Parallel Versus Distributed Data Access for Gigapixel-Resolution Histology Images: Challenges and Opportunities

Esma Yildirim and David J. Foran

***Abstract*—Recent advances in digital pathology technology have led to significant improvements in terms of both the quality and resolution of the resulting images, which now often exceed several gigabytes each. Today, several leading institutions across the country utilize whole-slide imaging (WSI) as part of their routine workflow. WSIs have utility in a wide range of diagnostic and investigative pathology applications. The fact that these images are both large in size (about 30 GB when uncompressed) and are generated in nonstandard proprietary formats has limited wider adoption of these technologies and makes the task of accessing, processing, and analyzing them in high-throughput fashion extremely challenging. The common approach for such data analytic applications is to preprocess the large whole-slide images into smaller size files and store them in a generic format. However, this approach limits the advantages that might be realized if different scalability levels and data unit sizes could be dynamically changed based on the specifications of the task at hand and the architectural limits of the infrastructure (e.g., node memory size). Such strategies also introduce extra processing time to the workflow. To address these challenges, we present, in this paper, novel scalable access methods for parallel file systems and distributed file/object storage systems. Experimental results gathered during the course of our studies show that these methods provide opportunities not realizable using traditional approaches. We demonstrate tangible, scalability, and high-throughput advantages using a Lustre parallel file system and AWS S3 distributed storage system.**

***Index Terms*—Amazon EMR, amazon S3, distributed file systems, Hadoop, HDFS, histopathology, Lustre, parallel file systems, whole-slide imaging (WSI).**

Manuscript received February 25, 2016; revised May 9, 2016; accepted June 3, 2016. Date of publication June 13, 2016; date of current version June 29, 2017. This work was supported, in part, by grants from the National Institutes of Health through Contracts 5R01CA156386-10 and 5R01CA161375-04 from the National Cancer Institute and Contracts 5R01LM009239-07 and 5R01LM011119-04 from the National Library of Medicine. This work used the Extreme Science and Engineering Discovery Environment, which was supported by the National Science Foundation under Grant ACI-1053575. The results published or shown in this paper are in whole or part based upon data generated by the TCGA Research Network (http://cancergenome.nih.gov/).

E. Yildirim is with the Rutgers Discovery Informatics Institute, Piscataway, NJ 08854 USA, and also with the Rutgers Cancer Institute of New Jersey, New Brunswick, NJ 08903 USA (e-mail: esma.yildirim@rutgers.edu).

D. J. Foran is with the Rutgers Cancer Institute of New Jersey, New Brunswick, NJ 08903 USA, and also with the Rutgers Robert Wood Johnson Medical School, New Brunswick, NJ 08901 USA (e-mail: foran@cinj.rutgers.edu).

Digital Object Identifier 10.1109/JBHI.2016.2580145

I. Introduction

WHOLE-SLIDE images (WSIs) are very large multiple resolution (e.g., multigigapixel) microscopy datasets produced by digital scanning devices. This relatively new technology has been adopted for use by many leading institutions worldwide for diagnostic, education, and research purposes. Whole-slide scanning offers some advantages over traditional microscopy methods including long-term archiving, ease of reuse, and remote consultation and analysis. Recent advances in whole-slide scanner technology makes it possible to scan a typical single glass slide in approximately 3 min, resulting in vast amounts of data in a very short amount of time [1]. The available big data produced by these scanners has become a rich source for large-scale data analytic applications.

Much of the software developed for WSI is focused on visualization [2], [3] with fewer applications concentrating on quantitative analysis where it might be useful to provide objective reproducible measures to support diagnostic decisions. Analytics applications such as automated content-based image retrieval and classification of cancer types require expensive computational operations such as low-level transformations, object segmentation, feature computation, filtering, and subsampling [4]. Recently, several studies have also begun to investigate the use of deep learning strategies on WSIs in an attempt to resolve complex problems in diagnostic pathology. For example, in [5], a multiple instance convolutional neural network (CNN) was used to classify glioblastoma and low-grade glioma cancers on the TCGA dataset successfully. Xu *et al.* [6], [7] used multiple instance learning with a combination of CNN, support vector machine, and principal component analysis on cancer image segmentation and classification problems. In each of these studies, the algorithms and workflow depended upon preprocessing the WSI data into smaller tiles because of the high cost of the computation in these types of neural networks.

The use of parallel processing algorithms and frameworks in WSI applications is very limited, and usually, access to the storage system is not considered as part of the parallelization process; therefore, no performance analysis is provided regarding this aspect of WSI data access. Zerbe, Hufnagl, and Schlüns [8] proposed a framework where a WSI server streams small patches to compute nodes; therefore, only sequential access to the storage system by the server is provided. Currently, their strategy supports a single server, which limits the throughput capacity of this approach.

In [4], several image-processing algorithms including object segmentation, feature extraction, classification, and tracking of pixel data access patterns are analyzed for CPU, GPU, and MIC processor architectures. However, in this paper, only memory access patterns are considered in their performance analysis, and a master/worker model similar to the work presented in [8] is adopted.

Qi *et al.* [9] present a cloud-computing-based parallel processing approach for content-based image retrieval in prostate cancer images. The WSIs are subdivided into smaller size images during the course of a preprocessing step and transferred to a storage system of the agent node within the worker site. Subsequently, the worker nodes access the disk system of the agent node of the site to process the image. The transfers to and from multiple disk systems introduce an additional overhead, which impedes overall performance.

A similar study to the parallel data access (PDA) method proposed in this paper is presented in [10]. Using this MPI-based approach, the master process assigns an image ID to the worker processes, and then, the workers access the storage system directly. However, in this application, the parallelization is provided for one single WSI image at a time, and the data size loaded into the memory of the worker processes changes according to the memory size of the node. This feature limits the scalability of the application to WSI size/memory size, without taking into account the characteristics of the dataset and the analytics algorithm. Also, fixing the tile size brought into the memory of the node based on the size of the WSI, and memory limit of the node may increase algorithm overhead. Although it is hard to make comparisons of this approach, since their results also include data-processing costs, their performance results show that it can only scale up to 17 processors. In this paper, our team presents a more flexible approach to the problem, which offers the benefit of significantly higher scalability results in data access.

