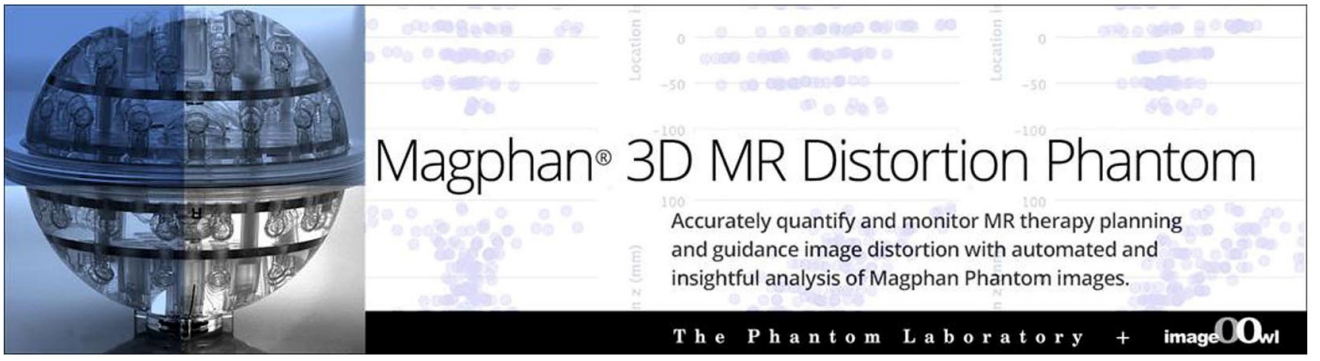
Medical Physics
The International Journal of Medical Physics Research and Practice

Ultrafast and scalable cone-beam CT reconstruction using MapReduce in a cloud computing environment

Bowen Meng, Guillem Pratx, and Lei Xing

Citation: Medical Physics **38**, 6603 (2011); doi: 10.1118/1.3660200
View online: http://dx.doi.org/10.1118/1.3660200
View Table of Contents: http://scitation.aip.org/content/aapm/journal/medphys/38/12?ver=pdfcov
Published by the American Association of Physicists in Medicine



Ultrafast and scalable cone-beam CT reconstruction using MapReduce in a cloud computing environment

Bowen Meng
Department of Electrical Engineering, Stanford University, California 94305 and Department of Radiation Oncology, Stanford University School of Medicine, Stanford, California 94305

Guillem Pratx
Department of Radiation Oncology, Stanford University School of Medicine, Stanford, California 94305

Lei Xing[a)]
Department of Radiation Oncology, Stanford University School of Medicine, Stanford, California 94305 and Molecular Imaging Program, Stanford University School of Medicine, Stanford, California 94305

(Received 1 June 2011; revised 20 September 2011; accepted for publication 20 October 2011; published 23 November 2011)

Purpose: Four-dimensional CT (4DCT) and cone beam CT (CBCT) are widely used in radiation therapy for accurate tumor target definition and localization. However, high-resolution and dynamic image reconstruction is computationally demanding because of the large amount of data processed. Efficient use of these imaging techniques in the clinic requires high-performance computing. The purpose of this work is to develop a novel ultrafast, scalable and reliable image reconstruction technique for 4D CBCT/CT using a parallel computing framework called MapReduce. We show the utility of MapReduce for solving large-scale medical physics problems in a cloud computing environment.

Methods: In this work, we accelerated the Feldcamp–Davis–Kress (FDK) algorithm by porting it to Hadoop, an open-source MapReduce implementation. Gated phases from a 4DCT scans were reconstructed independently. Following the MapReduce formalism, Map functions were used to filter and backproject subsets of projections, and Reduce function to aggregate those partial backprojection into the whole volume. MapReduce automatically parallelized the reconstruction process on a large cluster of computer nodes. As a validation, reconstruction of a digital phantom and an acquired CatPhan 600 phantom was performed on a commercial cloud computing environment using the proposed 4D CBCT/CT reconstruction algorithm.

Results: Speedup of reconstruction time is found to be roughly linear with the number of nodes employed. For instance, greater than 10 times speedup was achieved using 200 nodes for all cases, compared to the same code executed on a single machine. Without modifying the code, faster reconstruction is readily achievable by allocating more nodes in the cloud computing environment. Root mean square error between the images obtained using MapReduce and a single-threaded reference implementation was on the order of 10^{-7}. Our study also proved that cloud computing with MapReduce is fault tolerant: the reconstruction completed successfully with identical results even when half of the nodes were manually terminated in the middle of the process.

