compared with relational models in MySQL cluster and MongoDB.

The authors in [39] present an optimized HBase table schema focusing on merging information to fit in combination with customer cluster and constructing an index factor scheme to improve the calculation of strategy analysis formulas.

In summary, the above-referenced prior efforts tend to focus on optimizing the table schema, row key design for data fast access, update and query. For our work, we not only provide an innovative row key hierarchical design, but also optimize the default RegionSplitPolicy which goes deep into the HBase architecture. Our goal is to maximally collocate relevant data on same node for further and faster group processing. Moreover,

most prior works do not consider the cloud-based service aspect that we do.

In this following subsection we compare our work with a sampling of prior efforts. Specifically, we focus on related research along the following dimensions: supporting medical image processing in the cloud, efforts that estimate resource requirements and performance for medical image processing jobs, and efforts that conduct quality assurance.

2.3.1 System modeling - better understanding the data colocation in theory

Cluster based simulation toolkit is widely used to verify a new scheduling algorithm. Simgrid [40] and Gridsim [41] were designed based on this state-of-art. CloudSim is an extensible and popular simulation toolkit that enables modeling and simulation of Cloud computing systems and application provisioning environments. It supports both system and behavior modeling of Cloud system components such as data centers, virtual machines (VMs) and resource provisioning policies. It implements generic application provisioning techniques that can be extended with ease and limited effort. Currently, and it supports modeling and simulation of Cloud computing environments consisting of both single and inter-networked clouds [42]. Due to lack of advanced application models such as message passing applications and workflows or scalable network model of data center, Garg et al. extended CloudSim with a scalable network and generalized application model, which allows more accurate evaluation of scheduling and resource provisioning policies to optimize the performance of a Cloud infrastructure [43].

There are many simulators for energy and resource modeling [44, 45, 46]. The following perform Hadoop related modeling. Lin et al. [47] define the concept of relative computational complexity of MapReduce task to estimate the complexity of task, and illustrate the way to measure it. They analyze the detail composition of MapReduce tasks and relationships among them, decompose the major cost items, and present a vector style cost model with equations to calculate each cost items. The model includes in great detail including data parsing, data serialization, internal sorting with a platform independent term

to measure the costs of MapReduce. Song et al. [48] more focus on single job performance. They designed a light-weight hadoop job analyzer which can be used not only as a job performance analyzer but also a parameter collecter. They also proposed a prediction module, which combines two kinds of information given by the job analyzer and the history traces to predict job performance. Wang et al. [49] proposed a model predictor to help Hadoop cluster configuration by learning the practical experience of Hadoop configuration and narrow the configuration searching space we could optimize the Hadoop configuration with clear direction, not treating this optimization problem as a pure black optimization problem. Tian et al. [49] focuses on the relationship among the number of Map/Reduce slots, the amount of input data, and the complexity of application-specific components. The resulting cost model can be represented as a linear model which provides robust generalization power that allows one to determine the parameters with the data collected on small scale tests. The cost model furtherly helps resource allocation and financial cost. Nez et al. [50] targeted to conduct large experiments for large dataset. In their work , they provides a flexible and fully customizable global hypervisor for integrating any cloud brokering policy; and the simulator reproduces the instance types provided by a given cloud infrastructure.

While for our work, we focus more on large groups of jobs completion time and resource usage, namely wall-clock time and resource time. And our target models are cluster with shared network and HBase modeling that built upon Hadoop distributed file system.

2.4 Moving forward to software as a service and using cloud for medical imaging

The first two related work sections discuss the potential usage of creating a data colocation framework. Once such the medical image processing as-a-service is estabilished, we would like to browse more how it distinguish with other cloud based medical image analysis as a service approaches. Our aim is to find the current problem of processing monitoring and quality assurance of intermediate results. Wang et al. [51] 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 modified FeldcampDavisKress (FDK) algorithm to parallelization for using MapReduce and achieved 10 fold speedup. The Java Image Science Toolkit (JIST) is a tool that integrates with Amazon Web Serivce (AWS) EC2 and Amazon S3 storage to perform medical image processing, which submits local first level analysis to AWS to utilize the pay as go feature of the cloud [52, 22, 23, 53]. The work provides a cost/benefit model to predict the performance difference of using different categories of AWS cloud service. Huo et al. [54] provide a dockerizing approach for deploying large-scale image processing to High Performance Computing environment and potential affordable cloud.

Zhang et al. [55] implement a distributed computing framework using Apache Spark cloud for automatic segmentation of skeletal muscle cell image. The paper aims to split the muscle cell segmentation to available resources of the Spark cluster, and propose a parallelized hierarchical tree-based region selection algorithm to efficiently segment muscle cells. Roychowdhury et al. [56] 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 proposed method utilizes the system hardware independence of a cloud-based platform and builds a systematic workflow that further reduces the dependencies of feature selection and data modeling from medical classification tasks.

Chen et al. [57] presented a cloud-based collaborative and scalable image processing toolbox with a number of medical image utilities. The toolbox offers an open and web-wide collaboration platform for image processing by leveraging the user friendly interfaces and simple integration software architecture of Galaxy. They also explore technologies and software architecture study of using Hadoop for data-intensive image processing of large datasets. Bednarz et al. [58] provide a cloud based toolbox to allow user free access

and collaboration to create workflows for image processing algorithm designs for cellular imaging, advanced X-ray image analysis, computed tomography and 3D medical imaging and visualization. The system provides a way for carrying out image analysis, reconstruction and processing tasks using cloud based service provided on the Australian National eResearch Collaboration Tools and Resources (NeCTAR) infrastructure.

Mindcontrol is an open-source configurable web collaboration based quality control of brain segmentation using MongoDB and connects with Amazon S3 storage and Dropbox. The authors focus on integrating multiple QA metrics and providing better user-friendly interfaces, which is application-based rather than infrastructure-based [12]. Kagadis et al. [59] discuss the benefits of using the cloud to conduct algorithm validation by using real / synthesized datasets that are stored in cloud storage. The cloud can provide a benchmark and application environment for plugging in different algorithms. The Medical Image Archival and Analytics as-a-Service (MIaaS) is low-cost personal healthcare cloud service that provides a single apace for archiving medical images and analyzing medical images by software and/or physicians [60]. Chiang et al. [61] describe middleware to construct and develop a cloud service for medical image processing based on the service-oriented architecture (SOA) using ImageJ tools to process medical images. The authors also discuss security concerns of SOA's macro data. Mirarab et al. [62] present the Eucalyptus cloud infrastructure with image processing software "ImageJ" using improved genetic algorithm for the allocation and distribution of cloud VM resources. Liu et al. [63] propose the iMAGE cloud which is a three-layer hybrid software as a service (SaaS) cloud. It receives medical images and EMR data via the hybrid regional healthcare network. The images are processed in the high-performance cloud units using coronary extraction, pulmonary reconstruction, vascular extraction etc., which are helpful for clinical usage while the results are sent back to to regional users using the virtual desktop infrastructure (VDI) technology.

The above referenced works can be classified as software-as-a-service, platform-as-a-service, infrastructure-as-a-service, and analytics-as-a-service. They are more for clinical

usage, hosting the application for experiencing the benefit of the 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-level medical image processing of group based analysis. Moreover, none of them aim to solve the problem of processing monitoring and quality assurance of intermediate results.

2.5 Opportunity for big data multi-level medical image analysis

As introduced in introduction section, a framework with capability that can detect anomalies as early as possible in the processing stages in a multi-level medical image analysis is needed. In [64], 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 which contains common-join elements. The authors show that this technique avoids needless re-computation of joins, and were able to complete over 80 join operations using two Map/Reduce iterations. The work focuses on finding errors in a medical data-set though not specifically related to medical image processing.

While most of the studies [65, 66, 67, 68] incorporate some form of quality checking (though not cloud based) at individual scan level or outlier detection level (which are all at the first level) where data were checked for extreme outliers (relative to expected outcome for given study), these checks are not yet carried onto second level analysis stage. All the data is first processed and then based on the study design fed into second level analysis. While this approach is fine for smaller datasets, for studies involving large-scale data as mentioned above where a number of hypothesis get 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 getting processed than waiting until the end. This is a capability we provide in our work. Although quality assurance is shown in a two-level multi analysis use case, it can work for many different multiple level analysis, e.g. func-

tional MRI, positron-emission tomography (PET) and cortical surface reconstruction. For instance, functional MRI (fMRI) [69]- measures brain activity by detecting changes associated with blood flow; positron-emission tomography (PET) [70] - a nuclear medicine functional imaging technique that is used to observe metabolic processes in the body; optical coherence tomography (OCT) [71] - which is established medical imaging technique that uses light to capture micrometer-resolution, three-dimensional images from within optical scattering media and and cortical surface reconstruction [72] - to understand intrinsic two-dimensional structure of the cortical surface. Since current medical image processing software like FSL, SPM, ANTs and FreeSurfer [73, 74, 75, 9] all provide command-line based application for difference processing steps, we can embedded those application into HadoopBase-MIP, or find corporation to embed the concept of multi-level performance promoted monitoring strategy to their software for further cloud host service.

There are a number of studies that involve medical image acquisition and processing of large number of subjects [65, 66, 67, 68] like the Human Connectome Project (HCP) [65] with 1,200 subjects; Alzheimer's Disease Neuroimaging Initiative (ADNI), which is a longitudinal multisite observational study collecting imaging, clinical, and biologic samples at multiple time points in 200 elderly cognitively normal, 400 MCI, and 200 AD subjects [66]. The National Institute of Health Pediatric MRI data repository (https://pediatricmri.nih.gov/nihpd/info/index.html) comprises 554 studies for unsedated, healthy, psychiatrically normal children and adolescents/young adults that is useful to biomedical and bio-behavioral researchers, Autism Brain Imaging Data Exchange (ABIDE) [67] a grassroots consortium aggregating and openly sharing 1112 existing resting-state functional magnetic resonance imaging (R-fMRI) datasets with corresponding structural MRI and phenotypic information from 539 individuals with ASD and 573 age-matched typical controls (TC; 764 years) (http://fcon_1000.projects.nitrc.org/indi/abide/).

There are also a number of studies like multi atlas learner fusion [68] involving 3,464 MRI images. Large-scale proof of concept genetic study on schizophrenia that integrated

results from common variant studies of schizophrenia (33,636 cases, 43,008 controls) and volumes of several (mainly subcortical) brain structures (11,840 subjects) [76]. The Preprocessed Connectomes Project Quality Assessment Protocol (PCP-QAP) is QAP python toolbox that assembles several metrics for assessing the quality of both functional and structural MRI data, those quality check are done before data send to processing since it is very hard to do QA on processed data [77].

2.6 Opportunity for Apache Hadoop utilizing HPC computation resource and file system

In this section, we summarize several recent works that are related with HDFS with HPC based file system.

2.6.1 Integrate HDFS with existence HPC file system

Wang et al. presented an integration of HDFS with Gfarm file system, which is a globlal distributed file system [78]. The framework supports distributed data-intensive computation among multiple administrative domains using existing unmodified MapReduce applications across multiple clusters. Li et al. analyzed integrate HDFS with OrangeFS to build a hybrid Hadoop architecture that includes both scale-up and scale-out machines [79]. The framework can either deal with small and median (KB, MB) data sizes or with large (GB, TB) data size. They provide an automatically schedule method to tune cluster either scale-up or scale-out cluster to achieve the best performance. Li et al. compaired two storage backend for Hadoop in HPC cluster: dedicate and local storage [80]. Dedicated storage (OrangeFS) separates computation from shared storage resources. Local storage provides better locality. They analyze I/O intensive, data-intensive and CPU-intensive type of jobs and verify their finding with guidance on choosing the best pltform optimize the performance of different types of applications and reduce system overhead. Xuan et al. proposed a two-level storage by integrating the in-memory file system, Tachyon, and the parallel file system, OrangeFS [81]. In two-level storage, Tachyon is deployed on compute nodes

and OrangeFS is deployed on data nodes. Tachyon provides a temporal locality of data that is not needed to retrieve from data nodes through network. Their theoretical modeling and experimental evaluation storage can provide higher read and write throughput with limited number of compute nodes. Krishnan developed a framework named myHadoop, a framework for configuring Hadoop on-demand on traditional HPC resources, using standard batch scheduling systems such as TORQUE (PBS) or the Sun Grid Engine (SGE) [82]. With the help of myHadoop, users can utilize traditional HPC resources rather than a dedicated Hadoop cluster. It supports a regular non-persistent mode where the local file system on each compute node is used as the data directory for HDFS, and also a persistent mode where the HDFS can be hosted on a shared file system such as Lustre or GPFS. Yang et al. designed PortHadoop framework to solve the cross-platform (Lustre and GPFS) data access issue and keep default fault tolerance feature [83]. With PortHadoop, researchers can promptly analyze identified events without copying the entire data set from HPC parallel file system (PFS) to Hadoop, and consequently accelerate scientific discovery and significantly reduce costs in computation and storage. MapReduce can concurrently retrive data of PFS at runtime, and the data transfer operations from PFS to HDFS can overlap with MapReduce data processing. Baer et al. proposed pbs-spark-submit script to execute Apache Spark programs on PBS-based HPC cluster and shared-memory environments [84].

2.6.2 Utilize HPC environment to boost MapReduce and YARN resource management

Luckow et al. explores the trade-offs between running Hadoop/Spark applications on HPC environments and running HPC applications on YARN clusters [85]. They focus on the design of resource management middleware via integrating Pilot-based Dynamic Resource Management and YARN. Islam et al. proposed a design for HDFS on HPC Clusters with heterogeneous storage architecture called Triple-H [86]. Triple-H follows a hybrid storage usage pattern that can minimize the I/O bottlenecks and ensure efficient storage utilization with the same level of fault-tolerance as HDFS in HPC environments. They in-

troduced a RAM Disk and SSD-based Buffer-Cache to hide the cost of disk access during Read/Write operations and ensure reliability by utilizing SSD-based staging. They also propose effective data placement policies for Triple-H integrating with PFS installations, such as Lustre. Wasi-ur-Rahman et al. proposed a novel high-performance design for running YARN MapReduce on such HPC clusters by utilizing Lustre as the storage provider for intermediate data [87]. We identify two different shuffle strategies, RDMA and Lustre Read, for this architecture and provide modules to dynamically detect the best strategy for a given scenario. They also designed a dynamic adaptation technique to choose the better shuffle strategy during run-time of job execution. Islam et al. proposed to integrate HDFS with Lustre through RDMA-based key-value store [88]. We design a burst buffer system for Big Data analytics applications using RDMA-based Memcached and integrate HDFS with Lustre through this high-performance buffer layer. We also present three schemes for integrating HDFS with Lustre considering different aspects of I/O data locality and fault-tolerance. Wasi-ur-Rahman et al. proposed an accelerated execution framework (NVMD) for MapReduce and Directed Acyclic Graph (DAG) based processing engines to leverage the benefits of Non-Volatile Memory (NVM) on HPC systems [89]. Through NVMD, MapReduce is refined with integrating hybrid push and pull shuffle mechanism, non-blocking send and receive operations, and dynamic adaptation to the network congestion have been presented.

In summary, the above works either deal with integration of HDFS with existence HPC file system, or explore the optimization opportunities for MapReduce types of jobs in HPC environment. As a preliminary integration work of HDFS and L-Store will be introduced in chapter 9, the above related works can absolutely guide our future directions and treat as our future baselines.

Chapter 3

PERFORMANCE MANAGEMENT OF HIGH PERFORMANCE COMPUTING FOR MEDICAL IMAGE PROCESSING IN AMAZON WEB SERVICES

3.1 Problem Overview

The increasing accumulation of open-source multi-modal images makes it imperative to deploy local processing of images into cloud-based environments. An example of such a image processing framework is the Java Image Science Toolkit (JIST) [22, 23], a biomedical plugin for MIPAV [24], which is entirely developed in the Java programming language [90, 91] and is hosted on the Neuroimaging Informatics Tools and Resource Clearinghouse (NITRC). Amazon Web Services (AWS) provide reliable, on-demand, and affordable cloud computing services [20] that is promising for medical image processing. Different types of optimized Amazon Elastic Cloud Computing (EC2) instances are offered in AWS [92]. Amazon Simple Storage is easy to apply, cost oriented, and highly scalable as data/object storage [93]. Despite the promise, workflow definition/management and cloud configurations are two key challenges in this realm. We exploited the NITRC Computational Environment (NITRC-CE) [53] hosted by Amazon's AWS, to distribute each JIST "Execution Context" (process) on a separate cloud node. Through an applicable cost/benefit analysis, users are able to specify a grid size, automatically boot up a private network, install JIST into MIPAV, and set up all requirements for processing. All monitoring is managed through the standard JIST Process Manager. Our contributions are summarized as follows.

- We provide a concrete workflow definition/management for JIST to deploy a local project to be distributed and processed in the cloud.
- We integrate JIST to Amazon AWS (a reliable, on-demand, and inexpensive cloud computing services) to execute high performance computing.

- We demonstrate a cost/benefit analysis of a pipeline that can be easily parallelized and requires memory that would limit the processes to serially execute only on a local lab machine.

This work has been published in SPIE medical imaging 2016 [52].

3.2 Method

3.2.1 Workflow framework

Cloud configuration is based on Amazon AWS (of which S3 storage and EC2 are two components). Amazon S3 storage is mounted by every pay-for-use Amazon EC2 instance. Hence S3 is recognized as a shared cloud resource for all EC2 machines that are ordered by clients. S3 stores all necessary packages including MIPAV's install file, JIST plugins, JIST Library and other files related to configuration. Each Amazon EC2 machine can get the source packages from S3, and install related packages to run JIST.

Briefly, JIST pipelines are stored in a "layout" file, which specifies inputs and outputs as well as algorithm classes used for execution. The user's layout is later sent to S3 storage, and all processing is handled in the cloud virtual machines. The local host (local machine) meanwhile monitors the processing process in cloud. Rather than run whole layout on a typical cloud machine, each processing command (a JIST "execution context", which is a single process in the process manager) generated by the JIST scheduler is distributed to any of the free EC2 instances. The process manager periodically sends jobs that are ready to run to the cloud when idle machines are found. Each job has a local monitor that periodically pulls back its debug and error info in the cloud so that users can view the progress of a running job. Status "Complete" represents a process that has completed, and all files that have been produced during the proceeding have been replicated back to local workstation. Note, to copy files between systems, the directory paths must be modified to represent differences in mount points and local structures. Thus, outputs are available to

operate on locally even though processing happens on AWS. The workflow framework is presented in Figure 3.1.

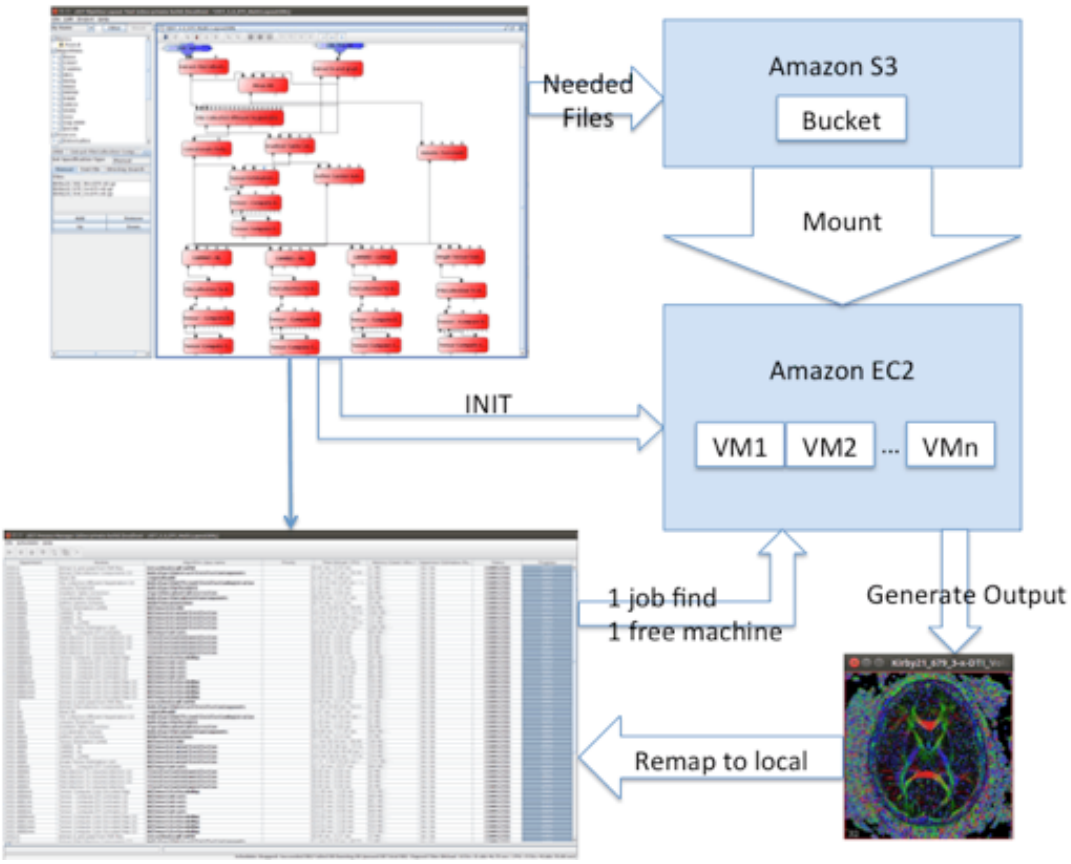


Figure 3.1: Throughput analysis for each of the test scenarios. (A) presents the number of datasets processed per minute by each of the scenarios as a function of the number of datasets selected for processing. (B) shows the fraction of time spent on overhead relative to the number of datasets.

3.2.2 Configuration

Several modes to run a layout can be selected in the JIST layout preference panel (e.g., run on a DRMAA compliant grid, run locally, or run in the Amazon Cloud). Users should choose use AWS EC2 option to activate cloud computing (Figure 3.2(a)).

Using the Cloud Dashboard, users are able to configure, initialize ("init"), boot, and terminate an Amazon AWS NITRC-CE cloud instance with predefined machine information (Figure 3.2(b)). Access Key Id, Secret access key and region information are used to

validate an AWS account so that user can access Amazon EC2 and S3 resources. Custom S3 buckets are a top-level folder on S3, the name of which must be globally unique in Amazon S3. This bucket is used to store necessary packages and JIST layout projects with the same structure.A standard cryptographic key pair is needed to securely log in and connect to remote cloud instances. Inbound and outbound connection rules including IP and port permission range are defined in security groups. Also, users can select several different types of pre-configured Amazon machines and any number of identical machines to boot. These machines have different processor speeds and memory installments. The initialize and install processes are executed once all necessary files and settings are prepared. In JISTs "Progress Manager", project input files are then uploaded to cloud and they wait for processing.

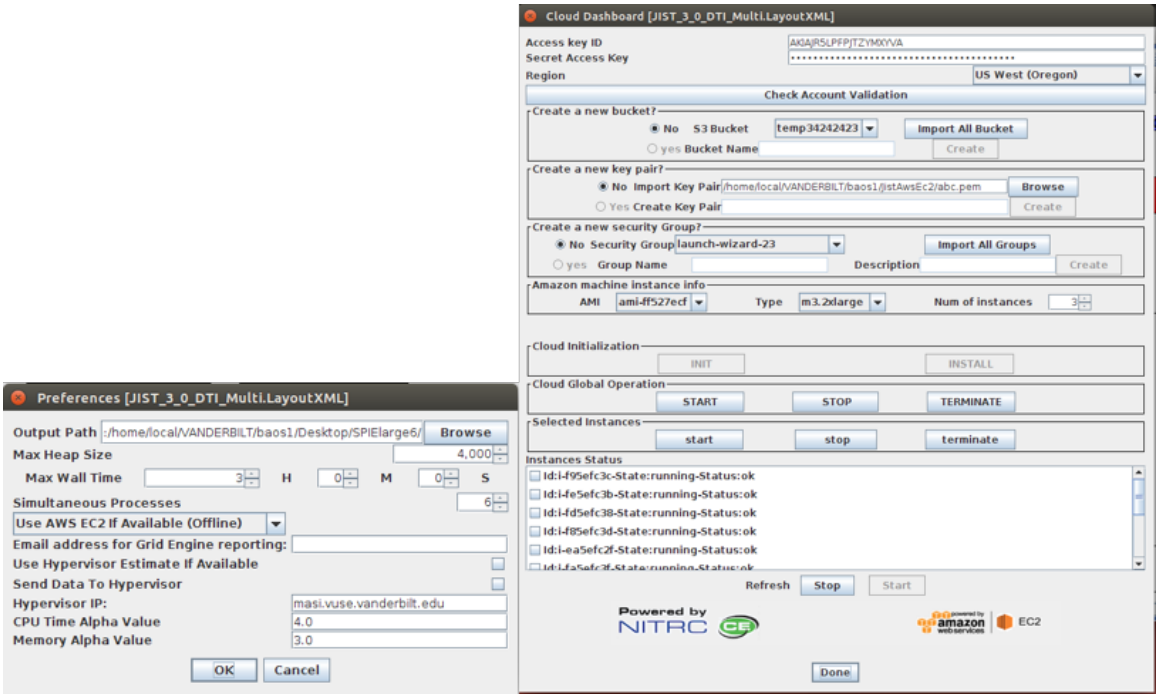


Figure 3.2: (a) JIST preferences panel (b) JIST Cloud Dashboard.

3.2.3 Cost/Benefit analysis

Time usage and expense are two integral factors to be considered cloud computing. Here, we consider if there is a way to estimate cloud usage costs relative to running serially on local lab machines. Total runtime in the cloud can be estimated by the following where $\boldsymbol{T_A}$ represents runtime in the cloud.

For the Amazon machine instance type A. The above variables are defined in detail in the following sections. Briefly, $\boldsymbol{B}$ is bandwidth, $\boldsymbol{S_{data}}$ is input data size, $\boldsymbol{S_{out}}$is the function to compute estimated execution time for all jobs, and $\boldsymbol{n}$ is the total number of cloud instances.

$$T_a = f(A, B, S_{data}, S_{out}, g(n)) = \frac{1}{A} \cdot (\frac{S_{data} + S_{out}}{B} + g(n)) \tag{3.1}$$

AWS provides a considerable spectrum of instances family for on-demand purpose, i.e., General Purpose (balance of compute, memory, and network resources), Compute Optimized (highest performing processors with lowest cost/compute), Memory Optimized (memory-intensive applications and have the lowest cost/GiB of RAM), and Storage Optimized (SSD storage with low cost high IOPS). Different types of instances have different processors, which need to be considered when predicting performance

Within each instance class, instances use same kind of processors with different total number. Memory should also be considered into install process (especially means installing MIPAV in each EC2 machines) and each job-processing time estimation. $\boldsymbol{A}$ is a weight value between (0, 1), when smaller cache and low memory allocation instance is applied, the value of A is more close to 0 (i.e., the machine is slower).

Input data size $\boldsymbol{S_{data}}$: Input image data size is also used to estimate processing time of a single job. In the example that we present, the process nearly linearly increases as the number of voxels increase. Amazon machine instance type $\boldsymbol{A}$ also affects the job time when taking processor cache and memory into consideration

Bandwidth $\boldsymbol{B}$: When running a big layout with lots of data that gets generated and mul-

tiple processing algorithms, upload/download time is not a primary factor in the model. However, if a layout is small, it is worth considering as the transfer process may end up taking as long, if not longer, than the processing time. On the same note, installation processes are also taken into consideration. The NITRC-CE includes many popular image processing toolkits, but is not guaranteed to have a compatible version of MIPAV and JIST — these are transmitted and installed for each newly created cloud node in the S3 bucket. In general, a small layout is a project when it is applied only one single process to run the entire job, the total time is smaller than an hour, and the process needs less than 500G memory. The uploading time depends on the size of the project, or $\boldsymbol{S_{data}}$, because the file size is usually bigger than any other type of input files in the JIST project (i.e., .input, .output, .LayoutXML). However, the output file size(s) $\boldsymbol{S_{out}}$ need to be estimated. Subsequently, install upload and download time is then computable via bandwidth $\boldsymbol{B}$.

Before explaining the details of $\boldsymbol{g(n)}$, we first introduce the job dependency tree $\boldsymbol{D}$. JISTs scheduler can handle the graph of execution dependencies. A lower priority value results in later execution and vice versa. The scheduler itself can also estimate execution time of a job. If several jobs are ready to run with the same priority at the same time, the job that has the smallest execution time will run first. For example, there are 4 jobs are all ready to run with same priority value, the estimation execution time is: job-a (15 minutes), job-b (18 minutes), job-c (50 minutes) and job-d (60 minutes). If two free instances are available, job-a and job-b will start first, and this will cost less time than any other combination of 2 jobs to run first. As a result, a job dependency graph can be generated. This can help a user to decide the total number of instances.

$\boldsymbol{D}$ is a depth-first-search tree, according to job dependency. We first build an entire tree, and denote each node as slow and fast to represent processing time of a job. Then we remove all fast nodes, and connect all slow nodes via the hierarchy. Figure 3 presents an example of how $\boldsymbol{D}$ is generated.

Based on $\boldsymbol{D}$, we first calculate the total number of nodes on each level and get the

maximum number among all levels. We can then get an estimated number of maximum processes that can run at once. The average number of machines can be derived from a level in $\boldsymbol{D}$, which has the maximum total execution time and maximum single running time in the selected level. If $\boldsymbol{T_{i\text{—}i=1,2,3m}}$ is processing time of m jobs in a chosen level, average number of instances then can be calculate by:

$$avg(n) = \left\lceil \frac{\sum_{i=1}^{m} T_i}{\max(T_{i|i=1,2..m})} \right\rceil \tag{3.2}$$

More machines can be booted with the benefit of decreased execution time at the expense of increased cost. Once $\boldsymbol{n}$ is set, if there are $\boldsymbol{x}$ paths in $\boldsymbol{D}$, we group $\boldsymbol{x}$ paths into $\boldsymbol{y}$ groups, where y equals $\boldsymbol{x/n}$, and $\boldsymbol{T_y}$ is max time to run all jobs in group $\boldsymbol{T_y}$. $\boldsymbol{T_{duplicate}}$ is the duplicate time among each groups. Finally, the estimation time to process all jobs in the cloud is

$$g(n) = \sum_{i=1}^{y} T_i - T_{duplicate} \tag{3.3}$$

According to $\boldsymbol{T_A}$, type of instance and the number of nodes, the expense $\boldsymbol{E}$ is easy to calculate as

$$E = \lceil T_A \rceil \cdot num(Inst) \cdot price(A) \tag{3.4}$$

A small amount of data transfer and data storage fee also occurs, but it is considerably less than the cost of renting EC2 machine.

The tradeoff between number of instances booted and cost is one of the most important considerations to make. For example, consider a layout where there are many fast jobs that can run simultaneously, and they are all depending on a single slow job. In this scenario, we should cautiously select the number of machines to reduce the idle time of instances, as this is payment for non-processing time.

Different types and numbers of instances lead to different $\boldsymbol{T_A}$, which incurs several

pairs of ($\boldsymbol{E}$,$\boldsymbol{T_A}$). Based on budget and expectant time, users can decide if JIST cloud mode is applicable.

Our test experiment is a multi-subject diffusion tensor imaging (DTI) project [53]. DTI is a method to non-invasively measure water diffusion in the human brain using magnetic resonance imaging (MRI). In this example, three different subjects are in the layout. The layout is used to fit tensors to the acquired data using four different tensor fitting routines (Figure 3.3). The input data size is 256*256*65*D, where $\boldsymbol{D}$ is the number of directions at which different gradients are applied and readout. In this case, there are 34. Table 3.1 illustrates the experiment top-down structure. Level means the job hierarchy and dependency.

The final Job dependency DFS tree $\boldsymbol{D}$ is shown in Figure 3.4. In summary, the total number of level-2 in $\boldsymbol{D}$ is larger than level-1, so the maximum number of processes that are running at the same time in our experiment is 12. Moreover, they are all slow jobs, so we should consider applying multiple instances rather than a single instance. In our experiments, in order to control variable, we apply the same Amazon instance family (namely M3), the processors of which is High Frequency Intel Xeon E5-2670 v2 (Ivy Bridge) Processors.

3.2.4 Case 1: Same total number of instance n, different Amazon instance type A

As a brute force approach, for each instance type, we all apply maximum number of instances 12. Table 3.2 presents the result of actual total running time of each option with total cpu time in cloud and expense. Total Cpu time in cloud is the time using only a single process to execute entire jobs in one virtual machine, which is also regarded as run layout in local workstation. Figure 5.1 visually shows the performance of JIST cloud mode compared with total cpu time.

The result shows the estimation time $\boldsymbol{T_A}$ is reasonable with 90% precision. It illustrates that Medium instances can accomplish processing the project for a low cost (a bit more

Table 3.1: Multi-slice DTI experiment layout top-down structure (one subject).

Job Id	Description	Dependent node	Timeestimation
T1	Extract b values and gradient table		Fast
T2	Makea scheme file	T1	Fast
T3	Registration	T1	Slow, 1.25 hrs/subject
T4	Tensor fitting-CAMINO-WL	T3	Slow, 15 mins/subject
T5	CAMINO-WL DTI contrast	T4	Fast
T6	CAMINO-WL DTI Color encode map	T4	Fast
T7	Tensor fitting-CAMINO-NL	T3	Slow, 2.5 hrs/subject
T8	CAMINO-NL DTI contrast	T7	Fast
T9	CAMINO-NL Color encode map	T7	Fast
T10	CAMINO-LLMSE	T3	Slow, 15 mins/subject
T11	CAMINO-LLMSE DTI contrast	T10	Fast
T12	CAMINO-LLMSE encode map	T10	Fast
T13	Single tensor estimation	T3	Slow, 1.25 hrs/subject
T14	Single tensor estimation DTI contrast	T13	Fast
T15	Single tensor estimation color encode map	T13	Fast

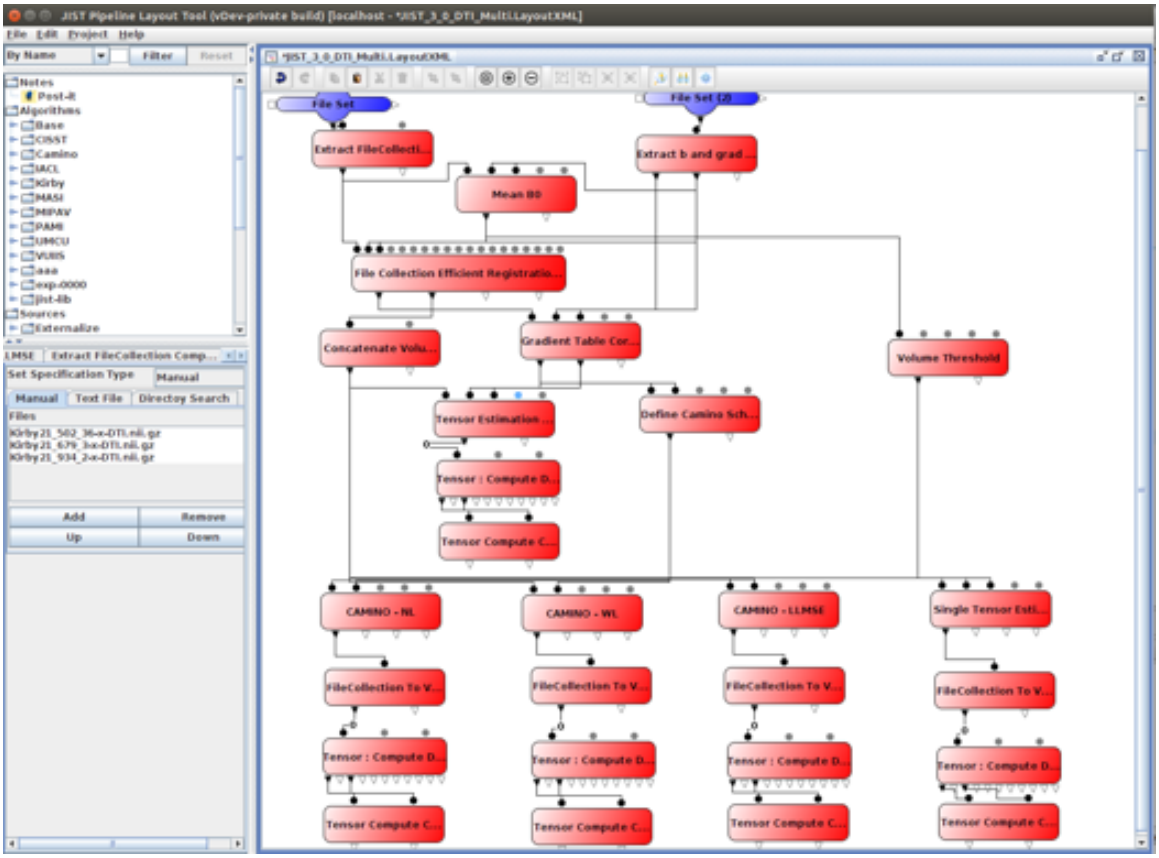


Figure 3.3: Multi-slice DTI test layout.

than Large instances), but approximately with double the needed time over other instances type because of lack of memory allocation. The result also demonstrates xLarge's memory is big enough for our experiment with a totalexperiment time of 4 hrs 10 mins. The time does not dramatically decrease even when we apply a more expensive type instance (e.g., 2xLarge). However, the total expense of 2xLarge is much more than the other 3 instance types. Note that the expense of Medium is even more than Large instances. Running 12 jobs at the same time is hard for a normal local station, while running 12 jobs with 12 available instances is easy to accomplish.

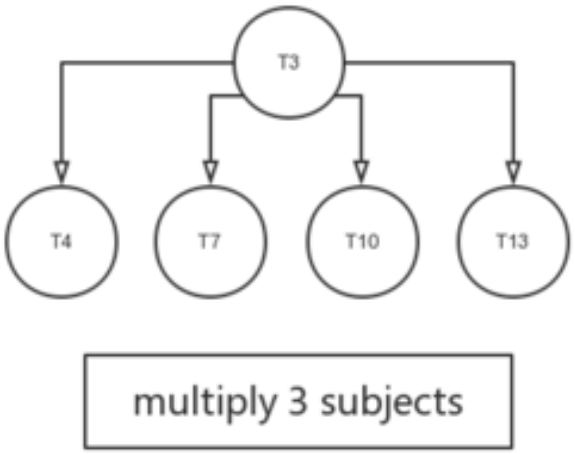


Figure 3.4: Job dependency DFS tree for multi-slice DTI experiment.

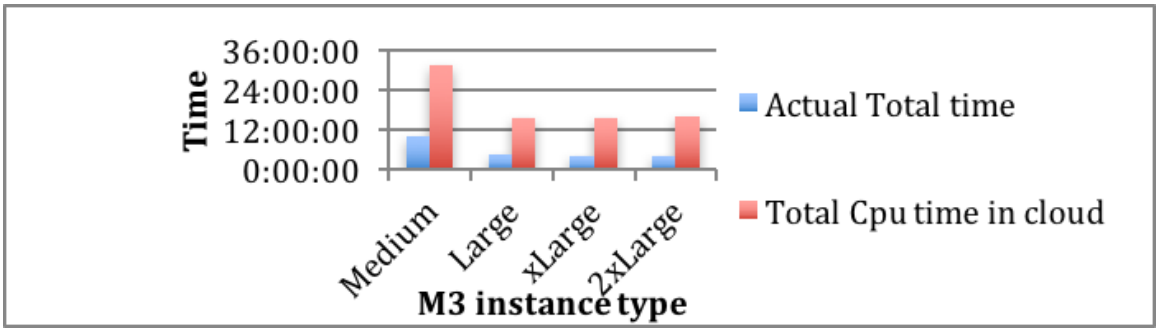


Figure 3.5: M3 Instance type.

