

# TECHNICAL REPORT

## Parallel Visualization on Large Clusters using MapReduce

*Huy T. Vo*<sup>\*</sup>, *Jonathan Bronson*<sup>\*</sup>, *Brian Summa*<sup>\*</sup>, *João L. D. Comba*<sup>†</sup>, *Juliana Freire*<sup>\*</sup>, *Bill Howe*<sup>‡</sup>, *Valerio Pascucci*<sup>\*</sup>, *Cláudio T. Silva*<sup>\*</sup>  
<sup>\*</sup>SCI Institute, University of Utah

<sup>†</sup>Instituto de Informática, Universidade Federal do Rio Grande do Sul, Brasil

<sup>‡</sup>eScience Institute, University of Washington, USA

UUSCI-2011-002

Scientific Computing and Imaging Institute  
University of Utah  
Salt Lake City, UT 84112 USA

March 2, 2011

### Abstract:

Large-scale visualization systems are typically designed to efficiently push datasets through the graphics hardware. However, exploratory visualization systems are increasingly expected to support scalable data manipulation, restructuring, and querying capabilities in addition to core visualization algorithms. We posit that new, emerging abstractions for parallel data processing, in particular computing clouds, can be leveraged to support large-scale data exploration through visualization. In this paper, we take a first step in evaluating the suitability of the MapReduce framework to implement large-scale visualization techniques. MapReduce is a lightweight, scalable, general-purpose parallel data processing framework increasingly popular in the context of cloud computing. Specifically, we implement and evaluate a representative suite of visualization tasks (isosurface extraction, mesh simplification, and polygon rasterization) as MapReduce programs, and report quantitative performance results applying these algorithms to realistic datasets. For example, we perform isosurface extraction of up to 16 isovalues for volumes composed of 8 billion voxels, simplification of meshes with 18GBs of data and subsequent rendering with image resolutions up to 800002 pixels. Our results indicate that the parallel scalability, ease of use, ease of access to computing resources, and fault-tolerance of MapReduce offer a promising foundation for a combined data manipulation and data visualization system deployed in a public cloud or a local commodity cluster

# Parallel Visualization on Large Clusters using MapReduce

Huy T. Vo<sup>1</sup>, Jonathan Bronson<sup>1</sup>, Brian Summa<sup>1</sup>, João L. D. Comba<sup>2</sup>, Juliana Freire<sup>1</sup>, Bill Howe<sup>3</sup>, Valerio Pascucci<sup>1</sup>, and Cláudio T. Silva<sup>1</sup>

<sup>1</sup>SCI Institute, University of Utah, USA

<sup>2</sup>Instituto de Informática, Universidade Federal do Rio Grande do Sul, Brasil

<sup>3</sup>eScience Institute, University of Washington, USA

---

## Abstract

*Large-scale visualization systems are typically designed to efficiently “push” datasets through the graphics hardware. However, exploratory visualization systems are increasingly expected to support scalable data manipulation, restructuring, and querying capabilities in addition to core visualization algorithms. We posit that new, emerging abstractions for parallel data processing, in particular computing clouds, can be leveraged to support large-scale data exploration through visualization. In this paper, we take a first step in evaluating the suitability of the MapReduce framework to implement large-scale visualization techniques. MapReduce is a lightweight, scalable, general-purpose parallel data processing framework increasingly popular in the context of cloud computing. Specifically, we implement and evaluate a representative suite of visualization tasks (isosurface extraction, mesh simplification, and polygon rasterization) as MapReduce programs, and report quantitative performance results applying these algorithms to realistic datasets. For example, we perform isosurface extraction of up to 16 isovalues for volumes composed of 8 billion voxels, simplification of meshes with 18GBs of data and subsequent rendering with image resolutions up to 80000<sup>2</sup> pixels. Our results indicate that the parallel scalability, ease of use, ease of access to computing resources, and fault-tolerance of MapReduce offer a promising foundation for a combined data manipulation and data visualization system deployed in a public cloud or a local commodity cluster.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

---

## 1. Introduction

Cloud computing has emerged as a viable, low-cost alternative for large-scale computing and has recently motivated both industry and academia to design new general-purpose parallel programming frameworks that work well with this new paradigm [DG04, ORS\*08, YIF\*08, CJL\*08]. In contrast, large-scale visualization has traditionally benefited from specialized couplings between hardware and algorithms, suggesting that migration to a general-purpose cloud platform might incur in development costs or scalability<sup>†</sup>.