Conclusions: An ultrafast, reliable and scalable 4D CBCT/CT reconstruction method was developed using the MapReduce framework. Unlike other parallel computing approaches, the parallelization and speedup required little modification of the original reconstruction code. MapReduce provides an efficient and fault tolerant means of solving large-scale computing problems in a cloud computing environment. [DOI: 10.1118/1.3660200]

Key words: CBCT, 4DCT, MapReduce, cloud computing, FDK

I. INTRODUCTION

Four-dimensional CT (4DCT) and cone beam CT (CBCT) are used clinically in radiation therapy for accurate and updated tumor target definition and localization.[1,2] Long reconstruction times in CBCT and 4DCT give rise to the need for a fast and reliable technique for various applications in treatment planning.[3] The computational complexity—linearly dependent on the volume size, phase number and the number of projections in each phase—prohibits the use of single-threaded processing. Moreover, the rapidly increasing size of CT projection data makes it challenging to perform reconstruction on a local machine with limited hardware capacity. Accordingly, practical 4D CBCT reconstruction is inevitably shifting to distributed and parallel architectures for higher efficiency.

Some of the cloud computing technologies used to solve Internet-scale problems can be applied to medical physics computational problems.[4–6] MapReduce,[7] developed at Google (Google, Inc., Mountain View, CA) for reliable computing on large-scale parallel architectures, is designed to facilitate the

development of data processing applications onto massively-parallel architectures, such as a cloud computing environment. It is the result of recent developments in virtualization technology and years of research in distributed computing. Virtualization software abstracts the underlying hardware architectures (such as servers, storage, and networking) and allows nodes to be created on demand with flexible specification of hardware parameters, such as the number of processors, memory, disk size, and operating system.

The most powerful feature of MapReduce is its simplicity. Existing codes can be naturally translated into the Map/Reduce paradigm, and developers can schedule many computer nodes on demand without worrying about the underlying hardware architecture. MapReduce also hides the parallelization, data distribution, fault tolerance, and load balancing from developers, allowing programmers to focus on the design of Map and Reduce functions.

Although MapReduce implementations can run on several types of architectures, including dedicated clusters and graphics processing units (GPUs),[8] its full potential is achieved in a cloud computing environment. Cloud computing providers offer a wide variety of services, including web-based software, data storage, and built-in MapReduce capability, facilitating the use of distributed computing for developers without in-depth knowledge on parallelization.

The MapReduce framework is presented in Sec. II. In Sec. III, we introduce a novel strategy for performing Feldcamp–Davis–Kress[10] (FDK) reconstruction using MapReduce. Section IV demonstrates the efficiency and reliability of the proposed method and Sec. V is discussion.

II. MAPREDUCE

MapReduce is a programming framework for processing large data sets on clusters of computers (nodes). In this framework, developers specify a Map function that takes input data and generates a set of intermediate Key/Value pairs, and a Reduce function that merges all the intermediate values that share the same intermediate key

$$\begin{aligned} \text{map}: \quad & v_1 \rightarrow \text{list}(k_2, v_2), \\ \text{reduce}: \quad & [k_2, \text{list}(v_2)] \rightarrow v_3. \end{aligned} \tag{1}$$

On a large cluster of commodity machines, the input data are first split into chunks that are used as input by Map tasks. Map and Reduce tasks are automatically distributed and executed in a parallel fashion. Data communication in MapReduce uses Key/Value pairs, where the key and value can be structured into different formats. The run-time system takes care of the low-level details, such as partitioning the input data, scheduling the tasks execution across a set of machines, handling machine failures, and managing inter-node communications. The number of Map and Reduce tasks can be specified by developers; typically, a larger number of tasks will result in better processing granularity but higher overhead. Figure 1 illustrates the workflow of the MapReduce framework.

MapReduce jobs are guaranteed to success, thanks to two features, namely, fault tolerance and speculative execution. In a large cluster, some nodes might fail while processing a job. MapReduce can reschedule tasks from a failed node to other nodes in order to avoid a crash of the entire job. When processing a task on a cluster of machines with heterogeneous performance, MapReduce can also balance the workload among different nodes to keep the overall speed optimized.