3.2.5 Case 2: Same instance type (large), different number of machines n.

Due to job top-down structure, for one subject, 4 tensor fitting methods can run simultaneously. We use it to c ompute ***avg(n)*** equals 1.68 which means an average of 2 instances can be applied for a subject. Thus, we should apply 6 instances in total for entire project. Table 3.3 demonstrates that 12 large machines just save 16 minutes (4%) more than 6 large machines, which is because of the structure of the project.Before tensor fitting, at most 6 machines will be used, so basically 6 machines will be idled when we apply 12 machines. While when we perform tensor fitting, 12 machines can save partially parallel time. When using 12 machines, 12 tensor fitting algorithm are run at the same time. While when 6 machines are ordered, owing to JIST scheduler, CAMINO-WL and CAMINO-LLMSE are completed first, and job CAMINO-NL and Single tensor Estimation are then run. In this

Table 3.2: Case 1 result summary

Instance Type	Total Machines	Weight value ***A***	Estimation time ***TA*** (hr:min:sec)	Actual Total time (hr:min:sec)	Total Cpu time in cloud (hr:min:sec)	Total Expense ($)
Medium (Cpu:1, Mem: 3.75G)	12	0.4	9:22:00	9:57:12	31:27:31.24	8.04
Large (CPU:2, Mem:7.5G)	12	0.9	4:10:00	4:42:40	15:38:8.35	7.98
xLarge (CPU:4, Mem:15G)	12	1	3:45:00	4:09:57	15:34:49.89	15.96
2xLarge (CPU:8, Mem:30G)	12	1	3:45:00	4:07:13	15:49:18.16	31.92

Table 3.3: 12 Large instances v.s. 6 Large instances.

	12 large instances	6 large instances
Weight value A	0.9	0.9
Estimation time TA (hr:min:sec)	4:10:00	4:26:00
Actual Total time,(hr:min:sec)	4:42:40	5:01:04
Total Cpu time in cloud,(hr:min:sec)	15:38:8.35	15:49:35.8
Expense($)	7.98	4.82

case, total number of 6 large instances should be ordered since it saves 40% expense and just need 16 more minutes to complete the entire jobs.

3.3 Conclusion

This chapter has presented a system to enable automatic deployment of JIST to the Amazon AWS cloud. Each processes of a user's project are processed in separate nodes in Amazon AWS. Mounting Amazon S3 storage to EC2 is automatically accomplished for efficient I/O and data transfer in cloud environment. Our cost/benefit analysis can help users to decide grid size and configuration relative to local computation. A performance

prediction scheme to accurately predict computing resource is found in [94], stricter cost estimation could be applied to current work. EC2 limit performance compared with conventional HPC platform has also introduced in [53]. So our future work is to maintain JIST cloud version and exploit cloud-based JIST to run very large parallel processes with efficient use of EC2 machines.

Chapter 4

CLOUD ENGINEERING PRINCIPLES AND TECHNOLOGY ENABLERS FOR MEDICAL IMAGE PROCESSING-AS-A-SERVICE

4.1 Problem Overview

Traditional grid computing approaches separate data storage from computation. To analyze data, each dataset must be copied from a storage archive, submitted to an execution node, processed, synthesized to a result, and results returned to a storage archive. This is the workflow traditionally adopted in processing medical imaging datasets. However, when imaging datasets become massive, the bottleneck associated with copying and ensuring consistency overwhelms the benefits of increasing the number of computational nodes. For example, consider the activity of converting Digital Imaging and Communications in Medicine (DICOM) files to NiFTI (a research file format); if converting a 50 MB volume takes 15 seconds, an ideal Gigabit network ($\approx 100MB/s$) saturates with slightly less than 30 simultaneous processes.

These challenges are further amplified considering the current trends where vast magnetic resonance imaging (MRI) and computed tomography (CT) databases are accumulating in radiology archives (at the rate of nearly 100 million examinations per year in the U.S.). However, we lack the image processing, statistical, and informatics tools for large-scale analysis and integration with other clinical information (e.g., genetics and medical histories). An efficient mechanism for query, retrieval, and analysis of all patient data (including imaging) would enable clinicians, statisticians, image scientists, and engineers to better design, optimize, and translate systems for personalized care into practice. Thus, a cloud-based service to address these needs holds promise.

It may however appear tempting to reuse the cloud-based solutions for social networks

and e-commerce, which provide a solution to this problem that is both simple and relatively inexpensive. These solutions combine the storage and execution nodes such that each task can be done with minimal copying of data. For example, the Apache Hadoop ecosystem [19], which provides the Big Data processing capabilities, has been extensively used in these contexts.

Two reasons preclude such a naïve reuse. First, although such big data architectures have been applied in online commerce, social media, video streaming, high-energy physics, and proprietary corporate applications, these technologies have not been widely integrated with medical imaging data formats (e.g., DICOM) for medical image processing. Second, several approaches have followed the path of general machine learning literature and seek to implement algorithms specifically designed to take advantage of big data architecture [95, 58, 96], exploit the MapReduce framework to sift through datasets [32], or use distributed file systems [97, 98]. While such approaches have been effective for genetics studies [98, 57], they have not yet proven effective within current medical image computing workflows.

The fundamental reason for this shortcoming is that substantial resources have been invested in creating existing algorithms, software tools, and pipelines, and hence there is a substantive (often prohibitive) cost associated with algorithm re-implementation and redesign specifically for big data medical imaging. Consequently, there is a need for new approaches that will not require algorithm re-implementation while still being able to exploit the potential of elastic, cloud-based frameworks, such as Apache Hadoop, that have shown promise in other application domains.

To address these problems, we present design principles and empirical validation for a new data model for use with cloud-based distributed storage and computation systems that provides practical access to distributed imaging archives, integrates with existing data workflows, and effectively functions with commodity hardware. Our approach makes specific improvements to the Apache Hadoop ecosystem, notably HBase, which is a NoSQL

database built atop Hadoop's distributed file system. Specifically, we make the following contributions:

- **A row-key design for Apache HBase:** A hierarchical key structure is proposed as a necessary step to accommodate nested layers of priority for data-collocation.
- **New RegionSplit policy:** A computationally efficient approach is proposed to optimally manage data collocation in the context of the hierarchical key structure.
- **Experimental results:** The proposed innovations are evaluated in the context of a routine image analysis task (file format conversion) in a private research cloud comprising a typical Gigabit network with 12 nodes.

The performance of this new system is evaluated on small (7 GB) to moderate-sized (530 GB) test cases to characterize the overhead associated with this model and demonstrate tangible gains on widely available network and computational hardware. We believe that the proposed improvements to the Apache Hadoop ecosystem will greatly reduce the technical barriers to performing high-throughput image processing necessary to integrate imaging data into actionable metrics for personalized medicine. The novelty of the approach lies in our integrated solution for a novel application. The row-key design and linearizing most heavily used fields have been used in other contexts [99, 100, 101]. Yet, realizing new opportunities to apply existing techniques to a different realm (medical image processing), with its specific ontologies and patterns of data size/access, is essential to driving innovations. Our effort is a mix of research and experimental work to demonstrate applicability to medical imaging, which, to date, has not used a data-collocation computational model, and instead typically relies on monolithic data warehouses.

The rest of the chapter is organized as follows: Section 4.2 describes our contributions; Section 4.3 describes our evaluation approach and presents experimental results; and finally Section 4.4 presents concluding remarks alluding to ongoing and future work, and discusses the broader applicability of our approach.

This work has been published in IEEE International Conference on cloud Engineering (IC2E) 2017 [102].

4.2 Design Principles for a Cloud-based Medical Image Processing Service

The task of processing medical images at scale requires a distributed (i.e., a cloud-based) image processing architecture that is aware of the underlying hierarchical imaging data and its meta-data. Our system is based upon the Hadoop framework, which was originally designed for file-system management and distributed processing [4, 103]. We combine Hadoop with Apache HBase, a NoSQL database which implements Google's BigTable [103, 31]. The specific contribution of this chapter is a novel data storage mechanism that uses the hierarchical structure of imaging studies to collocate data with physical machines. This proposed collocation provides an efficient processing environment in which data do not need to be transferred between machines, thereby avoiding network overhead and saturation.

To make this chapter self-contained, we first describe the properties of the DICOM and NiFTI file formats used by the medical imaging community alluding to the challenges. We also provide background information on Apache Hadoop's HBase and challenges in applying these frameworks to medical image processing.

4.2.1 Background on DICOM and NiFTI and Challenges

DICOM (Digital Imaging and Communications in Medicine) is the international standard for medical images and related information (ISO 12052). It defines the formats for medical images that can be exchanged with the data and quality necessary for clinical use (http://dicom.nema.org/). It has a hybrid structure that contains regular data (patient/clinical information), multimedia data (images, video). Data inside a DICOM file is formed as a group of attributes [6].

When a patient gets a Computed Tomography (CT) or Magnetic resonance imaging

(MRI) scan, for example a patient's brain image, a group of 2-dimensional DICOM images are generated slice by slice. A non-exhaustive set of medical imaging DICOM attributes for the slices include: project (i.e., a particular study), subject (i.e., a participant within the study), session (i.e., a single imaging event for the subject), and scan (i.e., a single result from the event).

In order to study the entire brain, all 2-dimensional DICOM images should be collected together. Even though medical imaging data is stored as DICOM images, a substantial amount of medical image analysis software are NiFTI-aware (e.g. FSL (http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/), AFNI (https://afni.nimh.nih.gov/afni/), SPM (http://www.fil.ion.ucl.ac.uk/spm/) and Freesurfer (http://freesurfer.net/)). NiFTI is a medical image data format, which is termed as a "short-term measure to facilitate inter-operation of functional MRI data analysis software package," developed and founded by the NiFTI Data Format Working Group (http://nifti.nimh.nih.gov/).

Converting a large group of slices of DICOM images belonging to one patient into a small number of NiFTI format images (many-to-many relationship) is a significant step in a medical imaging study. Any processing of DICOM datasets will need to determine which CT/MRI **scan** that slice belongs to and using the **Session** attribute which records when the CT/MRI scan volume is carried out. However, finding a **Session** needs to first know the attribute **Subject** that it belongs to. Finally, the **Project** attribute collects all subjects together. Thus, for medical imaging applications involving DICOM, the following attributes are necessary: $project \rightarrow subject \rightarrow session \rightarrow scan \rightarrow slice$.

As noted in Section 4.1, traditional grid computing approaches separate DICOM data storage from computation. To complete a DICOM to NiFTI conversion, DICOM datasets must be copied from a storage archive, submitted to an execution node, processed, synthesized to a result, and the results returned to a storage archive. When imaging datasets become massive, the bottleneck associated with copying and ensuring consistency overwhelms the benefits of increasing the number of computational nodes.

4.2.2 Background on Apache HBase and Challenges for Medical Image Processing

HBase [7] uses the Hadoop Distributed File System (HDFS) to provide distributed and replicated access to data. Zookeeper [104] handles distributed coordination to maintain the state in HBase cluster. It uses consensus to guarantee common shared state; hence the recommended cluster size is an odd number for cluster leader selection. The reason we choose HBase and not any other data store available in the open source community is that HBase can group data logically and physically based on row-key value. However, HDFS can only provide logical order, and files with similar names can also be placed on different remote machines. HBase also provides a flexible translation layer (region-split policy) for dealing with data collocalization statically or dynamically.

The key concepts from the HBase architecture are summarized in Table 4.1. Briefly, HBase maintains tables, which have a row key that is commonly used as an index, and where data columns are stored with the row key. All data in HBase is "type free," which are essentially in the format of a Byte Array. The table is sorted and stored based on the row key.

The HBase tables are divided into "regions" for distributed storage, where each region contains a continuous set of row keys from the table. The data in a region is physically collocated with an HDFS data node to provide data locality, which is performed by an operation called major compaction. When the region size grows above a pre-set physical size threshold, a "RegionSplitPolicy" takes effect and divides the region into smaller pieces. The newly created regions are automatically moved to different nodes for load balancing of the cluster. The row key and RegionSplitPolicy are thus integral to the performance and data retrieval of HBase and Hadoop.

There is no standard for the default row key design. Intuitively, the data should be placed as sparse as possible and distributed evenly across various points of the regions in the table. Such a strategy can avoid data congestion in a single region, which otherwise could give rise to read/write hot-spots and lower the speed of data updates. Because row

Table 4.1: HBase architecture key concepts summary

Concept	Comment
Table / HTable	A collection of related data with a column-based format within HBase.
Region	HBase Tables are divided horizontally by row key range into "Regions." A region contains all rows in the table between the region's start key and end key.
Store	Data storage unit of HBase region.
HFile / Storefiles	The unit of Store, which is collocated with a Hadoop datanode and stored on HDFS.
memStore	When write data is uploaded to a HTable, it is initially saved in a cache as memStore. Once the cache size exceeds a pre-defined threshold, the memStore is flushed to HDFS and saved as HFile.
HMaster	HBase cluster master to monitor a RegionServer's behavior for load balancing.Table operator. e.g., create,delete and update a table.
Regionserver	Serves read/write I/O of all regions in a cluster node. When Regionservers collocate with Hadoop datanode, it can achieve data locality. Subsequently, most reads are served by the RegionServer from the local disk and memory cache, and short circuit reads are enabled.
Rowkey	A unique identifier of a row record in table.
Column family	Columns in Apache HBase are grouped into column families.
Column identifier	The member in column family, also called as column qualifier. Multiple column identifiers can be used within one column family.

keys are sorted in HBase, using randomly generated keys when input the data to HBase can help leverage the data distribution in the table. As shown later, however, such an approach incurs performance penalties for medical imaging applications.

Despite using Big Data architectures, such as Apache Hadoop, two key challenges present themselves for medical image processing.

Challenge 1: It must be noted that the original DICOM file name is a unique identifier called Global Unique identifier [25]. If the task of interest is storing slice-wise DICOM data within HBase, a naïve approach would be to use the DICOM GUID. Since the GUID is a hash of the data, it will not collocate data together and thus will saturate the network while retrieving all DICOM images of a scan volumes.

Challenge 2: The standard RegionSplitPolicy will randomly assign files with hashed DICOM GUID file name as row keys to regions based on the key and the convenient split

point based on region size, which may incur significant data movement over the network.

We address these challenges in deploying a cloud-based medical image processing service.

4.2.3 Design Principle 1: Modified Row Key Design

To address Challenge 1 discussed above, we propose a modified row key design for HBase based on the row key design requirements that it must maintain the structure of DICOM comprising the project, subject, session, and scan. To maintain this structure, we propose using $< ProjectID > \_ < SubjectID > \_ < SessionID > \_ < ScanID >$ as the identifier with other optional characteristics such as the "slice" appended to this identifier. This is how our collection of images are named in the hierarchical manner.

For example, a row key is like $Proj1\_Subj2\_Session3\_Scan4\_Slice5\_example.dcm$, where ".dcm" is the filename extension for DICOM. Since HBase organizes data linearly based on row key, this new strategy will maintain data within a project that is split with a minimal, or just one more than the minimal number of splits across regions as possible, when used in conjunction with the default RegionSplitPolicy supported by HBase.

4.2.4 Design Principle 2: Modified RegionSplitPolicy for Medical Imaging

In practice, users always select a cohort (set of subjects, sessions) to do the processing. The data under the same subjects or sessions are always processed together, not individually. Specifically, for DICOM conversion, the unit of processing is scan volumes. Thus, it is important to maximally collocate relevant image data under the same level for further group retrieval and processing/analysis, while reducing the data movement in MapReduce operations pertaining to the DICOM to NiFTI conversion. The details of the MapReduce operation is discussed in Section 4.3.4.1.

Unfortunately, HBase's default RegionSplitPolicy does not have knowledge of the structure of medical imaging studies. Thus, it may split the regions non-ideally such as within

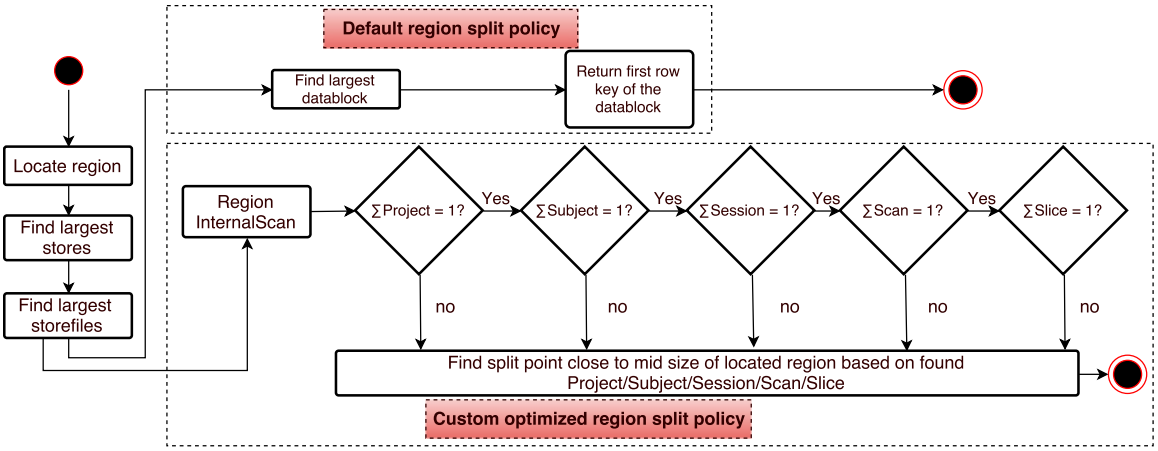


Figure 4.1: Comparison of the standard RegionSplitPolicy and our custom RegionSplitPolicy. The standard policy splits the data within a region equally based on the data in the region. The custom policy considers the projects, subjects, sessions and scans in the region and makes a split to maximize data co-locality.

a scan or between sessions for a particular subject. These splits cause increased processing time when data needs to be transferred between machines over the network. To overcome these bottlenecks, we propose a novel RegionSplitPolicy, which exploits the modified row key structure from our first contribution, thus maximizing data co-localization.

Our new RegionSplitPolicy first considers all row keys in a region. If multiple projects exist in the region, it splits the projects into separate regions. If the region is homogeneously a single project, it finds the highest available level (project, subject, session, scan) in the region on which it can split and balance the data between the new regions. This proposed RegionSplitPolicy relies on the modified row key design from Section 4.2.3. Figure 4.1 compares the operation of the standard RegionSplitPolicy with the custom one we have developed.

In detail, when a region split is triggered, the RegionServer first finds the largest files in the largest stores. The default RegionSplitPolicy defined in HBase finds the first row key of the largest data block in each storefile. This key is called the "midkey" of a region and is determined based on region mid size. Thus, this split point can separate an existing associated imaging dataset into two regions without considering row keys values in the

split region. The newly created two regions will move through the entire cluster for storage balancing.

The challenge for our optimized RegionSplitPolicy is to find a split point based on all row keys of a Region. HBase provides a client API to retrieve the data called scan (here, we refer to it as simple_scan). A user can customize the scan to define the range of row keys with which the column family and identifiers need to be retrieved. Users can also set customized filters to refine the query scan. A region has internal attributes that record the value of the start row key and end row key of the region. Since there are no attributes of records for any other row keys in a region except start/end row key, we need to use external means to retrieve all row keys of a region.

To be effective, the region must be split based on values of all row keys, however, HBase does not maintain all this information. Thus, to retrieve all keys in a region, two potential approaches can be used: (1) According to start/end key of region, a user specifies a column to scan. The scan is first executed on the entire table, finds the right RegionServer that hosts region from Zookeeper quorum, and retrieves the row key; (2) Use HBase default RowKey filter to customize the scan. However, both approaches are slower compared to our approach described below.

As shown in Figure 4.1, we are capable of locating the largest storefiles. In this way, we can apply a more advanced HBase scan API (called "Region Internal scan"), which we have found to be 163 times faster than simple_scan on average in our tests to find all hierarchy row keys involved in the region. The Internal scan can directly operate on the storefiles located on HDFS without starting a scan from the entire table. This gives us all the row keys of a split region. Next, the split point is selected according to the following conditions: (1) it ensures that related data is maximally collocated in the hierarchy, and (2) once we have identified the level of structure which will be the potential point to split, we traverse the candidates and return the point that can most evenly balance the size of the two new regions in order to avoid the overhead of many small regions emerging.

HBase provides PrefixSplitKeyPolicy as one of the default split policy which is designed for grouping rows sharing a fixed length of keys [3]. However, compared to our custom policy, it cannot dynamically group the subjects based on the order of the project, subject, session etc based on highest available level (project,subject, session, scan). Specifically, if there are many projects of a region, we should split rows by $< ProjectID >$; if all row keys start with same project, and there is not only one subject, we should split rows by $< ProjectID > \_ < SubjectID$, so on so forth. So we cannot define which is the appropriate fixed length for PrefixSplitKeyPolicy.

Another policy we evaluate and compare with is the IncreasingToUpperBoundRegionSplitPolicy [3], which splits a region from a small threshold; with the number of regions increased in a Regionserver, the split size threshold are increasing. It is a size-based policy without knowledge of what key values are in a split region.

The observed average run time range to determine one split point using our custom split policy is 28.22-58 ms, and 1.43-1.64 ms for the default HBase split policy with similar CPU usage (19.39% vs. 19.81%) indicating no substantial overhead in our approach compared with the default one. The increased time in our policy is due to the need to retrieve and analyze all row keys of a region. Despite this one-time initial cost, as we show in our experimental results, the performance improvements are substantial.

4.2.5 Putting the Pieces Together

Figure 4.2 presents the overall structure of our cloud-based medical image processing service, which is an Apache Hadoop ecosystem focusing on the HBase modifications. HBase resides upon HDFS. Zookeeper monitors the health status of RegionServer. When users create a HBase table, they need to pinpoint the RegionSplitPolicy to HMaster, and the pre-set split policy is automatically triggered once when a Table region needs to be split. Our custom split policy is made the default split policy. The input DICOM is de-identified (for privacy preservation purposes) and is normalized to the hierarchy structure by a local

row key generator before storing into HBase.

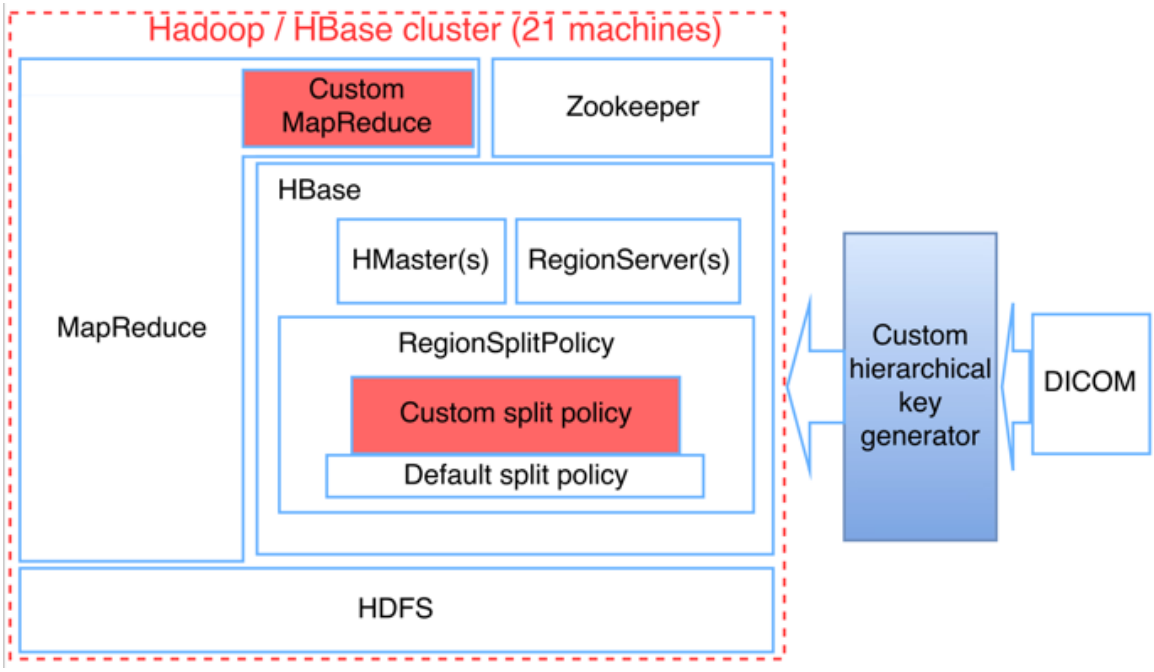


Figure 4.2: Overall structure of Hadoop / HBase / Zookeeper cluster with proposed custom row key and custom region split policy

Such a backend can then be offered as a service over the cloud using contemporary RESTful web technologies. In the current offering of our work, a shell-script interface (similar to Sun grid engine command line) is provided and multiple use cases have been tested [105]. Moreover, a template MapReduce job is provided that can be easily modified and packaged as a Maven project. Since our custom RegionSplitPolicy must be redesigned for different levels of hierarchy – our scenario has 5 levels (project/subject/session/scan/slice) – we have provided a template to generate the region split based on the levels, which alleviates the burden for the user. A more user-friendly version is in development and will be evaluated on larger datasets.

4.3 Evaluation Methodology and Experimental Results

This section presents results of evaluating our Apache Hadoop/HBase modifications for the cloud-based medical image processing service and comparing with default strategies.

4.3.1 Testing Scenarios

To investigate the performance of our HBase modifications, we evaluated the standard DICOM to NiFTI file format conversion using three test scenarios using HBase and Hadoop and one with Network Attached Storage (NAS) as follows.

• **Scenario: "Naïve HBase" –** The project data was anonymized such that the original GUIDs were lost prior to this project and could not be recovered associated with data retrieval. True DICOM GUIDs are globally unique and contain both source (root stem) and random components. The MD5 tag mimics the random components from a single source vendor. Using MD5 hash key value meets original intentional HBase key design for reduce hot-spot of table read/write. The DICOM files are distributed to all HBase regions, and we use an additional table to record the hierarchy structure of a scan dataset. We test using a random key, and MD5 hash of the data, as the key in HBase. With this comparison, we test the native capabilities of Hadoop and HBase without any of our proposed advances.

• **Scenario: "Custom Key/Default Split HBase" –** This scenario evaluates the custom key grouping and ordering of the DICOM file logically and physically in HBase by our custom key value prefix. When a HBase region exceeds a pre-defined size, we use the default split policy to split a region into two child regions without considering the key values of the split region as introduced in Section 4.2.3. In this case, the files belonging to the same project, subject, session, scan are distributed into two different regions. The two regions may move to different cluster nodes, and the replication of both regions may also be placed on random Hadoop datanodes. When retrieving all files of a cohort (i.e., a set of scan volumes) for further processing, a MapReduce job dispatches computations to nodes that contain the datasets of interest. When no single node contains all requested data for a single job (either due to a large request or local storage scarcity), the minimal necessary data will be retrieved over the network. So we test our proposed row key with the default RegionSplitPolicy.

• **Scenario: "Custom Key/Custom Split HBase" –** Our custom RegionSplitPolicy

has the capabilities to maximally collocate relevant data in the same group with the order of project, subject, session and scans. We test our complete design with our proposed row key and custom RegionSplitPolicy and compare it with Custom Key/Default Split HBase to see how data retrieval matters in MapReduce. Theoretically, this approach involves less data collection and movement over the network compared to the other two HBase methods and makes processing faster.

• **Scenario: "Grid Engine NAS" –** Traditional grid computing approaches separate data storage from computation. As a comparative method, we use a traditional Sun grid engine (SGE) to distribute portable bash script (PBS) jobs to computational nodes accessing data from a Network Attached Storage (NAS) device.

4.3.2 Hardware

We used a small, private research cloud for our experiments. Twelve physical machines were used consisting of 108 cores of AMD Operon 4184 processors, 40 cores of Intel Xeon E5-2630 processors and 8 cores of Intel Xeon W3550 processors running Ubuntu 14.04.1 LTS (64 bit). At least 2 GB RAM was available per core. In total, 190 GB of storage was allocated to HDFS and a Gigabit network connected all of machines. Local Disks type was Seagate ST32000542AS. Each machine was used as a Hadoop Datanode and HBase RegionServer for data locality. All machines were also configured using the Sun Grid Engine (Ubuntu Package: gridengine-* with a common master node). NAS was provided via CIFS using a Drobo 5N storage device (www.droboworks.com) with a 12 TB RAID6 array.

4.3.3 Data and Processing

To evaluate the test scenarios, 9,910,000 DICOM files from clinical CT scanners corresponding to 410 subjects and 8,120 scan volumes were retrieved in de-identified form under IRB approval from a study on traumatic brain injury. The processing system for each

scan applied a command line program to retrieve the data from storage (see test scenarios in Section 4.3.1) and convert the DICOM files to NiFTI using dcm2nii (www.nitrc.org\-/projects/dcm2nii/). We performed tests with subsets of data with different datasets to assess the scalability of each proposed system and relative overhead versus processing load. In Table 4.2, each dataset denotes the number of scans with average 126 DICOMs per scan.

Table 4.2: DICOM datasets size info

Datasets	**Total Scan size (GB)**
104	7.16
186	10.93
294	19.05
407	27.55
497	34.12
606	41
718	47.14
812	53.01
1624	106.02

Datasets	**Total Scan size (GB)**
2436	159.03
3248	212.04
4060	265.05
4872	318.06
5684	371.07
6496	424.08
7308	477.09
8120	530.1

4.3.4 Apache Hadoop/HBase Experimental Setup

The Hadoop/HBase cluster is configured by Hadoop (2.7.1), HBase (1.1.2) and Zookeeper (3.4.6). HDFS uses default 3 replicas with rack awareness. In our experimental setup, the Sun grid engine does the balancing and makes sure that the jobs ran as soon as space was available within the specified node list when processing is executed on a traditional grid [26]. For Hadoop scenarios, MapReduce is a programming model and an associated implementation for processing large datasets in the Hadoop ecosystem [4]. YARN is used for resource (CPU/Memory) allocation and MapReduce job scheduling [106]. We use the default YARN capacity FIFO (First in First Out) scheduler, which aims at maximizing the throughput of cluster with capacity guarantees when the cluster is being shared by multiple tenants.

The software tools to generate row keys from DICOM data were implemented in open source. The custom region split policy was implemented as a Hadoop extension class. All

software is made available in open source at NITRC project Hadoop for data collocation (http://www.nitrc.org/projects/hadoop_2016/). Manual inspection of region stores was used to verify data collocation under multiple configurations of Hadoop Datanodes to ensure that the desired data collocation and region splits were occurring.

4.3.4.1 MapReduce Setup for HBase Approach

The MapReduce model should complete two main tasks: data retrieval from HBase and data processing (DICOM to NIFTI conversion). The Map phase always tries to ensure that the computation tasks are dispatched where data resides, and such tasks are called data-local maps. A compromise scenario is when data is not on the local node where the running map task is located, but at least the data are on the same rack, and those maps are rack-local maps. In the Reduce phase, the output,$< key, value >$ pairs from Map phase are to be shuffled/sorted and sent to random cluster nodes.

If data retrieval is done in the Map phase while processing in the Reduce phase, then the computation and data can be on different nodes. A potential way to execute the processing is remote access (i.e.SSH) where data originally locates and applies processing. SSH limitation may occur, however, and block further connection, and as a result the processing cannot be fully completed. If both data retrieval and processing occurs in the Reduce phase, namely, each reduce task collects all row keys related with one scan volume, then downloads and collects DICOM files from HBase according to the keys, and finally executes the conversion in Reduce phase. In this way, network congestion occurs when the node that holds the Reduce task may not have all needed DICOM files. Since those DICOM files are aggregated in a same Region / node owing to proposed custom split policy, so Reduce task has to retrieve all datasets through the network leading to congestion.

Thus, these approaches break our main goal for data collocation with Hadoop and HBase with minimum data movement. As a result, for good use of our proposed Hadoop enhancements with data collocation in the context of the hierarchical key structure, the data

retrieval and processing occur in the Map phase and the Reduce phase is a no-op for our application.

In a traditional "word count" example, the input of MapReduce is a HDFS folder. The input folder is split into several pieces based on the files in the selected folder. Then each piece starts a map task with $< key, value >$ pair, the input Map Key is file names and input Map value is file content. However, this approach is not practical in HBase. The HBase region has a corresponding folder on HDFS, and all data stores/hfiles in this region are placed in the region. When the region collocates to a Hadoop datanode to achieve data locality, all data store/hfiles are compacted to a giant file, which means that a traditional MapReduce like wordcount strategy cannot split an input HBase folder for further processing.

Figure 4.3 shows the modified work-flow. HBase provides a default API for running HBase-oriented MapReduce. The input of the MapReduce is a HBase-scan, which represents an interval of consecutive table row key values of a selected column. The HBase-scan is split based on relevant regions, and the input $< key, value >$ pairs are values about row keys of a region and the content of the specified column. In short, if the input HBase-scan occurs across n regions, then only n map tasks are generated. The challenge for traditional HBase-oriented MapReduce for DICOM is that there are usually more than one datasets of DICOM files under the same scan in a region. So we refined the above approach to specify the input of MapReduce to be a selected cohort of scan volumes, and the number of Map tasks is based on the number of scans.

DICOM with same row-key prefix sticks together in order. Querying all DICOM images of a scan volumes does not need to iterate all input key values, however, we just need to define a search range (first/last row-key record of the selected cohort scan). Thus, we use an additional table to store the range of each scan volumes and do a one-time update once new images are uploaded to HBase. The Map Phase first retrieves the data from HBase and stores DICOM files to local node. Once done, it converts the DICOM files to NiFTI using dcm2nii as presented in Figure 4.3. For fair comparison between Hadoop methods

and approach on NAS, additional steps such as uploading the NiFTI result to HBase are not launched.

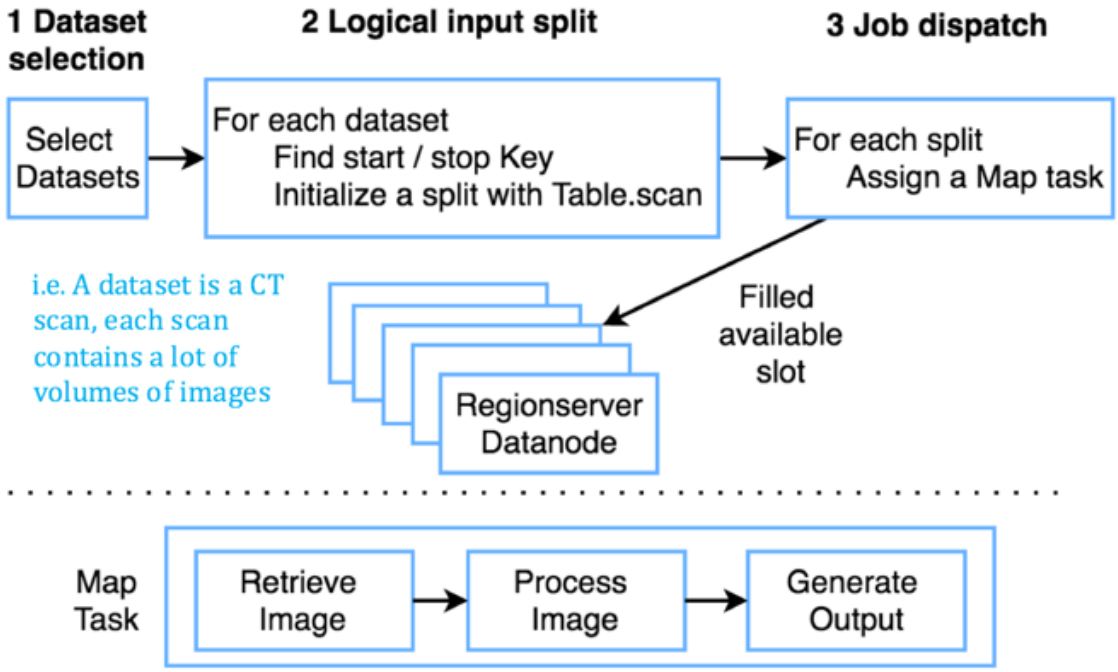


Figure 4.3: Custom HBase oriented MapReduce basing on input selected groups of scan volumes

4.3.4.2 Guidelines used for Scaling Hadoop / HBase Cluster

Scalability is one of the most important properties for Cloud usage. We test and scale our clusters for studying intrinsic scalability performance. The following summarizes how we scaled the Hadoop / HBase cluster step by step.

- For scaling down, the RegionServer should first be gracefully stopped [3], after which the relationship of data collocation between Datanode and RegionServer no longer exists. Then major compaction on the affected data from the stopped RegionServer must be applied to collocate to the rest of the cluster [3]. When all data-locality is achieved again, decommissioning the Datanode and re-balancing of the cluster is performed. If decommission order is reversed, redundant replications are to be stored

into HDFS which exponentially decreases the available size of the Hadoop cluster.

- For scaling up, a new Hadoop Datanode must be commissioned first and then a new HBase RegionServer is added, followed by a major compaction to achieve data locality. If there is no Datanode, adding a new RegionServer can collocate to nothing, which makes reverse commissioning order of no sense.

4.3.5 Results of Data Transfer Latency

First, we evaluate the latency in retrieving imaging data in each of the four scenarios. Table 4.3 shows average latency for all datasets. For naïve Hadoop, we retrieved data to a random node since the data were not collocated. For custom key / standard split, we retrieved the data to the machine which contained the first element in the scan. For custom key / custom split, we retrieved the data to the machine where the data were located entirely. For Grid Engine NAS, we retrieved the data from the NAS to a local machine serially (i.e., with one core in use).

Table 4.3: Latency results in seconds for each of the four test scenarios.

Approach	**Grid Engine NAS**	**Naive HBase**	**Custom key/ Standard split HBase**	**Custom key/ Custom split HBase**
Latency(s)	4.76	19.02	3.29	2.56

The naïve Hadoop strategy performed markedly worse than the other methods because it needs to open and close connections with multiple other machines in order to download the data, and the initialization and setup of each ZooKeeper connection involves overhead. Using the NAS with a single connection is relatively effective since the data are coming from one fixed location and there is low overhead in opening and closing connections. In comparing the default split policy to our proposed policy, we see an improvement in

average performance. Any increase comes from the cases where scans are split between machines and thus data needs to be retrieved from other locations on the network.

4.3.6 Data Processing Throughput for DICOM to NiFTI Conversion

Each of the four scenarios executed a DICOM to NiFTI conversion as described in Section 4.3.3. Figure 4.4-A presents an analysis of throughput. The Grid Engine NAS performed the worst (fewest datasets per minute, longest run times) across all dataset sizes. In all scenarios, the NAS device saturated at 20 MB/s (approximately 18 datasets per minute) throughput despite the gigabit network access. This was likely due to numerous small files that are generated with "classic" DICOM scanning as direct read/write to the NAS device demonstrated substantively higher performance.

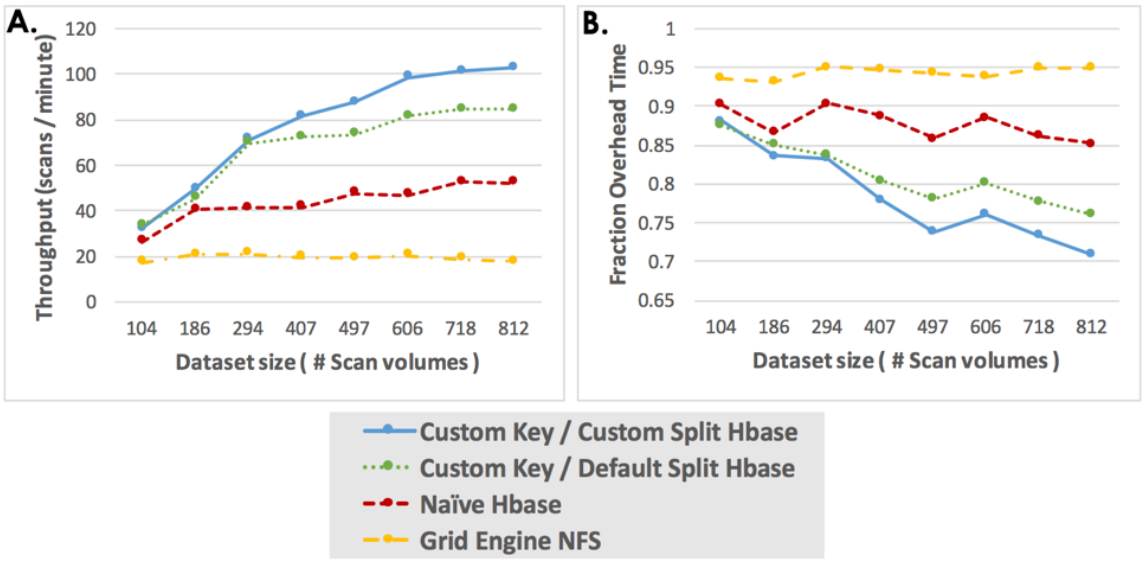


Figure 4.4: Throughput analysis for each of the test scenarios. (A) presents the number of datasets processed per minute by each of the scenarios as a function of the number of datasets selected for processing. (B) shows the fraction of time spent on overhead relative to the number of datasets.

