Terabyte-sized Image Computations on Hadoop Cluster Platforms

Peter Bajcsy, Antoine Vandecreme, Julien Amelot, Phuong Nguyen, Joe Chalfoun, Mary Brady
Software and Systems Division, Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD
e-mail: {peter.bajcsy, antoine.vandecreme, julien.amelot, phuong.nguyen, joe.chalfoun, mary.brady}@nist.gov

***Abstract*—We present a characterization of four basic Terabyte-sized image computations on a Hadoop cluster in terms of their relative efficiency according to the modified Amdahl's law. The work is motivated by the lack of standard benchmarks and stress tests for big image processing operations on a Hadoop computer cluster platform. Our benchmark design and evaluations were performed on one of the three microscopy image sets, each consisting of over one half Terabyte. All image processing benchmarks executed on the NIST Raritan cluster with Hadoop were compared against baseline measurements, such as the Terasort/Teragen designed for Hadoop testing previously, image processing executions on a multiprocessor desktop and on NIST Raritan cluster using Java Remote Method Invocation (RMI) with multiple configurations. By applying our methodology to assessing efficiencies of computations on computer cluster configurations, we could rank computation configurations and aid scientists in measuring the benefits of running image processing on a Hadoop cluster.**

Keywords- Big Data Industry Standards, Big Data Open Platform, Big Data Applications and Infrastructure

I. INTRODUCTION

Our objective is to characterize Terabyte-sized image processing computations in terms of their computational scalability on Hadoop computer cluster platforms with multi-processor nodes. The computations of interest include image background (flat field) correction, segmentation, image feature extraction and pyramid building for Deep Zoom visualization. These computations range from computationally intensive to data intensive, and operate either on thousands of Mega-pixel images (image tiles) or on hundreds of a half Giga-pixel images (stitched images). From an image processing perspective, these operations are very typical in the image to knowledge workflow consisting of intensity corrections, visualization and information extraction over regions of interest.

In general, computational platforms for processing large size images can be categorized as multi-processor desktop computers, computer clusters, high-performance computing (HPC) resources with big shared memory, grid computing on loosely coupled and networked computers, and computing on novel hardware architectures (e.g., Graphical Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs)). Advantages and disadvantages of several above categories for life science applications can be found in Schadt et al. [1]. Our work focuses on computer clusters in order to support run time and configuration decisions for research centers and governmental organizations with security concerns about the use of external computational resources. Nevertheless, the fundamental problems in characterizing Terabyte-sized image processing computations are similar to cloud platforms and are also tied to security (access to physical and virtual storage and compute resources).

Our specific focus is on Hadoop clusters because Hadoop includes file system and computation solutions for utilizing distributed computational resources. For example, the advantages of Hadoop Distributed File System (HDFS) are in data replication and collocation of data and computation [2]. The Map and Reduce computational paradigm of Hadoop has been shown very efficient especially for sorting computations [3], [4]. This work is exploring image processing computations on Hadoop clusters with HDFS while running Map tasks.

The main goal is to establish benchmarks and stress tests for big image data processing operations on a Hadoop cluster platform. The computational benchmarks provide experimental characteristics of the above image processing computations and image partitions for (a) assessing the efficiency of a computer cluster configuration with respect to a given computation (number of nodes, number of tasks per node, RAM per node, job resource managers such as Hadoop middleware or Java Remote Method Invocation (RMI) and Portable Batch System (PBS) scripts), and (b) predicting run times for various input data distribution patterns. The stress tests are useful for understanding the system limits, and how to manage hardware failures and deal with cluster heterogeneity.

The application specific motivation of our work comes from live cell imaging applications of very large fields of view (FOV). Given the advances in microscopy imaging and its applications in many bio-medical applications, a single microscope generates a large number of spatial image tiles with several measurements at each location over time. The image tiles can be stitched into a large FOV image with hundreds of Mega-pixels or tens of Giga-pixels. Multiple time slices of stitched images accumulate to Terabytes of image data. These image data cannot be analyzed without computations that calibrate, segment and visualize image channels, as well as extract

image features for further analyses. Such image processing computations are very common across many application domains and represent a sample subset for our work.

Our computer science motivation lies in the lack of benchmarks and stress tests for big image processing operations on a Hadoop computer cluster platform. For instance, as biologists increase robustness of their conclusions by increasing the number of experimental replicas, their access to elastic computational resources and to benchmarks for optimal reconfiguration become apparent. The image data type and the variety of image processing computations are different from the existing Hadoop benchmark tests that primarily focus on textual data and low level operations.

While the Apache Hadoop distribution includes several basic tests, it does not provide sufficient understanding about executions of image processing operations. The four common basic tests are MRBench, NNBench, TestDFSIO, and Gridmix [4], [5]. MRBench is designed to check whether small job runs are running efficiently. NNBench introduces a high HDFS management stress on the NameNode by requesting a large number of files to be created, read, renamed and deleted. TestDFSIO writes into or reads from a user specified number of files. Gridmix mimics a variety of data-access patterns seen in practice. There is also an interest in understanding the Hadoop performance as a function of processor and cluster hardware configurations [6] or against relational database management systems [7], [8]. These Hadoop tests together with general networking performance tests (Netperf [9], Iozone [10] or other [11]) provide excellent probes into a Hadoop cluster for low level operations. However, these tests are typically applied to many small-sized elements such as random bytes or words which are different from high-dimensional images with varying file sizes. Furthermore, the differences between the above computations and the image processing computations are in the additional support of various image formats on each cluster node and in spatial dependency of image computations (e.g., spatially local intensity changes versus spatially global image filtering or co-occurrence matrix-based feature extraction).