Although the original MapReduce software from Google is proprietary, several open-source alternatives are available. One of them, called Hadoop,[9] is broadly used for large-scale data processing, financial simulations, and bioinformatics calculations.[11–13] The Hadoop project includes the Hadoop distributed file system (HDFS) and Hadoop MapReduce. Hadoop is written in Java, but applications written in other programming languages can be implemented through a utility called Hadoop streaming. Hadoop streaming allows users to create and run MapReduce jobs based on any executable or script (Fig. 2). In this scheme, intermediate Key/Value pairs are transferred using standard UNIX streams, and formatted as strings or more efficient Typedbytes, which are a sequence of coded bytes in which the first byte is a type code.

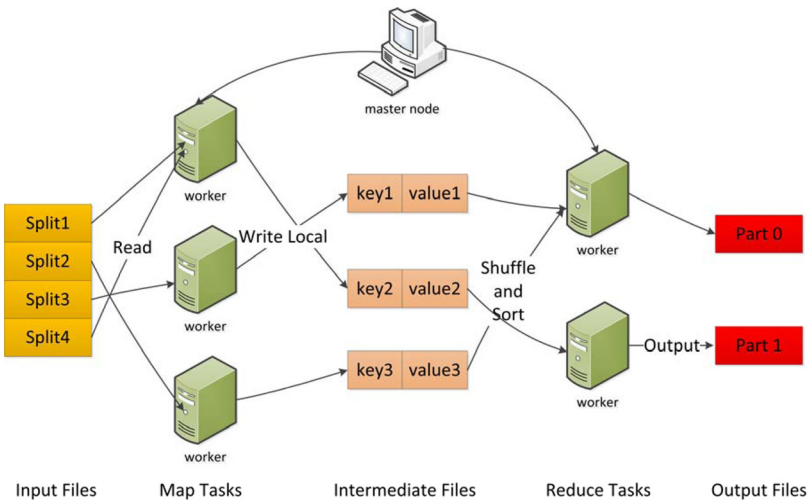


FIG. 1. An overview of the MapReduce framework. A user program specifies the Map and Reduce functions. A master node assigns and monitors the Map and Reduce task workers. The input files are divided into splits, which are then sent to the Map worker. Intermediate records, comprised of Key/Value pairs, are emitted from the Map worker and then fed to the Reduce workers, which combine data with same key and produces the final output.

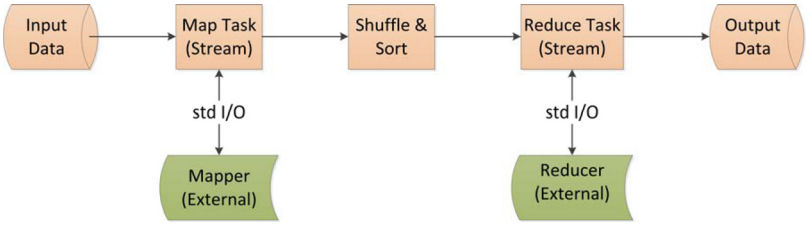


FIG. 2. A depiction of Hadoop streaming. User-defined external applications take place of the Map and Reduce functions.

III. METHOD

III.A. CBCT reconstruction

We assume that the source position, rotation center (also the reconstructed volume center), and projection center are along the same line. The source-to-detector distance is denoted as *sdd*, the source-to-rotation center distance as *srd* and the detector size is U by V. (All the notations for variables are directly used hereafter.) Given a series of 2D projections $P_1, P_2, ..., P_M$ collected with one rotation scan at angles $\theta_1, \theta_2, ...\theta_M$, the FDK algorithm first applies the Shepp–Logan filter to each projection, which helps to reduce the noise in the final image. The detector pixel value $P_k(u,v)$ at point (u,v) in the projection space is converted to value $Q_k(u,v)$ in the following form:

$$Q_k(u,v) = f_s(u) \otimes (W_1(u,v)P_k(u,v)), \tag{2}$$

$$W_1(u,v) = \frac{sdd}{\sqrt{(sdd^2 + u^2 + v^2)}}, \tag{3}$$

where $f_s(u)$ is 1D the Shepp–Logan filter and the convolution operation is 1D. $W_1(u,v)$ denotes the cosine weighting factor. The filtered data $Q_1, Q_2, ..., Q_M$ are then fed into the backprojection step to reconstruct the 3D volume $V(x,y,z)$

$$V(x,y,z) = \frac{2\pi}{M}\sum_{k=1}^{M} W_2(x,y,k)Q_k(u(x,y,z), v(x,y,z)), \tag{4}$$

$$W_2(x,y,k) = \left(\frac{srd}{srd - x\cos\theta_k - y\sin\theta_k}\right)^2, \tag{5}$$

$$u(x,y,z) = \frac{sdd \times (-x\sin\theta_k + y\cos\theta_k)}{(srd - x\sin\theta_k - y\cos\theta_k)}, \tag{6}$$

$$v(x,y,z) = \frac{sdd \times z}{(srd - x\sin\theta_k - y\cos\theta_k)}, \tag{7}$$

where $W_2(x,y,k)$ is the scaling factor. Since $u(x,y,z)$ and $v(x,y,z)$ often are not integers, linear interpolation is used to obtain the associated voxel value.