The naïve HBase approach scaled better than the NAS approach with a throughput ranging from 31 MB/s (with 104 datasets) to 58 MB/s (with 718-812 datasets). The performance leveled off at 52 datasets/minute for a factor of almost three-fold improvement

over NFS. The custom key / default policy HBase approach performed even better with a throughput of 34 MB/s (with 104 datasets) to 94 MB/s (with 718-812 datasets). The custom key / custom policy HBase approach further improved throughput performance from 37 MB/s (with 104 datasets) to 114 MB/s (with 812 datasets).

The naïve method's performance improves flatly because of uncertainty in the placement of data loading. It performs better than processing on the NAS device because not all data needs to be retrieved from other nodes; some of the files are placed on the same node with Map computation in most cases. On the other hand, the custom key / custom policy HBase involves smaller data movement with better performance rather than the custom key / default policy HBase, both of whose processing are executed within most data-local map and a few rack-local map according to YARN allocation.

4.3.6.1 Throughput Upper-bound

Figure 4.4-A illustrates the processing on NAS device, which saturates the Gigabit network. The HBase approach does not incur as much network congestion because most map tasks are data-local or rack-local. Thus, we were not able to observe any perceived network-imposed limitations even until 812 datasets. The upper limit on the throughput stems from other overheads in the framework, which we address in this chapter. This is further verified in Section 4.3.7.

Consequently, to identify the upper limit on the throughput of our system, we tested our system for more number of datasets. Figure 4.5 presents the result of processing scans per minute with more datasets according to Table 4.2. Our proposed method performs better than custom key / default policy HBase initially (with 407–4,060 datasets, 34.12–265.05 GB, respectively), and it reaches its throughput upper bound with 124.5 MB/s (with 4,060 datasets). Thereafter, both approaches perform basically the same. The throughput upper bound for custom key / default policy HBase is 120.5 MB/s (with 4,872 datasets). Since the network is not a factor, we conclude that to obtain even higher throughput, we will need to

scale the hardware by adding more cores since the number of cores is the limitation factor.

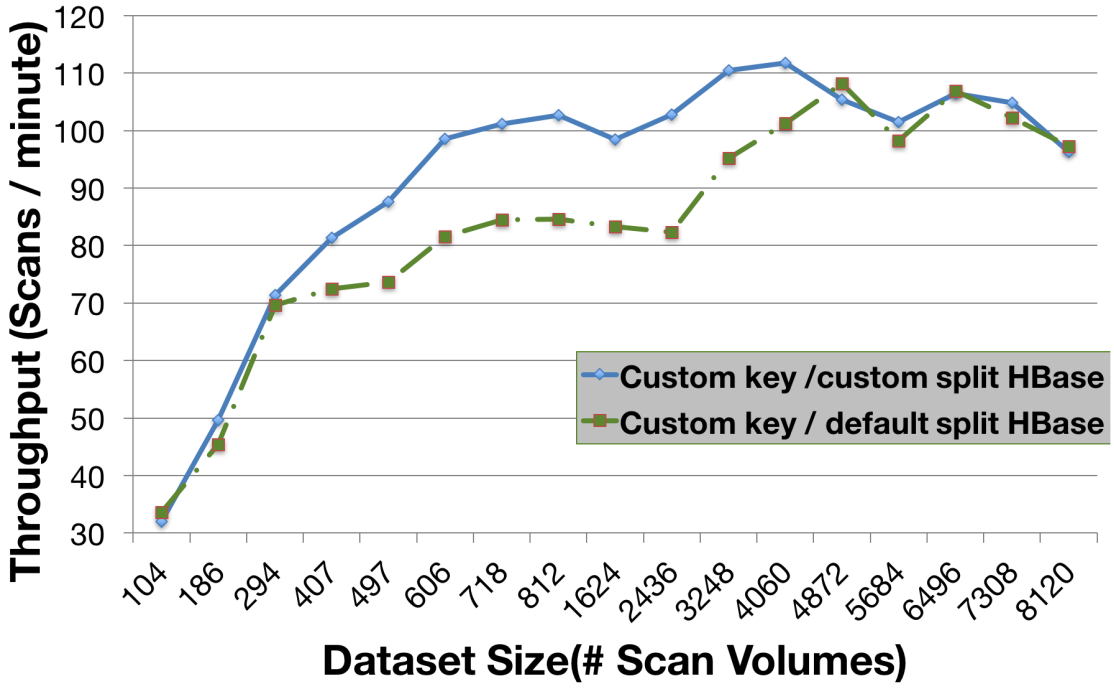


Figure 4.5: Throughput analysis for finding upper-bound of scenarios custom key / default policy HBase and custom key / custom policy HBase

4.3.6.2 Overhead Considerations with the Hadoop Framework

The computing grid had 156 cores available. Therefore, up to 156 jobs could run simultaneously in any of the test scenarios. With the three Hadoop scenarios, we have logs of both the time spent within each job on the compute node (including time to establish data connection, retrieve the data, and clean up the connection) and the actual wall time. For each of the Hadoop scenarios, we computed the average actual time spent executing the processing (including data retrieval), which ranged from 22 s to 35 s. For each of the data submission tasks, we can identify the minimum number of jobs that would need to run in serial by dividing the number of scan volumes by the number of cores. The fastest time that the Hadoop scheduler could run the jobs is the length of the serial queue times the job length, but in all cases the actual wall time exceeded this value. We define the overhead

time as the difference between the actual wall time and the theoretical minimum time.

The ratio of overhead time to total time is shown in Figure 4.4-B. Fitting a linear analysis to each of the three scenarios shows that the naïve HBase strategy had a marginal penalty of 1,003 ms per additional dataset. The custom key / default split policy reduced the overhead penalty to 547 ms per additional dataset. Finally, the custom key / custom split policy resulted in 398 ms per additional dataset.

4.3.6.3 Overhead Lower-bound

Similar to Section 4.3.6, the overhead of Hadoop scenarios have not reached the bottom line within 812 datasets. Figure 4.6 shows ratio of overhead in total processing time with multi-datasets. Both overhead values linearly decrease and then become steady at 33%. Finally, the Custom key / default split policy reaches the lower-bound in 159 ms and the Custom key / default split policy performs a bit better with 138 ms. When the size of datasets is small, the time for establishing data connection, retrieving the data, and cleaning up the connection dominates total time compared with data processing. When the framework reaches the upper limit of cluster cores treating processing capability, the overhead is balanced.

4.3.7 Evaluating the Scalability of the Framework

We wanted to understand how does the scale of the cluster impact performance. Thus, we experimented by linearly decreasing the size of the cluster and observe if the performance decreased in similar manner. In our experiments, each machine acted as a Hadoop Datanode and HBase RegionServer for data locality as introduced in Section 4.3.2. The order of decommisioning of Datanode and RegionServer is important when scaling the size of the cluster. Custom key with default and custom split policy are compared on scaled cluster (5-10 Hadoop/HBase nodes). Decreasing the size of the cluster can linearly increase the total time with processing 5,684 datasets, which is presented in the trends of Figure 4.7.

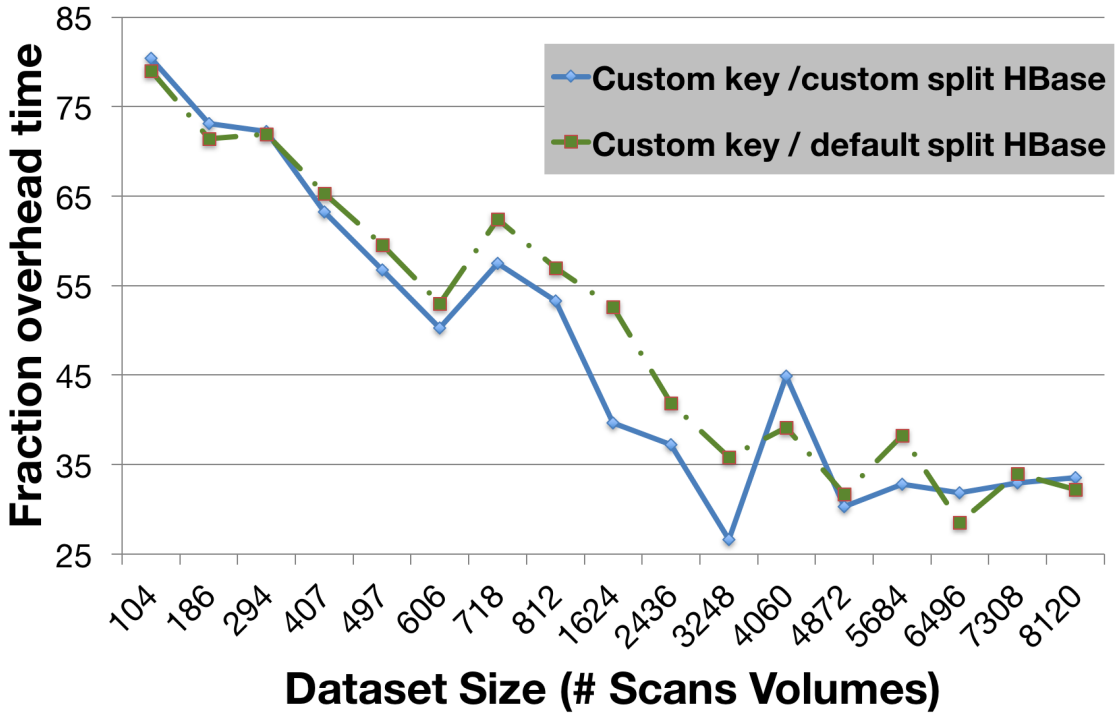


Figure 4.6: The fraction of time spent on overhead relative to the number of datasets for finding lower-bound of scenarios custom key / default policy HBase and custom key / custom policy HBase

Based on the previous discussion, we can conclude that the Hadoop scenario performance is not limited by the network bandwidth but by the total available CPU cores and memory. Thus, scaling up the size of cluster can increase high performance computing capability for medical imaging processing in an affordable local/cloud-based commodity grid.

4.4 Conclusions

Billions of magnetic resonance imaging (MRI) and computed tomography (CT) images on millions of individuals are currently stored in radiology archives [107]. These imaging data files are estimated to constitute one-third of the global storage demand [108], but are effectively trapped on storage media. The medical image computing community has heavily invested in algorithms, software, and expertise in technologies that assume that imag-

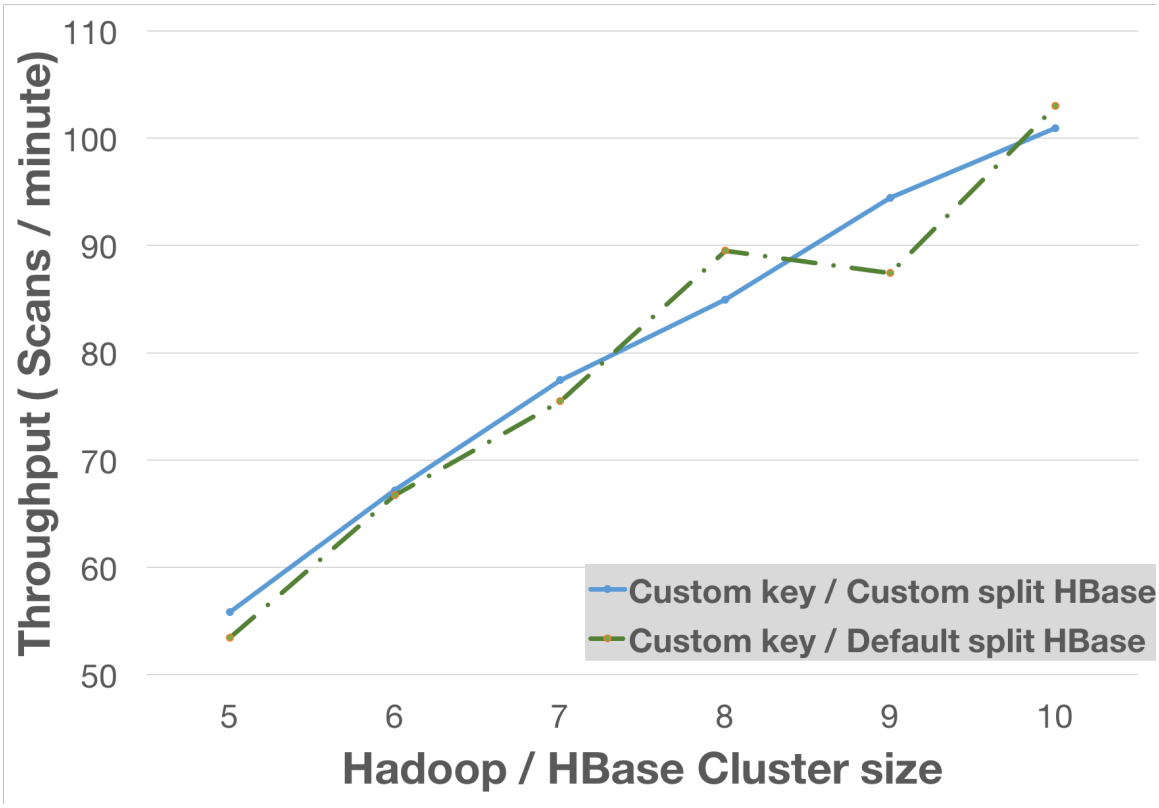


Figure 4.7: Throughput analysis for Hadoop scenarios with different size of cluster

ing volumes can be accessed in their entirety as needed (and without substantial penalty). Despite the promise of big data, traditional MapReduce and distributed machine learning frameworks (e.g., Apache Spark) are not often considered appropriate for "traditional" / "simple" parallelization.

4.4.1 Research Summary and Discussions

In this chapter we demonstrate that Apache Hadoop MapReduce can be used in place of a PBS cluster (e.g., Sun Grid Engine) and can be offered as a cloud-based service. Moreover, with our approach even a naïve application of HBase results in improved performance over NAS using the same computation and network infrastructure.

We present a row key architecture that mirrors the commonly applied Project / Subject / Session / Scan hierarchy in medical imaging. This row key architecture improves

throughput by 60% and reduces latency by 577% over the naïve approach. The custom split policy strongly enforces data collocation to further increase throughput by 21% and reduce latency by 29%. With these innovations, Apache Hadoop and HBase can readily be deployed as a service using commodity networks to address the needs of high throughput medical image computing.

Our experiments promote a general framework for medical imaging processing (e.g., structured data retrieval, access to locally installed binary executables/system resources, structured data storage) without comingling idiosyncratic issues related to image processing (e.g., parameter settings for local tissue models, smoothing kernels for denoising, options for image registration). DICOM2NiFTI is routine first step in processing and often a bottleneck for quality control on large datasets. Hence, the application demonstrates the system's correctness and scalablity across a complex organization of files.

The system was implemented on a small, private data center, which includes the Sun Grid Engine. As the number of machines increases, NFS becomes nonviable with a single host, and distributed storage (e.g., GPFS) is commonly used on large clusters with 10+ Gbps networks. The proposed data and computation co-location solution is an alternative and could scale to well-more CPU-cores than beyond a GPFS solution on the same underlying network.

Finally, as implied by the trends in Figure-4.4, the benefits of distributing computation with storage increase with larger datasets. Exploration of the asymptotic performance limits is of great interest, but beyond the scope of this chapter that illustrates meaningful gains on problems of widely applicable scale. The optimization of characterization of these approaches on heterogeneous grid is an area of great possibility. In particular, the Apache Hadoop YARN scheduler could be further optimized to exploit intrinsic relationships in medical imaging data.

4.4.2 Broader Applicability of our Approach

The approach was designed to provide a general framework for medical imaging processing. We have just submitted a work that presents theoretical models for estimating wall-clock time and resource time for the proposed Hadoop/HBase framework to answer when it will be relevant and non-relevant for medical imaging [105]. The models are empirically verified and can be applied to other domains if the necessary parameters, i.e. dataset size, job length, cluster setup, network environment etc, are supplied.

Thus, we envision a number of application domains that may benefit from our work in this chapter. For example, multimedia processing always involves "large" data set compared with medical imaging data. For instance, video transcoding has been applied on Hadoop, each video is about 200 MB, and experiment data sizes are from 1 - 50 GB [109, 110]. Based on video record time and content, we can easily create a hierarchy category to name the video, and do the group processing when omitting the reduce phase. The authors also discussed potential use of Hadoop/HBase for large image and video processing such as Discrete Fourier Transform (DFT), face detection, and template matching [111].

Gene data have many different styles with diverse attributes. Genes with similar expression patterns must be collocated for group analysis since genes that behave similarly might have a coordinated transcriptional response, possibly inferring a common function or regulatory elements [112]. Thus, genes data group/hierarchy storage, retrieval and analysis is applicable by our framework.

Another scenario where our work is applicable includes Satellite data/image processing on data about earth surface, weather, climate, geographic areas, vegetation, and natural phenomenon [113], which can be studied according to region-based, day-based, multiple-day-based, or seasonal-based [114]. As a result, time-oriented hierarchical structure can help group the data from the satellite for further processing. Similarly, Internet of things collect data from various facilities like sensors. According to the sensors' supervision

area, a component hierarchy-based data collection can be implemented. For instance, high-speed train fault and repair prediction is applied before a train runs [115]. Analyzing mass historical data from a group of Electric Multiple Unit (EMU) of a train's components has potential to be implemented in our framework.

The work in this chapter presented in this chapter is available in open source at www.nitrc.org/projects/hadoop_2016.

Chapter 5

THEORETICAL AND EMPIRICAL COMPARISON OF BIG DATA IMAGE PROCESSING WITH APACHE HADOOP AND SUN GRID ENGINE

5.1 Problem Overview

As imaging datasets and computing grid sizes grow larger, traditional computing's separation of data and computational nodes creates a problem. Moving data from where it is centrally stored to computational nodes can saturate a network with relatively few active processes. Under certain conditions, the bottleneck in the computing architecture becomes the network bandwidth. An inexpensive solution is to locate the data on the computational nodes to avoid the problem of saturating the network by copying data. This is already implemented by some big data architectures, e.g., Apache Hadoop [19, 116, 117, 33, 118]. Previously, DICOM to NiFTI conversion had been identified as an area where significant gains in scalability could be realized by using big data frameworks. In chapter 4, we use HBase that is built upon Hadoop to logically and physically sort the data by indexed row keys. We propose a novel data allocation policy within HBase to strongly enforce collocation of hierarchically row key for storing slice-wise DICOM data. In this way, a group-wise DICOM retrieval occurs locally without involving the network [3]. However, this creates new questions, e.g.: when does this novel Hadoop/HBase framework perform better than traditional high performance computing clusters like Sun grid engine (SGE) [26]? In this case, there are many parameters of concern, such as the cluster size, machine cores, node memory, distribution of resources, image processing job, etc. [30, 29, 28, 27]. This work develops theoretical models to characterize the performance of SGE and Hadoop and verify the models empirically. The theoretical models have two parts. The first is wall-clock time, which represents the total time as experienced by the user. The second part is resource

time, which measures elapsed time on each node when a process starts across all nodes. The models are further verified based on a real lab-based cluster environment focusing on custom image processing.

This work has been published in SPIE medical imaging 2017 [105] [119].

5.2 Methods

5.2.1 Computation modules

Hadoop and HBase is to enable co-location of data storage and computation while minimizing data transfer, while SGE separates data storage from computation. Figure 5.1 summarizes both methods' working flows. It is worth mentioning that there are mainly two kinds of map jobs in Hadoop [4]. A data-local map involves local data within a node. However, it is notable that if no single node contains all requested data for a single job (due to a large request or local storage scarity), the minimal necessary data will be retrieved over the internet. A rack-local or non-local map will retrieve data through other data nodes. Ideally only data-local maps will occur. In reality, around 5% of maps tend to be rack-local as evidenced by previous DICOM to NiFTI conversion experiments, which we had trained on a data / core balanced cluster based on chapter 4. The balanced means data is distributed equally to every machine, and every machine have the same number of cores. In the rest of the chapter, we use Hadoop to simplify representing our novel Hadoop/HBase framework.

5.2.2 Theoretical model

We summarize the modeled parameters that affect both wall-clock time and resource time in Table 5.1.

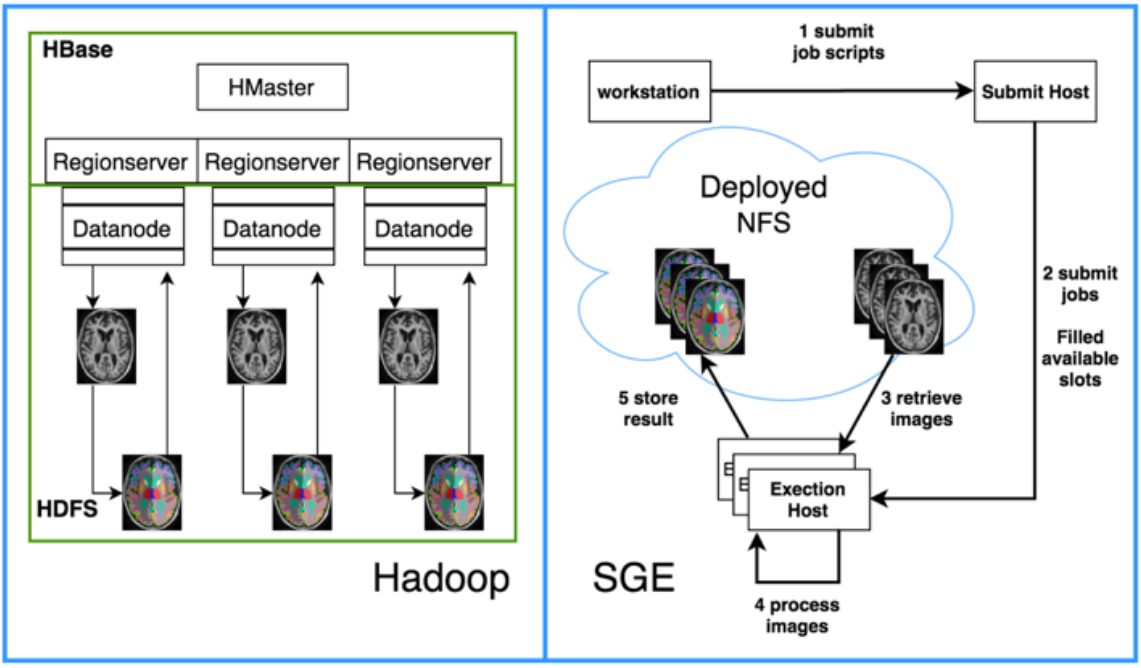


Figure 5.1: Hadoop and SGE data retrieval, processing and storage working flow basing on Multi-atlas CRUISE (MaCRUISE) segmentation [1, 2]. The data in an HBase table is approximately balanced to each Regionserver. The Regionserver collocates with a Hadoop Datanode to fully utilize the data collocation and locality [3]. We design our proposed computation models using only the map phase of Hadoop's MapReduce [4]. In this phase, the data is retrieved locally; if the result were moved to reduce phase, more data movement would occur, because the reduce phase does not ensure process local data. Within the map phase, all necessary data is retrieved and saved on a local directory and gets furtherly processed by locally installed binary executables command-line program. After that, the results of processing are uploaded back to HBase. For SGE, the user submits a batch of jobs to a submit host, and this host dispatches the job to execution hosts. Each execution host retrieves the data within a shared NFS and stores the result back to the NFS.

5.2.2.1 Wall-Clock Time

Wall-clock time is what the user sees and experiences. Equation (1) is an overview wall-clock time model basing on Figure 5.1. It contains three types of I/O: data retrieval, processing, storage. For SGE, data is loaded from and stored to network storage, so VsourceR and VsourceW is related with average bandwidth. All data I/O in execution host occur in the pre-allocated memory for the job, so VhostR and VhostW can be ignored compared with bandwidth. For Hadoop, the worst case of input data retrieval from HBase and out-

put data storage to HBase directly involves local hard disk [3]. When the input data is processed on a host, it is temporally saved on a place on hard disk, and the output also generated on a temp place of local drive. Thus, VsourceR, VsourceW, VhostR and VhostW are all related with the local disk reading/writing speed (VdiskR/VdiskW).

$$T_{wc} = \#round \cdot [(\frac{data_{in}}{V_{sourceR}} + \frac{data_{in}}{V_{hostW}}) + (\frac{data_{in}}{V_{hostR}} + \frac{data_{out}}{V_{hostW}}) + (\frac{data_{out}}{V_{hostR}} + \frac{data_{out}}{V_{sourceW}}) + T_j] + \eta \tag{5.1}$$

Wall-clock time summary Equation 5.2 is a summarized model for wall-clock time. $T_{wc\_network}$ is the time for jobs to load data from the network for both SGE and Hadoop scenarios. T_{wc_local} is for jobs that load data locally, and it only serves for the Hadoop scenario.

$$T_{wc} = \max(\alpha \cdot T_{wc\_local}, T_{wc\_network}) + \eta, Hadoop : \alpha = 1; SGE : \alpha = 0 \tag{5.2}$$

Wall-clock time for jobs only loading data locally T_{wc_local} Firstly, we make an assumption for the model T_{wc_local} that if the Hadoop map task is a data-local map task, all data retrieval/storing are happened locally, and the minimal necessary data movement from other nodes over network due to large request are ignored in this model. Therefore, #Round is determined by data and core allocation of the cluster - i.e., if the cluster is a core-balanced, the wall-clock time of all tasks is defined by the machine with the most data as Figure 5.2 illustrates. If the cluster is core-unbalanced, we need to take into consideration the ratio between number of jobs will be dispatched for each machine and the cores on that machine as in equation 5.3 presented. For the number of jobs per machine, according to the input dataset's row key in HBase, we can know the dataset belongs to which region of HBase table. The place of region stands for the node that the job will be run [3]. Then for each machine, once we find the maximum ratio of $\frac{job_i}{core_i}$, the short plate machine will decide the value of #Round.

Table 5.1: Theoretical model parameter definition

I/Ospeed ($V_{sourceR}$, VsourceW, VhostR, VhostW)	I/O speed is the data read/write rate on source where data stores and retrieves, and on host where data get processed. Source for SGE is the Network storage accessed by NFS. Source for Hadoop is the hard disk allocated for HBase.
Bandwidth(B, Breal)	B is the bandwidth ofcluster. In our case, it is based on a gigabit network. Breal is the real bandwidth that is shared by one job.Once there is a network congestion, the value of Breal is actually smaller than or equal to the cluster's given fixed bandwidth B.
Diskspeed (VdiskR, VdiskW)	Dataread/ write speed of local hard drive.
Input/output dataset (datain / dataout)	Thetotal data size for one job that is downloaded/uploaded while processing.
Core (#core, #allowed_core, corei)	#core is the number of processorcores in the cluster. #allowed_coreis the maximum number of allowed concurrent cores can be used without causing network congestion. We use coreito represent the number of individual cores on each machine.
Job(#job, jobi)	#job is the total number ofjobs. jobi is the number of jobs will be dispatched to each machine.
Jobtime per dataset (Tj)	Processingtime per dataset.
Round(#Round)	Thetotal number of round for parallel jobs.
Region(#region, regioni)	Eachpart of a Hbase table is split into several regions (blocks of data).The totalnumber of regions # region per Regionserveris approximately balanced [3]. Region_{i} denotes the number of region onspecified machine.
Machine(#machine)	Thenumber of machines (workstations) of the cluster.
File(#file)	Allfiles that are involved in the whole process.
Negligibleoverhead (η)	An insignificant overhead that could be eliminated and not counted
α	\alpha is a binary parameter. It helps the equationsexplain ' when do they only valid for Hadoop scenario (α =1) rather than SGE scenario(α =0.
β	\beta is a binary parameter. It helps the equationsexplain when do they only valid for SGE scenario (β=1) rather than Hadoopscenario (β=0).
γ	γ is an experimental empirical parameter to represent the ratio of rack-local map task for Hadoop scenario, namely the data is loaded/stored via network. Similarly, the value of γ is always 1 for SGE scenario since all data is transferred through network.

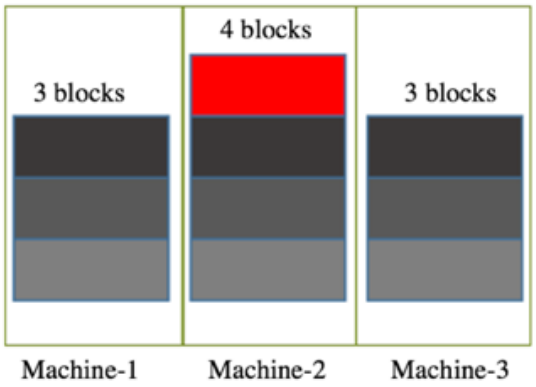


Figure 5.2: Assume there are 10 jobs are ready to process. Each block represents the input dataset for each jobs. Three machines are all have same number of cores. If there is no data transfer, all data processing time is defined by the medium's processors, if all datasets have the same processing speed.

$$T_{wc\_local} = \lceil \max_{i=1}^{\#machine} \frac{(1-\gamma)\cdot job_i}{core_i} \rceil \cdot (T_j + \frac{2data\_in + data\_out}{V_{diskR}} + \frac{data_{in} + 2data_{out}}{V_{diskW}}) + \eta, \gamma = 0.05 \tag{5.3}$$

If we need to process all HBase table's data, we can easily use the distribution of table's regions to find the number of local jobs for each machine as equation 5.4 demonstrated.

$$job_i = \frac{region_i}{\#region} \cdot \#region \in [1, \#file] \tag{5.4}$$

Wall-clock time for job containing data network transfer $T_{wc\_network}$ As equation 5.5 presented, all data in SGE is transferred through the network, and jobs are equally distributed to the free cores of machines, namely jobs do not care where data is. Hence the value of #Round for SGE is simply decided by $\lceil \frac{\#job}{\#core} \rceil$. Ideally, there is no data transfer in the Hadoop MapReduce approach. However, as mentioned in section 2.1, there are about 5% rack-local maps in our trained experiment so that it also involves network transfer. And for Hadoop, disk reading / writing speed should be considered in the time model due to the data retrieval, processing and storage working flow.

$$T_{wc\_network} = \beta \lceil \frac{\#job}{\#core} \rceil \cdot (T_j + \frac{data_{in} + data_{out}}{B_{real}}) + \alpha \lceil max_{i=1}^{\#machine} \frac{\gamma \cdot job_i}{core\_i} \rceil \cdot$$
$$[T_j + \frac{data_{in} + data_{out}}{B_{real}} + (\frac{2data_{in} + data_{out}}{V_{diskR}} + \frac{data_{in} + 2data_{out}}{V_{diskW}})], \quad (5.5)$$
$$Hadoop : \alpha = 1, \beta = 0, \gamma = 0.05; SGE : \alpha = 0; \beta = 1, \gamma = 1$$

Network saturation release point Under a fixed bandwidth, data traveling through a network can affect the number of running cores. We made an assumption that the value of allowed concurrent running cores $\#allowed\_core$ without arising network congestion is showed in equation 5.6. If the total number of running cores $\#core$ that a cluster can provide is more than $\#allowed\_core$, heavier data loading may cause network saturation and make real bandwidth B_{real} for each job smaller than or equal to the given fixed cluster bandwidth B. Under network congestion circumstances, job completion time gets delayed because job has to spend more time waiting for data movement by network I/O. The relationship of average B_{real} of all jobs and cluster's B within one round parallel job processing cycle is introduced in equation 5.7. Moreover, we define the network saturation release point (NSRP) is at the point when $\#allowed\_core$ equals to #core. Equation 5.8 summarizes the Breal before and after a NSRP.

$$\#allowed\_core = \frac{B}{\gamma^{\frac{data_{in} + data\_out}{T_j}}}, Hadoop : \gamma = 0.05; SGE : \gamma = 1 \quad (5.6)$$

$$B = \frac{\#core \cdot \gamma \cdot (data_{in} + data_{out})}{T_j + \frac{data_{in} + data_{out}}{B_{real}} + \alpha(\frac{2data_{in} + data_{out}}{V_{diskR}} + \frac{data_{in} + data_{2out}}{V_{diskW}})}, \quad (5.7)$$
$$Hadoop : \alpha = 1, \gamma = 0.05; SGE : \alpha = 0, \gamma = 1$$

$$B_{real} = \begin{cases} \min(B, \frac{B \cdot (data_{in} + data_{out})}{\#core \cdot \gamma \cdot (data_{in} + data_{out} - B \cdot T_j - \alpha \cdot B \cdot (\frac{2data_{in} + data_{out}}{V_{diskR}} + \frac{data_{in} + 2data_{out}}{V_{diskW}}))}), \\ \#allowed\_core < \#core \\ B, \\ Hadoop : \alpha - 1, \gamma = 0.05; SGE : \alpha = 0, \gamma = 1 \end{cases} \tag{5.8}$$

5.2.2.2 Resource Time

Resource time is time elapsed time on each node when a process starts across all nodes as displayed in equation 5.9. For SGE, the resource time is decided by the sum of all job's processing time and data transfer through network. B_{real} may affect the resource time of SGE since when network saturation occurs, the core has to wait for data being loaded to the node. Ideally, there is few data movements for Hadoop/HBase, except the small proportion for rack-local maps. The resource time usage for Hadoop is determined by total job time, data retrieval via the network in rack-local map and local disk reading/writing.

$$T_{resource} = [T_j + \frac{\gamma \cdot (data_{in} + data_{ut})}{B_{real}} + \alpha \cdot (\frac{2data_{in} + data_{out}}{V_diskR} + \frac{data_{in} + 2data_{out}}{V_diskW})], \\ Hadoop : \alpha = 1, \gamma = 0.05; SGE : \alpha = 0, \gamma = 1 \tag{5.9}$$

5.2.3 Experiment Design

Three parallel experiment environments are setup for both Hadoop and SGE. The experiment design does not aim to share the benefit of processing group hierarchical related imaging data similar in [102]. However, our goal is to verify the behavior of our Hadoop scenario that maximize the localization of data retrieval/processing/storage for a job. We make the experiment simpler that compressing 3,310 T1 images to the .gz format. Each

job compresses only one NiFTI image with 2GB memory available and generate one compressed images. All T1 images for Hadoop scenario are saved into a newly created HBase table. We estimate achievable empirical average bandwidth as 70 Mb/second; disk read speed as 100 Mb/second with write speed as 65 Mb/second. The total input size of the images is 70.7 GB and the processing generates 21.5 GB of compressed files as output. To explore the impacts of processing time, we manually increase the processing time by adding a sleep function without any data retrieval to make the job length of the experiment take an additional 15 105 seconds on a fixed dataset to mimic different job processing speed. Also, we vary cluster size to assess the scalability of SGE and Hadoop cluster from 6 21 machines. Table 5.2 presents the detail of the experiment setup. Each machine was used as a Hadoop Datanode and HBase RegionServer for data locality [3]. All machines were also configured using SGE. There is an additional Machine for both methods serving as cluster master.

Our goals are to empirically verify 1) if each of the scenarios can match the $\frac{wall-clock}{resource}$ time theoretical models basing on estimated network saturation release point NSRP; 2) test if the cluster can present a scalable performance, i.e., when SGE has more cores, the saturation length will be longer than seen with less cores with the increasing of data processing time per job (T_j), 3) how balanced/unbalanced data and core allocation can affect both computing architectures.

5.2.4 Datasets

The experiment uses 3,310 T1 images retrieved from a secure, shared web database application for MRI data that was gathered from healthy subjects/volunteers and subjects/volunteers with ADHD; the Tennessee Twin Study based on psychopathology risk and its subjects constitute a portion of the neuroimaging sub-study of the Baltimore Longitudinal Study on Aging.

Table 5.2: Hadoop v.s. SGE experiment cluster setup with same memory allocation and fixed datasets.

Core allocation	**Hadoop data allocation**	**Estimated NSRP**	**Experiment type**
209 cores, (3 machines with 32 cores, 10 machines with 3 cores, 7 machines with 12 cores, 1 machine with 11 cores).	43 regions (20 machines with 2 regions, 1 machine with 32 cores has 3 regions).	Until Tj reaches 85 s.	T1 NiFTI image compression (5s), and manually increase the processing time by adding a sleep function (10 s, 25 s, 40 s, 55 s, 70 s, 85 s, 100 s, 115 s respectively)
132 cores balanced on 11 machines (12 cores/machine).	34 regions (10 machines with 3 regions, 1 machine with 4 regions).	Until Tj reaches 54 s	
72 cores balanced on 6 machines (12 cores/machines).	11 regions (5 machines with 2 regions, 1 machines with 1 regions).	Until Tj reaches 29 s	

5.3 Results

Figure 5.3& 5.4 presents the verification result for Hadoop and SGE on wall clock time. The wall-clock time usage for SGE is a bit over what would be expected by theoretical model and could be explained by non-modeled overhead in SGE, such as job dispatching. When cores increase under the fixed number of jobs and size of datasets, the network saturation persists longer for SGE, and the wall clock time is limited because of data transfer. The turning points (NSRP) for SGE match the theoretical point when allowed maximum running cores without causing network saturation is larger than the cluster's cores, which can also be verified by result (when dataset processing time is 30 s for SGE 72 cores scenario, 60 s for SGE 132 cores scenario and 90 s for SGE 209 cores). When the cluster has 132 cores, Hadoop's time becomes gradually longer than SGE; the reason is that data in HBase is not perfectly distributed. According to the experiment setup in Table 2, the total processing time is decided by the node which has one more region. Our model can also predict Hadoop 72 cores scenario since the data / core allocation is approximately balanced. Hadoop 209 cores result is more complex and does not perfectly match theoretical model. Neither core nor data allocation for this scenario are balanced. Through the result, we can see that when most jobs are running on the limited machines, Hadoop randomly dispatches some jobs to idle cores, thus the final wall-clock time is smaller than theoretical models suggest. The final number of rack-local maps for Hadoop 209 cores is almost 30%, which is much larger than our previously observed model parameter (5%) based on a balanced cluster. On the other hand, SGE balanced the job and data distribution; namely, the average number of running cores is greater than seen with Hadoop.

For Figure 5.4, because only little data movement was allowed, Hadoop with 72 and 132 cores falls on the theoretical ideal resource usage line much faster than SGE with the same number of cores. Even on an unbalanced cluster, Hadoop with 209 cores can also match the theoretical trend. Variation in available network bandwidth is a potential explanation for the 132 core SGE not matching the theory-derived expected result for the 5-

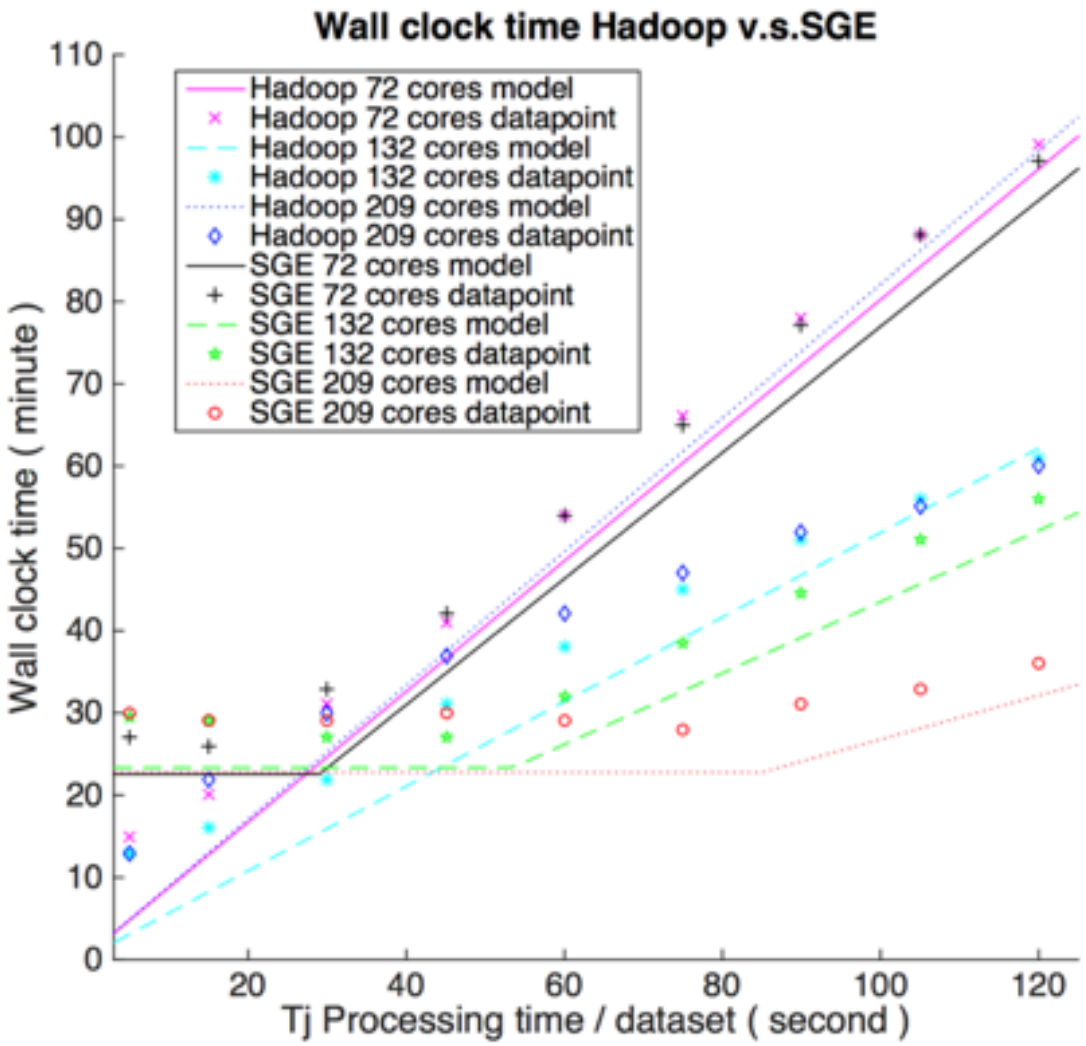


Figure 5.3: Wall-clock time performance for Hadoop and SGE with different cores.