In order to design a suite of benchmark tests for image processing computations on Hadoop clusters, one has to sample commonly used image computations and their various image inputs and outputs. The approach is similar conceptually to the design of the PUMA test suite [12], where the Hadoop tests are extended by additional types of text-based computations (e.g., K-mean clustering, inverted index, adjacency list, self-join). It can also be related to Almer's case study [13] that is comparing a Hadoop cluster and dual core personal computer run times. This case study considered a set of image format conversion, auto-contrast, sharpening and resizing operations applied to 200 Landsat satellite images of 139 Mega pixels each (total 83.3 GB of imagery). In our case, the sampling of computations tries to cover a wider range of input and output scenarios, and the run times are compared against a broader range of benchmarks on 0.6TB of imagery. The image input and output scenarios are summarized in Table 1. The details of each computation are provided in Section II.

Next, the designed suite of Hadoop cluster benchmark and stress tests for image processing are compared against baseline computations. In our work, the baseline computations are either (1) a well-benchmarked non-image computation on a Hadoop cluster such as Teragen or Terasort, or (2) the same image processing operations (a) running on a multicore desktop instead of a Hadoop cluster, and (b) running on a cluster without Hadoop middleware.

Terasort is one of the most popular text-based computations for sorting a Terabyte of numbers [14]. The algorithm running on a Hadoop cluster was initially developed by Yahoo! [3] and later advanced by Google [15]. Ideally, the well-understood Terasort computational performance on Hadoop clusters can be related to a priori unknown performance of image processing computations on Hadoop clusters for similar input file sizes. We reviewed the Hadoop Image Processing Interface (HIPI) library [16] that focused on filtering and bundling very small images

Table 1: Summary of computations, and input and output image data files.

Type of Image Processing	**Spatial Extent of Image Processing**	**Input & Output File Characteristics**	**Computational Complexity**	**Data-Access Pattern During Computations**
Flat Field Correction	**Local**	**Input & Output:** Tens of thousands of a few MB size files	**Low** (two subtractions and one division per pixel)	**Medium** (accessing three files and creating one file, data skew on two input files)
Segmentation based on convolution kernels	**Global** with fixed kernel	**Input & Output:** Hundreds of a half GB size files	**Medium** (tens of subtractions, multiplications, comparisons per pixel)	**Low** (accessing one file and creating one file)
Feature Extraction	**Global** with mask defined kernel	**Input:** Co-located pairs of hundreds of a half GB files **Output:** hundreds of KB size tables	**High** (several thousands of basic numerical operations per pixel)	**Medium** (accessing two files and creating one file)
Deep Zoom Pyramid Building	**Global** with fixed kernel	**Input:** Hundreds of a half GB size files **Output:** Millions of KB size files	**Medium** (tens of additions and divisions per pixel)	**High** (creating thousands of directories, accessing one input file and writing millions of pyramid tiles)

(HipiImageBundle & CullMapper classes) before running Hadoop. In our case, Terabyte collections of several megabyte files do not fit HIPI and the bundling was replaced by the Hadoop Sequence File and tar representations. Finally, two of the baselines are desktop and cluster configurations without Hadoop. The desktop has six cores of Intel Xeon @ 2GHz with 64GB of RAM and hyper threading activated. The cluster is the NIST Raritan cluster which is composed of more than 800 heterogeneous nodes controlled by Portable Batch System (PBS) as a job resource manager.

The novelty of the work lies in designing a suite of benchmark and stress tests for image processing computations on Hadoop clusters that can complement the existing tests in the Apache Hadoop project. The infrastructure could be integrated into a larger image processing community effort to build an open source image processing library running on a Hadoop cluster.

The paper is organized as follows. Section II introduces the image processing computations and their characteristics. Section III presents experimental data, hardware and software platforms, exploratory and baseline benchmarks, and relative efficiency evaluations. We conclude with a summary in Section IV.

II. IMAGE PROCESSING COMPUTATIONS

To address the algorithmic development of image processing, one approach is to take an existing well-established image processing library and enable running its functionality on a Hadoop cluster. A candidate for such a widely used library is an instance of NIH ImageJ/Fiji, ImageJDev or BioImageXD [17], [18]. This approach faces challenges in executing functionalities in a headless mode and in dealing with a code base that has not been designed with Terabyte image sizes in mind. People have blogged about "parallel ImageJ"[19] running in the Amazon cloud by adding a Hadoop InputFormat to handle image types in HDFS and encapsulating ImageJ operations in map and reduce methods. However, the encapsulation has been performed for a very small number of operations in the ImageProcessor class of the ImageJ library. In addition, the developed source code is not publicly available, the ImageProcessor-based execution lacks process control (not knowing when execution is done), and has the overhead of starting ImageJ and loading plugins.

These considerations led us to a development of our own image processing functionality in Java for the following computations: flat field correction, segmentation, image feature extraction and pyramid building for Deep Zoom visualization. These computations range from computationally intensive to data intensive, and have to operate either on thousands of Mega-pixel images (image tiles) or on hundreds of a half Giga-pixel images (stitched images). Next, we characterize these four computations.

A. Flat Field Correction

Flat field correction (FFC) is described mathematically below:

$$I^{FFC}(x,y) = \frac{I^{RAW}(x,y) - DI(x,y)}{WI(x,y) - DI(x,y)} \quad (1)$$

where $I^{FFC}(x,y)$ is the flat-field corrected image intensity, $DI(x,y)$ is the dark image acquired by closing the camera shutter, $I^{RAW}(x,y)$ is the raw uncorrected image intensity, and $WI(x,y)$ is the flat field intensity acquired without any object to correct primarily for spatial shading. This is the simplest computation that consists of two subtractions and one division per pixel. It needs the DI and WI images co-located from the distributed execution perspective.

B. Segmentation

There are many segmentation methods applied to cell microscopy images [20]. We selected a segmentation method that consists of four linear workflow steps: (1) Sobel-based image gradient computation, (2) thresholding by a value equal to twice the intensity histogram mode, (3) morphological opening (dilation of erosion) to remove small holes and islands, and (4) connectivity analysis to assign the same label to each contiguous group of 4-connected pixels.