III.B. MapReduce implementation

The FDK algorithm processes each projection independently and thus can be parallelized. We propose two novel implementations of the FDK algorithm using MapReduce. In both implementations, the Map tasks load an equal number of projections from the HDFS. The only difference between these two implementations is the types of Key/Value pair used to transfer data between Map and Reduce tasks. The first one, namely voxel-based FDK (FDK-VB), uses Key/Value pairs to transfer individual voxels. In this scheme, the Key encodes the voxel position (x,y,z) using an index calculated as $(z \times N^2 + y \times N + x)$, and the Value is the backprojected image intensity $V(x,y,z)$. Each Map task emits Key/Value pairs voxel by voxel. The Reduce function accumulates all the intermediate values that share a common same key, producing the final reconstructed volume. Figure 3 illustrates the process of the FDK-VB algorithm. The other method, termed slice-based FDK (FDK-SB), transfers an entire volume slice as one Key/Value pair in the Map function, wherein the key is the slice index, and the value is an array of voxel intensities. The Reduce function accordingly combines those partial data into one final output volume.

In both implementations, the Map and Reduce function were written in the C language and executed using Hadoop streaming.[9] A text file containing paths of projection files in HDFS was used as the input to MapReduce. The Map function downloads several projections from distributed storage to its local hard-drive. After loading these projections into memory, it filters and backprojects them into an image buffer, which is transmitted back to Hadoop. The Reduce function collects partial images from Hadoop and writes the final image back to the distributed storage (Fig. 3). All the communications through UNIX streams use the Typedbytes format for higher efficiency.

III.C. Computing environment

Two different Hadoop clusters were used. One cluster was set-up in a pseudo-distributed fashion, by installing Hadoop 0.21.0 on a 2.4 GHz Intel Core 2 Duo computer. It was used primarily for developing and debugging MapReduce jobs. All the data were stored in a local HDFS.

Amazon's elastic compute cloud (EC2) and the elastic MapReduce (EMR) service (an implementation of Hadoop 0.20) were employed to perform experimental characterization of the implementation on large-scale clusters. For fair comparison, all algorithms were run on Amazon EC2 high-memory extra large nodes (codenamed as m2.xlarge). These nodes are equipped with 17.1 GB of memory, and 6.5 EC2 compute units (one EC2 compute unit provides the equivalent CPU capacity of a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor).

The Map and Reduce applications were compiled remotely on the cloud using GCC version 4.3.2, and uploaded to Amazon's simple storage service (S3) together with projection files and a file list. EMR jobs were submitted from the local computer through a Ruby-based command-line interface.