45 seconds/dataset range. The result reveals the real average bandwidth Breal is faster than assumed on 70 Mb/s in section 2.3. That the SGE cluster with more cores was saturated longer also matches theoretical models, and once the dataset processing time is greater than NSRP, all three SGE scenarios fall on the Hadoop model.

5.4 Conclusion

The theoretical model indicates a trend of wall-clock time spent for a particular image processing job. For instance, converting a T1 NiFTI image to MNI using Aladin registration

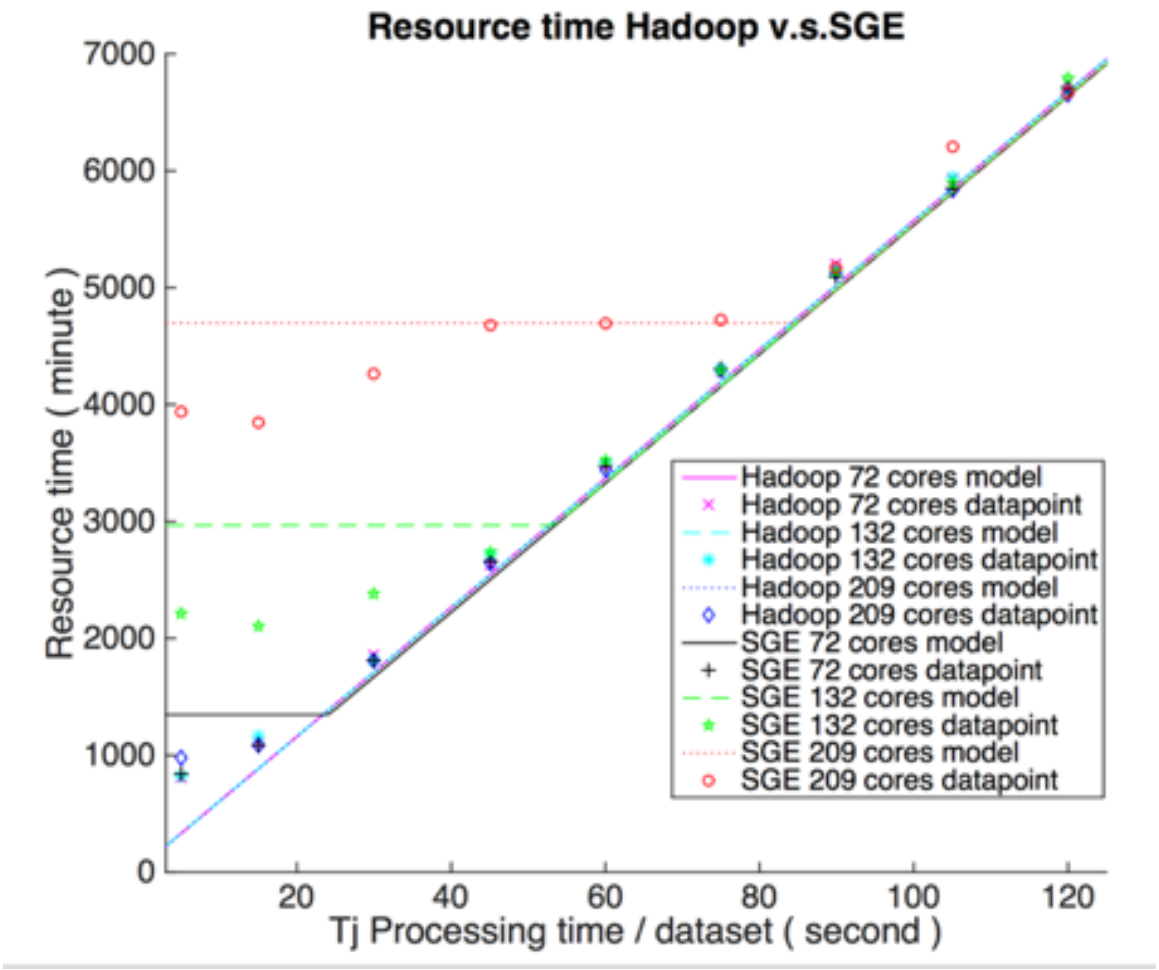


Figure 5.4: Resource time performance for Hadoop and SGE with different cores.

takes at least 2 minutes, so about 400 jobs can run concurrently before network saturation occurs. Thus, SGE cannot saturate the network under the experiments introduced in section 2. The theoretical model also conveys multiple relationships among job numbers, network environment and cluster setup. In Figure 5.5, the common logarithm ratio (log10) ratio for wall-clock and resource time performance transition of Hadoop's divide SGE's, and the log10 ratio is get scales in range from [-1,1].

We assume there are 5000 jobs as input. As Figure 5.5 (A) and (B) represents the ratio when core/data allocation are balanced (the ideal scenario for Hadoop), which are based on an assumed lab-based cluster (20 machines, 300 cores, gigabit bandwidth (70 Mb/s), hard drive (Read 100 Mb/s, Write 65 Mb/s), 2GB memory for each jobs). And the ratio reveals

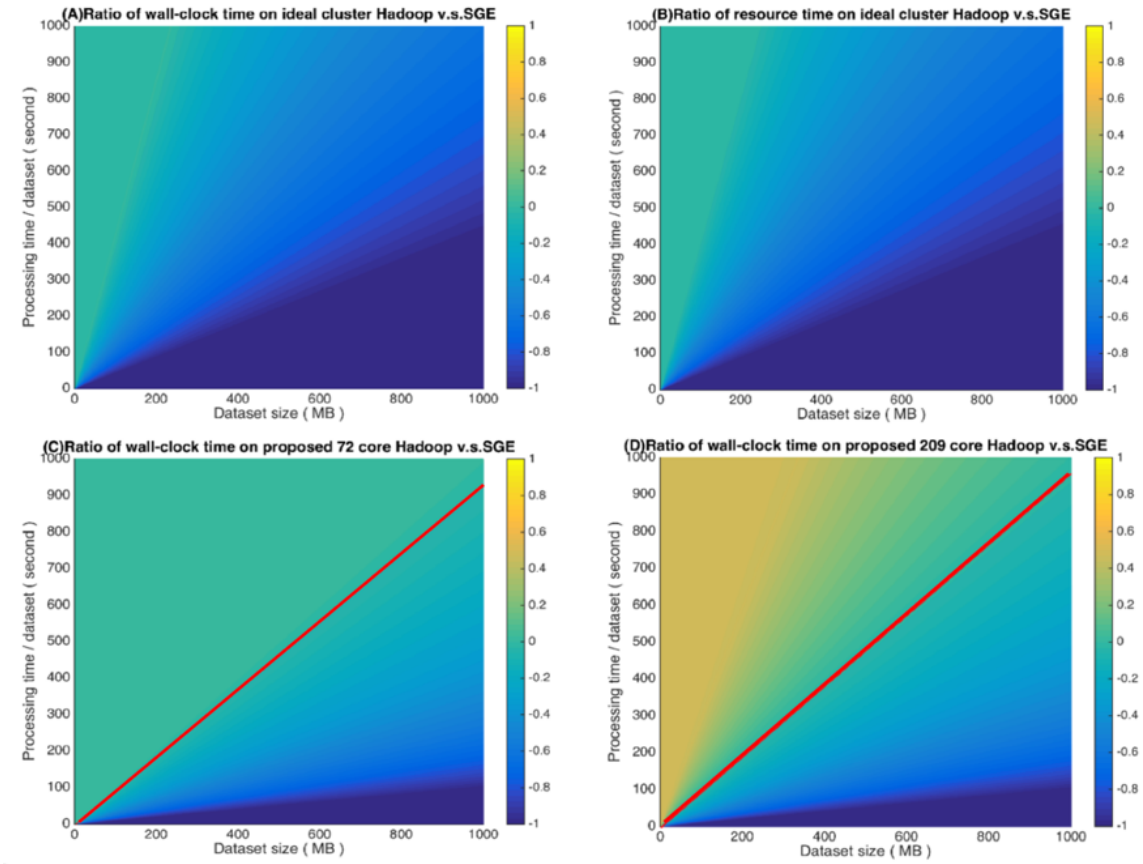


Figure 5.5: Time ratio of Hadoop v.s. SGE based on core / data balanced and unbalanced cluster. The balancing can only affect the total job running time. (A) Ratio of total running time on a data / core balanced cluster. (B) Ratio of resource time either on a core / data balanced cluster. (C) Ratio of total running time on a proposed 72 cores grid setup (approximate balanced cluster). (D) Ratio of total running time on a proposed 209 cores grid setup (core / data unbalanced cluster). The red lines in (C/D) indicate the parameters for which Hadoop and SGE result in equivalent performance for the specified setup.

SGE can perform similarly with Hadoop when the dataset size is relative small (200 MB) but running time is long, at around 1000 s. However, Hadoop performs much better, at least two-fold, when dataset size reaches over 500 MB and the processing time is around 1000 s either wall-clock time or resource time. We can also appreciate that when job processing time is very long (i.e., over 100 minutes), the resource time of both approaches are close, but Hadoop can still win on the time data transfer takes.

On a data / core unbalanced cluster, the wall-clock time for Hadoop is affected. Figure 5.5(C) and (D) presume the same data allocation proportion as the Hadoop with 72 cores

and 209 cores scenario in Table 5.2. The ratio value smaller than '1' can be treated as a break down time for SGE as the red line indicated, and the user should try to move from their 'traditional' grid framework to a Hadoop 'big data' framework. The smaller cluster is more balanced so the performance is better than in the Hadoop 209 scenario. This is because for bigger clusters, data is approximately balanced, but here there is one machine that contains 3 cores, which is much smaller than the average. So here the wall-clock time section distribution is different when compared with Figure 5.5(A). However, based on our experiment, Hadoop can randomly send some waiting jobs to other free cores (Figure 5.3(A) Hadoop 209 cores scenario), so the performance result should be considered a worst case. Additional investigation into factors with imbalanced clusters is warranted.

Chapter 6

A DATA COLOCATION GRID FRAMEWORK FOR BIG DATA MEDICAL IMAGE PROCESSING BACKEND HEURISTIC DESIGN

6.1 Problem Overview

When processing large medical imaging studies, adopting high performance grid computing resources rapidly becomes rapidly important. An inexpensive solution is to locate the data on the computational nodes to avoid the problem of saturating the network by copying data. For example, the Apache Hadoop Ecosystem can hold the promprovides an extensive suite of tools to co-locate storage and computation [19, 3, 4]. Hadoop is still not widely being integrated into medical image processing (MIP), although it has shown great success in online commerce [120, 121, 122], social media [123, 124, 125], and video streaming [125, 126, 127]. Several recent medical image processing (MIP) Current approaches in MIPhave aimed to take advantage of this big data architecture for specific usage use cases with like redesign general machine learning using MapReduce framework or understand use distributed systems [33, 118, 117]. We recently presented a "medical image processing-as-a-service" grid framework, Hadoop & HBase for Medical Image Processing (HadoopBase-MIP), which integrates - HadoopBase-MIP - offers promise in utilizing the Hadoop and HBase (a database built upon Hadoop) for data colocation by moving computation close to medical image storage as present in chapter 4 & 5. This system is a general framework for MIP (e.g., structured data retrieval, access to locally installed binary executables/system resources, structured data storage) without commingling idiosyncratic issues related to MIP. However, the system has not yet proved proven ease to use, and faces several key challenges for wide deployment as illustrated in Figure 6.1. Specifically,

- Heterogeneous hardware: cluster - Hadoop/HBase uses an approximately balanced

data allocation strategy by default. In [18], we observe that the throughput of a cluster is low especially when it combines with different types and / or different number of cores per machine. Thus, performance aware data collocation models are needed.

- Limit of Large dataset analysis : When we do large dataset summary statistical analysisanalyses are requested, huge volumes of data can easily saturate memory of one machine. Then Thus, one needs to we need to split whole a dataset into small chunks. M, most chunks would be sent to where their data is located and run in parallel on different machines, but few chunks would need to move data via network [cite data locality]. This, this process is also called Map phase in MapReduce computation paradigm [4, 3]. Once all intermediate results of all chunks are collected on one machine, they are further processed and final result is generated, this is depicted as Reduce phase [4, 3]. However, as presented in Figure 6.1(B), integrated processing of large datasets introduces dependencies with the number of jobs (which is based on chunk size) and a way to find a optimize chunking size is needs to be discovereded.

- Rapid NoSQL query: HBase is a column based NoSQL database, namely all data that are in same column are stored together and furtherly saved in Hadoop [31, 3]. Each row of table is a multi-record that are based on total number of columns are defined with a unique name as rowkey. If we store all medical images with other data like index, age and sex into same column, then when we do perform a query, linear search with image traversal is required, which rapidly decreases the speed of search. On the other hand, for HBase MapReduce, each single map needs to set a start/stop row and a typical column. When doing subset datasets analysis, data is scattered in HBase, and not all record between start/ stop row are needed. It is important to structure data in a manner to How to fast skip unnecessary row and image traversal for efficiency processing is challenging.

To resolve these challenges, in this chapter, we present a novel well-designed interface

application program interface (API) API that is developed based on built in Hadoop and HBase. By exploiting the backend interface of HadoopBase-MIP, we We make following contributions: 1) offer better data allocation in a heterogeneous cluster for faster processing through an off-line load balancer; 2) describe an optimal criteriona to find the splitting chunk size of large dataset; 3) do enable fast data query and boot up the of MapReduce performance by using a HBase table scheme.

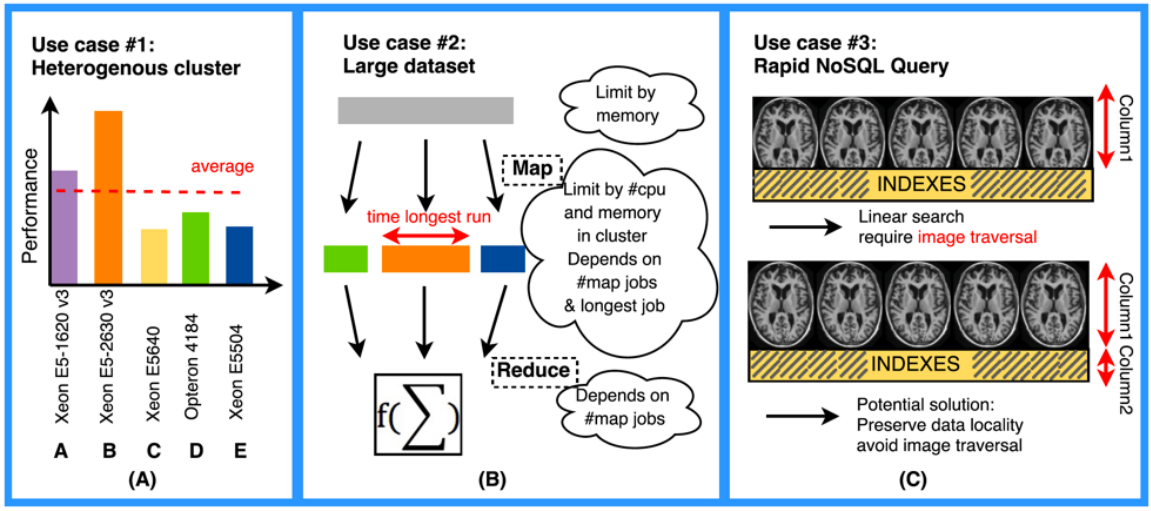


Figure 6.1: Use cases for three main challenges. (A) If a traditional cluster model is used, average throughput would be seen (red dash), which would leave some machines starved (e.g. A, B), while others overloaded (e.g. C, D and E). It Hence, a traditional approach will degrade the overall execution time due to those overloaded/starved cores. (B) The time to run a large dataset depends on total number of jobs and the longest map job to take. The total number of jobs should be neither too large or too small. It is limited clusters total cores and memory. (C) HBase is not designed default for storing image data since given variability of size and the volume of medical imaging studies. If information like age / sex / genetics are stored in same column with image data, when do query, image traversal is unavoidable, which degrades the search efficiency.

This work has been published in SPIE medical imaging 2018 [128].

6.2 Methods

6.2.1 HadoopBase-MIP system interface

Our interface is mainly based on HBase built in API. Before introducing our system backend, here is a quick introduction of HBase databases data storage model and abstract architecture. A simplified hierarchical data storage model of HBase is: $Table->Columnfamily->Columnqualifier->data$. An HBase table can contain many columns, and a column is usually denoted as column family. We can define many column qualifiers to specify a column family. Data is stored within qualifiers. EAnd each row of the table has a unique rowkey. To retrieve or delete a set of data, users need to specify (table name, rowkey, column family: column qualifer). Alternatively, one can , or retrievinge a series sequences of values by assigning a table rowkey range with relative column informationfamily and qualifier. As the aspect of architecture, aA table is split into different units named as regionsregions. - Iif any size of regions exceeds a pre-set threshold, (Of course, there is initially one region for a table), then the regions would be split into two new regions. This split process is activated through region split policy. And regions will be dispatched to different machines of cluster for balancing issue. Now we can see how our system backend interface works.

Figure 6.2 presents our system interface overview and how they may help tackle three challenges in section 6.1, which includes Upload, Retrieve, Load Balancer, MapReduce Template, Delete and Monitor. All operations are command-line based software tool with different parameters, and following enumerate list presents more details:

1. **Upload**:
    - Table name: will create a table if it does not exist.
    - A text file path for a file contains groups of $< columnfamily, qualifier >$ - to create / alter a table or change table scheme.

- A text file path for a file contains all images tuple of $< systemfilepath, fileuniquename, columr$ File unique name will be used as rowkey.
- Overwrite Boolean value. It helps update images or avoid uploading duplicate data.
- Region split policy : default policy, hierarchical policy [3]
- Pre-split : Boolean value. It is only valid when creating a new table.
- A text file path for a file contains all rowkeys for pre-split a table.

2. **Retrieve**:
   - Table name
   - Rowkey: set this value if retrieval is image based.
   - Start rowKey and / or stop row key: set them for a retrieval range, if both keys are empty, then retrieval is whole table column based.
   - Column family.
   - Column qualifier.
   - A text file path for a file contains all data retrieval destination path.
   - A text file path for a file contains all row keys need to skip to retrieve.
3. **Delete**:
   - All options are same with Retrieve except a text file path for a file contains all data retrieval destination path.
4. **Monitor**:
   - Hadoop built in job monitoring tool.
5. **MapReduce Template**:

- Table name: two names, one for source table to read data, another one for target table to write back result.
- Column family: three values, one for data query (will introduce more in next subsection), one for image data retrieval, the last one for target table.
- Column qualifier: three correspondence value with column family.
- A text file path for a file contains all start / stop row key pair, each pair is input for a map job.
- A text file path for a file contains all row keys need to skip to retrieve.
- Analysis level three options, image-based 5, dataset based 6 and large dataset based (will introduce more in next subsection).

6. **Load Balancer**:

- HBase default built in balancer is to balance the total number of region on each server. Our proposed load balancer is offline greedy allocation. First it finds all regions and images on each serve; second, moving images based on region; finally, the data allocation ratio of each machine meets the ratio of (total number of CPU * Million instructions per seconds (MIPS)) per nodes. MIPS is calculated by Linux perf and is varied based on different types of CPU.
- Table name.
- Column family.
- Column qualifier.

6.2.2 MapReduce model design and implementation for large datasets

As an exemplar use case, we consider processing of T1-weigthed brain MRI brain images are different inand construction of population specific averaged brain image. the context of cortical folding, the thickness, volume area across all age-specific range. Creating

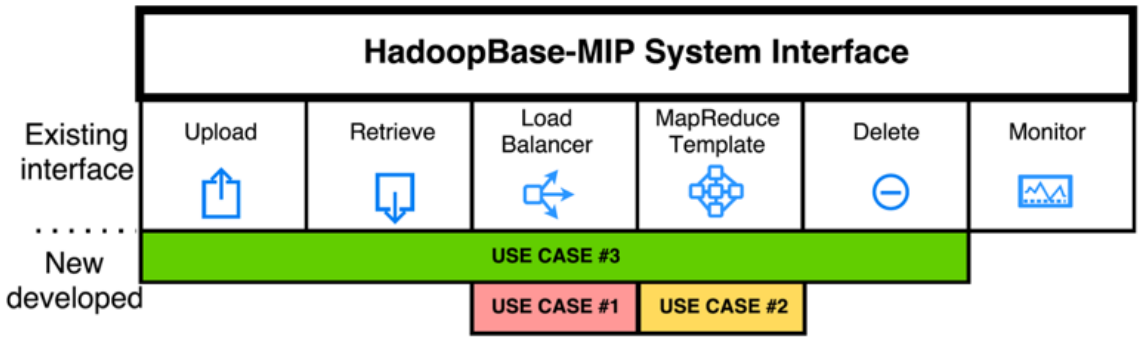


Figure 6.2: HadoopBase-MIP system interface overview. Except cluster monitoring, all operations are extended.

a specific template for a group of studies can help improve finding cortical folding pattern and reduce misclassification of brain tissue [129, 130, 131], we present a MapReduce computation model in HadoopBase-MIP to average a large dataset that is logically sticked together.

Figure 6.3 shows a full model pipeline to process a large dataset using MapReduce analysis. We first present a theoretical model to modelfor the pipeline on considering two important factors that users care the most: wall clock time (what the user sees anddirectly experiences) and resource time (time elapsed time on each node when a process starts across all nodes). We summarize the parameters and helper functions that affect both wall-clock time and resource time theoretical model for finding optimizing map task chunk size in and following enumerate list.

1. **#img** :
    - The total number of images that are needed to be averaged.
2. **#job** :
    - The total number of map jobs that are split from large datasets.
3. η :

- The total number of images per map task, which is the chunk size. It helps find the total number of map jobs: $\#job = \frac{\#img}{\eta}$. This is the variable that we are trying to find an optimizing value. We assume there is no local weighted concern in split map tasks, it means all map tasks share the same value of η.

4. **SizeBig** :

   - Maximum input file size of datasets, we use it for worst case scenario estimation..

5. **SizeSmall** :

   - Minimum input file size of datasets, we use it for upper and lower bound of η.

6. **SizeGen** :

   - Maximum output file size that is generated by image averaging software.

7. **Bandwidth** :

   - The bandwidth of cluster.

8. **VdisKR,VdisKW** :

   - Data read / write speed of local hard drive.

9. **#region** :

   - The total number of regions of a table in cluster.

10. **mem** :

    - The total memory of one machine. We presume all machines have same amount of memory.

11. **core** :

- The total number of CPU cores of the cluster.

When map tasks generate intermediate results, part of them are stored in network buffer, and others are flushed into local temporarily and transfer to reduce tasks shuffle phase later. is unbuffered ratio of map tasks results due to limit of heap size and cannot be held in memory.

12. α :

    - When map tasks generate intermediate results, part of them are stored in network buffer, and others are flushed into local temporarily and transfer to reduce tasks shuffle phase later. α is unbuffered ratio of map tasks results due to limit of heap size and cannot be held in memory.

13. β :

    - β is an experimental empirical parameter to represent the ratio of rack-local map task for Hadoop scenario, namely the data is loaded/stored via network. We empirically get its value with 0.9.

14. $diskR(x), diskW(x)$:

    - x is the size of a file. The function is used to calculate the time to read / write a file from / to local disc, namely $diskR(x) = x/VdiskR, diskW(x) = x/VdiskW$;

15. $bdx(x)$:

    - x is the size of a file. The function is used to calculate the time to transfer through network, namely $diskR(x) = x/Bandwidth$;

16. $avgANTS(\eta)$:

    - x is the size of a file. We use ANTS AverageImages [9] to empirically test average summary statistics analysis. We also do several profiling experiments

to model this image processing. However, it is hard to conclude a concrete model for ANTS average to match all profiling results. We can prove the total number of file sizes that are needed to be averaged grows much slower than chunk size itself. And the best solution is doing all average on 1 CPU. We found a worse case of $avgANTS(\eta) = 0.4 \cdot \eta + 5$ to illustrate this process.

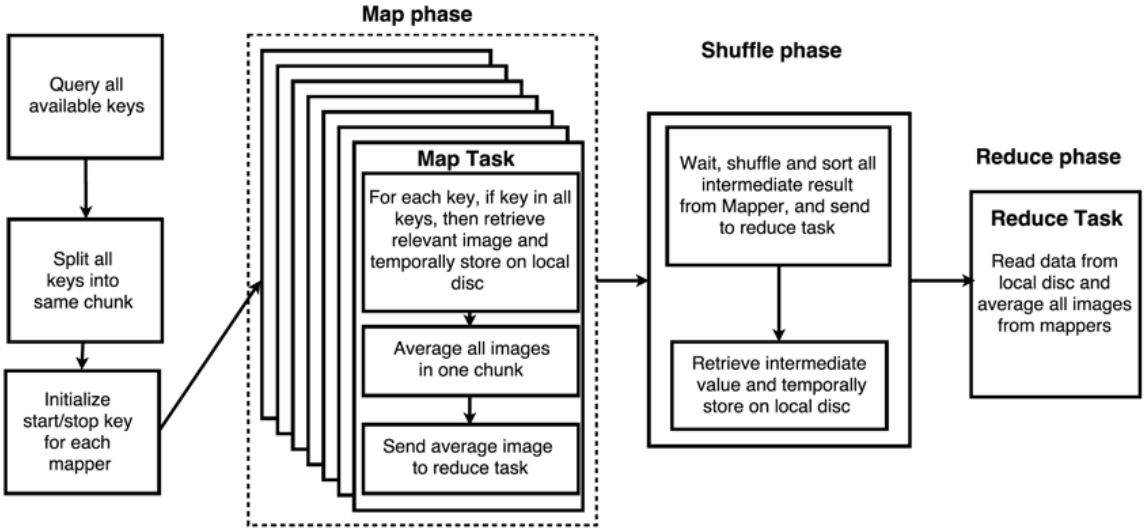


Figure 6.3: MapReduce model implementation for constructing population specific brain MRI templatesAverage images flow chart.

In Map phase, η is lower limited by total number of CPU cores since we aim to use one round to execute all map tasks. The upper bound is a confined by machines memory. In Rreduce phase, η depends on and further limited by machines memory. In summary, the range of valid η is defined $\eta \in [\max(\frac{\#img \cdot SizeSmall}{mem}), \frac{\#img}{core}]$. Equaltion 6.1 is an overview wall-clock time model.

$$WT_{wall} = WT_{init} + WT_{map} + WT_{shuffle} + WT_{reduce} + WT_{end} \tag{6.1}$$

where WT_{init}, WT_{end} are constant, which denote MapReduce job initialization and conclusion time. WT_{map} is wall time in Mmap phase that is presented in equation 6.2. It depends on the longest map task, namely the worst case a Mmap phase is decided by task(s) contain all large size of images.

$$WT_{map} = WT_{mapin} + WT_{mapProcess}$$
$$= diskR(SizeBig\ dot\eta) + btw(SizeBig \cdot \eta) + diskW(SizeBig \cdot \eta) + avgANTS(\eta) \tag{6.2}$$

The worst-case time for shuffle phase depends on all unbuffered data that needs to be retrieved from local disc, transfer through network and then transferred through band as shown in equation 6.3. WT_{reduce} is wall time in Reduce phase as equation 6.4 demonstrated.

$$WT_{shuffle} = diskR(SizeGen) + bdw(\alpha \cdot \lfloor \frac{\#img}{\eta} \rfloor \cdot SizeGen) + diskW(\lfloor \frac{\#img}{\eta} \rfloor \cdot SizeGen) \tag{6.3}$$

$$WT_{reduce} = WT_{reduceProcess} + WT_{reduceOut}$$
$$= avgANTS(\lfloor \frac{\#img}{\eta} \rfloor) + diskR(SizeGen) + diskW(SizeGen) \tag{6.4}$$

Equation 6.5 displays resource time, which sums up all the time from map, shuffle and reduce Reduce phase as Figure 6.3 shows.

$$RT = RT_{map} + RT_{shuffle} + RT_{reduce} \tag{6.5}$$

Similar with wall time, the worst-case resource time involve all big size image retrieval either through local disc or network, and it sums up all map tasks images average time presented in equation 6.6.

$$RT_{map} = RT_{mapin} + RT_{mapProcess}$$
$$= diskR(\#img \cdot SizeBig) + diskW(\#img\dot{S}izeBig) \tag{6.6}$$
$$+btw(\beta \cdot \lfloor \frac{\#img}{\eta} \rfloor \cdot \eta \cdot SizeBig) + \lfloor \frac{\#img}{\eta} \rfloor \cdot avgANTS(\eta)$$

$$RT_{shuffle} = \alpha \cdot \lfloor \frac{\#img}{\eta} \rfloor \cdot [diskW(sizeGen) + diskR(SizeGen)] \\ + bdw(\lfloor \frac{\#img}{\eta} \rfloor \cdot SizeGen) + diskW(\lfloor \frac{\#img}{\eta} \rfloor \cdot SizeGen) \tag{6.7}$$

$RT_{shuffle}$ is the resource shuffle time shown in equation (7), which includes all unbuffered data loading via local disc, all data transfer through network and all data writing to temporarily place to local disc in the end of shuffle. RT_{reduce} is wall time in Rreduce phase as equation 6.8 demonstrated that is identical with WT_{reduce}.

$$RT_{reduce} = RTreduceProcess + RT_{reduceOut} \\ = avgANTS(\lfloor \frac{\#img}{\eta} \rfloor) + diskR(SizeGen) + diskW(SizeGen) \tag{6.8}$$

6.2.3 NoSQL fast query new table design scheme

In this subsection, we will discuss how to create age-specific template in HadoopBase-MIP. As potential solution that is presented in Figure 6.1(C) potential solution discussed, we need to separate image data with relative indexes (i.e., age, sex, demographics, genetics etc.) into different column familiesy. It is to be noted that if they are in same column family using different column qualifiers, the query can still not avoid do image traversal. If there are no available indexes, but users only has have image files with unique file path/nameare needed to store and process, in this case, we recommend users using file path/name as table rowkeys, and same rowkey corresponds to one column family that stores one column familyimage data for image byte array data, and another column family for storing file size as index. Each file size record occupies only several bytes and can help hierarchical region split policy decides the right split point [3].

This table scheme is simple but useful especially for medical imaging data query owe to avoid image traversal. Here are several benefits for HadoopBase-MIP interface using our proposed table scheme. First, the separate index column family can help speed up checking if file exists when uploading, retrieving and deleting data. Second, it can also

fast collectlocate group or individual data and skip unnecessary record without traversing image data. Third, summary statistical analysis based on subset of data, i.e., average all females brain within 20-40 years old, and we have a table that column 1 storesall image data in one column, column 2 and stores indexes in a separate column. Both columns are in different column family. When firing lauching map tasks, our target is column 2 so that we can quickly filter which rowkeys are needed. Column 1 shares exactly same rowkey with Column 2, thus we can do data retrieval within the map and keep the data locality .

6.2.4 Experiment Design

Experiment cluster setups for both Hadoop and SGE are presented in Figure 6.4. Each job takes one CPU core with 4 GB memory available. We estimate achievable empirical average bandwidth as 70 Mb/second; disk read speed as 100 Mb/second with write speed as 65 Mb/second. The metrics to verify our proposed methods are wWall clock time and rResource time. SGE is empirically used as a baseline comparison.

6.2.4.1 Datasets

The experiment uses 5,153 T1 images retrieved from normal healthy subjectsgathered from [132].

6.2.4.2 Use case 1: Heterogenous cluster

In chapter 4, we found on a heterogonous cluster, if data is balanced allocated on each machines, that Hadoop would spent more wall time than SGE if data is balanced allocated on heterogeneous machines. A new HadoopBase-MIP load balancer result is shown at Table 2 based on number and the performance of the CPU per machine. Our goal is to empirically verify how load balancer can improve the performance of Hadoop in a heterogeneous cluster. We make use the same experimental design strategy as chapter 4, which is compressing 5,153 T1 images to the .gz format. Each job compresses only one NiFTI

image with 2GB memory available and generate one compressed image. The total input size of the images is 77.4 GB and the processing generates 45.7 GB of compressed files as output. To explore the impacts of processing time, we manually artificially increase the processing time by adding a sleep function without any data retrieval to make the job length of the experiment take an additional 15 105 seconds (10 s, 25 s, 40 s, 55 s, 70 s, 85 s, 100 s, 115 s respectively) on a fixed dataset to mimic different job processing speedrequirements. Each machine was used as a Hadoop Datanode and HBase RegionServer for data locality. All machines were also configured using SGE. There is aAn additional machine for both methods approaches serving serves as a cluster master.

6.2.4.3 Use case 2: Large dataset analysis

We first used NiftyReg [133] to do perform rigid affine transformation on all images to register to MNI-305space template [134, 135]. Our goal is to average all 5,153 datasets using ANTS AverageImages tool. We eEmpirically, the got biggestlargest file s size in the dataset is ,; the smallest file size is , and average generated files size are is MB. Based on clusters configuration, we get assessed the range of map chunk size of images as , so we manually increase the chunk size by 5 from 30, and empirically test both Hadoop and SGE scenarios.

6.2.4.4 Use case 3: Rapid NoSQL query

This use case aims to verify the benefit of using proposed table design scheme. Two major population based study features that we are concerned to do average are: age and sex. There are 10 experiments whose setup can be seen in Table 6.1. Two Hadoop approaches are deigned and compared. A nave table scheme is to store data of all images, index, sex and age information in a same column family. Our proposed one is to save image data in a separate column family, while the reset index and population info is in another column family. Two Hadoop scenarios are compared with baseline SGE performance. We

Table 6.1: Hadoop v.s. SGE experiment cluster setup with same memory allocation and fixed datasets.

Experiment	Age(years)	Female	Male
1	All	2370	
2			2120
3	4-20	1157	
4			698
5	20-40	651	
6			648
7	40-60	230	
8			280
9	>60	332	
10			494

empirically set 50 images per chunk for one map task.

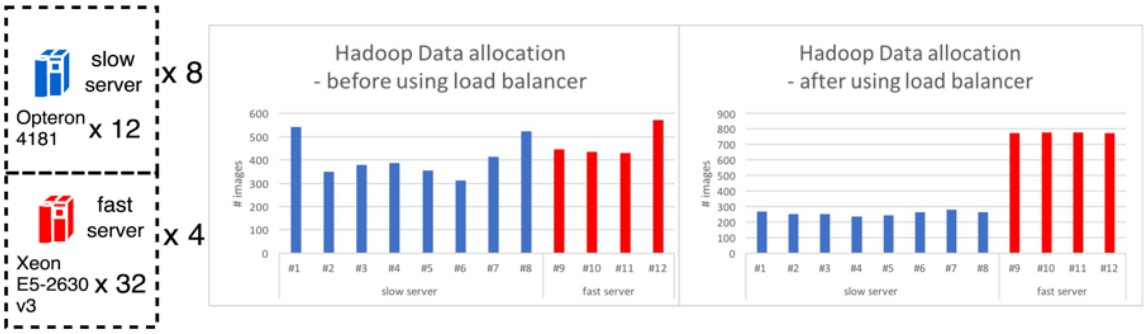


Figure 6.4: Experiment cluster setup and Data allocation for HadoopBase-MIP. Two2 different cores systems (eight machines with 12 slower cores and four machines with 32 fast cores) are used in cluster . 8 machines with slower cores, each machine contains 12 CPU. 4 machines with fast cores, and each machine has 32 CPU. Before applying the load balancer, each machine contains similar amount of image data. After using the load balancer, the data allocations meets match the ratio #CPU*MIPS.

6.3 Results

6.3.0.1 Use case 1: HeterogenousHeterogeneous cluster

Figure 6.5 presents the verification on both theatrical modeled and empirical result for Hadoop (before / after using load balancer) and SGE. The wall-clock time performance in Figure 6.5(A) for SGE is initially first stuck as one linelimited because of network

saturation. As the processing time of a single job increasesd, SGE performance is also linearly increasesd when with lessened network saturation impact is less. For both Hadoop scenarios, their initial overhead is very high when job processing time is small. We can see that as the job processing time increases, Hadoop without load balancing performs worse than SGE, while Hadoop with load balancer performs better than SGE with similar trend. Figure 6.5(B) shows an aggressive upper bound of theoretical upper bound for SGE; both Hadoop scenarios spend less resource time than SGE, and Hadoop with load balancer performs a little bit better when processing time is less than 40s.

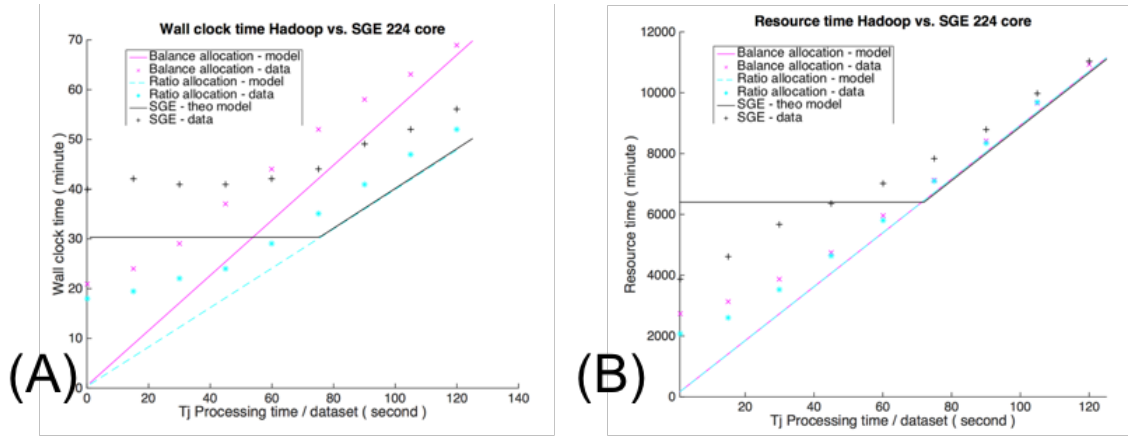


Figure 6.5: (A) Wall-clock time performance for Hadoop and SGE with different data allocation; (B) Resource time performance for Hadoop and SGE with different data allocation

6.3.0.2 Large datasets analysis

Figure 6.6 presents the validation of Hadoop and SGE on averaging 5,153 T1 images. SGE spends 5-fold of wall time and up to 20-fold of resource time more than Hadoop that are presented at Figure 6.6 (A, B). Figure 6.6 (C, D) shows the empirical wall time model has same trend with its theoretical model., which we can see wWhen the map task chunk size is 50-60, we can receive observe an optimizing optimized wall time. The Hadoop resource theoretical model shows an aggressive upper bound, and reveals when chunk size increases to more than 80, resource time are similar. And The empirical Hadoop real result shows when chunk size more than 80, the resource time becomes similar. Tthe qualitative

difference between 5,153 images average and MNI-305 space is shown in Figure 6.7.