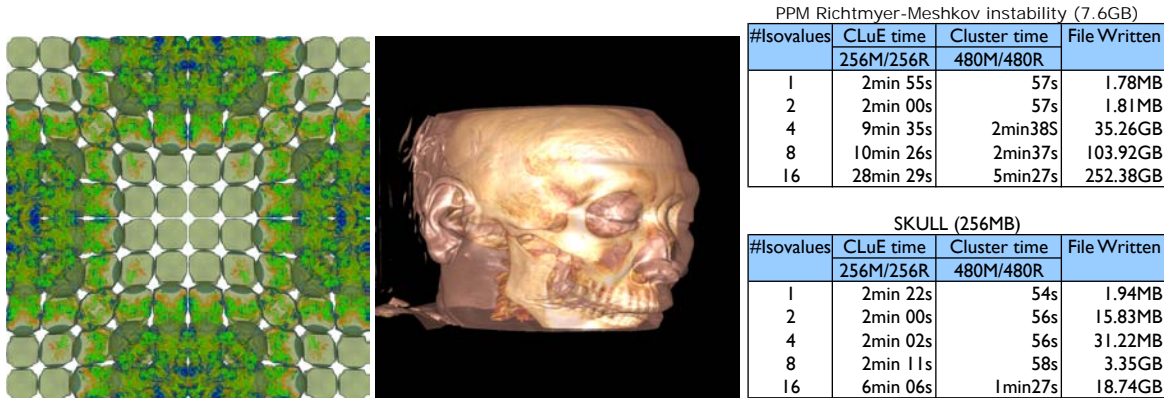
The MapReduce framework [DG04, DG08] provides a simple programming model for expressing loosely-coupled parallel programs using two serial functions, *Map* and *Reduce*. The *Map* function processes a block of input produc-

ing a sequence of (*key*, *value*) pairs, while the *Reduce* function processes a set of values associated with a single key. The framework itself is responsible for “shuffling” the output of the *Map* tasks to the appropriate *Reduce* task using a distributed sort. The model is sufficiently expressive to capture a variety of algorithms and high-level programming models, while allowing programmers to largely ignore the challenges of distributed computing and focus instead on the semantics of their task. Additionally, as implemented in the open-source platform Hadoop [had], the MapReduce model has been shown to scale to hundreds or thousands of nodes [DG04, PPR\*09]. MapReduce clusters can be constructed inexpensively from commodity computers connected in a shared-nothing configuration (*i.e.*, neither memory nor storage are shared across nodes). Such advantages have motivated cloud providers to host Hadoop and similar frameworks for processing data at scale [clu, yabb, aws].

These platforms have been largely unexplored by the visualization community, even though these trends make it ap-

---

<sup>†</sup> Scalability refers to the relative performance increase by allocating additional resources.



**Figure 1:** Isosurface results using the MapReduce framework for the PPM Richtmyer-Meshkov instability and Skull datasets. Performance results illustrate computation for varying isovalues using a cloud environment (CLuE) and a local cluster. Number of maps (M) and reduces (R) informed for each configuration, as well as the size of the output file written.

parent that our community must inquire into their viability for use in large-scale visualization tasks. The conventional modus operandi of “throwing datasets” through a (parallel) graphics pipeline relegates data manipulation, conditioning, and restructuring tasks to an offline system and ignores their cost. As data volumes grow, these costs — especially the cost of transferring data between a storage cluster and a visualization cluster — begin to dominate. Cloud computing platforms thus open new opportunities in that they afford both general-purpose data processing as well as large-scale visualization.

In this paper, we take a first step at investigating the suitability of cloud-based infrastructure for large-scale visualization. We have designed a set of MapReduce-based algorithms for memory-intensive visualization techniques, and performed an extensive experimental evaluation. Our results indicate that MapReduce offers a potential foundation for a combined storage, processing, analysis, and visualization system that is capable of keeping pace with growth in data volume (attributable to scalability and fault-tolerance) as well as growth in application diversity (attributable to extensibility and ease of use). We also found that common visualization algorithms can be naturally expressed using the MapReduce abstraction. Even simple implementations of these algorithms are highly scalable. For example, Figure 1 shows the results for isosurface extraction run on a shared cloud environment and a local cluster running Hadoop, *i.e.*, a private cloud environment. For our largest dataset, comprising a volume of 7.6GB, we extracted 16 isosurfaces in 5 mins and 27s.

In summary, the main contributions of the paper are:

- We have designed scalable MapReduce-based algorithms for three core, memory-intensive visualization techniques: isosurface extraction, mesh simplification, and polygon rasterization;
- We have performed an experimental evaluation of these

algorithms using both a multi-tenant cloud environment and a local cluster;

- We discuss the benefits and challenges of developing visualization algorithms for the MapReduce model, as well as the lessons we learned in this process.

## 2. Related Work

Recently, a new generation of systems have been introduced for data management in cloud computing environments. Examples include file systems [Bor07, Kos07], storage systems [amaa, CDG\*06, DHJ\*07], and hosted DBMSs [amab, goo, mic, Yaha]. MapReduce [DG04, YDHP07] and similar massively parallel data processing systems (*e.g.*, Clustera [DPR\*08], Dryad [IBY\*07], and Hadoop [had]) along with their specialized languages [CJL\*08, ORS\*08, PDGQ05, YIF\*08]) are having a great impact on data processing in the cloud. Despite the benefits these systems have given to other fields, they have not yet been evaluated in the context of visualization.