In [11], issues related to large scale of spatial data (e.g., satellite imaging and WSIs) and high computational complexity of spatial queries are addressed using a solution based on the Hadoop ecosystem. In this approach, the WSIs are converted into an HDFS file system compatible format as a preprocessing step; however, the paper provides only limited performance results. In the work we present here, an efficient parallel conversion step to binary format is conducted on data originating from multiple input sources using symmetrically separable compressed WSIs on HDFS/AWS S3. There are also different file formats such as HDF5 which allow multiple processes to access a single file [12]. However, the files need to be converted into their format and all the access logic should be implemented by the processing application.

In this study, the challenges traditionally presented by WSIs are identified in detail, and two novel scalable data access methods are presented for parallel and distributed storage systems and extensive scalability results on Lustre [13], HDFS [14], and AWS S3 [15] storage systems depending upon the different image resolution requirements. The drawbacks and opportunities presented by these approaches are discussed, and their respective performances are compared in terms of data access time and scalability. The experimental results show that both approaches have high scalability and are sufficiently flexible to conform to a range of different data analytic algorithms.

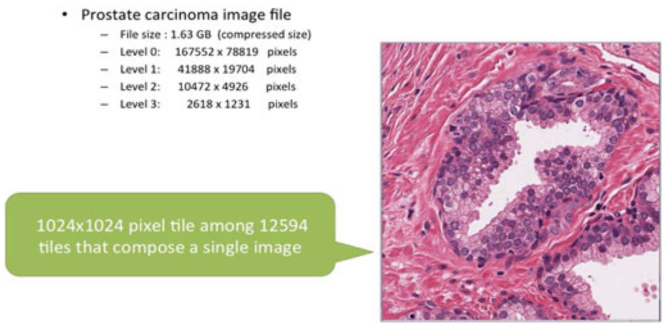
• Prostate carcinoma image file
– File size : 1.63 GB (compressed size)
– Level 0: 167552 x 78819 pixels
– Level 1: 41888 x 19704 pixels
– Level 2: 10472 x 4926 pixels
– Level 3: 2618 x 1231 pixels
1024x1024 pixel tile among 12594 tiles that compose a single image

Fig. 1. SVS image example (from Aperio scanner).

II. Challenges

This section describes the properties of WSIs and the challenges they present for large-scale medical applications. The section is grouped into three major categories.

A. Challenge 1: A WSI Can be Extremely Large

A single WSI may consist of a single or multiple files which can store pixel data at several different resolution levels depending on the brand and model of the whole-slide scanner device. At the highest resolution level, it can store around $100\,000 \times 100\,000$ pixel image data. Considering each pixel may be represented using RGB values, a single image can be around 30 GB when uncompressed. A typical WSI image file in compressed format may be as large as a few GBs. Fig. 1 shows a digitized section of prostate carcinoma tissue from the Cancer Genome Atlas [16] dataset in SVS image file format which has been generated using an Aperio [17] scanner. In this case, the compressed file is 1.6 GB and is stored at four different resolutions, the highest level (Level 0) resulting in a $167\,552 \times 78\,819$ (36-GB uncompressed) pixel image.

The uncompressed size of 30–40 GB makes it difficult and oftentimes inefficient to bring the entire image into memory for analytical processing. To address this difficulty, most approaches introduce a preprocessing step, where the image is divided into smaller size tiles and stored as separate files on the storage system. This might result in limitations on the analytics algorithm, extra and larger storage space in comparison to the original size to store the converted files, and limitations on the scalability of the application at hand. Considering the accuracy of the algorithms getting better with larger dataset sizes consisting of thousands of these images, this avalanche of big cancer image data places a heavy burden on storage systems and computational infrastructure and requires novel data access methods that can scale seamlessly.

B. Challenge 2: Proprietary File Formats

The vast majority of existing commercial WSI scanners utilize different digital data formats which rely upon proprietary

TABLE I
WHOLE-SLIDE SCANNER FILE FORMAT EXAMPLES [1], [18]

Brand	File Type	Format
Aperio-Leica	.svs	Single-file pyramidal tiled TIFF, with nonstandard metadata and compression
Hamamatsu	.vms, .vmu, .ndpi	Multifile JPEG/NGR with proprietary metadata and index file formats, and single-file TIFF-like format with proprietary metadata
Philips	.tiff	Single-file pyramidal tiled TIFF or BigTIFF with nonstandard metadata
Sakura	.svslide	BigTIFF, TIFF, JPG2000
Ventana	.bif, .tif	Single-file pyramidal tiled BigTIFF with nonstandard metadata and overlaps

metadata and compression techniques. Therefore, it becomes a very challenging task to develop generic applications which can work with data originating from one of these sources. Some of the file formats used by popular scanners are listed in Table I. To address this issue, several groups have begun to develop flexible interpreter libraries which can decipher these file formats and communicate with multiple storage system types in order to keep up with the big cancer data avalanche that is quickly emerging.

C. Challenge 3: Limited Access Libraries

The proprietary nature of the metadata and compression techniques generated by most vendors requires investigators and end users to utilize visualization tools supplied by the companies that design and manufacture these WSI devices. An often times the visualization software does not offer support for large-scale data analytics. There currently exist some libraries which provide compatibility with a relatively short sublist of the formats (e.g., Openslide [18]) from a WSI and read/write pixel data in different formats (e.g., Bio-Formats [19]). Unfortunately, none of them has parallel access capabilities and they do not support highly used distributed file systems such as HDFS or Cloud Storage Systems such as AWS S3. Furthermore, they require algorithms for balancing parallel access to WSIs stored on parallel file systems.