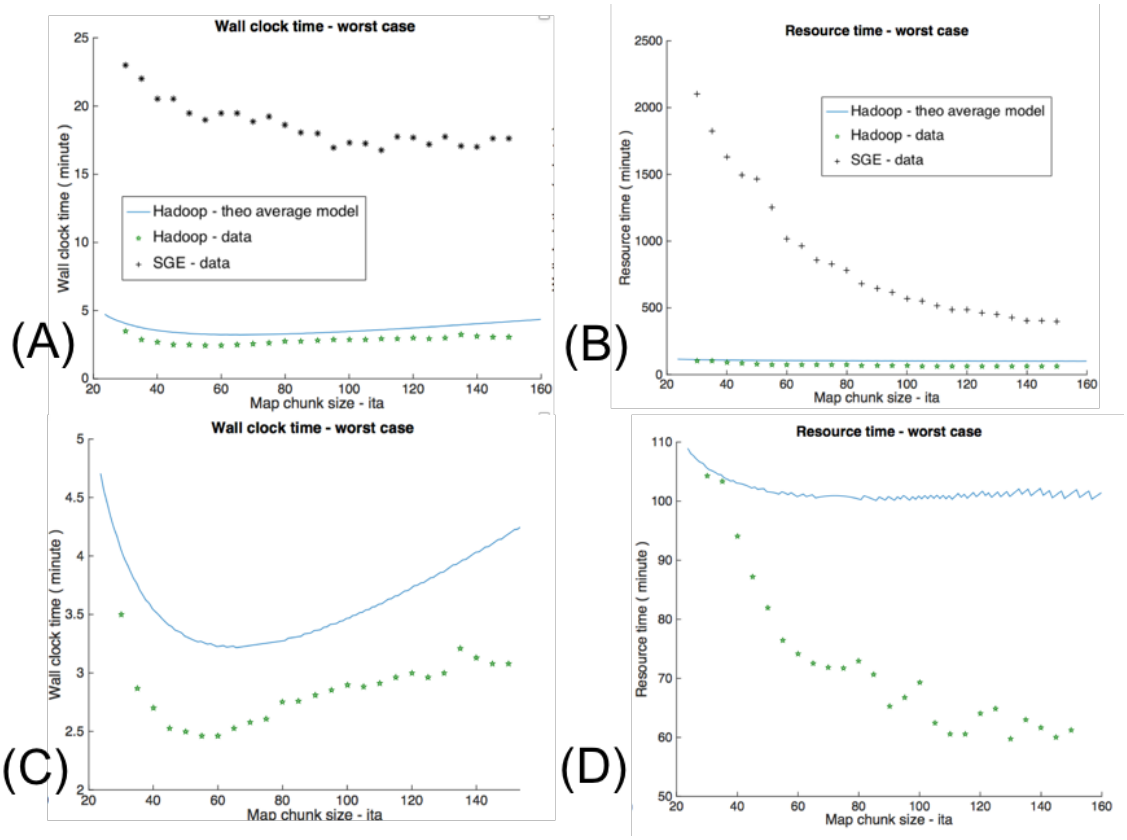


Figure 6.6: (A) Wall-clock time performance for Hadoop and SGE on large datasets analysis;(B) Resource time performance for Hadoop and SGE on large datasets analysis; (C) Wall-clock time performance for Hadoop and theoretical model; (D) Resource time performance for Hadoop and theoretical model

6.3.0.3 : Rapid NoSQL query

When averaging large subsets like all female and all males T1 images, SGE spends about 3-fold wall time and 6-fold resource time more than proposed Hadoop time. As the size of subsets decreases, we can see that SGEs wall / resource time also decreases, and proposed Hadoop table scheme design also generates similar decreasing trend, and use less wall / resource time than SGE. However, nave table design scheme leads to opposite trend, for example, averaging all males T1 images within 40-60 years old cost nave Hadoop scenario namely when subsets size is small, it spends 6.5- fold wall time more than SGE

and about 9- fold more than proposed Hadoop, and 7- fold resource time more than SGE with 12 fold12-fold more than proposed Hadoop.

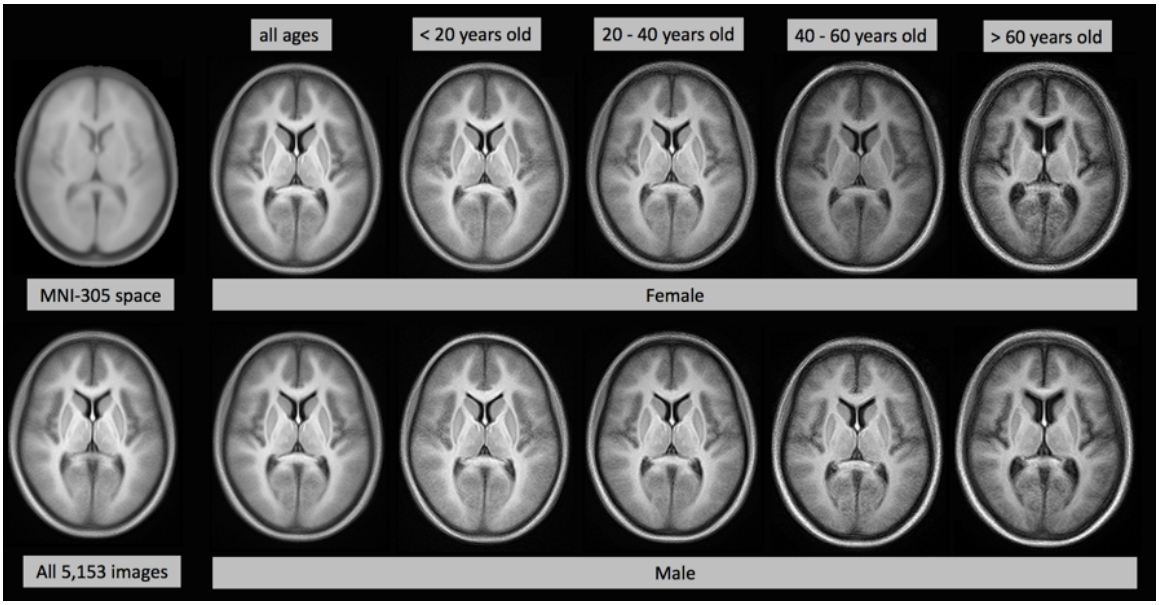
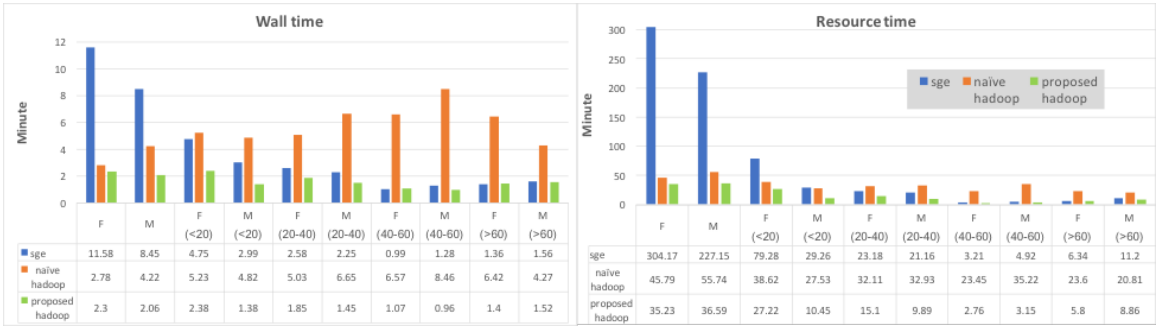


Figure 6.7: Qualitative results for summary statistics analysis on large datasets and age / sex-specific image averaging analysis.



	F	M	F (<20)	M (<20)	F (20-40)	M (20-40)	F (40-60)	M (40-60)	F (>60)	M (>60)
sge	11.58	8.45	4.75	2.99	2.58	2.25	0.99	1.28	1.36	1.56
naïve hadoop	2.78	4.22	5.23	4.82	5.03	6.65	6.57	8.46	6.42	4.27
proposed hadoop	2.3	2.06	2.38	1.38	1.85	1.45	1.07	0.96	1.4	1.52

	F	M	F (<20)	M (<20)	F (20-40)	M (20-40)	F (40-60)	M (40-60)	F (>60)	M (>60)
sge	304.17	227.15	79.28	29.26	23.18	21.16	3.21	4.92	6.34	11.2
naïve hadoop	45.79	55.74	38.62	27.53	32.11	32.93	23.45	35.22	23.6	20.81
proposed hadoop	35.23	36.59	27.22	10.45	15.1	9.89	2.76	3.15	5.8	8.86

Figure 6.8: Proposed table scheme design vs. nave scheme vs. SGE

6.4 Conclusion

The chapter presents a HadoopBase-MIP backend design which is a data Colocation Grid Framework for Big Data MIP. We use three use cases to validate the performance and the usage of the system interface, all cases are compared with SGE. For heterogamous

cluster case, SGEs trend reveals that its performance will be degraded by network saturation. When network saturations impact decreases, Hadoop without load balancer performs worse than SGE due to low utilization of cluster CPU resource. Hadoop with load performance can help reduce this resource overload / starvation issue. The reason of both Hadoop scenarios wall time is high when job processing time is very small reveals when data processing type is high read / write type, the write performance would make congestion of the whole process.

For large dataset analysis case, image averaging analysis is a read intensive with short computation processing. SGEs performance is limited due to data movement via network. Combining theoretical wall time with resource time model for Hadoop summary statistic, we can conclude 50-60 optimal map task chunk as map task chunk size is optimizing. For rapid NoSQL query case, SGEs performance is correlated with the subsets size that needs to be averaged, since there is no query efficiency issue, all data has to be retrieved from network storage to cluster computation node. For naïve Hadoop table scheme, it costs more time than proposed Hadoop and SGE especially when subsets size is relatively small. The reason for this is this scheme would force query traverse image data that is not needed which takes more time than proposed Hadoop scheme that skip small index without image traversal. In summary, HadoopBase-MIP system provides a complete backend interface developed upon built in Hadoop and HBase API.

Chapter 7

TECHNOLOGY ENABLERS FOR CLOUD-BASED MULTI-LEVEL ANALYSIS APPLICATIONS IN MEDICAL IMAGE PROCESSING

7.1 Problem Overview

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 7.1. 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 [136, 137]. 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

[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.

the processing times incurred by such multi-stage analysis.

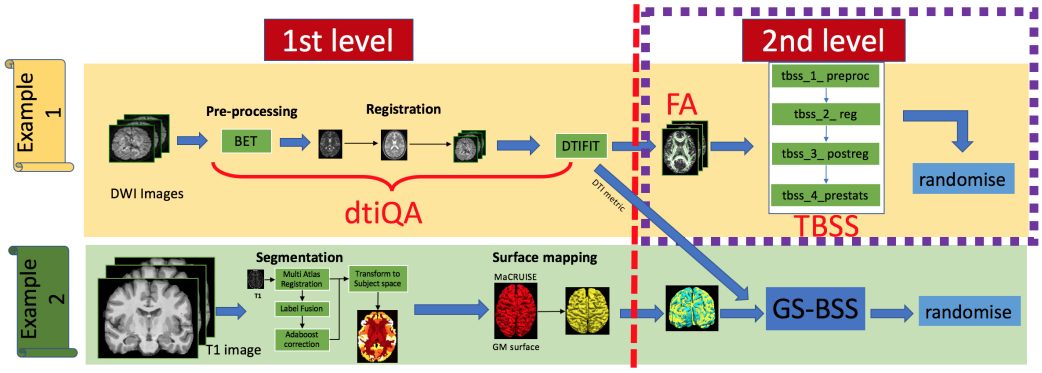


Figure 7.1: A classical medical image processing multi-level analysis Tract-Based Spatial Statistics and Gray Matter Surface-based Spatial statistics. Our target pipeline for this work is the first example dtiQA & TBSS pipeline. The purple box indicates our second contribution focus area.

To support our claim, we collected anonymous trace statistics logs of completed jobs from the XNAT system [138] 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 IBMs 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 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 [19], 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 7.2 describes the contributions that realizes the medical image processing-as-a-service; Section 7.3 describes our evaluation approach and presents experimental results; and finally Section 7.4 presents concluding remarks alluding to ongoing and future work, and discusses the broader applicability of our approach.

Ths work has been accepted at 2018 IEEE International Conference on BigData.

7.2 Supporting Medical Image Processing Multi-level Analysis in the Cloud

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.

7.2.1 Eliciting Challenges using Two Case Studies

Figure 7.1 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 [139]) 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 [8, 9]. 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 1.6×10^6 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.

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

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) [10] or Gray Matter Surface-based Spatial statistics (GS-BSS) [11], 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 7.1 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 [12, 13]. 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 prop-

[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.

agate 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.

7.2.2 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 7.2 illustrates our proposed real-time monitoring and intermediate checkpointing framework for multi-stage analysis. 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.

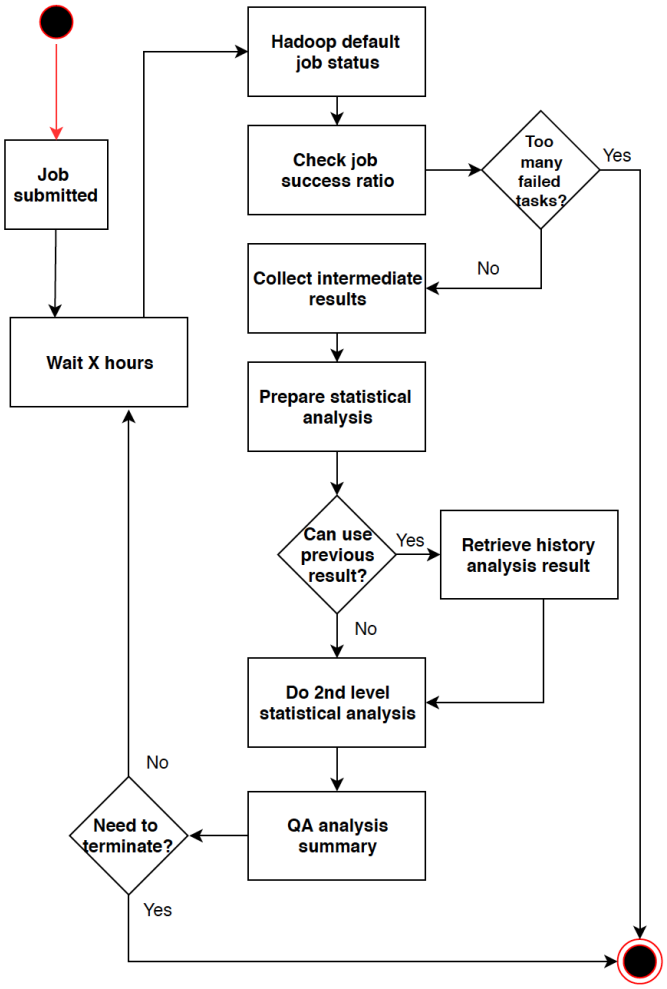


Figure 7.2: Real-time monitoring and intermediate checkpointing framework for multi-stage analysis (in this case 2-stage).

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 7.1, 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.

7.2.3 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 [140], 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 [4]. 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 HBasebased 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 [3]. 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 sub-

mission 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 [102, 119].

The full pipelines for both of our SLURM-simulator and HadoopBase-MIP-simulator are shown in Figure 7.3 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.

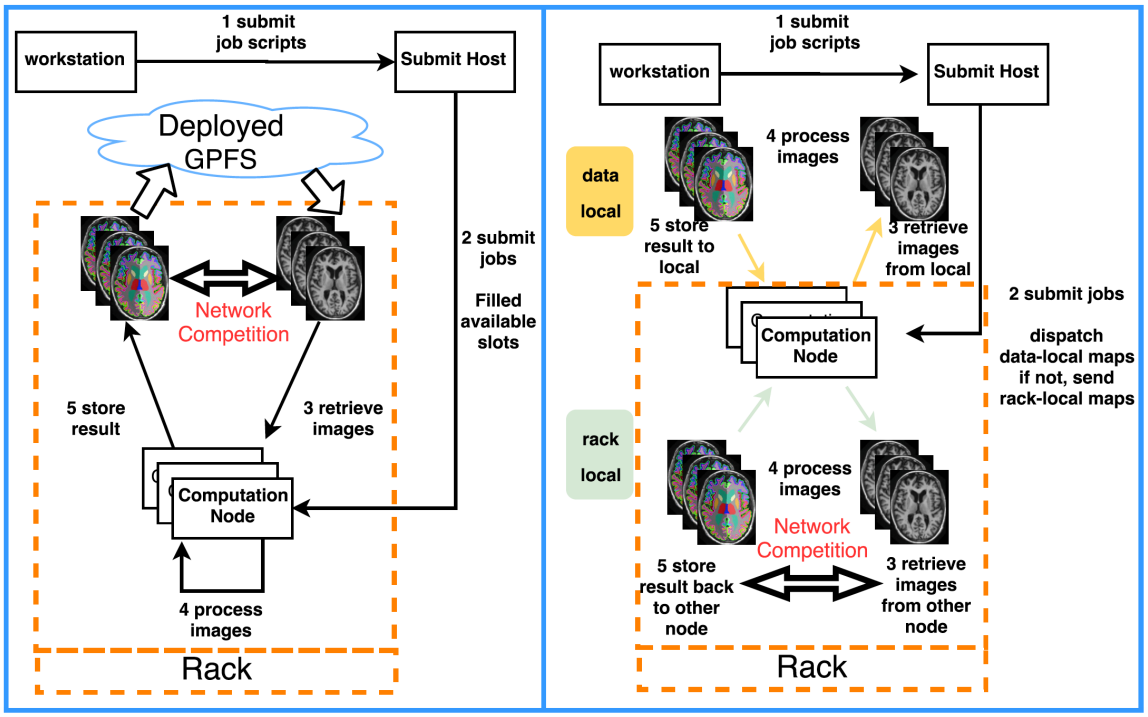


Figure 7.3: Full pipelines of SLURM-simulator for the traditional cluster (left) and the HadoopBase-MIP-simulator (right).

7.2.3.1 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.

7.2.3.2 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.

7.3 Evaluational 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.

7.3.1 Experiment Scenarios

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

7.3.1.1 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 [26] (SGE, a similar scheduler with SLURM), We use the same experimental design strategy as in [141], 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 [141] 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

[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.

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.

7.3.1.2 Experiment 2

The goal of experiment 2 is to use the simulator on the historical trace data we alluded to in Section 7.1 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 7.1, 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.

7.3.1.3 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 7.1. 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.

7.3.2 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 [132]. 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.

7.3.3 Experiments result

7.3.3.1 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$\pm$standard deviation throughput of 152.5$\pm$3.3 dataset/minute, which is 1.5-fold better on average than the traditional cluster using SGE whose mean$\pm$standard deviation throughput is 102.2$\pm$2.5 dateset/minute. In comparison, the simulator predicts that HadoopBase-MIP is 1.53-fold better than traditional cluster (163.9$\pm$4.6 vs. 106.9$\pm$5.7 dataset/minute). For the second test case, most jobs are long jobs (80% of jobs need to be processed in 120 seconds). Thus, the difference in throughput is smaller between the two systems compared with the first test case, and HadoopBase-MIP becomes 1.17-fold better than traditional cluster (91.3$\pm$2.3 vs. 179.4$\pm$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).

7.3.3.2 Experiment 2

Figure 7.4 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 7.4(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 7.4(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.

7.3.3.3 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 7.5(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 7.5(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 7.5(2). After removing all bad outputs from the first stage and re-running the second stage analysis, Figure 7.5(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 7.5(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 [139], 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 7.5(1), and users could conclude their findings when they see some p-value saturation around round 15-19 as shown by Figure 7.5(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 check-

pointing 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.

7.4 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 [141] has discussed the trade-offs between using HadoopBase-MIP and traditional in-house clusters.

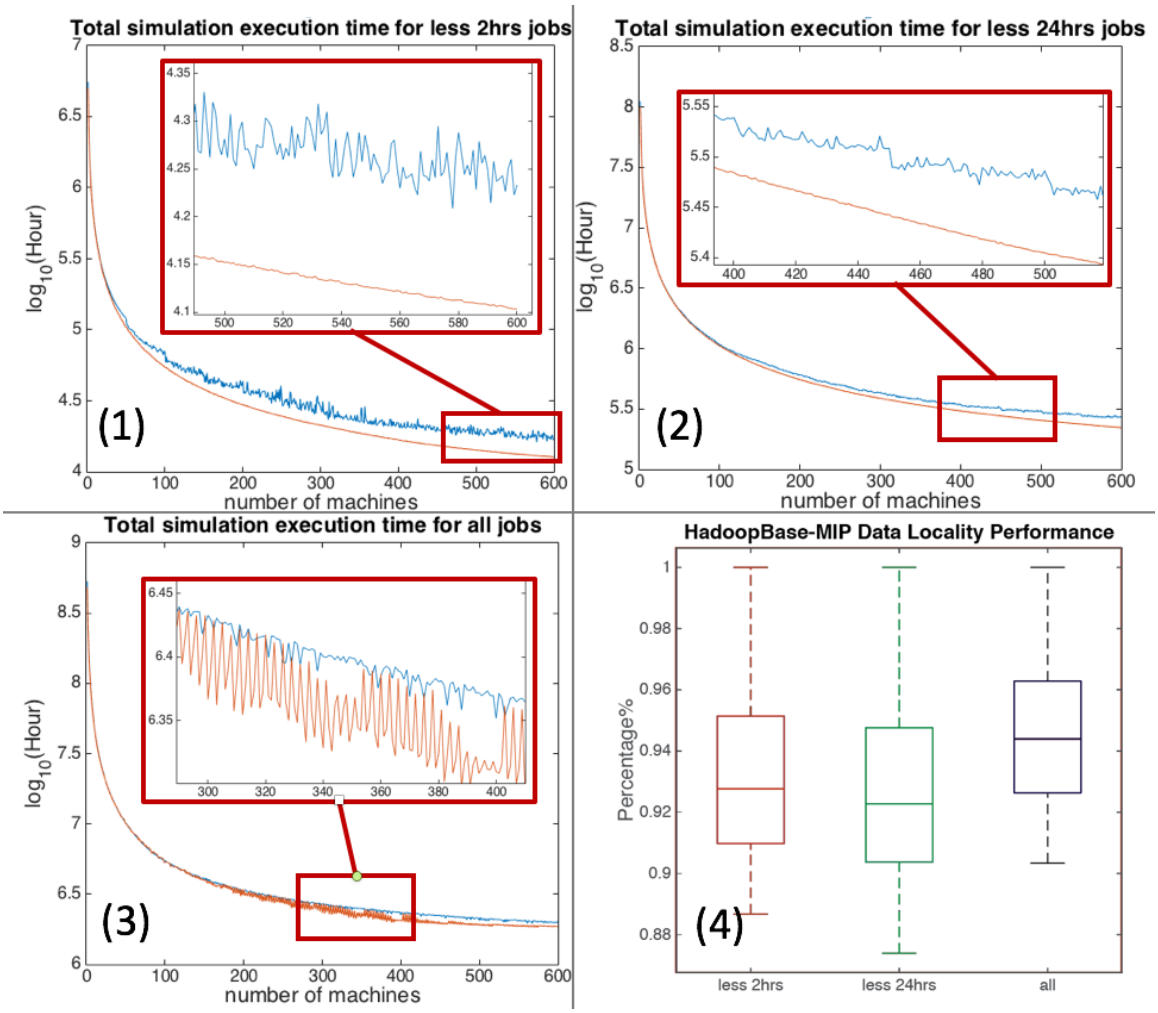
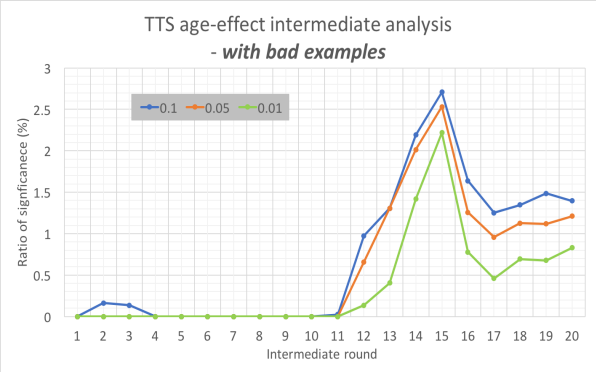
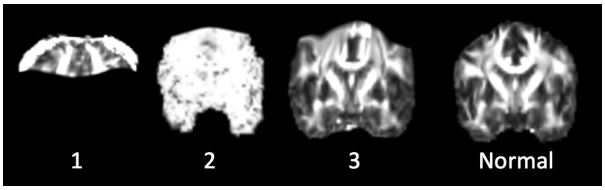


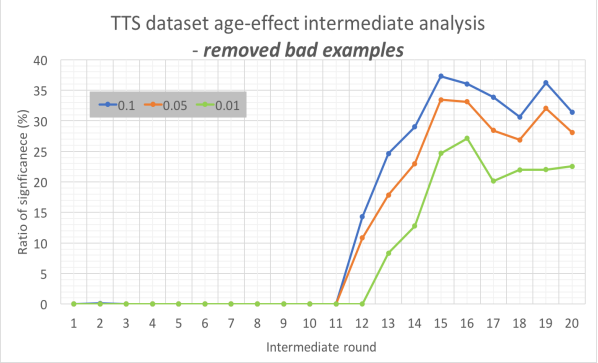
Figure 7.4: Simulation result of estimating total execution time (log10 (Hour)) according to historic jobs trace. (1)-(3) show running all jobs whose processing times are less than 2 hours, 24 hours and all 96,103 jobs, respectively. Blue line represents the performance for traditional cluster while 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.



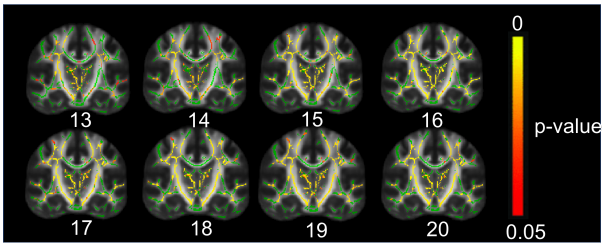
(1) Found weird p-value trend



(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

Figure 7.5: 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.

Chapter 8

BIG DATA NON-RIGID REGISTRATION-BASED IMAGE ENHANCEMENT IMPROVES DEEP BRAIN STRUCTURES

8.1 Problem Overview

Brain MRI is an important tool for analyzing deep brain structures. Visual inspection remains as the gold standard for identifying anatomical region boundaries in the brain. However, low contrast, spatial proximity of surrounding structures, and low resolution may inhibit clear visual assessment of deep brain structures [142]. Various post-acquisition image enhancement strategies have been proposed to address these issues. Traditional image enhancement employs a low-pass filter for denoising. Liu et al. propose an enhanced non-local means using Gaussian filter to reduce the disturbance of the noise [143]. Manjn et al. use a non-local PCA thresholding within a rotationally invariant non-local mean filter, in which the local noise level is automatically determined in the image to make noise estimation and denoising [144]. A graph-based redundant wavelet transform approach is shown in [145] to sparsely represent MR images. Another image enhancement classical research topic is to utilize low-resolution image to reconstruct high resolution image. In [146], Rousseau propose the use of anatomical inter-modality priors based on a physical model of image acquisition as a high resolution image reference to improve the resolution of the LR image. In [147], Bahrami et al, propose a hierarchical re-construction via group sparsity in a novel multi-level canonical correlation analysis (CCA) space, to improve the quality of 3T MR image to be 7T-like MRI. Recently, a trend has developed for deep learning for medical image enhancement area. Yang et al use an artificial neural network to lower the bound of the variance/resolution tradeoff and thus to decrease the size of the unachievable region in the variance/resolution plane of PET modality [148]. Oktay incorporates anatom-

ical prior knowledge into convolutional neural network to enhance cardiac image [149]. Yet, none of the above works intrinsically focus on MRI deep brain enhancement.

Our inspiration comes from a recent work [128] that can perform averaging of thousands of images (pre-processed affinely into MNI space) in near real-time (<3 mins). The resulting image reveals enhanced contrast in some brain structures. When considering the generation of MNI average brain stereotaxic registration model [134], we posit that it is useful to use non-rigid big data averaging. However, such an approach has been computationally infeasible as a single reasonable non-rigid image registration would take 2-4 hours; hence, registering a reference population would take about 2 months. Recently, ultra-fast image registration methods [150] have been developed that enable a 50-100-fold acceleration of the registration process. It is now feasible to routinely perform these computations. Herein, we propose big data registration-enhancement image (BDRE) to improve the intensity contrast of deep brain structures. Rather than parametric models of anatomy or specific sets of well-characterized atlases, we propose to use large archives of unlabeled data (e.g., big data) to capture and enhance image features.

Briefly, our approach registers thousands of datasets to a single target image using non-rigid image registration [150]. We then create a new average image by deforming the registered data to reflect patterns of target image. This approach reveals substantially increased detail in the deep brain structures. To the best of our knowledge, a proposal to routinely apply registration of a large number of images (i.e., >1,000) per target image has not yet been presented.

On standard 3T T1 images, deep brain structures are visible but substructures are not easily identifiable due to low contrast and low image resolution. The thalamic nuclei and hippocampal subfields are two sets of deep brain substructures that play significant roles in the pathophysiology and treatment of neurological and psychiatric disorders including Parkinsons disease, Alzheimers disease, depression, and schizophrenia [151, 152, 153]. However, delineation of these structures on MRI typically requires high resolution 7T or

specialized 3T images which are not commonly acquired in research studies or clinical practice. In Figure 8.1 and Figure 8.2, we present examples of images specialized for identification of thalamic nuclei (Figure 8.1: 7T T1) and hippocampal subfields (Figure 8.2: 3T T2) along with reference segmentations. The appearance of these structures on standard 3T T1-weighted images for both structures are also presented. As can be observed, the contrast of the thalamic nuclei and hippocampal subfields can be improved on standard 3T T1 images by de-noising (DN), super resolution (SR), and BDRE. Because labeling of the thalamic nuclei and hippocampal subfields requires another imaging modality (i.e., 7T MRI and 3T T2 MRI), it is difficult to determine if contrast within BDREs are spatially well aligned with the original T1 targets structures since the generation of BDRE is based on a more common widely used 3T scanner. In this chapter, rather than validate each boundarys correctness in BDRE due to difficulties of direct visualization of our target structures, we investigate the usefulness of BDRE for the multi-atlas (MA) segmentation on those deep brain structures and assume only T1 images are available.

Ths work has been submitted to Magnetic Resonance Imaging.

8.2 Methods

The pipeline for BDRE image generation is as follows Figure 8.3. Briefly, a T1-weighted 3T image is retrieved from an XNAT system [138] and affinely transformed into MNI space. All local atlases (a total of 5,150 T1-weighted images pre-processed into MNI space, and we call them as 5K atlas images for BDRE) undergo a linear transform to target image (in MNI space) followed by a non-rigid registration to the target image. Next, all registered atlases are averaged in target space with a uniform weight. Finally, the enhanced image is sent back to XNAT system for storage. Traditional registration tools, e.g., Advanced Normalization Tools (ANTS)s Symmetric Normalization (SyN) algorithm [9], are time consuming to perform at scale. With recent innovations in image registration efficiency, it is now becoming feasible to evaluate such scenarios. We evaluate non-parametric

discrete registration with convex optimization. The tool, Dense Displacement Sampling wbirLCC (deeds-wbirLCC), uses a fast implementation of a local cross-correlation metric evaluated densely over a discrete search region followed by Gaussian smoothing of the resulting fields, and the fields are coupled back into the similarity measure [150, 154]. While the quality of the fast registration is generally similar with a traditional neuroimaging centric tool - ANTs SyN, the registration process is exceptionally fast and sufficiently flexible to allow for high throughput registrations.

Specifically, we used NiftyReg [133] to perform rigid affine transformation to register target image to MNI-305 template. We then used deedss transformation tool linearBCV to operate linear transformation on 5K atlases to the target image in MNI for better compatiblility with deeds-wbirLCC [154]. After that, we applied the big data registration tool deeds-wbirLCC to non-rigid registered transformed 5K atlases to the target image in MNI. For averaging 5K non-rigid registered images, we used ANTS AverageImages tool. All software packages were configured using default parameter settings, except for the deeds-WBIRLCCs regularization Gaussian sigma parameter that was empirically set as 1. The 5K atlas images for big data registration were retrieved from normal healthy subjects collected by [132].

8.2.1 BDRE generation time

The proposed method takes less than 7 hours to perform 5150 non-rigid registrations to target image in MNI space using 200 CPU cores. For image averaging to generate one BDRE, we equally split 5150 registered atlases into 103 subgroups, average each group first and then average all intermediate results within 20 minutes.

In detail, the 5150 atlases images are all pre-rigid affine to MNI space since target image would be also registered to MNI space. This is only a one time cost (based on our experimental cluster, it would take about 50 minutes). We did not do other preprocessing (e.g. N4 bias correction. If we did so, it would be also a one-time cost. The most time

consuming part is the large volume of non-linear registration, the testbed used for our validation involved 188 cores. At least 2GB RAM was available per core. Each deeds-WBIRLCC takes 4 cores, thus in total 51 non-rigid registration can run simultaneously. Per each atlas, it would take about 4 mins (due to CPU resource competition, the speed is slower than deeds-WBIRLCC on a single image with empty cluster, which is 50x faster than ANTs around 2 minutes), and that is how we spent 7 hours per one BDRE (5150/51*4 $\approx$ 404 minutes). The averaging times depended on the network setup of our cluster. Our experimental cluster used shared network that was deployed with one gigabit ethernet on each computation nodes and took 20 minutes to do averaging 5150 atlases. Moreover, if we had multiple BDRE generating, the image averaging can be applied slowly on a single core, the processing time for averaging would be overlapped with an on-going BDRE generation.

8.2.2 Image Acquisition information

To explore the usefulness of BDRE, we focused on two deep brain structures: thalamic nuclei and hippocampal subfields. Under IRB approval, we collected MRI data from eight subjects with paired 7T-3T T1-weighted MRIs for full thalamus segmentation. The ground truth 46 labels (23 per each hemisphere) were manually delineated by visually fusing the information provided by the multiple image volumes in 7T space [155]. Table 8.1 presents the detail of 23 thalamic nuclei structures.

We retrieved paired T1 and T2 MRI datasets for 29 subjects in deidentified form from a first episode psychosis hippocampal imaging dataset. The reference standard hippocampal subfield labels were generated by automatic segmentation of hippocampal subfields (ASHS) pipeline on the T2 MRI (symmetric 4 labels for left / right side) as reference standards [156]: (1-2) CA1, CA2/3: cornu Ammonis. Here, we collapsed subfields CA2 and CA3 due to lack of distinguishing contrast in MRI and variability [157]; (3) DG: dentate gyrus; (4) SUB: subiculum.

Table 8.1: The detail of 23 thalamic nuclei substructures that were manually delineated.

Thalamic nuclei		
anterior medial nucleus (AM)	lateral posterior nucleus (LP)	ventral anterior nucleus (VA)
anterior ventral dorsal (AVD)	mediodorsal nucleus (MD)	ventral lateral anterior nucleus (VLa)
central medial nucleus (CeM)	mammillothalamic tract (mtt)	lateral posterior nucleus (VLp)
central lateral nucleus (CL)	parafascicular (Pf)	ventral medial nucleus (VM)
centre median (CM)	pulvinar (Pu)	ventral posterior inferior nucleus (VPI)
habenula (Hb)	anterior pulvinar (PuA)	ventral posterior lateral nucleus (VPL)
lateral dorsal nucleus (LD)	lateral pulvinar (PuL)	ventral posteromedial nucleus (VPM)
limitans (Li)	paraventricular (PV)	

8.2.3 Preprocessing

For each above 3T T1-weighted image, we created one BDRE generated image. Meanwhile, we also created one denoising image and one super resolution image by software BTK default setting. BTK is an Open-Source Toolkit intrinsically for fetal brain MR image processing, while the denoising technique can be applied for any 3D MRI, and super resolution can be applied to adults affected by pathologies (such as Parkinson disease) leading to involuntary movements during image acquisition [158, 159]. All images for MA segmentation pipeline were affinely transformed to the label space (352x352x350 voxels and 0.7mm isotropic voxel size in the axial direction for thalamus; 480x480x26 voxels with 0.48x0.48x2 mm3 voxel size for ASHS label space, for computation efficiency, we did not upsample atlases and targets.). Then, images were pre-processed by N4 bias field correction for correcting image intensity inhomogeneity. Next, we removed outliers by trimming each images and keep the intensity range within [0.15,99.85] percentiles. Finally, for image normalization, consider the original image I_0 with mean intensity of μ and standard devi-

ation of the intensity σ , the normalized image I_1, then a commonly used normalization based on z-scores can be described as follows equation 8.1.

$$I_1 = \frac{I_0 - \mu}{\sigma} \tag{8.1}$$

8.2.4 Experimental setup

We evaluate the usefulness of BDRE for MA segmentation on the above two deep brain structures. For the thalamic nuclei, we use three parallel MA pipelines as shown in Figure 2. We use leave-one-out cross-validation, where one subject is chosen as a target and the remaining 7 are used as atlases. The process is repeated 8 times using different subjects in traverse. For the hippocampal subfields, based on input imaging modality, four parallel pipelines are used for verification. We first randomly chose 15 images from 29 datasets as atlases and remainder of the 14 images for testing. We then reverse training and testing set for second fold validation.

For MA segmentation, we used NiftyRegs Reg Aladin algorithm for efficient rigid affine transformation. And we used ANTs SyN for non-rigid registration with the cross-correlation similarity metric (with radius 2) and a Gaussian regularizer with $\sigma = 3$. After registration, reference segmentations from each of the atlases were warped into the target image space. Finally, we used default joint label fusion [160] to make thalamus targets label. For hippocampal subfield, we empirically set both patch radius for similarity measures and local search radius to 1x1x1 because of limited number of slices in coronal view.

8.2.5 Validation

We quantified the performance of eight MAS approaches in terms of Dice Similarity Coefficient (DSC) to assess the contributions of BDRE in MA segmentation, which is the most commonly used accuracy metric in recent studies of image segmentation. Consider X as the segmentation result, Y the truth volume, and $\|\cdot\|$ the L1 norm operation, DSC is

Table 8.2: Average performance of thalamic nuclei MA segmentation approaches using different modalities as atlases and target. Average Dice similarity (± standard deviation(SD)) and median DSC are shown. Each Wilcox paired test is calculated between BDRE (as reference) and other corresponding scenarios, where "**" represents Wilcox paired t-test with $p<0.01$, and "*" represents $p<0.05$.

Atlases + target	**3T ***	**7T ****	**DN ***	**SR ****	**BDRE (ref)**
DSC (mean±SD)	0.447 (± 0.017)	0.422 (± 0.04)	0.457 (± 0.028)	0.420 (± 0.028)	0.488 (± 0.038)
DSC (median)	0.446	0.425	0.466	0.414	0.477

presented as follows equation 8.2.

$$DSC(A,B) = \frac{2\,\|A \cap B\|}{\|A\| + \|B\|} \tag{8.2}$$

All assessments of significance below were performed with the Wilcoxon paired signed rank test. Each Wilcox paired test is calculated between BDRE (as reference) and other corresponding scenarios.

Figure 8.4 illustrates qualitative results and Figure 8.5 presents quantitative results for full thalamus MA segmentation of using different modalities. Three sample datasets are selected based on overall DSC performance: poor, average and good. Table 8.2 lists quantitative for overall DSC performance for each imaging modalities in detail. BDRE gets overall DSC of mean (± standard deviation) of 0.488 (± 0.038), which significantly improves average thalamic nuclei segmentation compared with using 3T only, 7T only, DN only and SR only.

Specifically, BDRE MA performs significantly better with 10 out of total 46 thalamic nuclei structures when comparing BDRE and 3T MRI MA segmentation; 11 out of total 46 thalamic nuclei structures are significantly better when comparing BDRE and 7T MRI MA; 9 out of total 46 thalamic nuclei structures are significantly better when comparing BDRE and 3T images preprocessed by DN; and 10 out of total 46 thalamic nuclei structures are significantly better when comparing BDRE and 3T images preprocessed by SR. Mean-

Table 8.3: Average performance of hippocampal subfields MA segmentation approaches using different modalities as atlases and target. Average Dice similarity (± standard deviation) are shown. Each Wilcox paired test is calculated between BDRE (as reference) and other corresponding scenarios, where *** represents Wilcox paired t-test with $p<0.001$, and ** represents $p<0.01$.