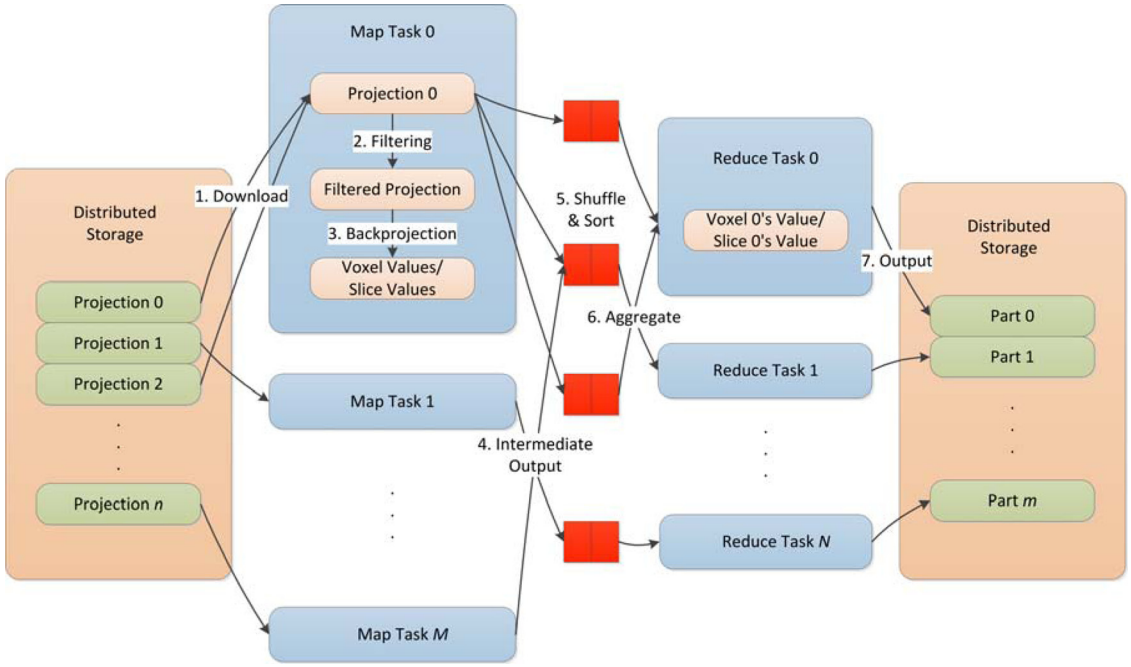


FIG. 3. An overview of the proposed FDK-VB algorithm. Projections are first downloaded from the distributed storage to local storage. Voxel-based Key/Value pairs are transferred between the Map and Reduce functions. The FDK-SB algorithm differs from FDK-VB only in the slice-based Key/Value pairs.

III.D. Evaluation

Evaluation of the new method was performed for a simulated digital phantom and for a physical anthropomorphic phantom. The digital phantom was the standard 3D Shepp–Logan phantom. The coordinates of the CBCT system were set as sdd = 1500 mm and srd = 1000 mm. Four hundred projections were collected in one rotation, each projection being 900×0.388 mm and 400×0.388 mm in size. The reconstructed image was $512 \times 512 \times 200$ voxels, with the voxel size being 0.388^3 mm^3.

A commercial calibration phantom CatPhan 600 (The Phantom Laboratory, Inc., Salem, NY) was used to evaluate the performance of the proposed method. CT projection data were acquired on a Varian TrueBeam STX (Varian Medical Systems, Palo Alto, CA) radiation treatment system using the on-board CBCT imaging system. The tube voltage and current were set to 100 kV and 138 mA, respectively, and the duration of the x-ray pulse at each projection view was set to 11 ms. In the TrueBeam system, the source-to-axis distance is 1000 mm and the source-to-detector distance is 1500 mm.

As a proof of concept, the phantom moved only in the longitudinal direction. The gantry rotation was purposely slowed down and synchronized to the phantom motion, such that there were six phases with each containing 328 projections. The dimension of each acquired projection image was 397 mm $\times$ 298 mm, containing 1024×768 pixels. The size of reconstructed image was $512 \times 512 \times 200$ and each voxel size was 0.388^3 mm^3.

In a first experiment, we reconstructed the digital and the CatPhan phantoms using three different algorithms, namely, single-threaded FDK (FDK-ST) performed on one node as a benchmark, and FDK-VB and FDK-SB algorithms executed on 200 nodes for parallelization. The FDK-ST implementation was adapted from plastimatch, a publicly-available library of tools for tomographic imaging.[14] All the output datasets were downloaded to a local computer and analyzed with MATLAB (MathWorks, Inc., Natick, MA).

In a second experiment, we tested the scalability of the FDK-SB algorithm by reconstructing the digital phantom on a variable number of m2.xlarge nodes, ranging from 20 to 200. For fine task granularity, the number of Map tasks was set to the number of projections and the number of Reduce task was set to the number of slices. This scheme was shown to optimally balance the MapReduce workload.[15] The total runtimes were recorded and the results were compared.

In a last experiment, we simulated the failure of some cluster nodes during the FDK-SB algorithm execution by manually terminating those nodes through the EC2 control panel. Half of the 100 nodes (m2.xlarge) were shut down 5 min after the start of the digital phantom reconstruction.

IV. RESULT

The reconstruction times are collected in Table I for the various phantoms and algorithms. The proposed FDK-SB method is more than ten times faster than the reference FDK-ST implementation. Furthermore, it also outperforms the FDK-VB method by about a factor of two. For algorithms based on MapReduce, the reconstruction time excludes the cluster initialization time.