III. APPROACH

Two novel scalable data access methods which utilize parallel file systems (e.g., Lustre) and distributed file/object storage systems (e.g., HDFS and AWS S3) are presented in this section. While there is a general idea that parallel file systems and distributed file systems serve the same purposes, there are significant differences in their design choices and target workloads.

A parallel file system is usually installed on top of enterprise storage systems and is connected to the compute nodes of a supercomputer with fast interconnects like Infiniband. Their target workloads are generally HPC applications which might require fast random PDA; therefore, their block sizes are usually very small (e.g., 64 kB).

On the other hand, a distributed file system usually allows append-only writes. The applications are data independent and

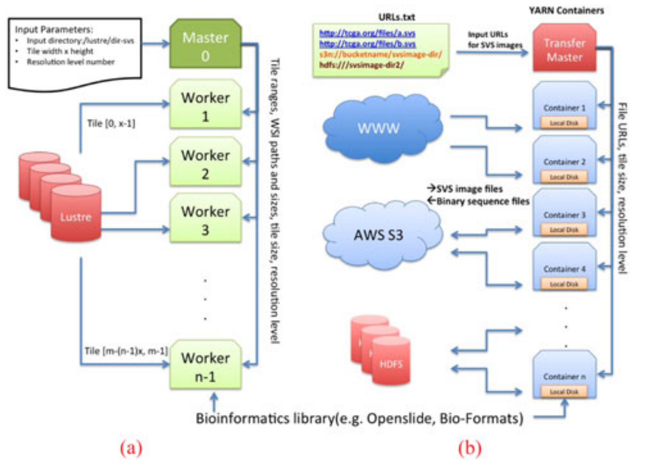

Fig. 2. (a) Parallel WSI access using MPI. (b) Distributed conversion of WSIs to binary sequence files.

sequential data access is more common resulting in larger block sizes (e.g., 128 MB). The WSI data access and their applications conform to both types of storage systems, which will be explained in detail in the following subsections.

A. Parallel Data Access

The PDA method presented in this section makes use of parallel file systems with fast interconnects. The MPI-based library balances the data access load evenly among processes, regardless of the dataset characteristics such as number of files, file size distribution, resolution level, and tile size, and the computational characteristics such as number of processes and memory size per node. The distribution algorithms at the core of this library are designed to achieve high scalability and maximization of parallel disk throughput of the file system.

1) PDA Design Choices: Fig. 2(a) illustrates communication steps between the master process and the corresponding worker processes. The important design choice that distinguishes it from studies presented in [8] and [4] is that file system access is not performed by a single-master process that serves the worker processes with input data, but multiple workers access the parallel file system at the same time utilizing its performance capabilities better. The PDA is also balanced to gain maximal scalability. The details of the tile distribution algorithm are given in Algorithm 1. The input parameters to the algorithm is the path of the WSI directory, the width and height of each tile which also represents unit read operation size performed by the workers, and, finally, the resolution level. The master process communicates with the file system to construct a list of file paths (Line 2) and the width and height of each resolution level for each image file (Line 3). Each worker is provided with the range of tiles they are supposed to read with a broadcast operation (Lines 4 and 5). This operation also sends the size of all images and their paths to the workers. The worker process calculates the coordinate of a specific tile in its range (Line 9) and accesses the file system by using the bioinformatics library (Openslide) to bring the tile into their memory for further processing (Line 10). The use of bioinformatics library helps identify the offsets in the file to read a tile at a specific

Algorithm 1 Balanced_Parallel_WSI_Access.

```
Require: $P_{WSI}$ : $Input\ path\ of\ WSI\ Images \vee$
$T_w$ : $Tile\ width \vee T_h$ : $Tile\ height \vee$
R : $Resolution\ level$
1: Master Process :
2: $L_{WSI} \leftarrow$ list_of_WSI_file_paths($P_{WSI}$)
3: $Sizes_{WSI_R} \leftarrow$
   retrieve_width_height_WSI($L_{WSI}, R$)
4: $Ranges_p \leftarrow$ calc_tile_range_per_process
   ($Sizes_{WSI_R}, T_w, T_h$)
5: $Broadcast\ L_{WSI} \vee Sizes_{WSI_R} \vee Ranges_i \vee T_w \vee$
   $T_h \vee R$
6: Worker Process :
7: $Receive\ L_{WSI} \vee Sizes_{WSI_R} \vee Ranges_i$
   $\vee T_w \vee T_h \vee R$
8: for all $Tile_{ij}\ in\ Ranges_i$ do
9:     $Tile_x, Tile_y \leftarrow$ calc_coords
       ($Sizes_{WSI_R}, Ranges_{ij}, T_w, T_h$)
10:    $buffer \leftarrow$ access_tile
       ($Tile_x, Tile_y, T_w, T_h, L_{WSI_k}, R$)
11: end for
```

coordinate plane. The number of tiles each worker process reads is evenly distributed, and consecutive tiles in the spatial plane are assigned to the same process.

The reason we leave the size of the tiles to the user to set as a parameter is to provide enough flexibility for the processing algorithm and keep the balanced PDA at the same time. It is important to note that once a tile as big as the user wants it to be is brought into the memory, using different data access patterns to access any coordinate in the tile does not matter, since the memory bandwidth is fixed. Therefore, different data access patterns are possible with our approach. This way, multiple WSIs with the chosen tile sizes that conform to the memory limit of the processes and the data analytic application's needs can be accessed in parallel and scale as high as the total disk throughput of the file system.

This method can be applied to 3-D WSI registration applications as well. These applications also work with pieces of WSIs, and only small adjustments are needed to stack cross sections together. This can easily be done by assigning tiles into processes in a fashion that conforms to the algorithm. Spatial locality in 2-D still needs to be protected, and adding an additional dimension would be just fine. Our distribution algorithm can be applied for each cross section separately to assign tile ranges to processes.