Atlases + target	**3T *****	**DN ****	**SR *****	**BDRE (ref)**
DSC (mean±SD)	0.607 (± 0.054)	0.628 (± 0.046)	0.608 (± 0.042)	0.660 (± 0.045)
DSC (median)	0.597	0.642	0.618	0.667

while, we found 3T MRI MA segmentation outperform than BDRE on left lateral posterior nucleus (LP) and right anterior ventral dorsal (AVD), and when 3T MRI preprocessed by DN, it performs better on left lateral pulvinar (PuL), right AVD and right pulvinar (Pu).

Similarly, Figure 6 illustrates qualitative results and Figure 7 presents quantitative results hippocampal subfields MA segmentation with three sample datasets. Table 8.3 lists quantitative for overall DSC performance in detail. BDRE performs overall DSC of mean (± standard deviation) of 0.66 (± 0.045) that significantly improves the average hippocampal subfield segmentation compared with using 3T only, DN only and SR only.

Specifically, BDRE MA performs significantly better with all 8 hippocampal subfields structures when comparing BDRE and T1 MA segmentation; 5 out of total 8 hippocampal subfields structures when comparing BDRE and T1 images preprocessed by DN; and all 8 hippocampal subfields structures when comparing BDRE and T1 images preprocessed by SR.

We also summarized Mean Surface Distance (MSD) and Hausdorff Distance (HD) performance as reference that present in Figure 8.8-8.11 and Table 8.4-8.5.

8.3 Discussion & Conclusion

Herein, we have presented a technique to use BDRE to enhance the image for deep brain structure MA segmentation. The full structure DSC performance results shows that BDRE

Table 8.4: Average performance of hippocampal subfields MA segmentation approaches using different modalities as atlases and target. Average overall Mean Surface Distance (MSD) ($\pm$ standard deviation(SD)) and median MSD are shown. Each Wilcox paired test is calculated between BDRE (as reference) and other corresponding scenarios, where *** represents Wilcox paired t-test with $p<0.001$.

Atlases + target	**3T *****	**DN *****	**SR *****	**BDRE (ref)**
MSD (mean$\pm$SD)	0.638 ($\pm$ 0.162)	0.601 ($\pm$ 0.138)	0.642 ($\pm$ 0.161)	0.492 ($\pm$ 0.107)
MSD (median)	0.607	0.586	0.598	0.454

Table 8.5: Average performance of hippocampal subfields MA segmentation approaches using different modalities as atlases and target. Average overall Hausdorff Distance (HD) ($\pm$ standard deviation(SD)) and median MSD are shown. Each Wilcox paired test is calculated between BDRE (as reference) and other corresponding scenarios, where *** represents Wilcox paired t-test with $p<0.001$.

Atlases + target	**3T *****	**DN *****	**SR *****	**BDRE (ref)**
HD (mean$\pm$SD)	3.935 ($\pm$ 1.031)	3.836 ($\pm$ 1.112)	3.795 ($\pm$ 0.933)	3.114 ($\pm$ 0.624)
HD (median)	3.816	3.574	3.448	2.989

can significantly improve full thalamus and hippocampal subfields segmentation accuracy. For full thalamus segmentation, BDRE significantly improves full thalamus segmentation accuracy with 9.1%, 15.6%, 6.9%, and 16.2% improvement over using 3T imaging only, 7T imaging only, DN only and SR only. The hippocampal subfields segmentation results in 8.7%, 8.4% and 8.6% improved DSC performance over utilizing T1 weighted modality only, DN only and SR only. We found that there were several structures when only using 3T MRI or 3T MRI preprocessed by DN outperformed BDRE. We also found that the overall performance when only using 7T MRI as atlases and target was worse than the other four approaches. Although the ground truth thalamus nuclei are segmented in 7T modality, the labels were not depicted by only using one 7T image but multiple imaging sequences. So it can explain 7T MA performs even worse. Due to our limit number of 8 datasets, and the manual reproducibility of full thalamus is also very difficult, more datasets to validate should help us better understand the utilities of BDREs enhanced thalamic nuclei. Furthermore, our work is not trying to design a best pipeline, (e.g., in [155], Liu et al. directly segment the thalamic nuclei in 3T MRI using high resolution shape models which are manually delineated in high-field MR images. They start from entire thalamus shape and hierarchically fit joint models to capture the relationship between the thalamus and the internal nuclei and leads to a better result compared with our work), rather, this chapter shows the value of the BDREs contrast.

The BDRE approach has the potential to improve multi-atlas segmentation for other deep brain structures and imaging protocols and offers enhanced contrast for other applications. We note that the intensity images produced by BDRE exaggerate contrast at boundaries and highlight intensity cues that might not be obvious in the original image. Some of the boundaries may be artefactual (i.e., a byproduct of the registration method and/or data archive). Here, we sidestep issues of the validity of these boundaries and focus instead on the utility of the contrasts for driving image processing in situations where a true anatomical label is available via an alternative imaging modality (which is less routinely

collected and more time consuming than typical 3T T1 MRI). For further understanding of BDRE and future validation, we plan to evaluate different numbers of atlases and/or utilize different time consuming non-rigid registration tools with smaller number of atlases for computation efficiency to browse if they impact the quality of deep brain structure contrast.

For this chapter, our main purpose is to find the usefulness of BDRE which is generated by large datasets. Using weighted averaging is a topic that has been highly studied, and tuning the weights is an important future step to improve BDRE. For MA segmentation validation, we also tested on using multimodal approach for atlases (combined with original T1 MRI + BDRE) and the target was either T1 MRI or BDRE, the performance of the above approach was worse than only using BDRE as input atlases and target (results not shown).

For the potential time consuming issue of generating a BDRE image, 7 hours for one image is time consuming, but trying fast non-linear on large datasets (i.e. >1000) is a new approach with interesting result (new contrast) which would take tremendous time when using traditional registration tool. Deep learning super resolution is in real time, but they are not focused on deep brain structures. In this chapter, we think currently it is irrelevant to compare BDRE with a deep learning approach since we focus on discovering an atlas with a new imaging modality, which also has potential to train a deep network.

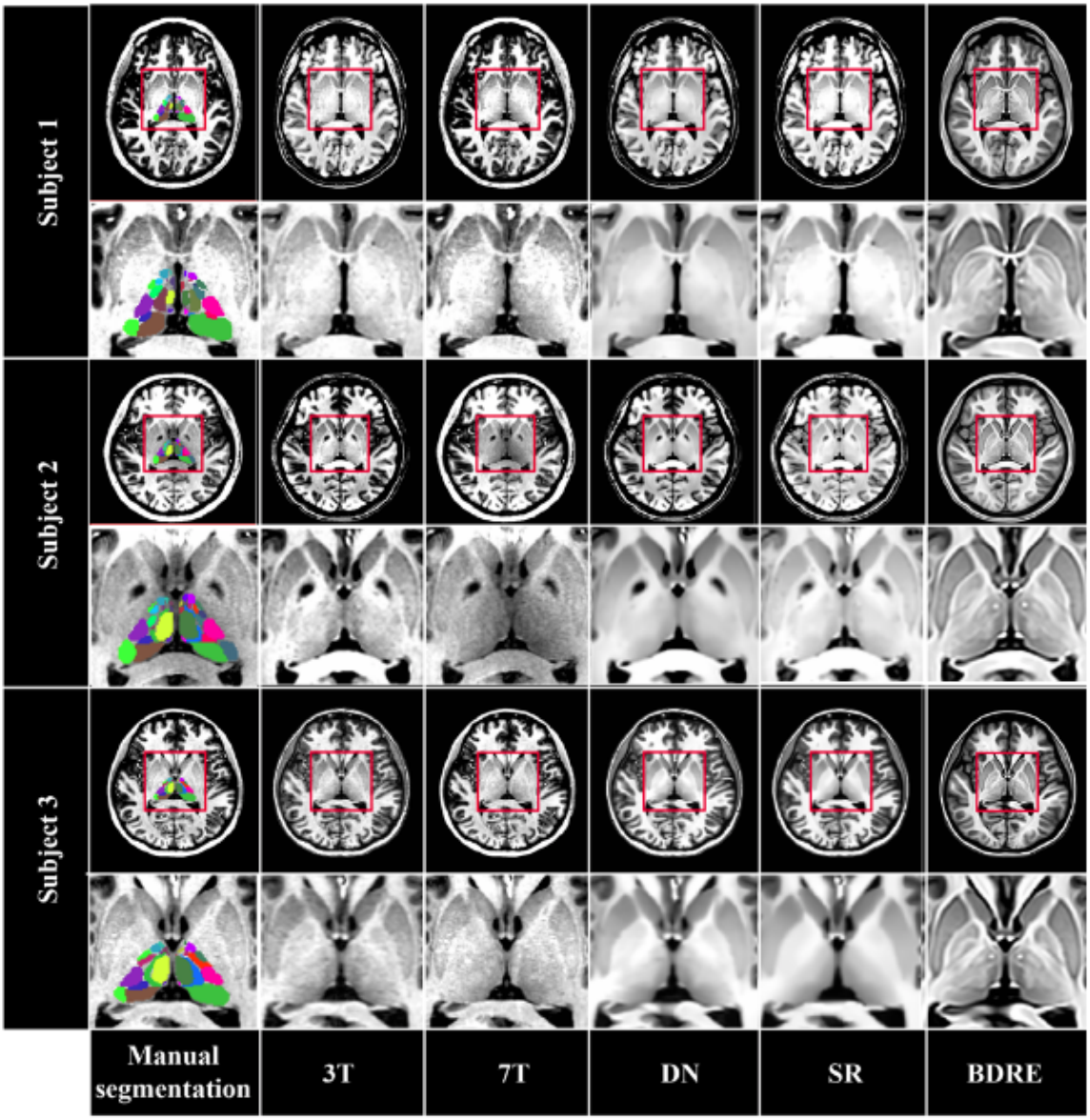


Figure 8.1: Three samples of 7T T1-weighted raw images with ground truth manual thalamic nuclei reference segmentation labels. Paired 3T T1-weighted MRI images are shown. Each BDRE, DN and SR preprocessed images are generated by the corresponding 3T image

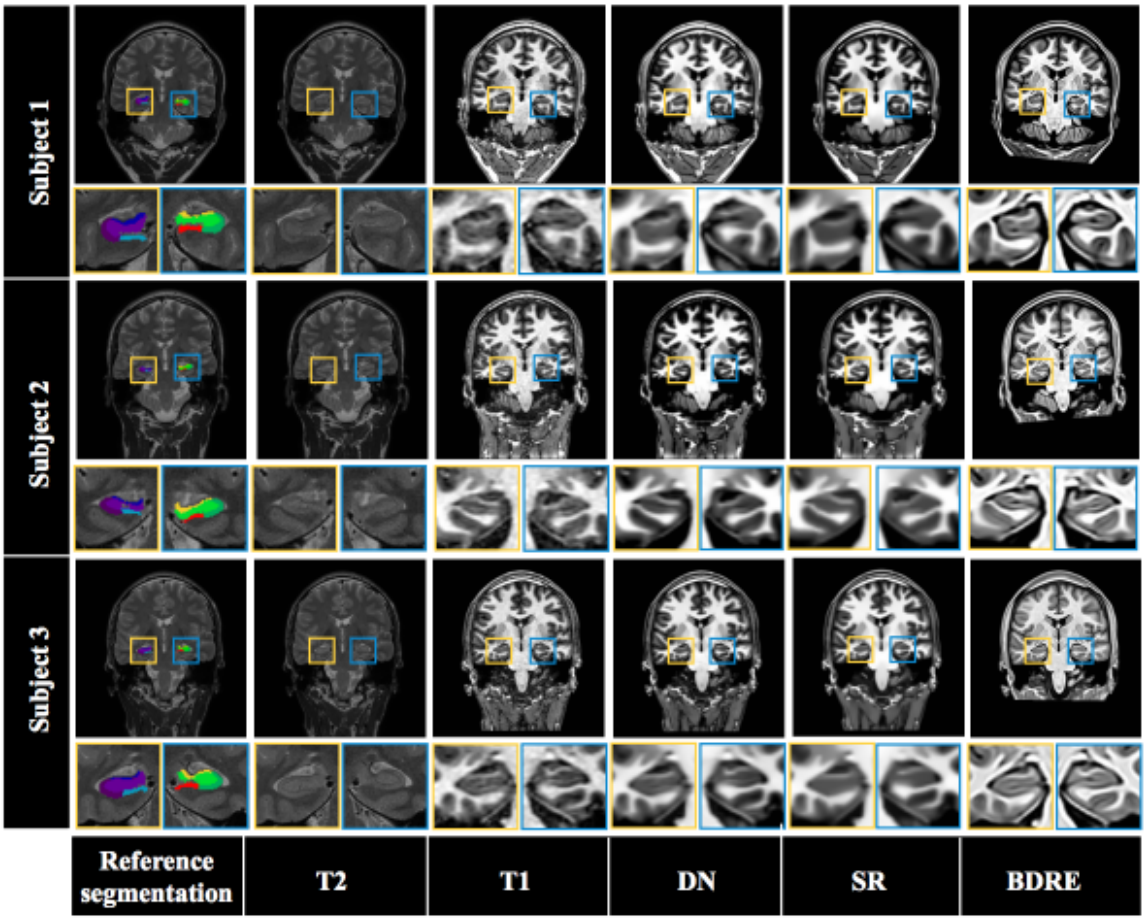


Figure 8.2: Three samples of T2-weighted MRI with hippocampal subfields reference segmentation standard labels (generated by automatic segmentation of hippocampal subfields (ASHS) pipeline). Paired T1-weighted MRI images are shown. Each BDRE, DN and SR preprocessed images are generated by the corresponding T1 image.

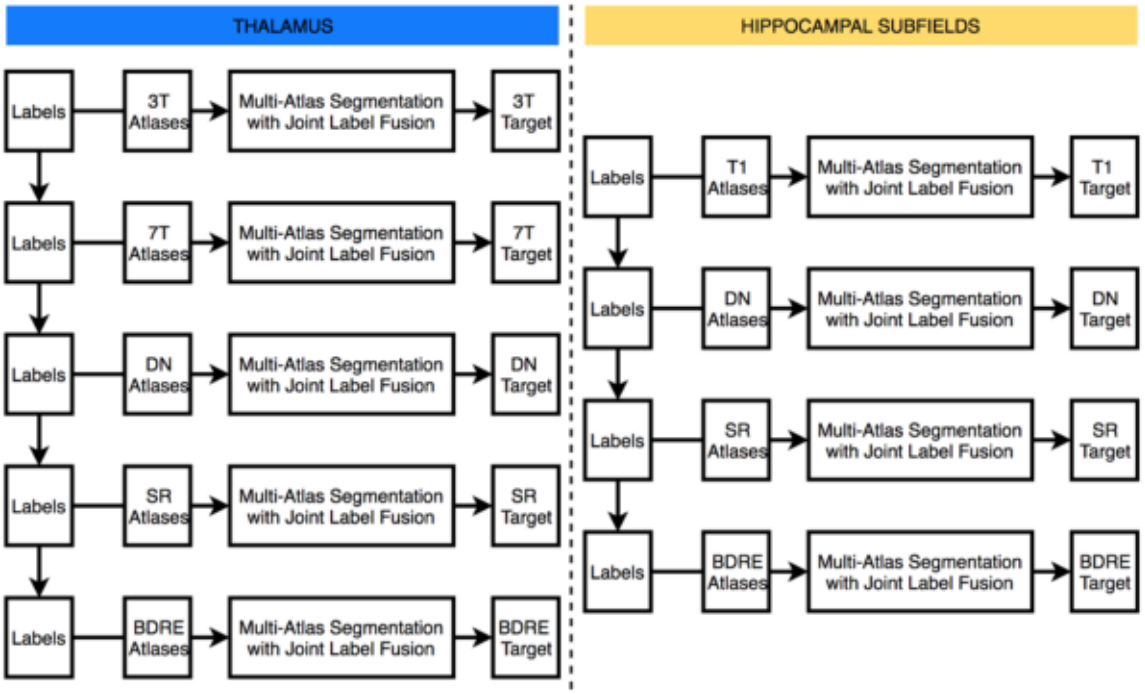


Figure 8.3: Workflow for validation of BDRE usefulness for MA. For the thalamic nuclei segmentation, BDRE atlases to BDRE target approach is compared with using 3T T1, 7T T1. For the hippocampal subfields, 3T T1 imaging is used. Each scenarios of BDRE is also compared with T1 images that are preprocessed by DN and SR.

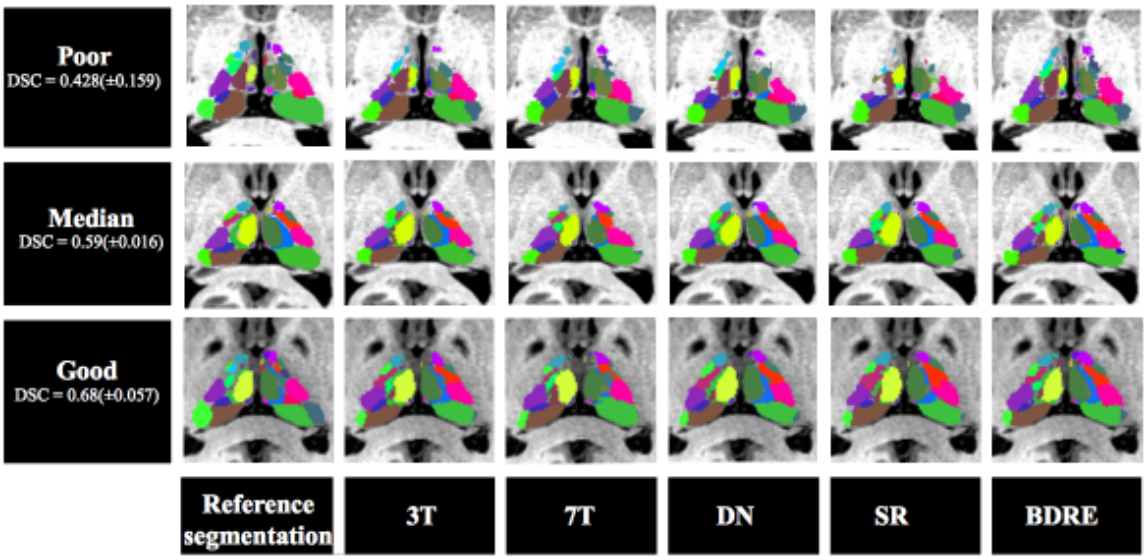


Figure 8.4: Qualitative result for thalamic nuclei MA segmentation validation including ground truth reference segmentation, 3T MRI only, 7T MRI only, DN only, SR only and BDRE only. Three samples were selected by average overall DSC across each MA segmentation approach using different modalities, which were ordered by worst, median and best mean DSC similarity ($\pm$ standard deviation).

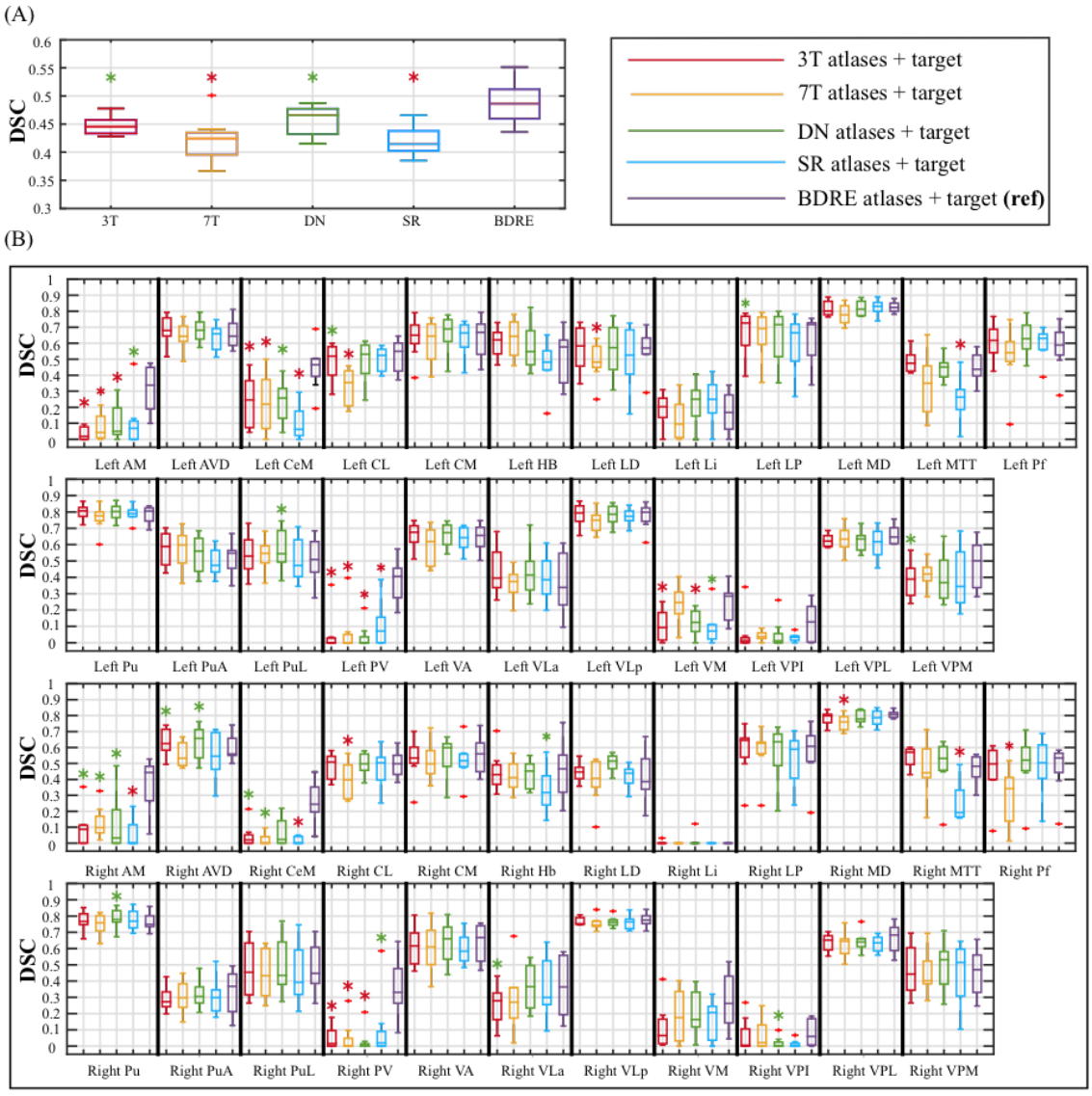


Figure 8.5: Quantitative result for thalamic nuclei MA segmentation including 3T MRI only, 7T MRI only, DN only, SR only and BDRE only. (A) is the full thalamic nuclei MA segmentation DSC performance for each scenario; (B) is left /right thalamic nuclei MA segmentation DSC on each label based. Each Wilcox paired t-test is calculated between BDRE (as reference) and other corresponding modalities. Red * represents Wilcox paired t-test with $p<0.01$, and green * represents $p<0.05$.

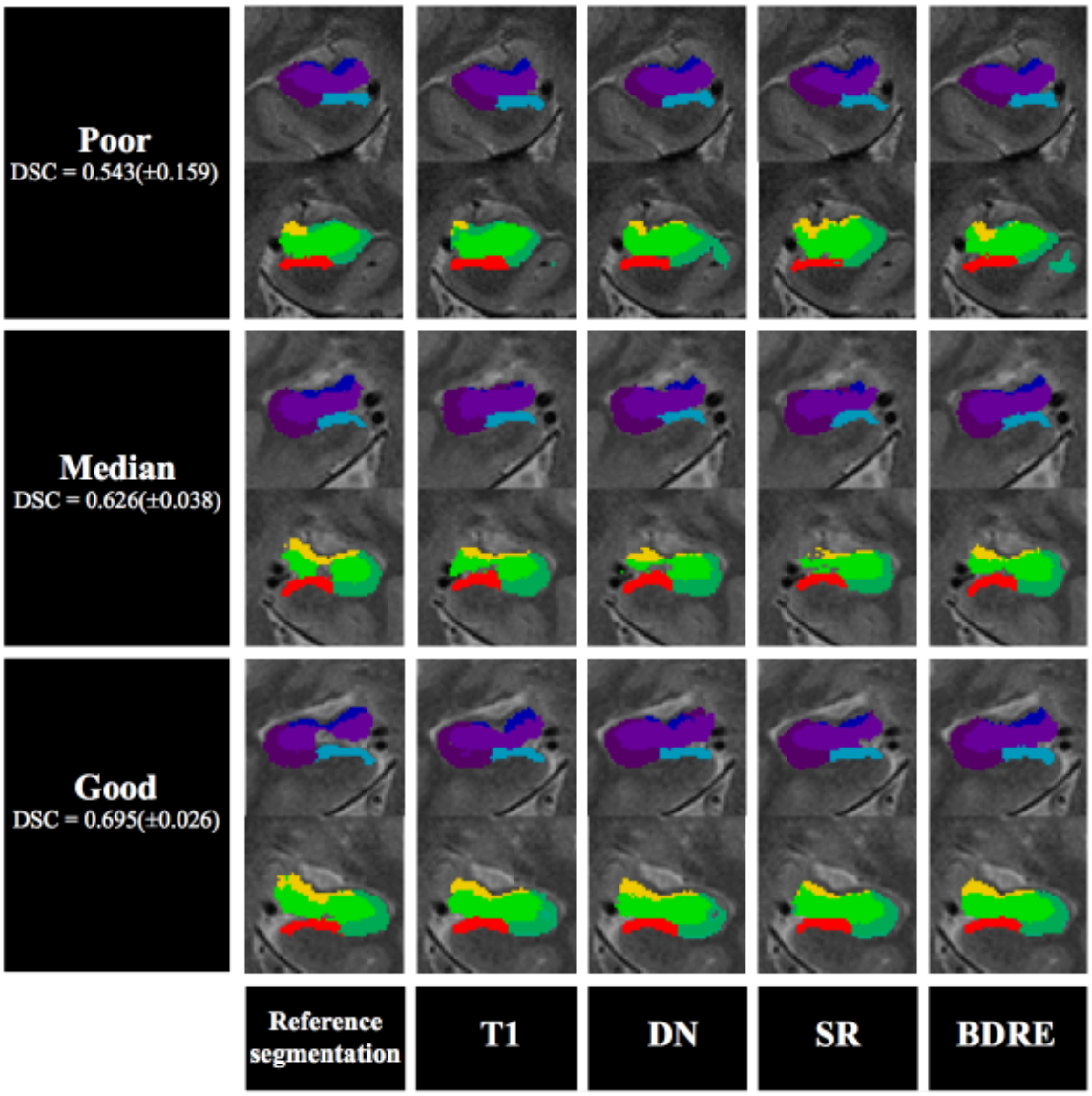


Figure 8.6: Qualitative result for hippocampal subfields MA segmentation validation including reference segmentation standards from ASHS pipeline, T1 MRI only, DN only, SR only and BDRE only. Three samples were selected by average overall DSC across each MA segmentation approach using different modalities, which were ordered by worst, median and best mean DSC similarity ($\pm$ standard deviation).

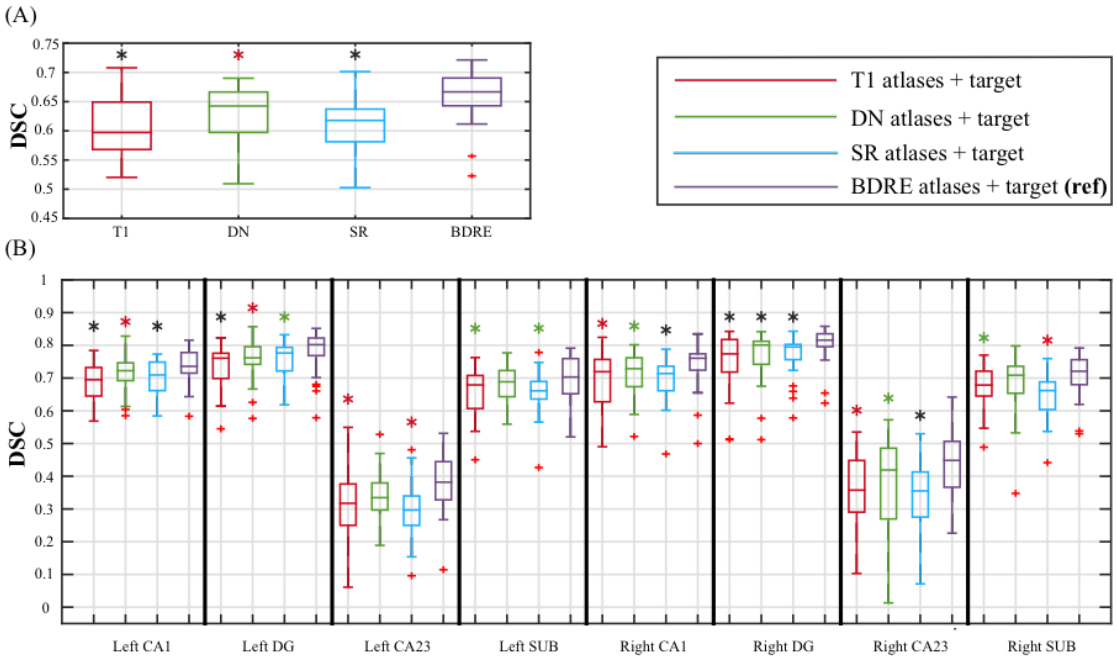


Figure 8.7: Average performance of hippocampal subfields MA segmentation approaches using different modalities as atlases and target. Average Dice similarity ($\pm$ standard deviation) are shown. Each Wilcox paired test is calculated between BDRE (as reference) and other corresponding scenarios, where *** represents Wilcox paired t-test with $p<0.001$, and ** represents $p<0.01$.

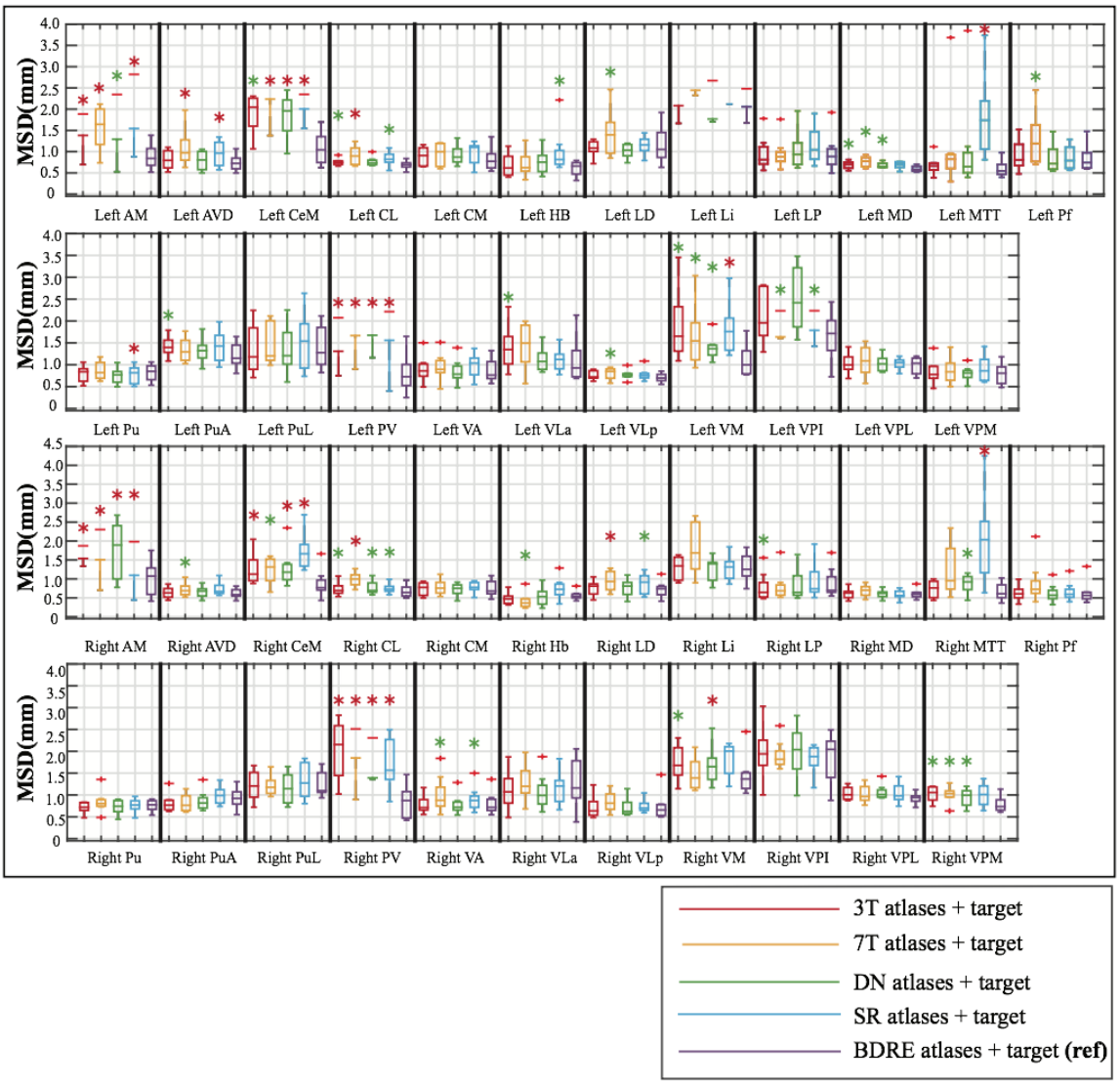


Figure 8.8: Quantitative result for thalamic nuclei MA segmentation including 3T MRI only, 7T MRI only, DN only, SR only and BDRE only. The plot shows left /right thalamic nuclei MA segmentation Mean Surface Distance (MSD) on each label based. The plots omit infinite MSD. Each Wilcox paired t-test is calculated between BDRE (as reference) and other corresponding modalities. Red * represents Wilcox paired t-test with $p<0.01$, and green * represents $p<0.05$.

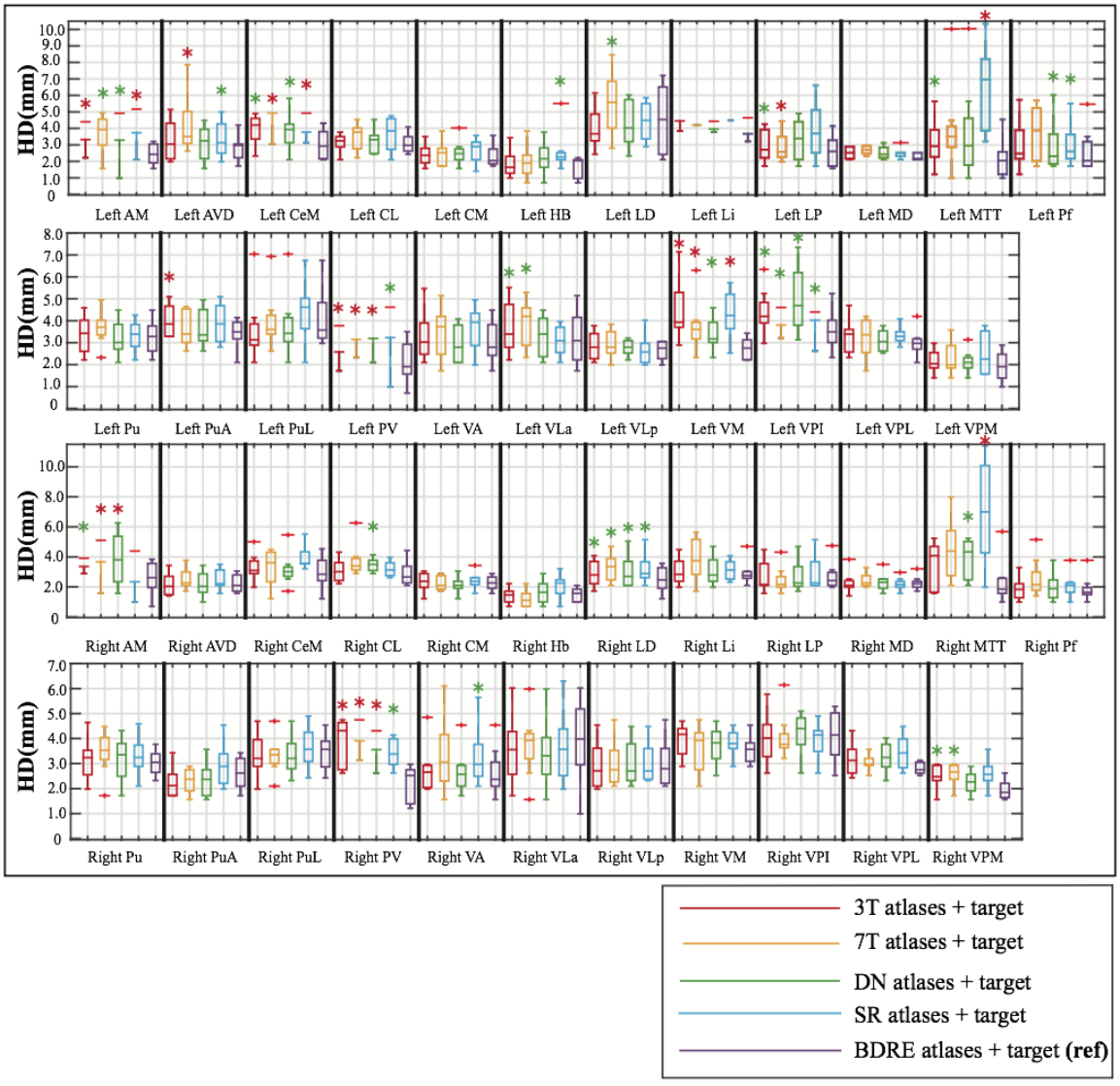


Figure 8.9: Quantitative result for thalamic nuclei MA segmentation including 3T MRI only, 7T MRI only, DN only, SR only and BDRE only. The plot shows left /right thalamic nuclei MA segmentation Hausdorff Distance (HD) on each label based. The plots omit infinite HD. Each Wilcox paired t-test is calculated between BDRE (as reference) and other corresponding modalities. Red * represents Wilcox paired t-test with $p<0.01$, and green * represents $p<0.05$.

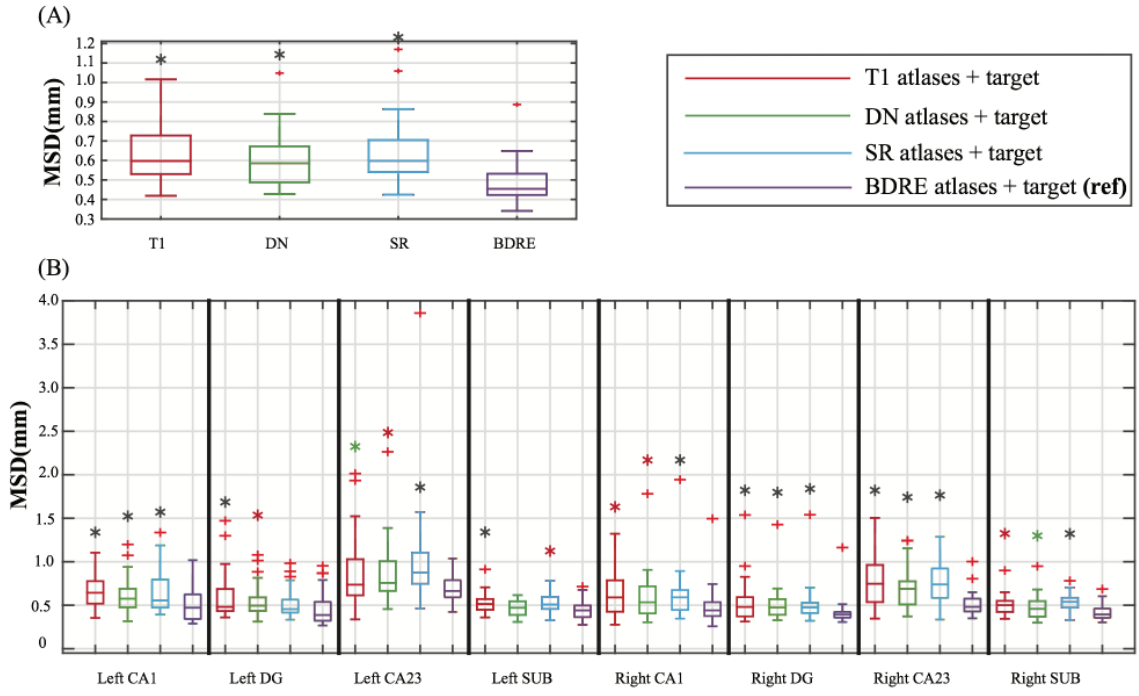


Figure 8.10: Quantitative result for hippocampal subfields MA segmentation including T1 MRI only, DN only, SR only and BDRE only. (A) is the hippocampal subfields MA segmentation Mean Surface Distance (MSD) for each scenario; (B) is left /right hippocampal subfields MA segmentation MSD on each label based. Each Wilcox paired t-test is calculated between BDRE (as reference) and other corresponding modalities. Black * represents Wilcox paired t-test with $p<0.001$, Red * represents $p<0.01$, and green * represents $p<0.05$.