Figure 4 shows the reconstructed images for different FDK implementations. For both cases, the deviations between the image obtained with the proposed methods and the FDK-ST implementation (ground truth) was on the order of 10^{-7} (root mean square error), with the small discrepancy due to quantization errors and data format conversion.

TABLE I. Simulation time comparison.

		Projection size	Volume size	# Node	# Map	# Reduce	Time (min)
Shepp–Logan phantom	FDK-ST	$900 \times 400 \times 400$	$512 \times 512 \times 200$	N/A	N/A	N/A	54.7
	FDK-VB	$900 \times 400 \times 400$	$512 \times 512 \times 200$	200	400	200	10.5
	FDK-SB	$900 \times 400 \times 400$	$512 \times 512 \times 200$	200	400	200	5.4
CatPhan phantom	FDK-ST	$1024 \times 678 \times 328$	$512 \times 512 \times 200$	N/A	N/A	N/A	65.3
	FDK-VB	$1024 \times 678 \times 328$	$512 \times 512 \times 200$	200	400	200	10.9
	FDK-SB	$1024 \times 678 \times 328$	$512 \times 512 \times 200$	200	400	200	5.7

The simulation time using the FDK-SB algorithm for different numbers of nodes is shown on Fig. 5. The numbers of nodes are set to be 10, 20, 50, 100, and 200, respectively. The fitting curve is calculated as $time = 6.4 + 202.4/N$, where the first term is a constant term, representing the total overhead and communication delay of the distributed system and the second term is an inverse term of the number of nodes, interpreted as the efficiency brought by more computing nodes.

In the last experiment, the FDK-SB algorithm successfully completed with identical results even though half of the worker nodes were manually terminated during the reconstruction. The overall run time increased by about 10 min due to reallocating tasks from terminated nodes to other nodes.

V. DISCUSSION

Cloud computing, multicore PC and GPU are types of hardware for parallel computing, while MapReduce and MPI are software frameworks running on those hardware. It should be noted that MapReduce can run on a wide range of architectures, including a cloud computing environment, a mutlicore workstation, a computer cluster or a GPU. Multicore PC have improved performance and better energy efficiency compared with single-core PC, however, their computing resources are limited (current multicore PC can have tens of cores maximally) and overall performance is contingent on the execution of multiple threads within applications. GPUs, originally designed for accelerating computer graphics, are increasingly used as massively-parallel coprocessors for scientific computation.[17] Compared with multicore CPUs, GPUs can run hundreds of threads with no context switching overhead, which makes it much more efficient than multicore CPUs. However, GPUs suffer from a few drawbacks, including complicated memory programming, slow random memory access, hardware constraints, and constraining single-program multiple data programming model.[17] Cloud computing connects shared computing resources, software, and information to computers and other devices by means of high-speed network links. Data exchange in cloud computing is generally slower than GPUs, because the latter is equipped with shared on-chip memory. However, MapReduce can avoid data access hazards which are problematic in GPUs by allocating its own data element locally. Additionally, MapReduce running in a cloud computing environment provides better scalability than GPUs. With multiple GPUs employed into a cluster, the software must be redesigned to provide internode and intercomputer communications. In contrast, the number of nodes in MapReduce can be modified by simply changing one parameter.

The results demonstrated the efficiency of the FDK-VB and FDK-SB algorithms. The FDK-SB algorithm achieved over tenfold speedup compared with the FDK-ST method. It is also observed that the FDK-SB algorithm outperforms the FDK-VB method by a factor of two, because it not only combines the data locally to lessen the Reduce function's effort to sort the data, but it also utilizes the I/O more