The Sobel-based image gradient is defined over a 3×3 convolution kernel, and estimates the gradient in the column and row directions. The gradient image contains the magnitude of each estimate as shown below.

$$G(x,y) = \sqrt{\left(\Delta_x\right)^2 + \left(\Delta_y\right)^2} \quad (2)$$

$$\Delta_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} * \begin{pmatrix} I(x-1,y-1) & I(x-1,y) & I(x-1,y+1) \\ I(x,y-1) & I(x,y) & I(x,y+1) \\ I(x+1,y-1) & I(x+1,y) & I(x+1,y+1) \end{pmatrix}$$

$$\Delta_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} * \begin{pmatrix} I(x-1,y-1) & I(x-1,y) & I(x-1,y+1) \\ I(x,y-1) & I(x,y) & I(x,y+1) \\ I(x+1,y-1) & I(x+1,y) & I(x+1,y+1) \end{pmatrix}$$

Sobel filtering has been explored on a Hadoop cluster by Almer [13], and serves as an input to parallel gradient domain computing [21] (e.g., computations of the divergence of gradient using Message Passing Interface (MPI)). Multiple reports of run time versus image file size (e.g., in [13] Fig. 2) improve general understanding of the performance benchmarks for various Hadoop cluster setups.

Another widely used computation of segmentation is morphological opening (dilation of erosion). During this operation, a thresholded binary image is convolved with the min and max operators over a 3×3 kernel. This computation has been explored in the past on distributed memory machines [22] (multiple-instruction-multiple-data (MIMD) Intel Paragon and single-instruction-multiple-data (SIMD) MasPar MP-1) showing advantages of data partitioning.

C. Feature Extraction

Similar to segmentation, there is a large number of feature extraction methods designed for cell microscopy images. The number ranges from hundreds [23], [24] to thousands [25] and serves as an input to image characterization and classification. We have categorized features according to their type as intensity, shape and texture descriptors. Each feature type is represented by sample representatives listed in Table 2.

Table 2. Description of extracted features.

Feature Type	Feature Name	Mathematical Model
Intensity	Basic central moments according to statistics: Mode, Mean, Standard Deviation, Skewness, Kurtosis, Fifth and Sixth moments	Integral computations
Shape	Hu's moment invariants are computed according to [26] : Centroid, area, perimeter, circularity, aspect ratio, extend, orientation, eccentricity	Integral computations
Texture	The Gray Level Co-Occurrence matrix is computed according to [27] and the following four features are extracted Correlation, Energy, Contrast and Homogeneity	Directional counting

The features of intensity and shape types are extracted by implementing integral equations over a masked region, and hence can be denoted as integral image computations. The extraction of texture features is also viewed as a directional counting computation of all intensity pairs within a masked region. In general, the directional counting computation has higher requirements on CPU & RAM than the integral computations because there are many more counting co-occurrence bins than the central moment accumulator variables. In terms of computation, the counting is less demanding than computing the higher moment powers. We have evaluated both integral and directional counting computations together because most microscopy image analyses do not have a priori knowledge about preferred sets of features [28].

D. Pyramid Building

One way to view images with very large pixel counts is to use the Deep Zoom image pyramid representation [29] and the OpenSeadragon javascript library [30]. The visualization of Terabyte-sized images is critical for many applications and requires building a multi-resolution pyramid. This computation is I/O bound because it generates many small files for fast data transmission and rendering purposes. The pyramid building computation was explored by Kooper and Bajcsy [31], [32] on a cluster with 16 nodes, each node with 8 cores and 16GB RAM. It took 30 days to build a pyramid for a 79 Gigapixel image and generated 1.6M tiles stored on redundant array of independent disks (RAID-5). A speed-up by a factor of 20 was achieved after custom image loading. These previous benchmarks as well as the benchmarks with storage on NO RAID, RAID 0, and RAID 1 configurations contribute to a better understanding of the pyramid computation on a Hadoop cluster and serve as comparison benchmarks.

III. Experimental Results

We have executed about one hundred runs of image processing computations with the data set described in Section III.A and with the Terasort generated data. The hardware and software specifications are provided in Section III.B as well as observations about our computational reliability in Section III.E. Sections III.D and III.E. document the relative efficiency of each computation and the suitability of individual image processing operations based on these benchmarks.

A. Characteristics of Image Data Sets

We experimented with three data sets where each raw image data set is about 0.6TB. For benchmarking and stress testing, we have selected a data set that consists of 18×14= 252 image tiles covering approximately 180 square millimeters of a stem cell colony dish, over five days under both phase contrast and green fluorescence channels, with images acquired every 15 minutes. This example data set is composed of 195,552 images at 2.8 MB/image equal to 0.527 TB of data (388 time samples of 252 images with 2 color channels acquired over 97 hours every 15 minutes). The H9 human embryonic stem cell line was engineered to produce green fluorescent protein (GFP) under the influence of Oct4 promoter and cells were cultured under feeder-free conditions on Matrigel™. After correcting the GFP image tiles, all tiles were stitched and yielded 388 stitched time frames stored in a TIFF file format (0.527 TB; GFP only: 0.264 TB). Table 3 summarizes the data with respect to image processing computations. Note that the "Size per File" refers to compressed files via pack bits of the TIFF format that become much larger in RAM after loading.

Table 3. A summary of inputs and outputs for each benchmarked computation.