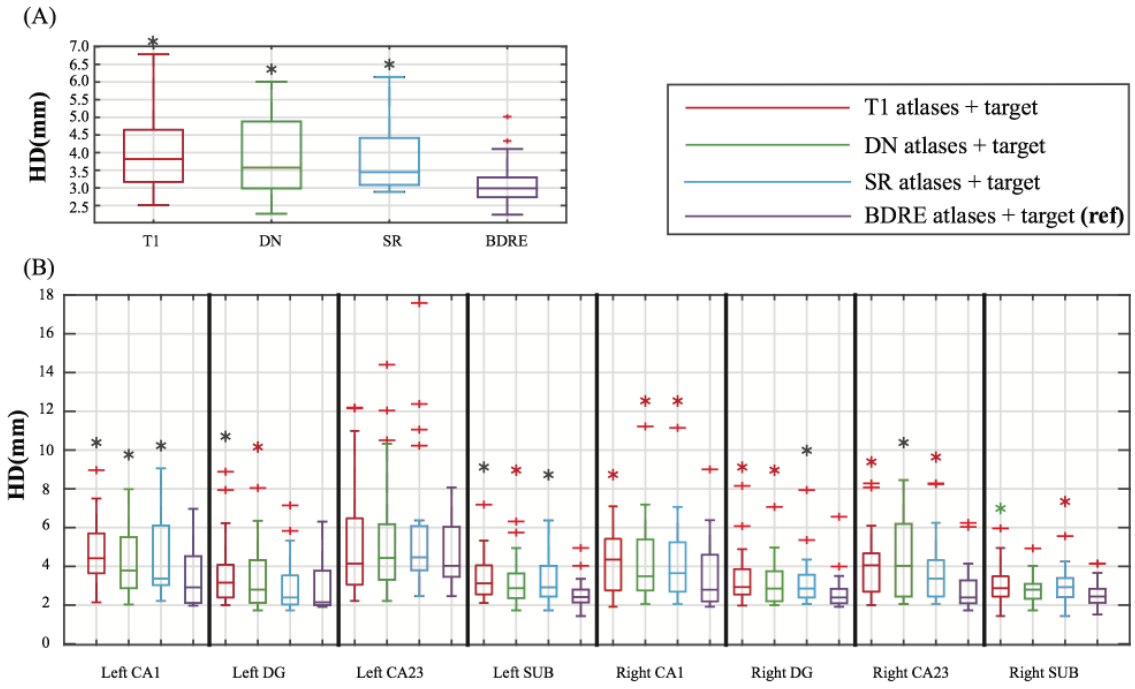


Figure 8.11: Quantitative result for hippocampal subfields MA segmentation including T1 MRI only, DN only, SR only and BDRE only. (A) is the hippocampal subfields MA segmentation Hausdorff Distance (HD) for each scenario; (B) is left /right hippocampal subfields MA segmentation HD on each label based. Each Wilcox paired t-test is calculated between BDRE (as reference) and other corresponding modalities. Black * represents Wilcox paired t-test with $p<0.001$, Red * represents $p<0.01$, and green * represents $p<0.05$.

Chapter 9

INTEGRATE HADOOP DISTRIBUTED FILE SYSTEM WITH GEOGRAPHICALLY DISTRIBUTED HIGH PERFORMANCE FILE SYSTEM

9.1 Problem Overview

The Apache Hadoop ecosystem aims to use commodity hardware and network setup to provide a distributed storage and computation environment. The Hadoop Distributed File System (HDFS) is a data storage system. It can split a large file into several small chunks, and distributed / replicated each chunk across a cluster. MapReduce is a built-in Hadoop computation template that can process split large files chunk in parallel and summarize intermediate result efficiently. To achieve best efforts with such types of applications, data locality is very important since moving computation to storage is cost friendly especially for those computation clusters with limited network bandwidth (e.g, one Gigabit). However, when moving forward to high performance computing (HPC) environment, fast speed network is (>ten Gigabit) commonly used. To run MapReduce types of jobs, we need a separate Hadoop cluster set inside the HPC cluster since resource management (YARN) is needed to operate MapReduce jobs. However, if we can fully employ existence HPC storage, separation between computation and storage is achievable without involving much data transfer overhead owing to fast network, and we can still use existing computation nodes in HPC for MapReduce tasks.

Currently, in Vanderbilt's high performance computing cluster called Advanced Computing Center for Research and Education (ACCRE), IBMs General Parallel File System (GPFS) is the main storage solution for routinely HPC application. Meanwhile, we have an alternative cheaper solution: LStore file system, which is a distributed, scalable and geographically dispersed storage. L-storage aims to provide a peta level storage for the

compact muon solenoid (CMS) experiment [17] that uses commodity hard drives.

In our recent work [128, 102], we developed a simulator for modeling the behavior of processing different types and amounts of medical image processing jobs between solely using a separate Hadoop cluster and regular SLURM job scheduling mechanism with GPFS in HPC environment. To support our claim, we collected anonymous trace statistics logs of completed jobs from the XNAT system [138] for 4 years worth of multi-stage analysis without accessing any Protected Health Information (PHI) or imaging data. These jobs were executed in ACCRE. The ACCRE cluster is deployed with a 10 Gigabit Ethernet on 12 racks with IBMs 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 (in total 152.45 CPU years). We present a simulation engine suite to estimate the performance of running medical image processing on Hadoop-based approach (decentralized storage) versus the traditional cluster (centralized storage). The simulation result shows that in order to maximize the benefit for I/O intensive types of application (processing time less than 2 hours), data locality is important and matters network saturation. However, when considering more long runing jobs (1 day up to 9 days), data locality matters not that much. It is important for us that when moving forward to use Hadoop ecosystem to HPC environment with fast network, seeking a feasible, cheap and geographically distributed storage is urgent. GPFS is feasible but very expensive to utilize and maintain. Thus we consider integrating Hadoop cluster with LStore file system.

LStore has two interfaces for client to attach the data: a fuse mountable client and HPC command line interfaces. A naïve solution is to copy data from LStore and load it to a Hadoop cluster, process the data and send the result back to LStore. However, it would inevitably involve much redundant cost and unnecessary operations, since data would be stored and replicated in both LStore and HDFS side. Another naïve solution is put all Hadoop related directory for Namenode and Datanode on fuse mount directory. Current

fuse mount interface can only take 128 Kbytes chunk size per read / write operation by default. LStore aims to store and process peta-level's file, the limited I/O chunk size will easily degrade data streaming performance and furtherly impact HDFS performance.

The fundamental solution is to build up a bridge from HDFS to LStore (before involving HBase inspired from our previous effort HadoopBase-MIP [128]) to separate computation and storage, and still use commodity hardware with keeping all features of LStore storage (e.g., distributed, geographically dispersed, data integrity schemes). Because HDFS read block size is tunable (>10 MB), and for write, the size of buffer for loading sequence files is also tunable from (4KB-16 MB). Thus, if we can bypass fuse mount and directly integrate a plugin of HDFS native interface to LStore command line interfaces (command line tool can configure flexible read block size and stream buffer for read / write), data streaming performance of the plugin should be much better than LStore fuse mount. In summary, the main challenges of this work are (1) how to avoid data redundancy on HDFS side once data already available on LStore side and (2) how to ease the effort of implementing the plugin integration by choosing LStore fuse client interface or command line based interface. Figure 9.2 illustrates the problem that we deal with of this work among Hadoop, GPFS and LStore.

Our contributions are summarized as follows:

- We integrated a plugin between HDFS interface and LStore to provide regular input / output stream and other file system operation. The plugin can avoid data copying redundancy between HDFS and LStore.
- We empirically verified our HDFS LStore plugin on in-house shared environment HPC and standalone HPC. The performance of the plugin was compared with a two current LStore client interface: LStore command line interface (ideal case) and LStore fuse mounted client interface (baseline).

This work has not been submitted yet.

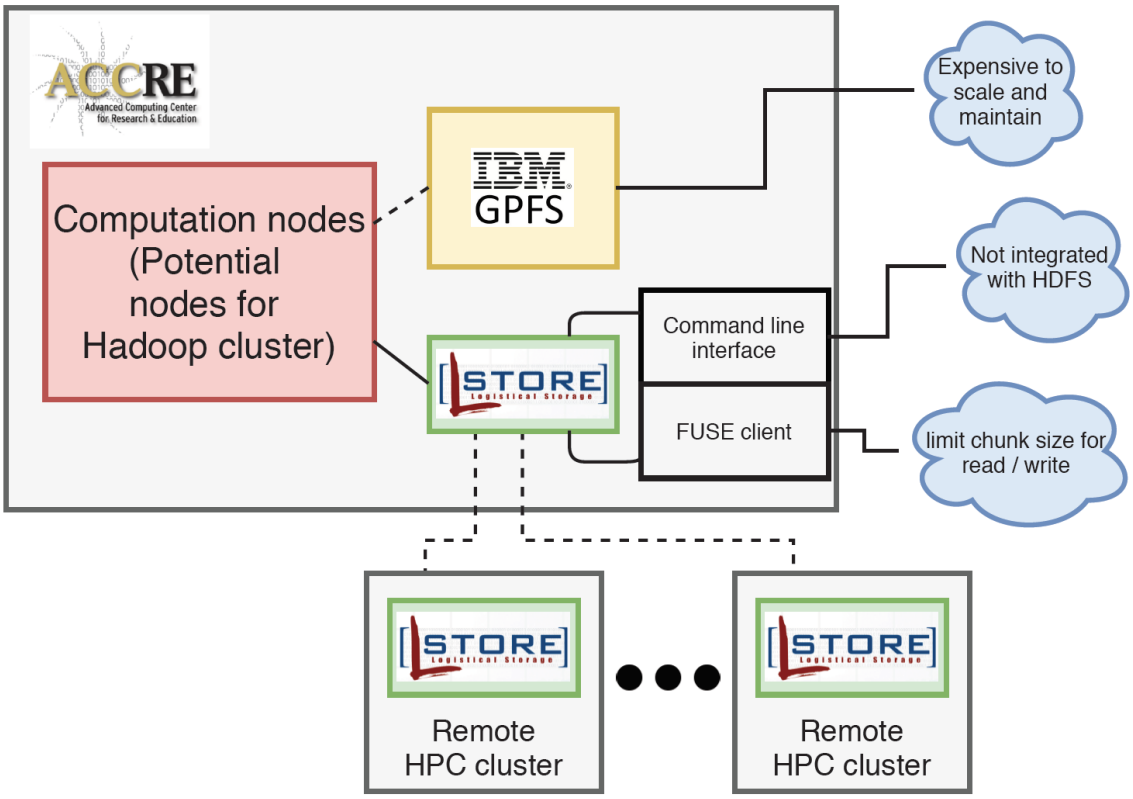


Figure 9.1: Main problems for ACCRE cluster integrates HDFS with HPC file system.

9.2 Methods

Considering our previous effort on HadoopBase-MIP, moving forward to HPC environment is reasonable to scale up the system and utilize HPC computation power. GPFS is the current solution, which is expensive to maintain and scale up. LStore as a cheap, distributed and geographically dispersed is ideally to use to integrate with Hadoop. The physical storage unit of LStore can be a single hard drive or a single disk or a collection of disks in RAID 6 array. The unit is called depot. The file is stored as block separately. Similar with UNIX's metadata mechanism inode [161], LStore uses a container called ExNodes to store the file's physical location, attributes etc, which provide a mapping from LStore to physical data layer. For this preliminary work, we are not dealing or optimizing underlying LStore system design. We designed, implemented and verifed the integration

of HDFS and LStore first.

9.2.1 System Design

As we discussed above, our goal is to bypass the usage of LStore fuse mount client and directly integrate HDFS interface with LStore command line tool. It can ease the effort of input / output data stream between HDFS and LStore and further benefit MapReduce throughput. Besides HDFS block size per read operation and the size of buffer for use in sequence files is tunable, we can also customize data blocks size for each of split large file. In HDFS, the block size is pre-set as 256 MB by default. LStore high performance command line interface is implemented in C language. The key component of this structure is implement a Java Native Interface (JNI) to LStore client. For LStore side, we make a shared wrapper library. A customized schema (LStore://) for LStore is registered in HDFS. For Hadoop side, a class LStoreFilesystem is extended from org.apache.hadoop.fs.FileSystem, and a class LStoreBaseFile defines file operation of open / close / seek / read / write / append. Figure-2 illustrates the data loading workflow of a native call involving JNI. And Table 9.1 shows the detail of each JNI call that weve implemented.

9.2.2 Experiment design

Currently there are two system metrics we care most: the throughput of (multiple) read and the throughput (multiple) write. And there should be three test scenarios: (1) LStore command line interface, which is ideal case (2) LStore fuse client interface, which is the baseline and potential worst case and (3) HDFS LStore plugin interface. Due to the limit of test machines memory and buffer setting on LStore command line interface, there are maximally 8 files with size of 10 GB can be sequentially read simultaneity. Thus 8 identical files with size of 10 GB are pre-write to both dedicate network connected depots and in-house shared LStore. Moreover, files loading via in-house standalone depot are an ideal test cases since it is not shared publicly. For in-house shared LStore, it is real world verification

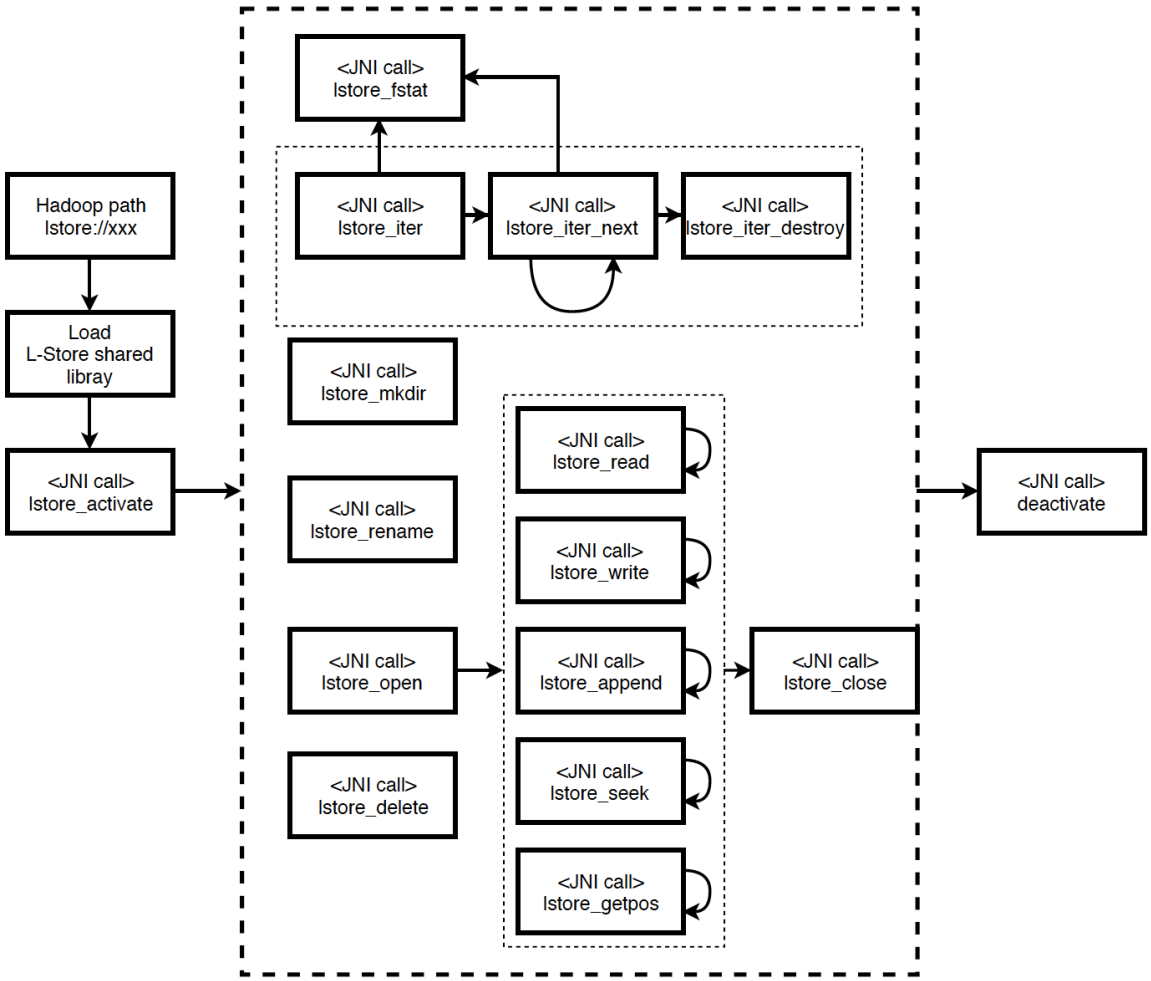


Figure 9.2: JNI implementation workflow via HDFS to LStore

since the depot belongs to routinely on-going CMS research, which is shared by multiple groups on campus. For write performance, we only test on in-house HPC [1].

Specifically, 8 sets of multiple reads are performed for each of three test interfaces. Namely we perform multiple read range from 1-8 files simultaneously. Similarly, writing test are based on performance of in parallel loading same file on test machine and output identical 1-8 files to LStore. For fuse scenario, cleaning cache is necessary before each set of test. We empirically set read buffer size for HDFS as 80MB (same with LStore command

[1]For dedicate connection scenario, the depots heterogeneous clustered, some of them are old and slow. So the write performance would be random which depends on which depots that the file would be put on. It would result 50% read degradation if the file is written to old depots with poor drive. In order to get consistent write performance, we ignore the write testing to dedicate depot sets

Table 9.1: JNI implementation from HDFS to integrate with LStore command line tool's core interface

JNI	**Description**
LStore_activate	Start LStore file system when loading the shared library
LStore_deactivate	Shutdown LStore file system
LStore_delete	Remove a file with optionally recursive the path
LStore_fstat	Parse the returned file attributes and stores them in a customized data structure for Hadoop FileStatus object
LStore_fstat_iter	Returns a fstat iterator
LStore_fstat_iter_next	Returns the next iterator object
LStore_fstat_iter_destroy	Destroys a fstat iterator
LStore_mkdir	make a directory in LStore
LStore_rename	Rename an object
LStore_open	Opens an existing targeting object
LStore_close	Closes an existing targeting object
LStore_read	Read data from the target
LStore_write	Write data to the target
LStore_append	Write data to the target (append mode)
LStore_seek	Set the current file position
LStore_getpos	Get the current file position

line interface), and set files io.file.buffer.size in Hadoop core-site.xml as 8MB. The rest of parameters setting in Hadoop will be available soon at https://github.com/accre/LStore/.

9.2.3 Hardware

The Hadoop package version we use is 2.7.5. LStore version is 0.5. Because this work focus on evaluating the data read / write performance of sequential files via HDFS LStore plugin integration, only single machine as Namenode and Datanode is enought for HDFS. The test client machine is deployed with twelve Xeon(R) E5520 CPU cores with 23.5 memory 23.5 GB available. We tested LStore uses single LServer with 80 depots on ACCRE, which are shared by multiple cients; and 9 depots are also set within ACCRE using dedicate connection for testing ideal scenairo. Each depot is connected with a 10 Gigabit WAN Ethernet.

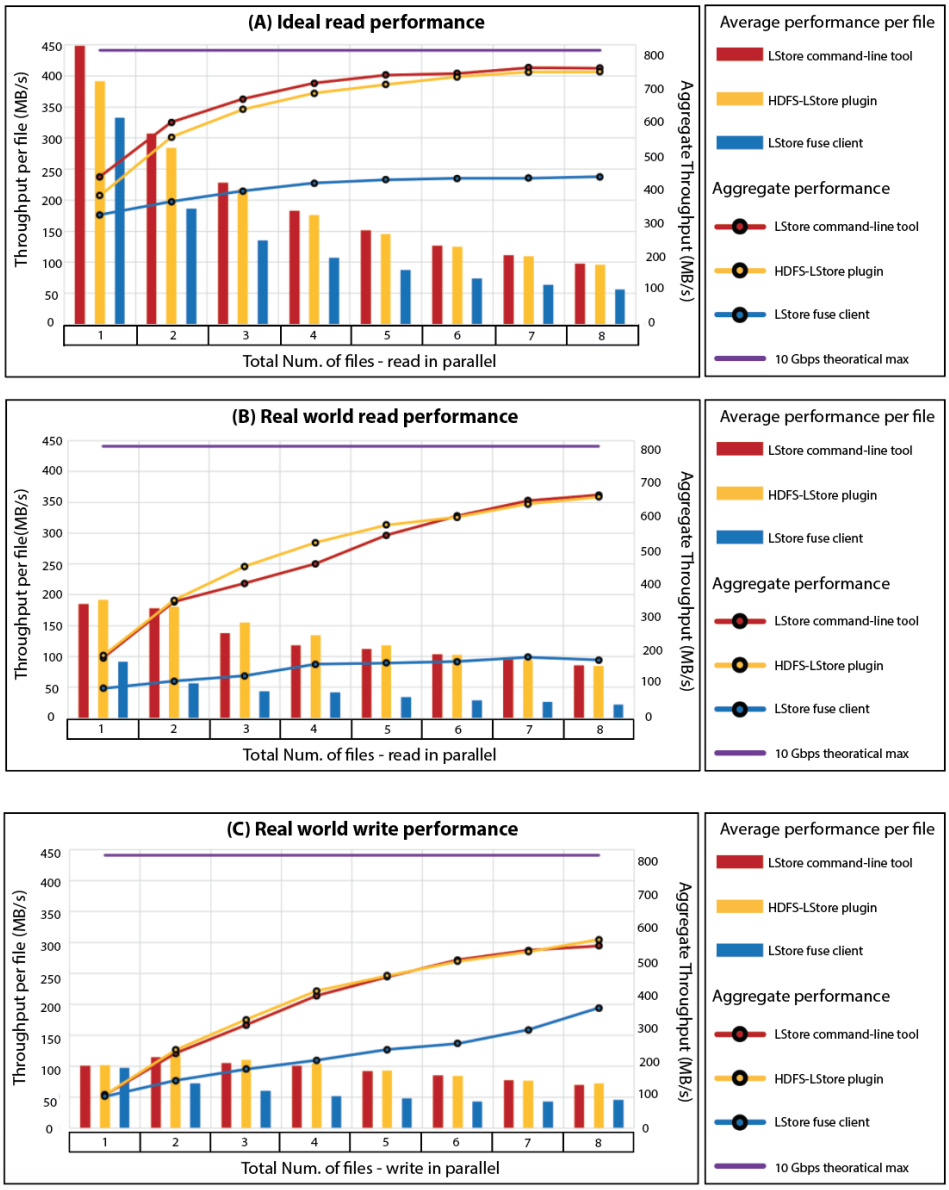


Figure 9.3: Full empirical experiment result summary.

Figure 9.3(A) and Table 9.2 present standalone read performance via three interfaces when data retrieval incurs long distance transfer, which is a ideal scenario since the depot bandwidth is only shared by us. With the total number of files that are simultaneously read increased, the read throughput is decreased, which is our expected result. We can also see that when reading more files in parallel, the performance difference between LStore

command line interface and HDFS LStore plugin are moderate. Although LStore command line scenario represents the ideal case, some of read performance are a bit worse than HDFS LStore plugin. It is acceptable because of the variation of network environment. In overall, HDFS LStore achieves 52%-73% throughput multi-file read improvement when it compares with corresponding fuse mount.

Similarly, Figure 9.3(B) and Table 9.2 present in-house HPC LStore read performance for three different data access interfaces. As a real world test environment, we can see the performance LStore command line interface and HDFS LStore plugin tie to each other. LStore command line interface seems to involve more variance within same test case, but it needs further to verify. As the aspect of HDFS LStore plugins performance versus fuse mount client, HDFS LStore plugin outperform with 219% - 280% throughput improvement.

Finally, for write performance through in-house LStore, the result is displayed in Figure 9.3(C) and Table 9.2. Fuse mount client write performance seems to saturate around 45 MB/s when writing more than five files simultaneity. HDFS LStore plugin employs 57% - 102% better improvement than mountable fuse interface. The write performance of LStore command line tool and HDFS LStore plugin are still comparable among each sub-test cases.

9.4 Discussion and Conclusion

When moving Apache Hadoop ecosystem to HPC environment, cost efficiency is still an important factor since Hadoop aims to use commodity hardware and network, while HPC cluster usually deploys with expensive file system such as lsture and GPFS. As an alternative cheap HPC file system, LStore brings us an opportunity to integrate HDFS with such a distributed, scalable and geographically dispersed storage. In this work, we introduce how the integration is implemented between LStore command line interfaces and HDFS, which bypass LStore fuse client due to its limited chunk size per read / write oper-

Table 9.2: Full empirical experiment result summary for mean±standard deviation read write throughput performance. Two tails t-test is calculated between HDFS LStore plugin (the reference) and other two interfaces. "*" means statistically difference with p-value <0.05.

		mean +/- SD (MB/s)		
Test scenario	**Total number of files**	**LStore command line tool**	**HDFS LStore plugin (ref)**	**LStore fuse client**
Read from dedicate depots in parallel	1	448 +/- 2.3 *	392 +/- 13	333 +/- 7.9 *
	2	307 +/- 4.6 *	285 +/- 6	187 +/- 0.9 *
	3	228 +/- 5 *	218 +/- 5.6	135 +/- 1.3 *
	4	183 +/- 2.8 *	176 +/- 4	107 +/- 0.9 *
	5	152 +/- 2.7 *	146 +/- 8.6	88 +/- 0.9 *
	6	127 +/- 3.5 *	125 +/- 5.4	74 +/- 0.9 *
	7	112 +/- 1.8 *	110 +/- 5.2	64 +/- 0.9 *
	8	97 +/- 3.6	96 +/- 4.2	56 +/- 0.8 *
Read from in-house HPC cluster in parallel	1	185 +/- 3.7 *	192 +/- 2.2	92 +/- 7.7 *
	2	178 +/- 28.5	181 +/- 20	56 +/- 2.9 *
	3	137 +/- 29.8 *	155 +/- 11.5	43 +/- 2.4 *
	4	118 +/- 30.1 *	134 +/- 7.8	41 +/- 1.3 *
	5	112 +/- 17.5 *	118 +/- 6.2	34 +/- 0.9 *
	6	103 +/- 16.1	103 +/- 15.1	29 +/- 1 *
	7	95 +/- 11	94 +/- 9.6	27 +/- 1.9 *
	8	85 +/- 8.7	85 +/- 7.5	22 +/- 0.6 *
Write to in-house HPC cluster in parallel	1	101 +/- 21.7	102 +/- 33.4	97 +/- 8.1
	2	114 +/- 5.6	119 +/- 14.9	73 +/- 6.6 *
	3	104 +/- 14.6	110 +/- 8.1	60 +/- 3.9 *
	4	101 +/- 1.4	105 +/- 3.4	52 +/- 1.1 *
	5	92 +/- 8.2	93 +/- 11.6	48 +/- 1.3 *
	6	86 +/- 5.7	85 +/- 3.1	43 +/- 1.4 *
	7	77 +/- 8.9	77 +/- 5.6	43 +/- 1.7 *
	8	70 +/- 13.5	72 +/- 9	46 +/- 0.5 *

ation. Our empirical experiment result verifies the HDFS LStore data loading efficiency.

Chapter 10

CONCLUSION

Billions of magnetic resonance imaging (MRI) and computed tomography (CT) images on millions of individuals are currently stored in radiology archives [107]. These imaging data files are estimated to constitute one-third of the global storage demand [108], but are effectively trapped on storage media.

In this work, we first demonstrated how medical image processing can share the benfit of using cloud (AWS) to deploy large dataset processing when user short of computation resource. We then introduce the HadoopBase-MIP, which is a data colocation framework for dealing with big data problem. Specifically, the problem is about high volume of unstructured / small medical image files. The medical image computing community has heavily invested in algorithms, software, and expertise in technologies that assume that imaging volumes can be accessed in their entirety as needed (and without substantial penalty). Despite the promise of big data, traditional MapReduce and distributed machine learning frameworks (e.g., Apache Spark) are not often considered appropriate for "traditional" / "simple" parallelization. In this paper, we demonstrate that HadoopBase-MIP can be used in place of a PBS cluster (e.g., Sun Grid Engine) and can be offered as a cloud-based service. Moreover, with our approach, even a naïve application of HBase results in improved performance over NAS using the same computation and network infrastructure. We proposed theoretical way to better understanding the feasibility of HadoopBase-MIP and developing more applications with different scale and levels of analysis including a novel multi-level incremental quality assurance framework. During the development of HadoopBase-MIP, we found a new MRI contrast via doing big data non rigid registration averaging that enhanced deep brain structures. Rather than validate each boundarys' correctness in BDRE due to difficulties of direct visualization of our target structures, we

investigated the usefulness of BDRE for the multi-atlas (MA) segmentation on those deep brain structures (thalamic nuclei and hippocampal subfields) were assume only T1 images were available. And finally, we presented a preliminary plugin framework that integrate HDFS with LStore, where LStore is a HPC based distributed, scalable and geographically dispersed file system.

10.1 Summary of Ph.D. Contributions

• Integration of the Java Image Science Toolkit with Amazon Web Service

1. For the Java Image Science Toolkit (JIST) to Amazon Web Service (AWS)

2. Completed cost/benefit analysis of a pipeline that can be easily parallelized and required memory that would limit the processes to serially execution only on a local lab machine.

• HadoopBase-MIP: A data colocation grid framework for big data Medical Image Processing using Hadoop & HBase

1. Implemented a novel data colocation strategy for improving latency and throughput of processing.

2. Defined Big Data by comparing the performance between HadoopBase-MIP and a traditional cluster with Sun Grid Engine in theoretical and empirical way.

3. Developed MapReduce templates for different types of analysis: single image processing; group based analysis; large dataset summary statistics (also described an optimal criterion to find the splitting chunk size of the large dataset).

4. Implemented an off-line load balancer to better allocating data in a heterogeneous cluster for processing efficiency.

5. Designed a HBase Table design scheme to enable fast data query and boot up the of MapReduce performance.

6. Developed a simulation engine suite to estimate the performance of running medical image processing on traditional cluster (centralized storage) versus the HadoopHBae-MIP (decentralized storage).

7. Designed and implemented a semi-automated, real-time quality assurance (QA) model monitor and checkpoint framework which aims to optimize the performance of medical image processing by finding anomalies in the first level processing in a timely manner thereby expediting the entire multi-level analysis.

• Big data non-rigid registration-based image enhancement improves deep brain structures

1. Discovered a new contrast that enhance the contrast of deep-brain structure by large volume of fast non-rigid registration. The target structures were two deep brain structures: thalamic nuclei and hippocampal subfields.

2. Explored the usefulness of proposed new contrast imaging modality by multi atlas segmentation.

• Integrate Hadoop Distributed File System with Geographically Distributed High Performance File System

1. We integrated a plugin between HDFS interface and LStore to provide regular input / output stream and other file system operation. The plugin can avoid data copying redundancy between HDFS and LStore.

2. We empirically verified our HDFS LStore plugin on in-house HPC and long distance geographically WAN connected remote HPC cluster. The performance of the plugin was compared with a two current LStore client interface: LStore command line interface (ideal case) and LStore fuse mounted client interface (baseline).

10.2 Future Work

First of all, there is an open task of writing a summary journal paper for HadoopBase-MIP is necessary, which should introduce all features and innovations with system design strategies.

For deep brain enhancement via BDRE, we would like to verify different medical image registration tool with different number of atlases for big data averaging. We would like to evaluate the usefulness not only for Multi-Atlas segmentation, but also for deep learning. Our current effort shows that to make a correct verification, it needs to be very careful to align the enhanced image affine correctly to original target space. Of course, a cleverer alignment strategy is ideal rather than manual checking.

The performance of whole brain segmentation is improved by pre-registering all training volumes to 305 MNI template, which is a routinely steps. However, there is no such an abdomen template like MNI space template for brain currently. Creating a standard template for abdomen is challenging due to high variability of structure in abdomen area. Our goal is to create an unbiased abdomen template as a reference for deep learning network - which is a very popular recent trend for doing medical image segmentation - to improve the accuracy of learning process. Of course, without a promise of using fast non-rigid registration tool and availability of big volume of abdomen data, this work is not achievable. We have successfully configured an automatic cropping as first pre-processing step developed in [162, 163]. Then registration would be based on cropped images. We have found cropping help better registration but is still not good enough. We are currently using a classic 3D Unet to verify the performance of potential template, it seems that cropping leads to better whole abdomen segmentation, unfortunately, potential template makes result worse. Some manual corrections are needed for correct registration and we need to further work on this promising project.

Another innovation should be easy to verify, namely using BDRE mechanism to enhance a group of abdomen atlases. Those enhanced images would be trained as second

channel for helping whole abdomen organ segmentation.

Finally, for HDFS LStore integration project, we have not fully tested the MapReduce types of jobs performance using LStore as backend, our next step is to setup a multi-nodes commercialized cluster for HDFS and compare the performance of between using local storage and LStore plugin. Since HDFS LStore integration has been implemented, more empirical tests should be done to find the bottleneck and challenging problem. Another future work is to seek a mapping from HDFS Namenode to LStores Exnode, which should give us several optimizing opportunities to design a HDFS LStore customized fault tolerant and replication strategy for large dataset processing. Due to HDFS and LStores namespace is easily exploded by large small volume of data, deploying HadoopBase-MIP to deal with medical image processing in HPC environment is challenging but optimistic since it uses HBase, which is built upon HDFS and deals with large volume of small files. Finally, our ultimate goal is to propose a cheap solution of using LStore to replace GPFS solution without experience a lot of performance loss for Hadoop system. Finally, we fully understand LStore is an existence solution available for us to utilize and optimize, my ultimate goal is to employ the power of domain specific language modeling from my previous expertise in [164],to provide a generic integration from HDFS to any high performance computing storage.

10.2.1 Dissertation time line

Finally, here is a summary of my thesis' tasks and research time line.

- Task 0: Experiment with medical image analysis in cloud - chapter 3
- Task 1: Data colocation grid prototype for Medical image processing-as-a-service - chapter 4
- Task 2: Establishing theoretical bounds - chapter 5
- Task 3: HadoopBase-MIP optimization -chapter 6

- Task 4: HadoopBase-MIP enablers for optimzing traditional multi-level Medical image analysis - chapter 7
- Task 5: Big data registration based image enhancement - chapter 8
- Task 6: Integrate HDFS with LStore (a geographically distributed high performance file system) - chapter 9

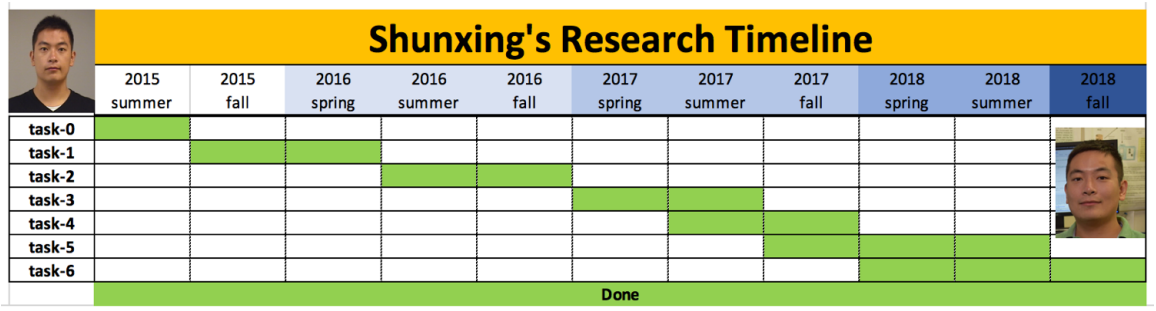


Figure 10.1: Research timeline

Appendix A

Publications

A.1 Journal Articles

1. **Shunxing Bao**, Camilo Bermudez, Yuankai Huo, Prasanna Parvathaneni, William Rodriguez, Pierre-Franois DHaese, Maureen McHugo, Stephan Heckers, Benoit M. Dawant, Ilwoo Lyu, Bennett A. Landman. "Registration-based Image Enhancement Improves Multi-Atlas Segmentation of the Thalamic Nuclei and Hippocampal Subfields" Magnetic Resonance Imaging (Submitted)

2. Yuankai Huo, Zhoubing Xu, Hyeonsoo Moon, **Shunxing Bao**, Albert Assad, Tamara K. Moyo, Michael R. Savona, Richard G. Abramson, Bennett A. Landman. "SynSegNet: Synthetic Segmentation Without Target Modality Ground Truth". IEEE Transactions on Medical Imaging. 2018.

3. Yuankai Huo, Justin Blaber, Stephen M. Damon, Brian D. Boyd, **Shunxing Bao**, Prasanna Parvathaneni, Camilo Bermudez Noguera, Shikha Chaganti, Vishwesh Nath, Greer M. Jasmine, Ilwoo Lyu, William R. French, Allen T. Newton, Baxter P. Rogers, Bennett A. Landman. "Towards Portable Large-Scale Image Processing with High-Performance Computing." Journal of Digital Imaging (2018): 1-11.

4. Yuankai Huo, Zhoubing Xu, **Shunxing Bao**, Camilo Bermudez, Hyeonsoo Moon, Prasanna Parvathaneni, Tamara K. Moyo, Michael R. Savona, Albert Assad, Richard G. Abramson, and Bennett A. Landman. "Splenomegaly Segmentation on Multi-modal MRI using Deep Convolutional Networks ". IEEE Transaction on Medical Imaging

5. Steve Damon, Sahil Panjwani, **Shunxing Bao**, Peter Kochunov, Bennett A. Land-

man. ”Integration of the Java Image Science Toolkit with E-Science Platform.” In-Sight Journal. 2016.

A.2 Highly Selective Conference Publications

1. **Shunxing Bao**, Prasanna Parvathaneni, Yuankai Huo, Yogesh Barve, Andrew J. Plassard, Yuang Yao, Hongyang Sun, Ilwoo Lyu, David H. Zald, Bennett A. Landman and Aniruddha Gokhale. ” Technology Enablers for Cloud-based Multi-level Analysis Applications in Medical Image Processing ” IEEE BigData 2018, Seattle. (accepted)

2. **Shunxing Bao**, Andrew J. Plassard, Bennett A. Landman, and Aniruddha Gokhale. ”Cloud engineering principles and technology enablers for medical image processing-as-a-service.” In Cloud Engineering (IC2E), 2017 IEEE International Conference on, pp. 127-137. IEEE, 2017.

3. **Shunxing Bao**, Joe Porter, and Aniruddha Gokhale. ”Reasoning for CPS Education Using Surrogate Simulation Models.” In Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual, vol. 1, pp. 764-773. IEEE, 2016.

4. Yuankai Huo, Zhoubing Xu, Katherine Aboud, Prasanna Parvathaneni, **Shunxing Bao**, Camilo Bermudez, Laurie E. Cutting, Susan M. Resnick, and Bennett A. Landman. ”Spatially Localized Atlas Network Tiles Enables 3D Whole Brain Segmentation from Limited Data” MICCAI 2018

A.3 Conference Publications

1. **Shunxing Bao**, Yuankai Huo, Prasanna Parvathaneni, Andrew J. Plassard, Camilo Bermudez, Yuang Yao, Ilwoo Lyu, Aniruddha Gokhale, and Bennett A. Landman. ”A data colocation grid framework for big data medical image processing: backend

design." In Medical Imaging 2018: Imaging Informatics for Healthcare, Research, and Applications, vol. 10579, p. 105790A. International Society for Optics and Photonics, 2018.

2. **Shunxing Bao**, Frederick D. Weitendorf, Andrew J. Plassard, Yuankai Huo, Aniruddha Gokhale, and Bennett A. Landman. "Theoretical and empirical comparison of big data image processing with apache hadoop and sun grid engine." In Medical Imaging 2017: Imaging Informatics for Healthcare, Research, and Applications, vol. 10138, p. 101380B. International Society for Optics and Photonics, 2017.