2) PDA Experimental Results: The performance results of the developed library are presented in Fig. 3. The experiments are conducted on the Stampede supercomputer of the XSEDE environment [20]. The characteristics of the testbeds used for experiments are given in Table II. Stampede has a Lustre parallel file system. The dataset used in the experiments consists of ten SVS images downloaded from the TCGA prostate carcinoma dataset (5.2-GB compressed size). The developed library is used to read the SVS directory and write them as uncompressed .ppm tile files.

Level 0 results [see Fig. 3(d)–(f)] represent the highest resolution level and the total data size becomes around 150 GB when it is uncompressed. Level 1 resolution level is 16x smaller then Level 0 and it is around 9.5 GB when uncompressed [see Fig. 3(a)–(c)]. The tile sizes selected for comparison are 1024×1024 and 4096×4096 for Level 1 and 0 resolution. The number of processors ranges between 1 and 1024 (64 Stampede nodes). Read and write speed ups are calculated by the average total read/write times for each process, and they represent a closer characteristics to the linear speed up line. The total speed up is calculated as the total execution time when all the processes are synchronized after they read and write all the tiles assigned to them.

The tile size affects the read and write times significantly. For Level 1 resolution, the total number of tiles for 1024×1024 pixel tile size is 2622, while it is only 212 for 4096×4096 pixel tiles. Therefore, if we keep the tile size large, there is a possibility that the scalability is limited by the total number of tiles. On the other hand, keeping the tile size large increases the throughput. The results indicate that at the Level 1 resolution, using 256 processes with 4096×4096 pixel tiles and using 1024 processes with 1024×1024 pixel tiles gives similar total times. Level 0 resolution results also indicate 4096×4096 pixel tile size takes less time to read/write. One of the reasons that smaller tile sizes cause larger read/write times with the same parallelism level than the larger tiles is because the total number of tiles is larger for smaller tile sizes. This number also corresponds to the number of disk accesses. Therefore, one can say that there is a tradeoff between maximum number of parallelism level which can be applied in data access and the total time if we range the tile sizes. We leave the choice of tile size to the user so that it is dynamically set based on the dataset size and the needs of the processing algorithm.

The speed up results are higher for smaller tile sizes, because there is still room for improvement of the throughput one can achieve from the file system. Based on that, the highest speed up results gained is a little over 600 and with Level 0 resolution and 1024×1024 tile size. The logarithmic increase in speed up might be due to the file system's capacity, imbalance between datasets due to left-over tile sizes, and synchronization overhead.

One important point to note is that there is still room for improvement as we have not observed a drop down in speed up in our experiments, and increasing the number of processes might help that. However, the total tile number and the maximum number of processes allowed in Stampede queue limit the expansion of the experiments. Looking at the time results, we can read and write 9.5 GB of data in 15 s with 256 cores and 150-GB data in 22 s with 1024 cores, which is a very good start for real-time applications.

3) PDA Opportunities: This section presents some of the drawbacks and also opportunities presented by the proposed PDA method. Opportunities are itemized as follows.

1) The preprocessing step which exists in most of the previous studies is eliminated. This allows us to archive the data in its original format without needing additional storage space for intermediate file formats also the analytic algorithm gains from processing time and can apply

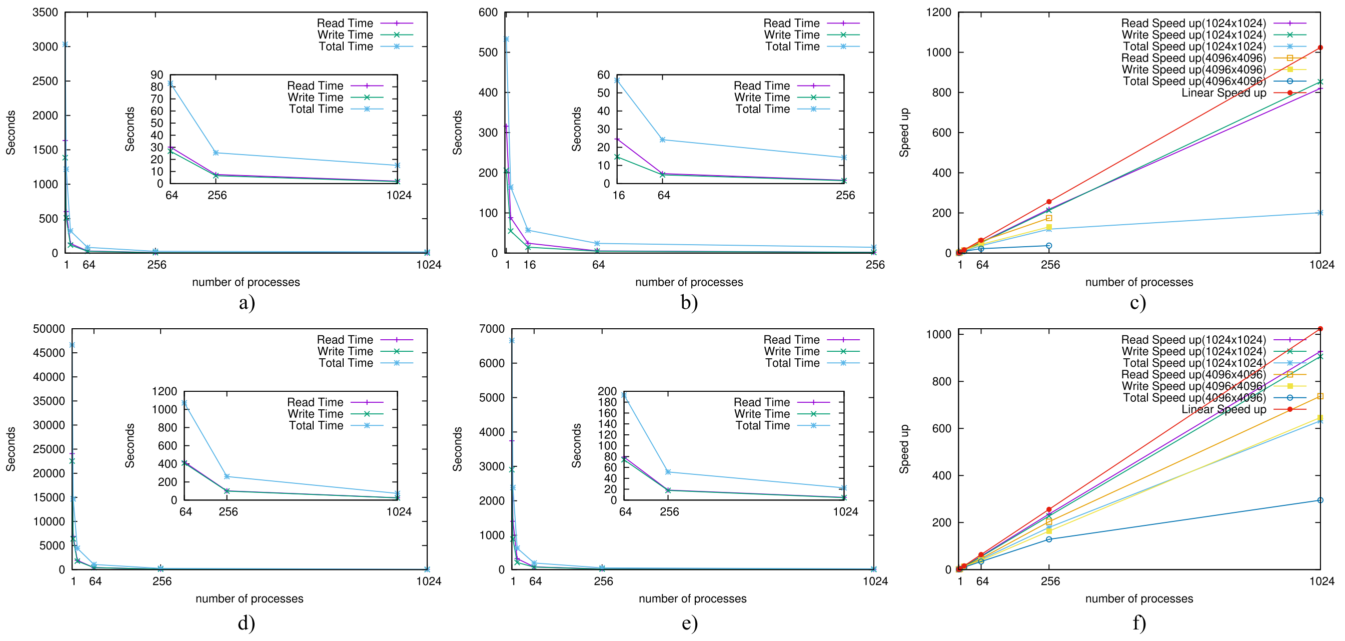

Fig. 3. Stampede Lustre file system, PDA performance results. (a) Level1 (1024 × 1024 pixel tiles)-Time to read/write, (b) Level1 (4096 × 4096 pixel tiles)-Time to read/write, (c) Level1-Speed up, (d) Level0 (1024 × 1024 pixel tiles)-Time to read/write, (e) Level0 (4096x4096 pixel tiles)-Time to read/write, (f) Level0-Speed up.