Type of Image Processing	Input Data in TIFF File Format		Output Data mostly in TIFF File Format	
	Total Files	Per-File Size	Total Files	Per-File Size
Flat Field Correction	97,776 GFP channel corrected tiles~264 GB	2 bytes per pixel: 2.83 MB	97,776 GFP channel corrected tiles~527GB	4 bytes per pixel: 5.6MB
Segmentation based on convolution kernels	388 frames of stitched images ~219GB	2 bytes per pixel: 593 MB	388 frames of mask images ~86GB	2 bytes per pixel 71MB-331MB
Feature Extraction	388 frames of stitched and segmented images ~ 219GB + 80.7GB	2 bytes per pixel: ~593 MB + ~66MB-336MB	388 files ~ 40MB	CSV file format ~100KB

Deep Zoom Pyramid Building	388 frames of stitched images ~ 219GB	2 bytes per pixel: 593 MB	6,596 folders; 2,476,683 files; 151 GB	JPG file format 2-18KB per file

B. Computer Hardware and Software Characteristics

We ran the benchmarks on the NIST Raritan cluster and on a desktop computer. Table 4 summarizes the cluster and desktop hardware and software specifications. The cluster nodes differ in terms of CPU speed and RAM, and are allocated to jobs based on the requested resources in a Portable Batch System (PBS) script. We installed Hadoop and Java 1.7 on the cluster to support Java code execution and Java Remote Method Invocation (RMI). The desktop computer had similar software configuration to the cluster.

Table 4. NIST Raritan cluster and test desktop characteristics.

	Specs	**Cluster**	**Desktop**
Hardware	Cluster Nodes	800 computer nodes having from 2 to 16 virtual processors with 4 to 32GB of RAM	Intel Xeon @ 2GHz 6 cores, 64GB of RAM and hyper threading activated
	Networking	1Gbit/second	
Software	Java Virtual Machine	Java version "1.7.0_17" Java(TM) SE Runtime Environment (build 1.7.0_17-b02) Java HotSpot(TM) 64-Bit Server VM (build 23.7-b01, mixed mode)	Java version "1.7.0_15" Java(TM) SE Runtime Environment (build 1.7.0_15-b03) Java HotSpot(TM) 64-Bit Server VM (build 23.7-b01, mixed mode)
	Hadoop	hadoop-2.0.3-alpha	
	Operating System	CentOS 5.9 Linux 2.6.18-274.3.1.el5 x86_64	Ubuntu 12.10 Linux 3.5.0-28-generic x86_64
	File System	Lustre parallel distributed file system	ext4 on top of Logical Volume Manager (LVM)

C. Characteristics of Image Processing Benchmarking Software

All four computations were implemented as independent Java libraries and used on the desktop, Java RMI and Hadoop cluster platforms. Thus, the same image processing code is invoked regardless of the platform. The desktop implementation is a Java program starting a fixed number of threads (specified on the command line) using a fixed thread pool. One image is processed by each thread. As soon as a thread has finished processing an image, it starts processing a new image.

The Java RMI Application Programming Interface (API) invokes methods on remote computers (i.e., on the cluster nodes). We build a simple job scheduler with a client-server architecture using the Java RMI API. The Java RMI-based server provides a method to fetch the next image to process. When a client node has finished processing an image, it notifies the server with another RMI call and fetches the next image to process. Similar to the desktop implementation, the client nodes can run multiple threads (one per image).

In order to invoke the four image processing computations from Hadoop, we implemented a new Hadoop FileInputFormat named ImageInputFormat. This class reads an image file from HDFS and submits it as a map task input. The map tasks then retrieve a BufferedImage from the ImageInputFormat and pass it to the processing implementations. Once the processing is done, the result is saved back to HDFS directly from the map task. Thus, the jobs do not have any reduce task. The design of our Hadoop implementation leverages Hadoop data and computation colocation as summarized in Table 5. The key differences among the three platform-specific implementations lie in the computational elasticity associated with each platform and in the locality of the data and computation. We have observed on average 98.4% of tasks getting input data from local node for image segmentation and 99.8% of tasks for flat field correction computations except accessing two common files from HDFS.

Table 5. Characteristics of computational elasticity and data & computation collocation for the three experimental platforms.

Computational Platform	**Computational Elasticity**	**Data & Compute Collocation**
Desktop	**Low:** limited by the RAM and CPU of the executing computer	Yes: all data are on a local disk
Java Remote Method Invocation (RMI)/Raritan Cluster	**High:** nodes can be requested as needed	No: all data are transferred over network to the computing node
Hadoop/Raritan Cluster	**High:** nodes can be requested as needed	Yes with high probability: After pushing 3 data replicas to HDFS, Hadoop launches computations where the data are.

D. Benchmark and Baseline Runs

We have documented the following samples in the space of image processing computations, and hardware and software configurations applied to the selected data set.

- Four image processing computations described in Table 1 and the Terasort computation from the Apache Hadoop test suite [4].
- Raritan hardware cluster configurations used for computations (8 virtual processors @2.5 GHz available per node with local storage: >100GB):
    - Number of client nodes: 5, 10, 20, 30, 40, 50, 60
    - RAM on cluster nodes: 16GB (Segmentation Java RMI), 24GB (Feature extraction), 32GB RAM (all other computations)
- Raritan software cluster configurations used for computations:

- Cluster computation management: Simple Java RMI server started with PBS script or Hadoop middleware.
- Number of map tasks per node: 1 (all computations), 2 (Pyramid building)

- Desktop software configurations:
  - Number of threads: 1 (all computations), 2 (Feature extraction), 5,10,12,16 (Pyramid)

The numerical results are shown in Figure 1 through 6. Figure 1 sets our desktop baseline and represents run times of processing a Terabyte volume on a desktop with a 6-core CPU and 64GB of RAM. Figure 2 compares flat field correction with Teragen on the Hadoop Raritan cluster since both image and text operations are very I/O intensive. Figure 3 shows image segmentation and Terasort computations executed on the Raritan cluster for the same input file sizes and file numbers. The image spatial filtering (kernel multiplications of an image matrix) and sorting (number/word comparisons) are frequently the key discriminators between image and text processing. Figure 4 illustrates the comparison of Hadoop and Java RMI based management of the Raritan cluster for image pyramid building in terms of the number of parallel processes. The map tasks in Hadoop and the number of threads per node in Java RMI show similar performance with Hadoop outperforming Java RMI for larger number of nodes. Figure 5 and Figure 6 focus on run time decomposition of pyramid building and feature extraction computations to understand the overheads and gains for various cluster configurations. The overhead of pushing data to HDFS in Hadoop could be minimized by running multiple computations on the same data over many nodes since the Hadoop-based computation is only slightly outperforming the Java RMI based computation (~9% at 60 nodes, feature extraction).