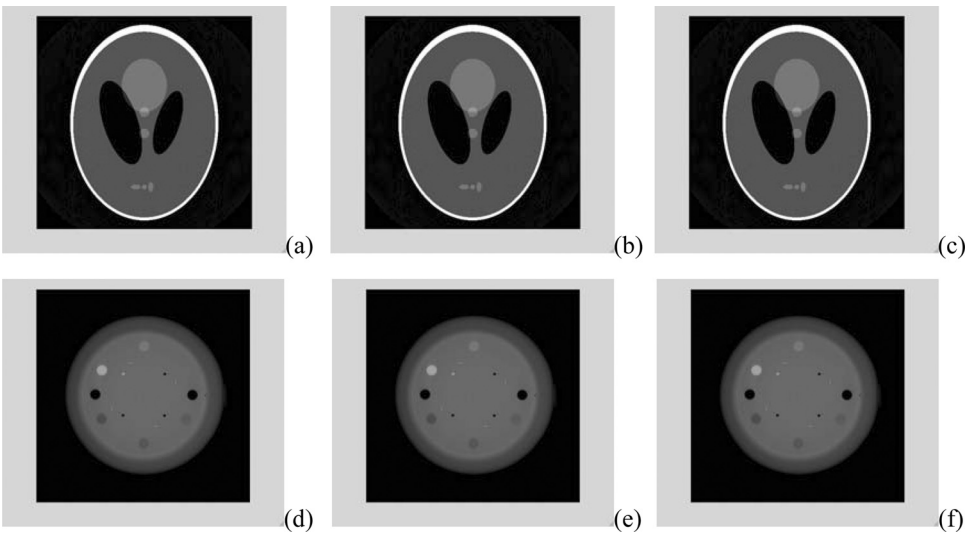


FIG. 4. Digital Shepp–Logan phantom (first row) and 4D CatPhan phantom (second row) reconstructed with FDK-ST (left column), FDK-VB algorithm (middle column) and FDK-SB algorithm (right column). Normalized image window: [0 1].

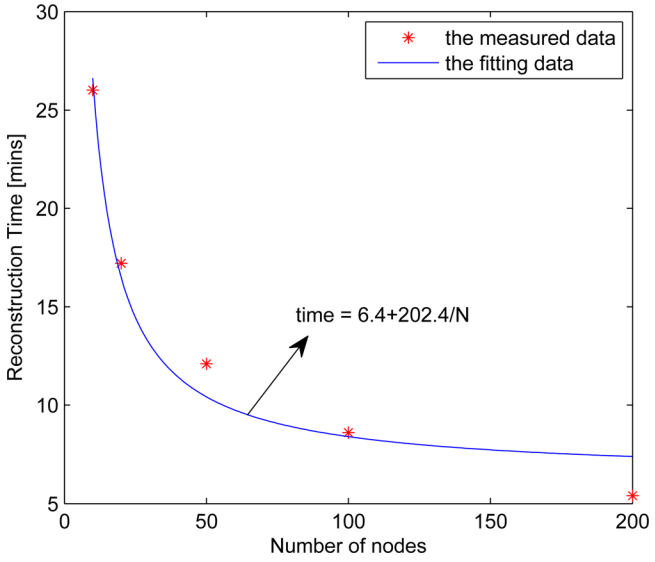


FIG. 5. The reconstruction times using FDK-SB method with different numbers of nodes for the $512 \times 512 \times 200$ Shepp–Logan phantom were collected. Linear regression shows that the computation time can be approximated by an affine function of the inverse of number nodes.

efficiently by transferring large chunks of data for each record. For 4DCT or other iterative reconstruction methods,[16] the advantage of using MapReduce in a cloud environment would be more remarkable. For example, iterative reconstruction can be achieved by executing a sequence of Map and Reduce tasks. MapReduce allows the output of a Reduce task to be fed in as the input of a new set of Map tasks. This is the basic principle that can be used for iterative reconstruction. For instance, iterative weighted least-square reconstruction would first require a Map/Reduce step to compute the forward projection along integration lines. These projections would be compared to the measurements and the error would be back-projected using another Map/Reduce step. This process can be repeated for a certain number of iterations and the overall reconstruction time will approach only the serial port time.

MapReduce is designed in a highly decentralized manner to optimize internode data transfer for massive problems. Both data input and output are stored to HDFS, a distributed file system, and both Map and Reduce tasks are performed on various nodes throughout the cluster. Hence, every stage of the computation workflow is scalable and no stage constitutes a bottleneck. However, in our experiments, the speedup achieved by the distributed MapReduce implementation was lower than the number of worker nodes (Fig. 5), which suggests that the overall performance is limited by communication between nodes and overhead. In this work, we model the reconstruction time as a function of a constant term, accounting for the overhead and communication delay, plus an inverse of number of nodes term, representing the benefit of distributed computation. This finding highlight the fact that while Hadoop is a powerful tool for processing terabytes of text, it is less adapted to processing smaller binary data such as CT scans. With further optimization of the Hadoop platform for numerical computations, more acceleration is expected. Moreover, the simplicity, scalability, and reliability of MapReduce remain very attractive for the developer. Further improvements in the technology could easily address its current shortcomings, in particular with respect to data sorting and caching.