TABLE II
TESTBED

Environment	Cluster/Site	cores/node	Processor	Memory Size/node	Interconnect	File System
XSEDE	Stampede	16 cpus	Xeon E5-2680	32 GB	FDR Infiniband	Lustre
Amazon Web Services	Oregon, US-Standard	4vcpus(m3.xlarge instance)	Xeon E5-2670	15 GiB	500 Mpbs	S3 Object Storage
iMac Desktop	Local	8vcpus	Core i7	16 GB	1 Gbps	HDFS

different scalability levels and data unit sizes dynamically based on their needs and the architectural limits of the infrastructure (e.g., node memory size).

2) Parallel file systems are connected with fast interconnects and can provide high-speed access, which is absent in distributed file systems that are designed for commodity infrastructures. On the other hand, the scalability is limited by the maximum file system throughput, while it is not an issue in distributed file systems.

3) Use of MPI as a data access medium allows us to support data-dependent applications where interprocess communication occupies a high portion of the total execution time. For example, once tiles are loaded into the memory of the node, there is nothing preventing them to share boundary pixels with neighboring processes.

B. Distributed Data Access (DDA)

The current bioinformatic libraries that can decipher data formats of WSIs do not comply with distributed file systems (e.g., HDFS) or object storage systems (e.g., AWS S3); therefore, preprocessing becomes necessary to convert the original file format to a symmetrically splittable format in a distributed storage system so that it can be read by parallel libraries in partial data splits. However, this step can also be applied dynamically and conducted in a performance efficient way using our methods. For the data conversion and access methods which will be described in the following subsections, we have used Hadoop environment to communicate with HDFS and AWS S3 storage systems, where data transferred from different input sources using different protocols can be transformed and stored.

1) DDA Design Choices: A flexible design choice we have adopted is that the data can originate from different types of sources and can be accessible by specific set of APIs and protocols. We do not have the restriction that it should reside in a parallel file system anymore. Fig. 2(b) illustrates the steps of the conversion process and communication patterns in data transfer. The input to the conversion process is a text file, which contains the URLs of WSIs. The first two lines indicate two SVS files from a web repository, the third path is an AWS S3 directory that contains the images, and the fourth one is an HDFS directory path.

The distributed conversion algorithm is written as a native YARN application which can communicate with YARN, the job scheduler of Hadoop ecosystem. The outline of the algorithm is given in Algorithm 2. The algorithm takes the input text file which contains the URLs for the WSI images, the resolution level and tile size as input parameters. The transfer master

Algorithm 2 Distributed_WSI_Conversion_To_Sequence _Files.

```
Require: $P_{WSI}$ : $Input\ path\ of\ text\ file\ containing\ WSI\ URLs \vee T_w$ : $Tile\ width \vee T_h$ : $Tile\ height \vee R$ : $Resolution\ level$
 1: $\underline{Transfer\ Master}$ :
 2: $L_{URL} \leftarrow$ list_of_file_URLs($P_{WSI}$)
 3: for all $URL_i$ in $L_{URL}$ do
        4: launch_worker_container($URL_i, R, T_w, T_h$)
 5: end for
 6: $\underline{Worker\ Container}$ :
 7: $URL\_type\_x \leftarrow$ recognize_URL_type($URL_i$)
 8: $local\_file\_path \leftarrow$ transfer_to_local_disk ($URL\_type\_x, URL_i$)
 9: $Width, Height \leftarrow$ retrieve_size_of_WSI ($local\_file\_path$)
10: $List_{tiles} \leftarrow$ retrieve_tile_coords ($Width, Height, local\_file\_path$)
11: for all $Tile_i$ in $List_{tiles}$ do
12:     $Key \leftarrow file\_name, x\_coord_i, y\_coord_i, tile\_width, tile\_height$
13:     $Value \leftarrow$ read_tile_pixels_from_WSI ($local\_file\_path, x\_coord_i, y\_coord_i$)
14:     write_to_sequence_file ($Key, Value, destination\_path$)
15: end for
```

goes over the URLs in the text file and recognizes URL types. Then, by using the appropriate protocols for each URL type, it interrogates the storage systems to make a global list of image files (Line 2). For each file URL in the list, the master launches worker YARN containers who are responsible with the transfer, transformation, and storage of the files on the local HDFS system or back on AWS S3 storage system (Lines 3–5). A simple scheduling algorithm is applied in which each container is assigned a single URL to process. The application running on the worker container recognizes the URL type and uses appropriate transfer protocol or API to transfer the file into its local disk (Lines 7 and 8). It then uses the bioinformatic library to read the metadata of the file and gathers a list of coordinates based on the resolution level and tile size parameters (Lines 9 and 10). For each tile coordinate, it creates a key which consists of the WSI file name, x and y coordinates, and size of the tile. The metadata about each tile is stored in the key value because it makes it very easy to implement specific Hadoop partitioner and groupby classes that can be used to implement select-where clauses or join objects. This property allows various analytic applications and different access patterns to be possible with sequence files. The worker application then reads the raw partial pixel data in the form of a tile, converts it into a value object, and writes key-value pair into HDFS or S3 as part of a sequence file (Lines 11–15). Sequence files are compressed binary object files which can be split by programming paradigms like Hadoop MapReduce [14] or Spark [21] automatically.