By way of an aside, the numerical results in the graphs are reported for the number of client nodes that were allocated by the Raritan cluster. Thus, the number of requested nodes was higher than the number shown in the figures below. Computations that deal with a large number of small input files (flat field correction) or output files (pyramid building) were programmed to package the files by either tarring them or creating a serialized Hadoop Sequence File for better I/O efficiency.

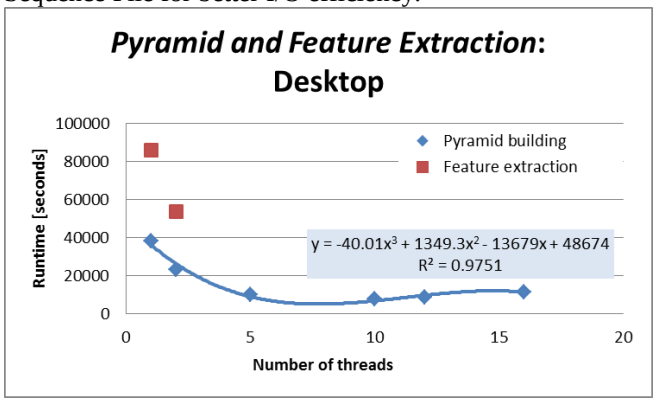

Figure 1: Pyramid and feature extraction computations executed on a 6 core desktop with various number of threads. Feature extraction with more than 2 threads is limited by the desktop RAM size equal to 65GB (each feature extraction job requires more than 24GB of RAM).

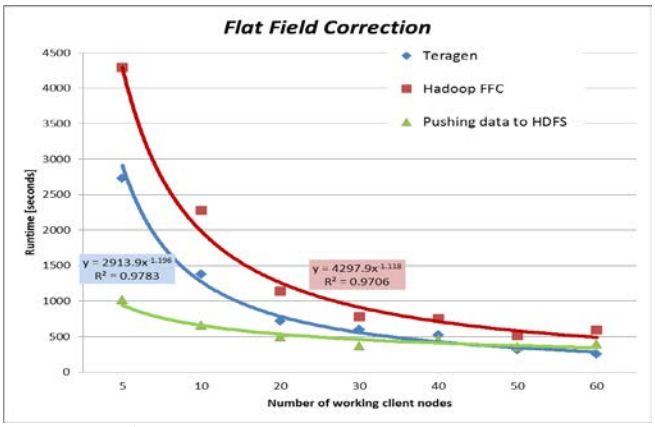

Figure 2: Flat field correction (FFC) computation executed on the Hadoop Raritan cluster and compared to Teragen that generates the same number of files as FFC.

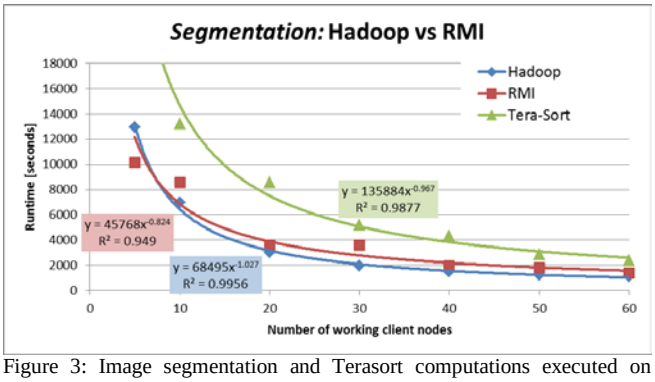

Figure 3: Image segmentation and Terasort computations executed on Raritan cluster for the same input file sizes and file numbers. The Hadoop execution (blue) slightly outperforms RMI execution (red) for more than 10 nodes and is faster than the Terasort computation. All power approximations have a residual R^2 higher than 0.949 indicating a very good fit.

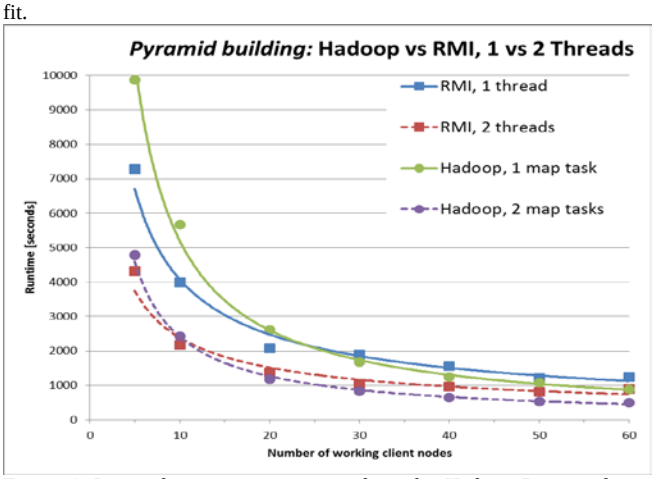

Figure 4: Pyramid computation executed on the Hadoop Raritan cluster with one or two map tasks per node and on the Java RMI cluster with one or two threads per node. Hadoop with K map tasks outperforms Java RMI with the same number of K threads for more than 25 nodes (K=1) or 10 nodes (K=2).