3. **Shunxing Bao**, Stephen M. Damon, Bennett A. Landman, and Aniruddha Gokhale. "Performance management of high performance computing for medical image processing in Amazon Web Services." In Medical Imaging 2016: PACS and Imaging Informatics: Next Generation and Innovations, vol. 9789, p. 97890Q. International Society for Optics and Photonics, 2016.

4. Yuankai Huo, Zhoubing Xu, **Shunxing Bao**, Albert Assad, Richard G. Abramson, and Bennett A. Landman. "Adversarial synthesis learning enables segmentation without target modality ground truth." In Biomedical Imaging (ISBI 2018), 2018 IEEE 15th International Symposium on, pp. 1217-1220. IEEE, 2018.

5. Yuankai Huo, Zhoubing Xu, **Shunxing Bao**, Camilo Bermudez, Andrew J. Plassard, Jiaqi Liu, Yuang Yao, Albert Assad, Richard G. Abramson, and Bennett A. Landman. "Splenomegaly segmentation using global convolutional kernels and conditional generative adversarial networks." In Medical Imaging 2018: Image Processing, vol. 10574, p. 1057409. International Society for Optics and Photonics, 2018.

6. Yuankai, **Shunxing Bao**, Prasanna Parvathaneni, and Bennett A. Landman. "Improved stability of whole brain surface parcellation with multi-atlas segmentation." In Medical Imaging 2018: Image Processing, vol. 10574, p. 1057438. International

Society for Optics and Photonics, 2018.

7. Meg F. Bobo, **Shunxing Bao**, Yuankai Huo, Yuang Yao, Jack Virostko, Andrew J. Plassard, Ilwoo Lyu et al. "Fully convolutional neural networks improve abdominal organ segmentation." In Medical Imaging 2018: Image Processing, vol. 10574, p. 105742V. International Society for Optics and Photonics, 2018.

8. Prasanna Parvathaneni, **Shunxing Bao** , Allison Hainline , Yuankai Huo , Kurt G. Schilling , Hakmook Kang , Owen Williams , Neil D. Woodward , Susan M. Resnick , David H. Zald , Ilwoo Lyu , Bennett A. Landman "Harmonization of white and gray matter features in diffusion microarchitecture for cross sectional studies. In International Conference on Clinical and Medical Image Analysis 2018 (ICCMIA18)

9. Jiachen Wang, Yuankai Huo, **Shunxing Bao**, Yunxi Xiong, Sanja L. Antic, Travis J. Osterman, Pierre P. Massion, Bennett A. Landman. "Lung Cancer Detection using Co-learning from Chest CT Images and Clinical Demographics" In SPIE Medical Imaging, International Society for Optics and Photonics, 2019. (Accepted)

10. Yuankai Huo, James G. Terry, Jiachen Wang, Vishwesh Nath, Camilo Bermudez, **Shunxing Bao**, Prasanna Parvathaneni, J. Jeffery Carr, and Bennett A. Landman "Coronary Calcium Detection using 3D Attention Identical Dual Deep Network Based on Weakly Supervised Learning." In SPIE Medical Imaging, International Society for Optics and Photonics, 2019. (Accepted)

11. Cailey I. Kerley, Yuankai Huo, Shikha Chaganti, **Shunxing Bao**, Mayur B. Patel, Bennett A. Landman. "Montage based 3D Medical Image Retrieval from Traumatic Brain Injury Cohort using Deep Convolutional Neural Network." In SPIE Medical Imaging, International Society for Optics and Photonics, 2019. (Accepted)

A.4 Conference Abstracts

1. **Shunxing Bao**, Bennett Landman, and Aniruddha Gokhale. ”Algorithmic Enhancements to Big Data Computing Frameworks for Medical Image Processing.” In Cloud Engineering (IC2E), 2017 IEEE International Conference on, pp. 13-16. IEEE, 2017.

2. **Shunxing Bao**, Aniruddha Gokhale, Sherif Abdelwahed, and Shivakumar Sastry. ”Model-Predictive Controllers for Performance Management of Composable Conveyor Systems.” In 21st IEEE Real-Time and Embedded Technology and Applications Symposium, p. 25. 2015.

Appendix B

Biography

Shunxing Bao was born in Shaoxing, in Oct 1989, and raised in Beijing, China. He received his B.S. from the Huazhong University of Science and Technology (China) in 2008 with software engineering major. Then he joined Vanderbilt University and earned his master degree in 2014. His master thesis was about cloud-based domain-specific modeling design. He continued to work as a Ph.D. student at Vanderbilt University. As a graduate research associate, his primary research was employing and optimizing the distributed computing system technologies to solve challenging issues in big data medical image analysis. During his graduate student life, he worked with three research groups, Medical-image Analysis and Statistical Interpretation (MASI) Lab, Distributed Object Computing (DOC) Group and Advanced Computing Center for Research and Education (ACCRE).

BIBLIOGRAPHY

[1] Yuankai Huo, Andrew J Plassard, Aaron Carass, Susan M Resnick, Dzung L Pham, Jerry L Prince, and Bennett A Landman. Consistent cortical reconstruction and multi-atlas brain segmentation. *NeuroImage*, 138:197–210, 2016.

[2] Yuankai Huo, Aaron Carass, Susan M Resnick, Dzung L Pham, Jerry L Prince, and Bennett A Landman. Combining multi-atlas segmentation with brain surface estimation. In *Proceedings of SPIE–the International Society for Optical Engineering*, volume 9784. NIH Public Access, 2016.

[3] Apache HBase Team. *Apache hbase reference guide*. Apache, version 2.0.0 edition, April 2016.

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

[5] Rebecca Smith-Bindman, Diana L Miglioretti, and Eric B Larson. Rising use of diagnostic medical imaging in a large integrated health system. *Health affairs*, 27(6):1491–1502, 2008.

[6] Baraa Mohamad, Laurent d'Orazio, and Le Gruenwald. Towards a Hybrid Row-column Database for a Cloud-based Medical Data Management System. In *Proceedings of the 1st International Workshop on Cloud Intelligence*, page 2. ACM, 2012.

[7] Carol McDonald. An in-depth look at the hbase architecture, 2015.

[8] Brian B Avants, Nicholas J Tustison, Gang Song, Philip A Cook, Arno Klein, and James C Gee. A reproducible evaluation of ants similarity metric performance in brain image registration. *Neuroimage*, 54(3):2033–2044, 2011.

[9] Brian B Avants, Nick Tustison, and Gang Song. Advanced normalization tools (ants). *Insight j*, 2:1–35, 2009.

[10] Stephen M Smith, Mark Jenkinson, Heidi Johansen-Berg, Daniel Rueckert, Thomas E Nichols, Clare E Mackay, Kate E Watkins, Olga Ciccarelli, M Zaheer Cader, Paul M Matthews, et al. Tract-based spatial statistics: voxelwise analysis of multi-subject diffusion data. *Neuroimage*, 31(4):1487–1505, 2006.

[11] Prasanna Parvathaneni, Baxter P Rogers, Yuankai Huo, Kurt G Schilling, Allison E Hainline, Adam W Anderson, Neil D Woodward, and Bennett 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.

[12] Anisha Keshavan, Esha Datta, Ian M McDonough, Christopher R Madan, Kesshi Jordan, and Roland G Henry. Mindcontrol: A web application for brain segmentation quality control. *NeuroImage*, 2017.

[13] Jimit Doshi, Guray Erus, Yangming Ou, Susan M Resnick, Ruben C Gur, Raquel E Gur, Theodore D Satterthwaite, Susan Furth, Christos Davatzikos, Alzheimer's Neuroimaging 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.

[14] Sandra González-Villà, Arnau Oliver, Sergi Valverde, Liping Wang, Reyer Zwiggelaar, and Xavier Lladó. A review on brain structures segmentation in magnetic resonance imaging. *Artificial intelligence in medicine*, 73:45–69, 2016.

[15] Mark W Woolrich, Saad Jbabdi, Brian Patenaude, Michael Chappell, Salima Makni, Timothy Behrens, Christian Beckmann, Mark Jenkinson, and Stephen M Smith. Bayesian analysis of neuroimaging data in fsl. *Neuroimage*, 45(1):S173–S186, 2009.

[16] Frank B Schmuck and Roger L Haskin. Gpfs: A shared-disk file system for large computing clusters. In *FAST*, volume 2, 2002.

[17] Serguei Chatrchyan, EA de Wolf, et al. The cms experiment at the cern lhc. *Journal of instrumentation.-Bristol, 2006, currens*, 3:S08004–1, 2008.

[18] Micah Beck, Terry Moore, and James S Plank. An end-to-end approach to globally scalable network storage. In *ACM SIGCOMM Computer Communication Review*, volume 32, pages 339–346. ACM, 2002.

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

[20] Amazon Elastic Compute Cloud. Amazon web services. *Retrieved November*, 9:2011, 2011.

[21] Donald Robinson. *Amazon Web Services Made Simple: Learn how Amazon EC2, S3, SimpleDB and SQS Web Services enables you to reach business goals faster*. Emereo Pty Ltd, 2008.

[22] Blake C Lucas, John A Bogovic, Aaron Carass, Pierre-Louis Bazin, Jerry L Prince, Dzung L Pham, and Bennett A Landman. The java image science toolkit (jist) for rapid prototyping and publishing of neuroimaging software. *Neuroinformatics*, 8(1):5–17, 2010.

[23] Bo Li, Frederick Bryan, and Bennett A Landman. Next generation of the java image science toolkit (jist): visualization and validation. *The insight journal*, 2012:1, 2012.

[24] Matthew J McAuliffe, Francois M Lalonde, Delia McGarry, William Gandler, Karl Csaky, and Benes L Trus. Medical image processing, analysis and visualization in clinical research. In *Computer-Based Medical Systems, 2001. CBMS 2001. Proceedings. 14th IEEE Symposium on*, pages 381–386. IEEE, 2001.

[25] V Hernández et al. Bridging clinical information systems and grid middleware: a medical data manager. In *Challenges and Opportunities of Healthgrids: Proceedings of Healthgrid 2006*, volume 120, page 14. IOS Press, 2006.

[26] Wolfgang 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.

[27] Antoine Rosset, Luca Spadola, and Osman Ratib. Osirix: an open-source software for navigating in multidimensional dicom images. *Journal of digital imaging*, 17(3):205–216, 2004.

[28] Naidila Sadashiv and SM Dilip Kumar. Cluster, grid and cloud computing: A detailed comparison. In *Computer Science & Education (ICCSE), 2011 6th International Conference on*, pages 477–482. IEEE, 2011.

[29] Emmanuel Medernach. Workload analysis of a cluster in a grid environment. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 36–61. Springer, 2005.

[30] Raja Appuswamy, Christos Gkantsidis, Dushyanth Narayanan, Orion Hodson, and Antony Rowstron. Scale-up vs scale-out for hadoop: Time to rethink? In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 20. ACM, 2013.

[31] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A Distributed Storage System for Structured Data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[32] Dimitrios Markonis, Roger Schaer, Ivan Eggel, Henning Müller, and Adrien Depeursinge. Using MapReduce for Large-scale Medical Image Analysis. *arXiv preprint arXiv:1510.06937*, 2015.

[33] Said Jai-Andaloussi, Abdeljalil Elabdouli, Abdelmajid Chaffai, Nabil Madrane, and Abderrahim Sekkaki. Medical Content-based Image Retrieval by using the Hadoop Framework. In *20th International Conference on Telecommunications (ICT), 2013*, pages 1–5. IEEE, 2013.

[34] Yahia Chabane, Laurent d'Orazio, Le Gruenwald, Baraa Mohamad, and Christophe Rey. Medical Data Management in the SYSEO Project. *ACM SIGMOD Record*, 42(3):48–53, 2013.

[35] Tinghuai Ma, Xichao Xu, Meili Tang, Yuanfeng Jin, and Wenhai Shen. MHBase: A Distributed Real-Time Query Scheme for Meteorological Data Based on HBase. *Future Internet*, 8(1):6, 2016.

[36] Youzhong Ma, Jia Rao, Weisong Hu, Xiaofeng Meng, Xu Han, Yu Zhang, Yunpeng Chai, and Chunqiu Liu. An efficient index for massive iot data in cloud environment. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2129–2133. ACM, 2012.

[37] Qingcheng Li, Ye Lu, Xiaoli Gong, and Jin Zhang. Optimizational method of hbase multi-dimensional data query based on hilbert space-filling curve. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2014 Ninth International Conference on*, pages 469–474. IEEE, 2014.

[38] Shicai Wang, Ioannis Pandis, Chao Wu, Sijin He, David Johnson, Ibrahim Emam, Florian Guitton, and Yike Guo. High dimensional biological data retrieval optimization with nosql technology. *BMC genomics*, 15(8):1, 2014.

[39] Seungkyun Hong, Minhee Cho, Sungho Shin, CN Seon, SK Song, et al. Optimizing hbase table scheme for marketing strategy suggestion. In *2016 8th International Conference on Knowledge and Smart Technology (KST)*, pages 313–316. IEEE, 2016.

[40] Henri Casanova. Simgrid: A toolkit for the simulation of application scheduling. In *Cluster computing and the grid, 2001. proceedings. first ieee/acm international symposium on*, pages 430–437. IEEE, 2001.

[41] Rajkumar Buyya and Manzur Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and computation: practice and experience*, 14(13-15):1175–1220, 2002.

[42] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011.

[43] Saurabh Kumar Garg and Rajkumar Buyya. Networkcloudsim: Modelling parallel applications in cloud simulations. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 105–113. IEEE, 2011.

[44] Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62(3):1263–1283, 2012.

[45] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012.

[46] Zoltán Ádám Mann. Allocation of virtual machines in cloud data centersa survey of problem models and optimization algorithms. *Acm Computing Surveys (CSUR)*, 48(1):11, 2015.

[47] Xuelian Lin, Zide Meng, Chuan Xu, and Meng Wang. A practical performance

model for hadoop mapreduce. In *Cluster Computing Workshops (CLUSTER WORKSHOPS), 2012 IEEE International Conference on*, pages 231–239. IEEE, 2012.

[48] Ge Song, Zide Meng, Fabrice Huet, Frederic Magoules, Lei Yu, and Xuelian Lin. A hadoop mapreduce performance prediction method. In *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*, pages 820–825. IEEE, 2013.

[49] Kewen Wang, Xuelian Lin, and Wenzhong Tang. Predatoran experience guided configuration optimizer for hadoop mapreduce. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 419–426. IEEE, 2012.

[50] Alberto Núñez, Jose L Vázquez-Poletti, Agustin C Caminero, Gabriel G Castañé, Jesus Carretero, and Ignacio M Llorente. icancloud: A flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing*, 10(1):185–209, 2012.

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

[52] Shunxing Bao, Stephen M Damon, Bennett A Landman, and Aniruddha 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.

[53] DN Kennedy and C Haselgrove. The three nitrc's: software, data and cloud computing for brain science and cancer imaging research. *Front Neuroinform*, 2013.

[54] Yuankai Huo, Justin Blaber, Stephen M Damon, Brian D Boyd, Shunxing Bao, Prasanna Parvathaneni, Camilo Bermudez Noguera, Shikha Chaganti, Vishwesh

Nath, Jasmine M Greer, et al. Towards portable large-scale image processing with high-performance computing. *Journal of digital imaging*, 31(3):304–314, 2018.

[55] Zizhao Zhang, Fuyong Xing, Fujun Liu, and Lin Yang. High throughput automatic muscle image segmentation using cloud computing and multi-core programming.

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

[57] Shiping Chen, Tomasz Bednarz, Piotr Szul, Dadong Wang, Yulia Arzhaeva, Neil Burdett, Alex Khassapov, John Zic, Surya Nepal, Tim Gurevey, et al. Galaxy+ Hadoop: Toward a Collaborative and Scalable Image Processing Toolbox in Cloud. In *Service-Oriented Computing–ICSOC 2013 Workshops*, pages 339–351. Springer, 2013.

[58] Tomasz Bednarz, Dadong Wang, Yulia Arzhaeva, Ryan Lagerstrom, Pascal Vallotton, Neil Burdett, Alex Khassapov, Piotr Szul, Shiping Chen, Changming Sun, et al. Cloud Based Toolbox for Image Analysis, Processing and Reconstruction Tasks. In *Signal and Image Analysis for Biomedical and Life Sciences*, pages 191–205. Springer, 2015.

[59] George C Kagadis, Christos Kloukinas, Kevin Moore, Jim Philbin, Panagiotis Papadimitroulas, Christos Alexakos, Paul G Nagy, Dimitris Visvikis, and William R Hendee. Cloud computing in medical imaging. *Medical physics*, 40(7), 2013.

[60] Han Ter Jung, Ku Hwan An, Jin Cheul Park, Soo Dong Kim, and Hyun Jung La. Miaas: Medical image archival and analytics as-a-service. In *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*, pages 766–773. IEEE, 2016.

[61] Wen-Chung Chiang, Hsiu-Hsia Lin, Tung-Shen Wu, and Chin-Fa Chen. Bulding a cloud service for medical image processing based on service-orient archtecture.

In *Biomedical Engineering and Informatics (BMEI), 2011 4th International Conference on*, volume 3, pages 1459–1465. IEEE, 2011.

[62] Ali Mirarab, Najmeh Ghasemi Fard, and Mahboubeh Shamsi. A cloud solution for medical image processing.

[63] Li Liu, Weiping Chen, Min Nie, Fengjuan Zhang, Yu Wang, Ailing He, Xiaonan Wang, and Gen Yan. image cloud: medical image processing as a service for regional healthcare in a hybrid cloud environment. *Environmental health and preventive medicine*, 21(6):563–571, 2016.

[64] Stephen Bonner, Andrew Stephen McGough, Ibad Kureshi, John Brennan, Georgios Theodoropoulos, Laura Moss, David Corsar, and Grigoris 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.

[65] David C Van Essen, Kamil Ugurbil, E Auerbach, D Barch, TEJ Behrens, R Bucholz, Acer Chang, Liyong Chen, Maurizio Corbetta, Sandra W Curtiss, et al. The human connectome project: a data acquisition perspective. *Neuroimage*, 62(4):2222–2231, 2012.

[66] Clifford R Jack, Matt A Bernstein, Nick C Fox, Paul Thompson, Gene Alexander, Danielle Harvey, Bret Borowski, Paula J Britson, Jennifer L Whitwell, Chadwick Ward, et al. The alzheimer's disease neuroimaging initiative (adni): Mri methods. *Journal of magnetic resonance imaging*, 27(4):685–691, 2008.

[67] Adriana Di Martino, Chao-Gan Yan, Qingyang Li, Erin Denio, Francisco X Castellanos, Kaat Alaerts, Jeffrey S Anderson, Michal Assaf, Susan Y Bookheimer, Mirella 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.

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

[69] P Rinck. Magnetic resonance: a critical peer-reviewed introduction. In *Magnetic resonance in medicine. The basic textbook of the European magnetic resonance forum,*, pages 21–01, 2014.

[70] Dale L Bailey, David W Townsend, Peter E Valk, and Michael N Maisey. *Positron emission tomography*. Springer, 2005.

[71] Adolf F Fercher, Wolfgang Drexler, Christoph K Hitzenberger, and Theo Lasser. Optical coherence tomography-principles and applications. *Reports on progress in physics*, 66(2):239, 2003.

[72] Anders M Dale, Bruce Fischl, and Martin I Sereno. Cortical surface-based analysis: I. segmentation and surface reconstruction. *Neuroimage*, 9(2):179–194, 1999.

[73] Stephen M Smith, Mark Jenkinson, Mark W Woolrich, Christian F Beckmann, Timothy EJ Behrens, Heidi Johansen-Berg, Peter R Bannister, Marilena De Luca, Ivana Drobnjak, David E Flitney, et al. Advances in functional and structural mr image analysis and implementation as fsl. *Neuroimage*, 23:S208–S219, 2004.

[74] John Ashburner. Computational anatomy with the spm software. *Magnetic resonance imaging*, 27(8):1163–1174, 2009.

[75] Bruce Fischl. Freesurfer. *Neuroimage*, 62(2):774–781, 2012.

[76] Barbara Franke, Jason L Stein, Stephan Ripke, Verneri Anttila, Derrek P Hibar, Kimm JE Van Hulzen, Alejandro Arias-Vasquez, Jordan W Smoller, Thomas E

Nichols, Michael C Neale, et al. Genetic influences on schizophrenia and subcortical brain volumes: large-scale proof of concept. *Nature neuroscience*, 19(3):420, 2016.

[77] Zarrar Shehzad, Steven Giavasis, Qingyang Li, Yassine Benhajali, Chaogan Yan, Zhen Yang, Michael Milham, Pierre Bellec, and Cameron Craddock. The preprocessed connectomes project quality assessment protocola resource for measuring the quality of mri data. *Frontiers in neuroscience*, 2015.

[78] Lizhe Wang, Jie Tao, Rajiv Ranjan, Holger Marten, Achim Streit, Jingying Chen, and Dan Chen. G-hadoop: Mapreduce across distributed data centers for data-intensive computing. *Future Generation Computer Systems*, 29(3):739–750, 2013.

[79] Zhuozhao Li and Haiying Shen. Designing a hybrid scale-up/out hadoop architecture based on performance measurements for high application performance. In *Parallel Processing (ICPP), 2015 44th International Conference on*, pages 21–30. IEEE, 2015.

[80] Zhuozhao Li, Haiying Shen, Jeffrey Denton, and Walter Ligon. Comparing application performance on hpc-based hadoop platforms with local storage and dedicated storage. In *Big Data (Big Data), 2016 IEEE International Conference on*, pages 233–242. IEEE, 2016.

[81] Pengfei Xuan, Walter B Ligon, Pradip K Srimani, Rong Ge, and Feng Luo. Accelerating big data analytics on hpc clusters using two-level storage. *Parallel Computing*, 61:18–34, 2017.

[82] Sriram Krishnan, Mahidhar Tatineni, and Chaitanya Baru. myhadoop-hadoop-on-demand on traditional hpc resources. *San Diego Supercomputer Center Technical Report TR-2011-2, University of California, San Diego*, 2011.

[83] Xi Yang, Ning Liu, Bo Feng, Xian-He Sun, and Shujia Zhou. Porthadoop: Support

direct hpc data processing in hadoop. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 223–232. IEEE, 2015.

[84] Troy Baer, Paul Peltz, Junqi Yin, and Edmon Begoli. Integrating apache spark into pbs-based hpc environments. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, page 34. ACM, 2015.

[85] Andre Luckow, Ioannis Paraskevakos, George Chantzialexiou, and Shantenu Jha. Hadoop on hpc: integrating hadoop and pilot-based dynamic resource management. In *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*, pages 1607–1616. IEEE, 2016.

[86] Nusrat Sharmin Islam, Xiaoyi Lu, Md Wasi-ur Rahman, Dipti Shankar, and Dhabaleswar K Panda. Triple-h: A hybrid approach to accelerate hdfs on hpc clusters with heterogeneous storage architecture. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pages 101–110. IEEE, 2015.

[87] Md Wasi-ur Rahman, Xiaoyi Lu, Nusrat Sharmin Islam, Raghunath Rajachandrasekar, and Dhabaleswar K Panda. High-performance design of yarn mapreduce on modern hpc clusters with lustre and rdma. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 291–300. IEEE, 2015.

[88] Nusrat Sharmin Islam, Dipti Shankar, Xiaoyi Lu, Md Wasi-Ur-Rahman, and Dhabaleswar K Panda. Accelerating i/o performance of big data analytics on hpc clusters through rdma-based key-value store. In *Parallel Processing (ICPP), 2015 44th International Conference on*, pages 280–289. IEEE, 2015.

[89] Md Wasi-ur Rahman, Nusrat Sharmin Islam, Xiaoyi Lu, and Dhabaleswar K DK Panda. Nvmd: Non-volatile memory assisted design for accelerating mapreduce

and dag execution frameworks on hpc systems. In *Big Data (Big Data), 2017 IEEE International Conference on*, pages 369–374. IEEE, 2017.

[90] Ronald F Boisvert, José Moreira, Michael Philippsen, and Roldan Pozo. Java and numerical computing. *Computing in Science & Engineering*, 3(2):18–24, 2001.

[91] J Mark Bull, Lorna A Smith, Lindsay Pottage, and Robin Freeman. Benchmarking java against c and fortran for scientific applications. In *Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande*, pages 97–105. ACM, 2001.

[92] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.

[93] Mayur R Palankar, Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel. Amazon s3 for science grids: a viable solution? In *Proceedings of the 2008 international workshop on Data-aware distributed computing*, pages 55–64. ACM, 2008.

[94] Hongli Zhang, Panpan Li, Zhigang Zhou, Xiaojiang Du, and Weizhe Zhang. A performance prediction scheme for computation-intensive applications on cloud. In *Communications (ICC), 2013 IEEE International Conference on*, pages 1957–1961. IEEE, 2013.

[95] Jeremy Freeman, Nikita Vladimirov, Takashi Kawashima, Yu Mu, Nicholas J Sofroniew, Davis V Bennett, Joshua Rosen, Chao-Tsung Yang, Loren L Looger, and Misha B Ahrens. Mapping Brain Activity at Scale with Cluster Computing. *Nature methods*, 11(9):941–950, 2014.

[96] BK Tripathy and Dishant Mittal. Hadoop based Uncertain Possibilistic Kernelized C-means Algorithms for Image Segmentation and a Comparative Analysis. *Applied Soft Computing*, 2016.

[97] Tiago S Soares, Mario AR Dantas, Douglas DJ De Macedo, and Michael A Bauer. A Data Management in a Private Cloud Storage Environment Utilizing High Performance Distributed File Systems. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2013 IEEE 22nd International Workshop on*, pages 158–163. IEEE, 2013.

[98] Chao-Tung Yang, Wen-Chung Shih, Lung-Teng Chen, Cheng-Ta Kuo, Fuu-Cheng Jiang, and Fang-Yie Leu. Accessing Medical Image File with Co-allocation HDFS in Cloud. *Future Generation Computer Systems*, 43:61–73, 2015.

[99] Shoji Nishimura, Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. Md-hbase: A scalable multi-dimensional data infrastructure for location aware services. In *2011 IEEE 12th International Conference on Mobile Data Management*, volume 1, pages 7–16. IEEE, 2011.

[100] Jianling Sun and Qiang Jin. Scalable rdf store based on hbase and mapreduce. In *2010 3rd international conference on advanced computer theory and engineering (ICACTE)*, volume 1, pages V1–633. IEEE, 2010.

[101] Kisung Lee, Raghu K Ganti, Mudhakar Srivatsa, and Ling Liu. Efficient spatial query processing for big data. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 469–472. ACM, 2014.

[102] Shunxing Bao, Andrew J Plassard, Bennett A Landman, and Aniruddha 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.

[103] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.

[104] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *USENIX Annual Technical Conference*, volume 8, page 9, 2010.

[105] Shunxing Bao, Frederick D Weitendorf, Andrew J Plassard, Huo Yuankai, Aniruddha Gokhale, and Landman A Bennett. Theoretical and empirical comparison of big data image processing with apache hadoop and sun grid engine. SPIE Medical Imaging, 2017(accepted).

[106] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM, 2013.

[107] AT&T. Medical imaging in the cloud. Technical Report AB-2246-01, AT&T, July 2012.

[108] Frost and Sullivan. U.S. Data Storage Management Markets for Healthcare, November 2004.

[109] Rainer Schmidt and Matthias Rella. An approach for processing large and non-uniform media objects on mapreduce-based clusters. In *International Conference on Asian Digital Libraries*, pages 172–181. Springer, 2011.

[110] Myoungjin Kim, Yun Cui, Seungho Han, and Hanku Lee. Towards efficient design and implementation of a hadoop-based distributed video transcoding system in cloud computing environment. *International Journal of Multimedia and Ubiquitous Engineering*, 8(2):213–224, 2013.

[111] Yuzhong Yan and Lei Huang. Large-scale image processing research cloud. *Cloud Computing*, pages 88–93, 2014.

[112] Tanya Barrett and Ron Edgar. [19] gene expression omnibus: Microarray data storage, submission, retrieval, and analysis. *Methods in enzymology*, 411:352–369, 2006.

[113] Dorian Gorgan, Victor Bacu, Teodor Stefanut, Denisa Rodila, and Danut Mihon. Grid based satellite image processing platform for earth observation application development. In *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2009. IDAACS 2009. IEEE International Workshop on*, pages 247–252. IEEE, 2009.

[114] Navid Golpayegani and Milton Halem. Cloud computing for satellite data processing on high end compute clusters. In *Cloud Computing, 2009. CLOUD'09. IEEE International Conference on*, pages 88–92. IEEE, 2009.

[115] Bin Wang, Fan Li, Xinhong Hei, Weigang Ma, and Lei Yu. Research on storage and retrieval method of mass data for high-speed train. In *2015 11th International Conference on Computational Intelligence and Security (CIS)*, pages 474–477. IEEE, 2015.

[116] Dhruba Borthakur. The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 11(2007):21, 2007.

[117] Chao-Tung Yang, Lung-Teng Chen, Wei-Li Chou, and Kuan-Chieh Wang. Implementation of a medical image file accessing system on cloud computing. In *Computational Science and Engineering (CSE), 2010 IEEE 13th International Conference on*, pages 321–326. IEEE, 2010.

[118] Ronald C Taylor. An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics. In *BMC bioinformatics*, volume 11, page S1. BioMed Central, 2010.

[119] Shunxing Bao, Bennett Landman, and Aniruddha 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.

[120] Jing Jiang, Jie Lu, Guangquan Zhang, and Guodong Long. Scaling-up item-based collaborative filtering recommendation algorithm based on hadoop. In *Services (SERVICES), 2011 IEEE World Congress on*, pages 490–497. IEEE, 2011.

[121] Sachin Gulabrao Walunj and Kishor Sadafale. An online recommendation system for e-commerce based on apache mahout framework. In *Proceedings of the 2013 annual conference on Computers and people research*, pages 153–158. ACM, 2013.

[122] Zhi-Dan Zhao and Ming-Sheng Shang. User-based collaborative-filtering recommendation algorithms on hadoop. In *Knowledge Discovery and Data Mining, 2010. WKDD'10. Third International Conference on*, pages 478–481. IEEE, 2010.

[123] Saint John Walker. Big data: A revolution that will transform how we live, work, and think, 2014.

[124] Marco Pennacchiotti and Siva Gurumurthy. Investigating topic models for social media user recommendation. In *Proceedings of the 20th international conference companion on World wide web*, pages 101–102. ACM, 2011.

[125] Adriana Garcia, Hari Kalva, and Borko Furht. A study of transcoding on cloud environments for video content delivery. In *Proceedings of the 2010 ACM multimedia workshop on Mobile cloud media computing*, pages 13–18. ACM, 2010.

[126] Chungmo Ryu, Daecheol Lee, Minwook Jang, Cheolgi Kim, and Euiseong Seo. Extensible video processing framework in apache hadoop. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 2, pages 305–310. IEEE, 2013.

[127] Hanlin Tan and Lidong Chen. An approach for fast and parallel video processing on apache hadoop clusters. In *Multimedia and Expo (ICME), 2014 IEEE International Conference on*, pages 1–6. IEEE, 2014.

[128] Shunxing Bao, Yuankai Huo, Prasanna Parvathaneni, Andrew J Plassard, Camilo Bermudez, Yuang Yao, Ilwoo Llyu, Aniruddha Gokhale, and Bennett A Landman. A data colocation grid framework for big data medical image processing-backend design. *arXiv preprint arXiv:1712.08634*, 2017.

[129] Marko Wilke, Scott K Holland, Mekibib Altaye, and Christian Gaser. Template-o-matic: a toolbox for creating customized pediatric templates. *Neuroimage*, 41(3):903–913, 2008.

[130] Carmen E Sanchez, John E Richards, and C Robert Almli. Age-specific mri templates for pediatric neuroimaging. *Developmental neuropsychology*, 37(5):379–399, 2012.

[131] Uicheul Yoon, Vladimir S Fonov, Daniel Perusse, Alan C Evans, Brain Development Cooperative Group, et al. The effect of template choice on morphometric analysis of pediatric brain data. *Neuroimage*, 45(3):769–777, 2009.

[132] Yuankai Huo, Katherine Aboud, Hakmook Kang, Laurie E Cutting, and Bennett 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.

[133] Sébastien Ourselin, Alexis Roche, Sylvain Prima, and Nicholas Ayache. Block matching: A general framework to improve robustness of rigid registration of medical images. In *International Conference on Medical Image Computing And Computer-Assisted Intervention*, pages 557–566. Springer, 2000.

[134] Alan C Evans, D Louis Collins, SR Mills, ED Brown, RL Kelly, and Terry M Peters. 3d statistical neuroanatomical models from 305 mri volumes. In *Nuclear Science Symposium and Medical Imaging Conference, 1993., 1993 IEEE Conference Record.*, pages 1813–1817. IEEE, 1993.

[135] John Mazziotta, Arthur Toga, Alan Evans, Peter Fox, Jack Lancaster, Karl Zilles, Roger Woods, Tomas Paus, Gregory Simpson, Bruce Pike, et al. A probabilistic atlas and reference system for the human brain: International consortium for brain mapping (icbm). *Philosophical Transactions of the Royal Society B: Biological Sciences*, 356(1412):1293–1322, 2001.

[136] Yangming Ou, Hamed Akbari, Michel Bilello, Xiao Da, and Christos 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.

[137] Satrajit S Ghosh, Sita Kakunoori, Jean Augustinack, Alfonso Nieto-Castanon, Ioulia Kovelman, Nadine Gaab, Joanna A Christodoulou, Christina Triantafyllou, John DE Gabrieli, and Bruce 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.

[138] Robert L Harrigan, Benjamin C Yvernault, Brian D Boyd, Stephen M Damon, Kyla David Gibney, Benjamin N Conrad, Nicholas S Phillips, Baxter P Rogers, Yurui Gao, and Bennett 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.

[139] Chandana Kodiweera, Andrew L Alexander, Jaroslaw Harezlak, Thomas W McAllister, and Yu-Chien 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.

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

[141] Shunxing Baoa, Frederick D Weitendorfa, Andrew J Plassarda, Yuankai Huob, Aniruddha Gokhalea, and Bennett 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.

[142] Zang-Hee Cho, Hoon-Ki Min, Se-Hong Oh, Jae-Yong Han, Chan-Woong Park, Je-Geun Chi, Young-Bo Kim, Sun Ha Paek, Andres M Lozano, and Kendall H Lee. Direct visualization of deep brain stimulation targets in parkinson disease with the use of 7-tesla magnetic resonance imaging. *Journal of neurosurgery*, 113(3):639–647, 2010.

[143] Hong Liu, Cihui Yang, Ning Pan, Enmin Song, and Richard Green. Denoising 3d mr images by the enhanced non-local means filter for rician noise. *Magnetic resonance imaging*, 28(10):1485–1496, 2010.

[144] José V Manjón, Pierrick Coupé, and Antonio Buades. Mri noise estimation and denoising using non-local pca. *Medical image analysis*, 22(1):35–47, 2015.

[145] Zongying Lai, Xiaobo Qu, Yunsong Liu, Di Guo, Jing Ye, Zhifang Zhan, and Zhong Chen. Image reconstruction of compressed sensing mri using graph-based redundant wavelet transform. *Medical image analysis*, 27:93–104, 2016.

[146] François Rousseau. A non-local approach for image super-resolution using intermodality priors. *Medical image analysis*, 14(4):594–605, 2010.

[147] Khosro Bahrami, Feng Shi, Xiaopeng Zong, Hae Won Shin, Hongyu An, and Dinggang Shen. Reconstruction of 7t-like images from 3t mri. *IEEE transactions on medical imaging*, 35(9):2085–2097, 2016.

[148] Bao Yang, Leslie Ying, and Jing Tang. Artificial neural network enhanced bayesian pet image reconstruction. *IEEE Transactions on Medical Imaging*, 2018.

[149] Ozan Oktay, Enzo Ferrante, Konstantinos Kamnitsas, Mattias Heinrich, Wenjia Bai, Jose Caballero, Stuart Cook, Antonio de Marvao, Timothy Dawes, Declan ORegan, et al. Anatomically constrained neural networks (acnn): application to cardiac image enhancement and segmentation. *IEEE transactions on medical imaging*, 2017.

[150] Mattias P Heinrich, Bartlomiej W Papież, Julia A Schnabel, and Heinz Handels. Non-parametric discrete registration with convex optimisation. In *International Workshop on Biomedical Image Registration*, pages 51–61. Springer, 2014.

[151] Morten L Kringelbach, Ned Jenkinson, Sarah LF Owen, and Tipu Z Aziz. Translational principles of deep brain stimulation. *Nature Reviews Neuroscience*, 8(8):623, 2007.

[152] Giulio Pergola, Pierluigi Selvaggi, Silvestro Trizio, Alessandro Bertolino, and Giuseppe Blasi. The role of the thalamus in schizophrenia from a neuroimaging perspective. *Neuroscience & Biobehavioral Reviews*, 54:57–75, 2015.

[153] Scott A Small, Scott A Schobel, Richard B Buxton, Menno P Witter, and Carol A Barnes. A pathophysiological framework of hippocampal dysfunction in ageing and disease. *Nature Reviews Neuroscience*, 12(10):585, 2011.

[154] Mattias Paul Heinrich, Mark Jenkinson, Bartlomiej W Papież, Michael Brady, and Julia A Schnabel. Towards realtime multimodal fusion for image-guided interventions using self-similarities. In *International conference on medical image computing and computer-assisted intervention*, pages 187–194. Springer, 2013.

[155] Yuan Liu, Pierre-François D'Haese, Allen T Newton, and Benoit M Dawant. Thalamic nuclei segmentation in clinical 3t t1-weighted images using high-resolution 7t shape models. In *Medical Imaging 2015: Image-Guided Procedures, Robotic Interventions, and Modeling*, volume 9415, page 94150E. International Society for Optics and Photonics, 2015.

[156] Paul A Yushkevich, John B Pluta, Hongzhi Wang, Long Xie, Song-Lin Ding, Eske C Gertje, Lauren Mancuso, Daria Kliot, Sandhitsu R Das, and David A Wolk. Automated volumetry and regional thickness analysis of hippocampal subfields and medial temporal cortical structures in mild cognitive impairment. *Human brain mapping*, 36(1):258–287, 2015.

[157] Juan Eugenio Iglesias, Jean C Augustinack, Khoa Nguyen, Christopher M Player, Allison Player, Michelle Wright, Nicole Roy, Matthew P Frosch, Ann C McKee, Lawrence L Wald, et al. A computational atlas of the hippocampal formation using ex vivo, ultra-high resolution mri: application to adaptive segmentation of in vivo mri. *Neuroimage*, 115:117–137, 2015.

[158] François Rousseau, Estanislao Oubel, Julien Pontabry, Marc Schweitzer, Colin Studholme, Mériam Koob, and Jean-Louis Dietemann. Btk: An open-source toolkit for fetal brain mr image processing. *Computer methods and programs in biomedicine*, 109(1):65–73, 2013.

[159] François Rousseau, Kio Kim, Colin Studholme, Meriam Koob, and J-L Dietemann. On super-resolution for fetal brain mri. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 355–362. Springer, 2010.

[160] Hongzhi Wang, Jung W Suh, Sandhitsu R Das, John B Pluta, Caryne Craige, and

Paul A Yushkevich. Multi-atlas segmentation with joint label fusion. *IEEE transactions on pattern analysis and machine intelligence*, 35(3):611–623, 2013.

[161] Marshall K McKusick, William N Joy, Samuel J Leffler, and Robert S Fabry. A fast file system for unix. *ACM Transactions on Computer Systems (TOCS)*, 2(3):181–197, 1984.

[162] Ke Yan, Le Lu, and Ronald M Summers. Unsupervised body part regression via spatially self-ordering convolutional neural networks. In *Biomedical Imaging (ISBI 2018), 2018 IEEE 15th International Symposium on*, pages 1022–1025. IEEE, 2018.

[163] Ke Yan, Xiaosong Wang, Le Lu, Ling Zhang, Adam Harrison, Mohammadhadi Bagheri, and Ronald M Summers. Deep lesion graphs in the wild: relationship learning and organization of significant radiology image findings in a diverse large-scale lesion database. In *IEEE 2018 Conf. Computer Vision Pattern Recognition*, 2018.

[164] Shunxing Bao, Joe Porter, and Aniruddha Gokhale. Reasoning for cps education using surrogate simulation models. In *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, volume 1, pages 764–773. IEEE, 2016.