There are two important questions which must be addressed here:

1) *Why NOT use a MapReduce job instead of a YARN job?* There are multiple reasons why we cannot use a traditional MapReduce job. First, the input file is a small file consisting of URLs which will be split into 1 by a Mapreduce job because it is much smaller than the block size of the HDFS system. In this case, all URLs will be processed by a single worker container. A YARN job on the other hand is a lower level job where the input data can be split according to custom algorithms. Second, MapReduce does not know how to read a WSI file. To be able to split it into multiple workers, it would have to bring the entire file in uncompressed format into the memory, which is not feasible as each file can be as big as 40 GB. Finally, we are not limited to HDFS and S3 storage systems as sources of the input files. As it is clear from the architecture, the files can even reside on a web server.
2) *Why NOT keep the transferred file in the memory but store on local disk of the worker container?* The bioinformatics library, which is used to read raw pixel data cannot read from memory and does not know how to communicate with a distributed file system. Additionally, the file might be too big in uncompressed format to bring into memory.

2) DDA Experimental Results: The same dataset used in the PDA method experiments is also used in these experiments using a 8vcpu iMac computer with a Hadoop ecosystem. To calculate the speed up of the proposed method, a single file performance is compared to the performance of the entire dataset conversion process as the scheduling algorithm will set parallelism level based on the number of files in the dataset. Tables III and IV present the conversion time and speed up results for Level 1 and Level 0 resolutions. Two different compression techniques are used to see the tradeoff between performance and storage space. DEFLATE is the default compression technique, which is the technique used by gzip with a few metadata differences. DEFLATE's performance is better than the BZIP2 method; however, BZIP2 compresses the data better (see Table V). The conversion process time for Level 1 resolution drops from 248 to 162 s, which corresponds to a 34% decrease in execution time when DEFLATE is used instead of BZIP2; however, datasize only decreases 12% if BZIP2 is used. In this case, sacrificing from storage space to gain from performance seems to be a better choice; therefore, the rest of the experiments are conducted using the DEFLATE compression technique. The percentages for Level 0 are 51% and 28%, respectively, indicating the same trend.

In addition to the compression techniques, different source–destination URL pairs are tested as well. The best speed up results are obtained with transfers occurring between S3-HDFS and S3-S3 source destination pairs because of wide area transfer costs. The speed up results of Level 0 and Level 1 also show similarities. The maximum speed up that can be obtained on the system is 7, since the master application reserves one vcpu for itself and the rest of the seven vcpus can be allocated to worker containers. The highest speed up achieved by the conversion process is 4.7 and 3.17 for Level 1 and Level 0, respectively.

TABLE III
LEVEL 1—1024 × 1024 PIXEL TILES—CONVERSION RESULTS

Codec Type	Source	Destination	Conversion Time (secs)-Ten Files	Conversion Time (secs)-One File	Speed Up
BZIP2	HDFS	HDFS	**248.74**	**67.42**	2.95
DEFLATE	HDFS	HDFS	**162.50**	**39.22**	2.56
DEFLATE	AWS S3	HDFS	322.38	145.22	4.77
DEFLATE	AWS S3	AWS S3	407.49	177.19	4.61
DEFLATE	HDFS	AWS S3	238.27	77.08	3.43
DEFLATE	HTTP	HDFS	204.50	53.20	2.76

TABLE IV
LEVEL 0—4096 × 4096 PIXEL TILES—CONVERSION RESULTS

Codec Type	Source	Destination	Conversion Time (secs)-Ten Files	Conversion Time (secs)-One File	Speed Up
BZIP2	HDFS	HDFS	**4324.30**	**1107.25**	2.53
DEFLATE	HDFS	HDFS	**2120.47**	**483.06**	2.18
DEFLATE	HDFS	AWS S3	3138.35	1038.09	3.17

TABLE V
DATASET SIZE IN MB—BEFORE AND AFTER CONVERSION

Codec Type	Orig. Size (Ten Files)	Orig. Size (One File)	L1 Size (Ten Files)	L1 Size (One File)	L0 Size (Ten Files)	L0 Size (One File)
BZIP2	5350.11	609.37	2066.72	190.10	24092.35	2442.90
DEFLATE	5350.11	609.37	2370.63	223.65	33757.91	3526.62

Once the WSIs are converted into binary sequence files, they can be accessed by any programming paradigm that supports the Hadoop ecosystem (e.g., Spark).

To measure the performance of the sequence file access, AWS Elastic MapReduce service in the US-Standard region was used to gather scalability results. In these experiments, m3.xlarge instances were used, each equipped with four vcpus. The details of the architecture was given in Table II. The instance numbers ranged between 1 and 16 (64vcpus at max.) for Level 1 resolution sequence files, because the storage system automatically divided it into 36 splits, while the Level 0 sequence files were divided into around 380 splits. Therefore, the instance numbers for them ranged between 1 and 64 (256vcpus at max.).

The results indicate that there is not a significant difference in read times if we range the tile size (see Fig. 4). Unlike PDA approach in which the tile size affects the throughput a lot, in the DDA approach, it does not matter. The Hadoop ecosystem retrieves data in splits. Since the split size is fixed based on the total data size, the results are similar for different tile sizes. At the Level 1 resolution, both 1024 × 1024 and 4096 × 4096 pixel tile datasets are divided into 36 splits, and for the Level 0 resolution, they are divided into 383 and 380 splits, respectively. The speed up reaches to 5 for Level 1 results for 36 vcpus and 26 for Level 0 results with 256 vcpus as we increase the number of instances. The Level 1 dataset can be read in 50 s, while the Level 0 dataset is read in 105 s.

3) DDA Opportunities: Several opportunities arise in using DDA and are itemized as follows.