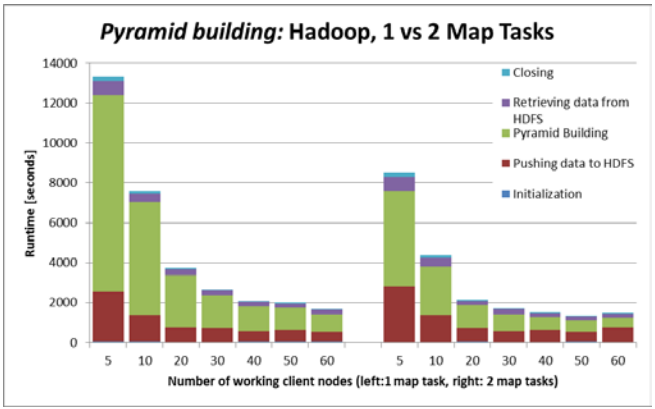

Figure 5: Pyramid computation executed on the Hadoop Raritan cluster with one or two map tasks per node and presented with 5 contributions to each run time. The key benefit of launching multiple map tasks is the reduction of the pyramid building time (green color bars).

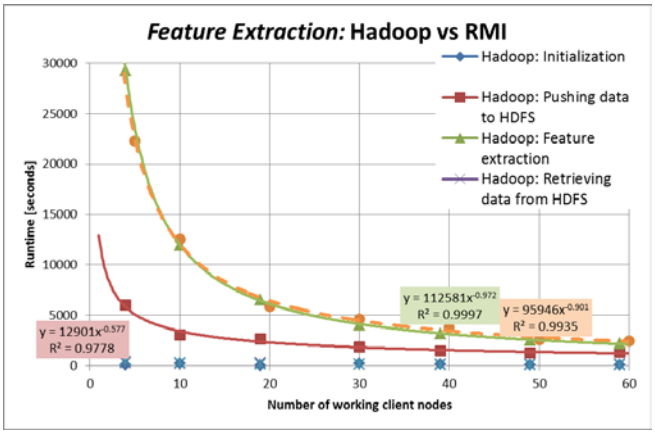

Figure 6: Feature extraction executed on the Raritan cluster using Hadoop (solid green line) and Java RMI (dashed orange line). Feature extraction using Hadoop is slightly faster than using Java RMI if the overhead of pushing data to HDFS is viewed as a one-time overhead. All power approximations have a residual R^2 higher than 0.9778 indicating a very good fit.

E. *Computational Reliability*

In our case, running benchmarks for image processing operations has also served the purpose of stress testing the Raritan Hadoop cluster. By default, Hadoop middleware has a built-in mechanism for handling possible hardware failures or straggler tasks through speculative execution [33]. We have used the default speculative execution setting and documented the errors of failed executions. The NIST Raritan cluster is quite heterogeneous. Although we did not observe single straggler tasks, we have encountered task failures due to the entire system overload such as many tasks failing at the same time, SSH connection closed as a function of possible network overload, socket timeout exception from Hadoop or sometimes failing to push data to HDFS without a notification. Future stress tests for image processing computations can be designed for the users running image processing on Hadoop clusters similar to the analyses reported based on text logs [34], [35].

F. *Relative Efficiency of Image Processing Computations*

As the number of nodes in the cluster increases, computations can be run in shorter amounts of time, but the efficiency of computation naturally decreases. We adopted Amdahl's law [36] for measuring parallel efficiency and computed the relative efficiency of cluster computations (elapsed time for 1 worker/client divided by the multiplication of M workers and elapsed time for M workers). Our modification is introduced on the right side of Equation (3) for two reasons. First, benchmarking of image processing computations is very time consuming when using only one node of a cluster. For all practical purposes, users should be able to compute the parallel efficiency based on the elapsed run time on any small number of cluster nodes. Second, each benchmark has an inherent temporal uncertainty due to variable system loads no matter how many times the execution is re-run. Relative efficiency values larger than one do not have practical meaning and can be avoided by considering the minimum sample for the numerator of the following equation

$$E_{RELATIVE} = \frac{T_1}{MT_M} = \frac{LT_L}{MT_M} \qquad (3)$$

Here T_1 is the run time on a single cluster node, T_M is the runtime on M nodes where M is in the sample set of working nodes of size NumSamples ($M \in \{w_i\}_{i=1}^{NumSamples}$), and T_L is the runtime on the number of L cluster nodes for which $L*T_L = \min_M\{M*T_M\}$. Ideally, the parallel efficiency would be always one for any number of nodes. The efficiency formula can also be extended in the future to include power considerations [37]. Modified relative efficiency coefficients for all four image processing and Terasort computations are reported in Figure 7.

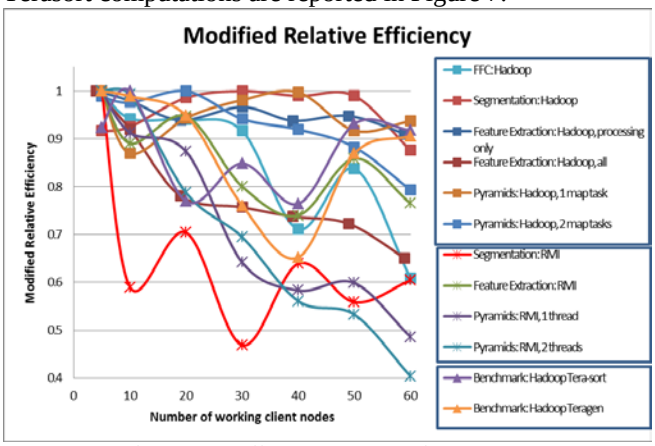

Figure 7: Modified relative efficiency computed for all sample points.

Based on the results in Figure 7, we computed a cluster computing suitability score S per computation configuration according to Equation (4). It is an average of deviations from the ideal relative efficiency (equal to one).

$$S^{CONFIG} = \frac{1}{NumSamples - 1} \sum_{i=1}^{NumSamples} \left(1 - E_{RELATIVE}^{CONFIG}(i)\right) \quad (4)$$

According to the scores shown in Figure 8, the majority of image processing operations executed on a Hadoop cluster outperformed their corresponding benchmarks run on Java RMI clusters or compared against number/text sorting computations for the same input file sizes.