Fault resilience is a standard feature of Hadoop and other MapReduce implementations. In EMR, new nodes are automatically provisioned to replace failed nodes. Failed Map and Reduce task are placed back in the queue and eventually redistributed to these newly allocated nodes. The guaranteed reliability of the compute framework is crucial for applications in medical physics.

Parallel computing is commonly performed on clusters of commodity computers. Though such systems have a few advantages such as better control over the hardware, the maintenance, and energy costs can be much higher compared to a cloud computing environment. For instance, the current price for an m2.xlarge node is 0.57$ per hour for N. California users on the Amazon EC2. However, if used intensively, a small cluster of GPUs can be more economical over time than purchasing on-demand clusters.

For best performance, the system parameters of MapReduce need to be finely tuned. In order to execute each task most efficiently, developers need to adjust parameters such as the heap size for child Java Virtual Machines of Map and Reduce functions and the memory limits for transferring data. This requires an in-depth understanding of the computing problem. However, reasonable performance can usually be achieved using the default settings.

VI. CONCLUSION

In this work, a fast, scalable and reliable 4D CBCT/CT reconstruction technique was developed and evaluated using the MapReduce framework. The FDK-SB algorithm on the cloud showed more than ten times speedup compared with the single-threaded implementation. Furthermore, the simplicity, reliability and scalability of the implementation enabled by MapReduce are attractive features not only for CBCT reconstruction, but also for many other projects in medical physics.

ACKNOWLEDGMENT

This project was supported in part by Grant Nos. from NCI (1R01 CA133474) and NSF (0854492).

[a]Author to whom correspondence should be addressed. Electronic mail: lei@stanford.edu. Telephone: (650) 498-7896.

[1]D. Paquin, D. Levy, and L. Xing, "Multiscale registration of planning CT and daily cone beam CT images for adaptive radiation therapy," Med. Phys. **36**, 4–11 (2009).
[2]Y. Xie, M. Chao, P. Lee, and L. Xing, "Feature-based rectal contour propagation from planning CT to cone beam CT," Med. Phys. **35**, 4450 (2008).
[3]Y. Okitsu, F. Ino, and K. Hagihara, "Accelerating cone beam reconstruction using the CUDA-enabled GPU," *High Performance Computing HiPC*, 15th Annual IEEE International Conference on High Performance Computing (2008), pp. 108–119.
[4]B. Hayes, "Cloud computing," Commun. ACM **51**, 9 (2008).
[5]P. Mika and G. Tummarello, "Web semantics in the clouds," IEEE Intell. Syst. **23**, 82–87 (2008).
[6]E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan, "Computational solutions to large-scale data management and analysis," Nat. Rev. Genet. **11**, 647–657 (2010).
[7]J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Commun. ACM **51**(1), 107–113 (2008).

[8]B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: A MapReduce framework on graphics processors," Intelligent Systems **23**(5), 82–87 (2008).
[9]T. White, *Hadoop: The Definitive Guide* (O'Reilly Media, 2009).
[10]L. A. Feldkamp, L. C. Davis, and J. W. Kress, "Practical cone-beam algorithm," J. Opt. Soc. Am. A **1**, 612–619 (1984).
[11]S. MC, "CloudBurst: Highly sensitive read mapping with MapReduce," Bioinformatics **25**, 1363–1369 (2009).
[12]C. Moretti, K. Steinhaeuser, D. Thain, and N. V. Chawla, "Scaling up classifiers to cloud computers," *ICDM* (2008).
[13]F. Wang, V. Ercegovac, T. Syeda-Mahmood, A. Holder, E. Shekita, D. Beymer, and L. H. Xu, "Large-scale multimodal mining for healthcare with mapreduce," *Proceedings of the 1st ACM International Health Informatics Symposium* (ACM New York, 2010).
[14]G. Sharp, N. Kandasamy, H. Singh, and M. Folkert, "GPU-based streaming architectures for fast cone-beam CT image reconstruction and demons deformable registration," **52**, 5771–5783 (2007).
[15]G. Pratx and L. Xing, "Monte-Carlo simulation in a cloud computing environment with MapReduce," J. Biomed. Opt. (in press).
[16]B. Meng, J. Wang, and L. Xing, "Sinogram preprocessing and binary reconstruction for determination of the shape and location of metal objects in computed tomography (CT)," Med. Phys. 5867 (2010).
[17]G. Pratx and L. Xing, "GPU computing in medical physics: A review," Med. Phys. 2685 (2011).