1) Most parallel programming paradigms that are derived from the ideas of the Hadoop ecosystem try to be backward compatible with HDFS file system (e.g., Spark, Tez). So, the range of programming tools which complies with the sequence file format varies.
2) The parallel conversion step provides flexible access to any remote system as long as the transfer means are supplied. In the experiments, examples with HTTP, S3, and HDFS were shown as use cases. YARN's flexible scheduling features allow any type of transfer scheduling algorithm to be developed.
3) While a parallel file system is limited with its interconnect and disk capacities, a distributed file system can scale with the node numbers, because it uses the local disks of the nodes. Cloud storage also becomes possible option.
4) The plethora of data-independent applications in which communication is negligible also allows a good range of analytic applications to use DDA access.
5) Several compression formats are possible and custom formats can be developed as well.

IV. PERFORMANCE COMPARISON

In this section, the performances of two data access methods are compared in terms of read time, speed up, and throughput. Since the architectures of the two testbeds are very different from each other, it is hard to make a fair comparison. Nevertheless, the results presented in Fig. 5 match the vcpu numbers of AWS

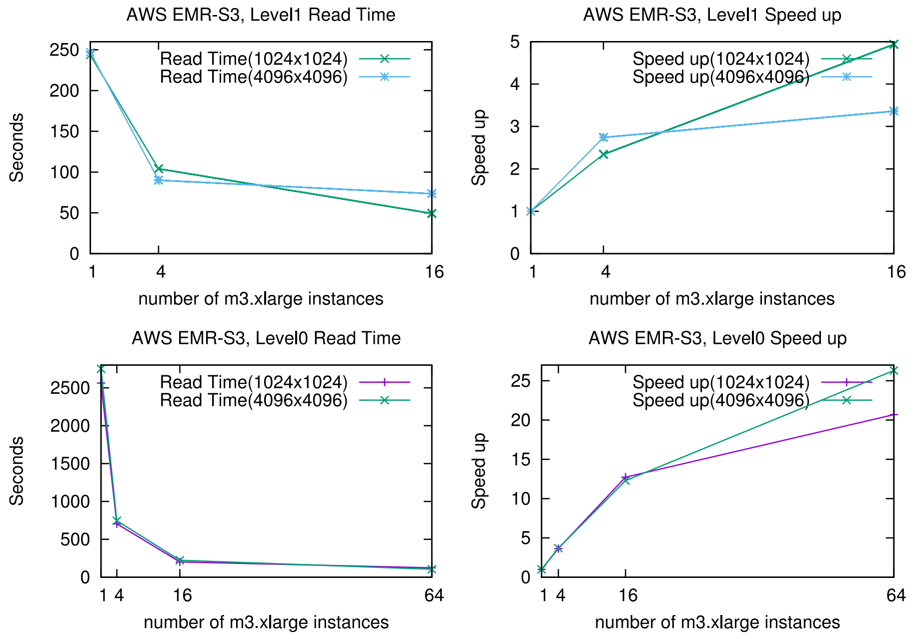

Fig. 4. AWS EMR-S3, DDA performance results (Level 1 (#splits: 36(1024 × 1024), 36(4096 × 4096)), Level 0 (#splits: 383(1024 × 1024), 380(4096 × 4096))).

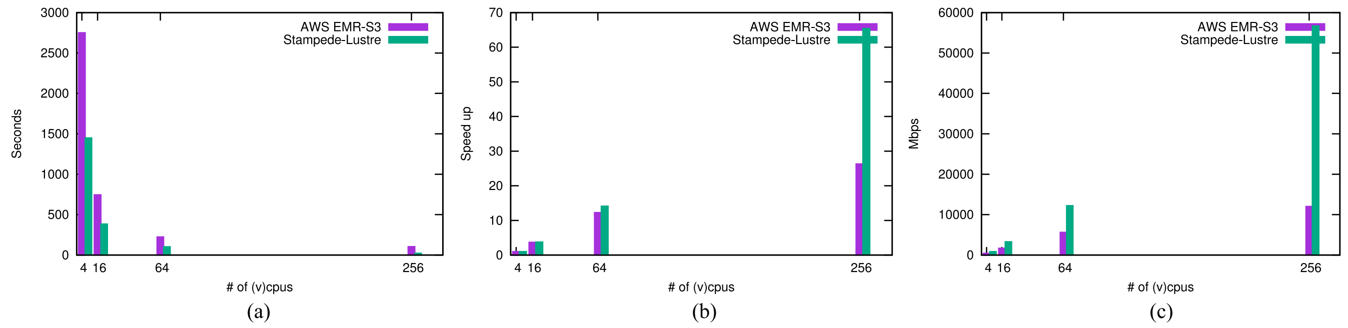

Fig. 5. Performance comparison of DDA approach on AWS EMR-S3 to PDA approach on Stampede-Lustre at Level 0 resolution. (a) Read Time - AWS EMR-S3 vs. Stampede-Lustre, (b) Speed up - AWS EMR-S3 vs. Stampede-Lustre, (c) Throughput - AWS EMR-S3 vs. Stampede-Lustre

m3.xlarge instances to the physical core number of Stampede nodes. Since the data parallelism level is higher (380 splits), the analysis is made based on the Level 0 4096 × 4096 tile size results.

The data read time results indicate that Stampede-Lustre performance results are at least 2× better than AWS S3 results [see Fig. 5(a)]. This is understandable, because a vcpu on an AWS instance is actually a hyperthread, which may share the physical core with other threads. If we look at the speed up results, it is obvious that both methods scaled with similar ratios until we change the (v)cpu number from 64 to 256. The linear scalability of AWS S3 performance changes into a logarithmic scalability, while Stampede Lustre continues to scale at the same ratio [see Fig. 5(b)]. This decrease may be related to many parameters. However, the network interconnects and the parallelism level of S3 storage system would be among the main reasons. The hardest comparison match would be in the throughput results because while PDA approach reads from the original files compressed with a custom technique (5.2 GB), the DDA approach reads from sequence files compressed with DEFLATE algorithm (32 GB). Therefore, the throughput calculations were made based on the original uncompressed data size which is around 150 GB. According to this, the throughputs of both approaches show similar characteristics to the read time metric [see Fig. 5(c)]; however, if the original compressed data sizes were to be used in calculations, then S3 performance would have been much better.