IV. SUMMARY

There is a lack of algorithms for processing Giga to Terabyte-sized images that can leverage very powerful parallel and distributed hardware architectures such as Hadoop clusters. We have researched four image processing algorithms and the corresponding Hadoop infrastructure for running these algorithms on Hadoop clusters. The software enabled (a) quantitative characterization of a Hadoop cluster against several baseline measurements, and (b) assessment of relative efficiency of image processing computation running on a cluster and their suitability for Hadoop platforms. The research also serves as a potential contribution to the suite of benchmark and stress tests for the Apache Hadoop project and to a future development of a standard for big image data processing on distributed platforms. Similar results to those showed in Figure 8 could be developed as reference measurements to aid cluster performance 'calibration' across smaller groups and larger institutions or to assess benefits of cluster configurations for various computation types.

We concluded that image processing operations benefit from scalability on a Hadoop cluster because of the computational elasticity of cluster platforms and the collocation of data and computation in Hadoop. The spatial extent of image processing operations defined the data access pattern during many computations. It was a factor for the RAM requirements per node since a half GB size files had to be loaded without sub-dividing the input images. On the other side, close to one hundred images as inputs (flat field correction) or 2.4 million images as outputs (pyramid) pose challenges on efficient data transmission to the cluster nodes, and handling I/O operations. Image processing on a Hadoop cluster would not be efficient without additional considerations of RAM requirements, data transmission packaging and I/O tasks, and therefore a Hadoop infrastructure for benchmarking image analyses becomes important. To illustrate this point, although the total runtime of all reported benchmarks was about 95 hours, the actual runtime was about three times as much due to test replicates of runs, configuration efficiency experimentations and failed computations.

In the future, we plan to design specific stress tests for image processing computations running on Hadoop clusters. Our aim is to make the transitions from a desktop solution for image processing to a solution running on Hadoop cluster hardware architectures much easier for all scientists.

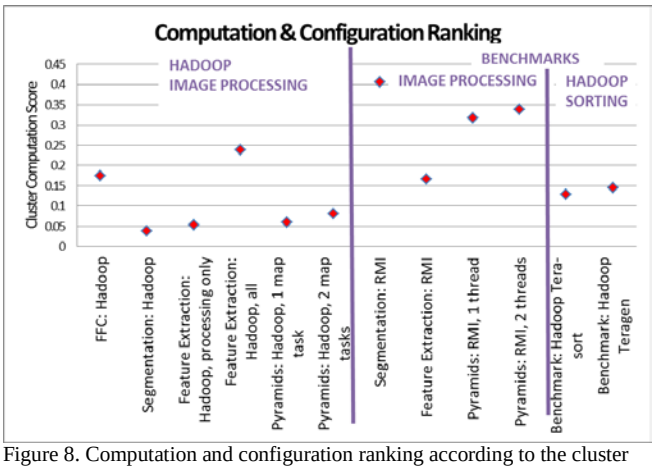

Figure 8. Computation and configuration ranking according to the cluster computation efficiency score. Lower scores imply better cluster computing suitability.

V. ACKNOWLEDGMENT

This work was sponsored by NIST as a part of the Computational Science in Biological Metrology project. We would like to acknowledge all project team members for their contributions.

VI. DISCLAIMER

Commercial products are identified in this document in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the products identified are necessarily the best available for the purpose.

VII. REFERENCES

[1] E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan, "Computational solutions to large-scale data management and analysis.," *Nature reviews. Genetics*, vol. 11, no. 9, pp. 647–57, Sep. 2010.

[2] Xyratex, "Using Lustre with Apache Hadoop," 2010. [Online]. Available: http://wiki.lustre.org/images/1/1b/Hadoop_wp_v0.4.2.pdf. [Accessed: 31-May-2013].

[3] O. O. Malley, "TeraByte Sort on Apache Hadoop," 2008. [Online]. Available: http://sortbenchmark.org/YahooHadoop.pdf. [Accessed: 31-May-2013].

[4] O. O. Malley, "Hadoop Benchmarking," in *Workshop on Big Data Benchmarking*, 2012, no. May, pp. 1–13.

[5] Apache, "Hadoop Benchmarking Code," *Grepcode*, 2013. [Online]. Available: http://grepcode.com/file/repository.cloudera.com/content/repositories/releases/org.apache.hadoop/hadoop-test/0.20.2-cdh3u0/org/apache/hadoop/mapred. [Accessed: 31-May-2013].

[6] INTEL, "Optimizing Hadoop* Deployments," 2010. [Online]. Available: http://software.intel.com/sites/default/files/m/f/4/3/2/f/31124-Optimizing_Hadoop_2010_final.pdf. [Accessed: 31-May-2013].

[7] M. Stonebreaker, D. Abadi, D. J. Dewitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin, "MapReduce and Parallel DBMSs : Friends or Foes ?," *Communications of the ACM*, vol. 53, no. 1, 2010.

[8] P. Kent, "Hadoop Benchmarking from a SAS Perspective," in *Workshop on Big Data Benchmarking*, 2012, pp. 1–9.

[9] R. Jones, “NetPerf Benchmark, Computer Program,” 2013. [Online]. Available: http://www.netperf.org/netperf/. [Accessed: 31-May-2013].

[10] W. D. Norcott. and D. Capps, “Iozone Filesystem Benchmark,” 2013. [Online]. Available: http://www.iozone.org/docs/IOzone_msword_98.pdf. [Accessed: 31-May-2013].

[11] D. Heger, “Hadoop Performance Tuning - A Pragmatic & Iterative Approach,” 2013. [Online]. Available: http://www.cmg.org/measureit/issues/mit97/m_97_3.pdf. [Accessed: 31-May-2013].

[12] F. Ahmad, S. Lee, M. Tothenthodi, and T. N. Vijaykumar, “PUMA : Purdue MapReduce Benchmarks Suite, Tech. Report,” 2012. [Online]. Available: http://web.ics.purdue.edu/~fahmad/benchmarks.htm. [Accessed: 31-May-2013].

[13] M. H. Almeer, “Cloud Hadoop Map Reduce For Remote Sensing Image Analysis,” *Journal of Emerging Trends in Computing and Information Sciences*, vol. 3, no. 4, pp. 637–644, 2012.

[14] C. Nyberg, M. Shah, and N. Govindaraju, “Sort Benchmark Web Page,” 2013. [Online]. Available: http://sortbenchmark.org/. [Accessed: 15-May-2013].

[15] Google, “Google I/O conference,” *Google’s developer conference*, 2012. [Online]. Available: http://www.engadget.com/event/googleio2012/articles/. [Accessed: 05-May-2013].

[16] J. Lawrence, S. Arietta, C. Sweeney, and L. Liu, “Hadoop Image Processing Interface, Computer Program,” 2010. [Online]. Available: http://hipi.cs.virginia.edu/about.html. [Accessed: 31-May-2013].

[17] W. Rasband, “ImageJ & Fiji & ImageJA & ImageJ2, Computer Program,” 2013. [Online]. Available: http://rsbweb.nih.gov/ij/. [Accessed: 15-May-2013].

[18] “BioimageXD, Computer Program,” 2013. [Online]. Available: http://www.bioimagexd.net/. [Accessed: 31-May-2013].

[19] T. Plunkett, M. A. Sick, and J. Su, “Cloud Analytics Tools by Serene Software Inc.,” *SBIR DOD/OSD*, 2010. [Online]. Available: http://www.sbir.gov/sbirsearch/detail/13117. [Accessed: 15-May-2013].

[20] A. A. Dima, J. T. Elliott, J. J. Filliben, M. Halter, A. Peskin, J. Bernal, M. Kociolek, M. C. Brady, H. C. Tang, and A. L. Plant, “Comparison of segmentation algorithms for fluorescence microscopy images of cells.,” *Cytometry. Part A: the journal of the International Society for Analytical Cytology*, vol. 79, no. 7, pp. 545–59, Jul. 2011.

[21] S. Philip, B. Summa, P. Bremer, and V. Pascucci, “Parallel Gradient Domain Processing of Massive Images,” in *Eurographics Symposium on Parallel Graphics and Visualization (2011)*, 2011, p. 9.

[22] M. D. Theys, R. M. Born, M. D. Allemang, and H. J. Siege, “Morphological Image Processing on Three Parallel Machines,” in *Frontiers of Massively Parallel Computing, Sixth Symposium*, 1996, pp. 327–334.

[23] C. Bakal, J. Aach, G. Church, and N. Perrimon, “Quantitative morphological signatures define local signaling networks regulating cell morphology.,” *Science (New York, N.Y.)*, vol. 316, no. 5832, pp. 1753–6, Jun. 2007.

[24] K. Huang and R. F. Murphy, “From quantitative microscopy to automated image understanding.,” *Journal of biomedical optics*, vol. 9, no. 5, pp. 893–912, 2004.

[25] N. Orlov, L. Shamir, T. Macura, J. Johnston, D. M. Eckley, and I. G. Goldberg, “WND-CHARM: Multi-purpose image classification using compound image transforms,” *Pattern Recognition Letters*, vol. 29, no. 11, pp. 1684–1693, Aug. 2008.

[26] M.-K. Hu, “Visual Pattern Recognition by Moment Invariants,” *IRE Transactions on Information Theory*, pp. 179–188, 1962.

[27] R. M. Haralick, M. Shanmugan, and I. Dinstein, “Textural features for Image Classification,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-3, no. 6, pp. 610–621, 1973.

[28] Y. Saeys, I. Inza, and P. Larrañaga, “A review of feature selection techniques in bioinformatics.,” *Bioinformatics (Oxford, England)*, vol. 23, no. 19, pp. 2507–17, Oct. 2007.

[29] Microsoft, “Deep Zoom Silverlight,” *Microsoft Developer Network (MSDN)*, 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/cc645050(v=vs.95).aspx. [Accessed: 27-May-2013].

[30] “Open Seadragon,” *Open Seadragon project*, 2013. [Online]. Available: http://openseadragon.codeplex.com/. [Accessed: 15-May-2013].

[31] R. Kooper and P. Bajcsy, “Computational Scalability of Large Size Image Dissemination,” in *IS&T/SPIE Electronic Imaging*, 2011, pp. 7872–23.

[32] R. Kooper, P. Bajcsy, and N. M. Hernández, “Stitching Giga Pixel Images Using Parallel Computing,” in *IS&T/SPIE Electronic Imaging*, 2011, pp. 7872–17.

[33] Yahoo!, “Hadoop Tutorial from Yahoo!,” *On-Line Tutorial*, 2013. [Online]. Available: http://developer.yahoo.com/hadoop/tutorial/module4.html. [Accessed: 14-May-2013].

[34] R. Dudko, A. Sharma, and J. Tedesco, “Effective Failure Prediction in Hadoop Clusters,” 2012. [Online]. Available: https://wiki.engr.illinois.edu/download/attachments/195766887/JAR-2nd.pdf?version=3&modificationDate=1333424381000. [Accessed: 31-May-2013].

[35] E. Bortnikov, A. Frank, K. Tivon, and E. Hillel, “Predicting Execution Bottlenecks in Map-Reduce Clusters,” 2010. [Online]. Available: https://www.usenix.org/system/files/conference/hotcloud12/hotcloud12-final50.pdf. [Accessed: 31-May-2013].

[36] I. Foster, *Designing and Building Parallel Programs*. Chicago, IL: ADDISON WESLEY Publishing Company Incorporated, 1995, p. 381.

[37] D. H. Woo and H. S. Lee, “Extending Amdahl’s Law for Energy-Efficient Computing in the Many-Core Era,” *IEEE Computer*, pp. 24–32, 2008.