V. Conclusion

WSIs present many challenges for data analytic algorithms in terms of their size, proprietary metadata formats, and compression techniques. Novel scalable data access methods leveraging the speed of parallel and distributed file/storage systems can significantly improve the performance of these algorithms. The parallel and DDA methods presented in this paper overcome the challenges presented by WSIs and show high scalability results. PDA method eliminates the preprocessing step and leverages the fast interconnects of parallel file systems of supercomputers, and, therefore, presents superior performance.

On the other hand, the DDA method can convert data coming from different types of source to a symmetric splittable format accessible by distributed file systems and Cloud storage systems and can run on cheap hardware which can catch PDA performances.

As future work, we are in the process of applying these data access methods to design an extremely scalable Content-based Image Retrieval Workflow and our preliminary results for PDA method indicate that we are able to obtain 170 times faster execution times for a similar dataset on a 80 node cluster (24 core per node) where each node is used as a single data access process and multiple threads in each node are used for the processing algorithms on tiles brought into the memory.

References

[1] N. Farahani, A. Parwani, and L. Pantanowitz, “Whole slide imaging in pathology: advantages, limitations, and emerging perspectives,” *J. Pathol. Lab. Med. Int.*, vol. 7, pp. 23–33, 2015.

[2] Imagescope, 2016. [Online]. Available: http://www.leicabiosystems.com/pathology-imaging/aperio-digital-pathology/integrate/imagescope/

[3] C. A. Schneider, W. S. Rasband, and K. W. Eliceiri, “NIH image to imageJ: 25 years of image analysis,” *Nature Methods*, vol. 9, no. 7, pp. 671–675, 2012.

[4] G. Teodoro, T. Kurc, J. Kong, L. Cooper, and J. Saltz, “Comparative performance analysis of Intel (R) Xeon Phi (TM), GPU, and CPU: A case study from microscopy image analysis,” in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, 2014, pp. 1063–1072.

[5] L. Hou, D. Samaras, T. M. Kurc, Y. Gao, J. E. Davis, and J. H. Saltz, “Efficient multiple instance convolutional neural networks for gigapixel resolution image classification,” *arXiv preprint arXiv:1504.07947*, 2015.

[6] Y. Xu *et al.*, “Deep learning of feature representation with multiple instance learning for medical image analysis,” in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2014, pp. 1626–1630.

[7] Y. Xu, J.-Y. Zhu, I. Eric, C. Chang, M. Lai, and Z. Tu, “Weakly supervised histopathology cancer image segmentation and classification,” *Med. Image Anal.*, vol. 18, no. 3, pp. 591–604, 2014.

[8] N. Zerbe, P. Hufnagl, and K. Schlüns, “Distributed computing in image analysis using open source frameworks and application to image sharpness assessment of histological whole slide images,” *Diagn Pathol*, vol. 6, no. Suppl 1, p. S16, 2011.

[9] X. Qi *et al.*, “Content-based histopathology image retrieval using cometcloud,” *BMC Bioinformat.*, vol. 15, no. 1, p. 287, 2014.

[10] G. Bueno *et al.*, “A parallel solution for high resolution histological image analysis,” *Comput. Methods Programs Biomed.*, vol. 108, no. 1, pp. 388–401, 2012.

[11] A. Aji *et al.*, “Hadoop GIS: A high performance spatial data warehousing system over mapreduce,” *Proc. VLDB Endowment*, vol. 6, no. 11, pp. 1009–1020, 2013.

[12] Hdf5, 2016. [Online]. Available: https://www.hdfgroup.org/HDF5/

[13] Lustre parallel file system, 2016. [Online]. Available: https://lustre.org

[14] Hadoop distributed file system, 2016. [Online]. Available: http://hadoop.apache.org

[15] Amazon simple storage service, 2016. [Online]. Available: https://aws.amazon.com/s3/

[16] The cancer genome atlas, 2016. [Online]. Available: http://cancergenome.nih.gov

[17] Aperio, 2016. [Online]. Available: http://www.leicabiosystems.com/pathology-imaging/aperio-digital-pathology/

[18] Openslide, 2016. [Online]. Available: http://openslide.org

[19] Bio-formats, 2016. [Online]. Available: http://www.openmicroscopy.org/site/products/bio-formats

[20] J. Towns *et al.*, “Xsede: Accelerating scientific discovery,” *Comput. Sci. Eng.*, vol. 16, no. 5, pp. 62–74, 2014.

[21] Apache spark, 2016. [Online]. Available: http://spark.apache.org

Esma Yildirim received the Ph.D. degree from the Computer Science Department, Louisiana State University, Baton Rouge, LA, USA, in 2010.

She is currently a Research Associate with the Rutgers Discovery Informatics Institute, Piscataway, NJ, USA, and the Rutgers Cancer Institute of New Jersey, New Brunswick, NJ. She worked at the University at Buffalo as a Scientific Researcher until 2012. She was an Assistant Professor at the Computer Engineering Department, Fatih University, until 2015. Her research interests include data-intensive distributed computing, bioinformatics, cloud computing and optimization in high-performance computing, networking, and storage systems.

David J. Foran is currently a Professor of pathology with the Laboratory Medicine and Radiology and the Chief of the Division of Medical Informatics, Rutgers Robert Wood Johnson Medical School, New Brunswick, NJ, USA. He also serves as Executive Director of Computational Imaging and Biomedical informatics and Chief Informatics Officer at the Rutgers Cancer Institute of New Jersey, New Brunswick. His research interests include the design, development, and implementation of new approaches in computer-assisted diagnostics, medical imaging, and statistical pattern recognition for resolving challenging clinical problems in pathology, radiology, and oncology.