One of the first remote visualization applications is described in [SME02]. In this system, the X Window System’s transport mechanism is used in combination with Virtual Network Computing (VNC) [RSFWH98] to allow remote visualization across different platforms. IBM’s Deep Computing Visualization (DCV) system [IBM05], SGI’s OpenGL Vizserver [Sil] and the Chromium Renderserver (CRRS) [PAB\*08] perform hardware accelerated rendering for OpenGL applications. A data management and visualization system for managing finite element simulations in materials science, which uses Microsoft’s SQL Server database product coupled to IBM’s OpenDX visualization platform is described in [HG05]. Indexes provide efficient access to data subsets, and OpenDX renders the results into a manipulable scene allowing inspection of non-trivial simulation features such as crack propagation. However, this architecture is un-

likely to scale beyond a few nodes due to its dependency on a conventional database system.

A different approach to distributed visualization is to provide access to the virtual desktop on a remote computing system [Law, IBM05, Sil, PAB\*08], such that the data remains on the server and only images or graphics primitives are transmitted to the client. Other applications, such as VisIt [Law] and ParaView [par], provide a scalable visualization and rendering back-end that sends images to a remote client. Many scientific communities are creating shared repositories with increasingly large, curated datasets [iri, lss, slo]. To give an idea of the scale considered by these projects, the LSST [lss] is predicted to generate thirty terabytes of raw data per night for a total of six petabytes per year. Existing systems associated with these repositories support only simple retrieval queries, leaving the user to perform analysis and visualization independently.

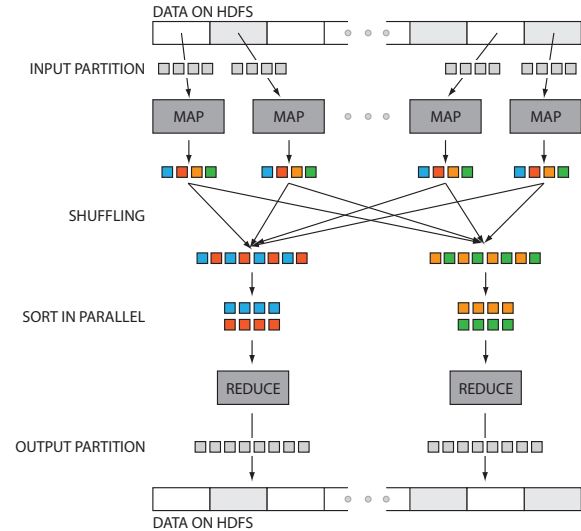
### 3. MapReduce Overview

MapReduce is a framework to process massive data on large distributed systems. It provides an abstraction inspired by functional programming languages such as Lisp, relying on two basic operations:

- Map: Given input, emit one or more (*key, value*) pairs.
- Reduce: Process all values of a given key and emit one or more (*key, value*) pairs.

A MapReduce job is comprised of three phases: map, shuffle and reduce. Each dataset to be processed is partitioned into fixed-size blocks. In the map phase, each task processes a single block and emits zero or more (*key, value*) pairs. In the shuffle phase, the system sorts the output of the map phase in parallel, grouping all values associated with a particular key. In Hadoop, the shuffle phase occurs as the data is processed by the mapper (*i.e.*, the two phases overlap). During execution, each mapper hashes the key of each key/value pair into bins, where each bin is associated with a reducer task and each mapper writes its output to disk to ensure fault tolerance. In the reduce phase, each reducer processes all values associated with a given key and emits one or more new key/value pairs. Since Hadoop assumes that any mapper is equally likely to produce any key, each reducer may potentially receive data from any mapper. Figure 2 illustrates a typical MapReduce job.

MapReduce offers an abstraction that allows developers to ignore the complications of distributed programming — data partitioning and distribution, load balancing, fault-recovery and interprocess communication. Hadoop is primarily run on a distributed file system, and the Hadoop File System (HDFS) is the default choice for deployment. Hadoop has become a popular runtime environment for higher-level languages for expressing workflows, SQL queries, and more [ORS\*08, Hiv]. These systems are becoming viable options for general purpose large-scale data processing, and leveraging their computational power to new fields can be a very promising prospect. For example, MapReduce systems are



**Figure 2:** Data transfer and communication of a typical MapReduce job in Hadoop. Data blocks are assigned to several Maps, which emit key/value pairs that are shuffled and sorted in parallel. The reduce step emits one or more pairs.

well-suited for *in situ* visualization, which means that data visualization happens while the simulation is running, thus avoiding costly storage and post-processing computation. There are several issues in implementing *in situ* visualization systems as discussed in [Ma09]. We posit that the simplicity of the implementation, inherent fault-tolerance, and scalability of MapReduce systems make it a very appealing solution.

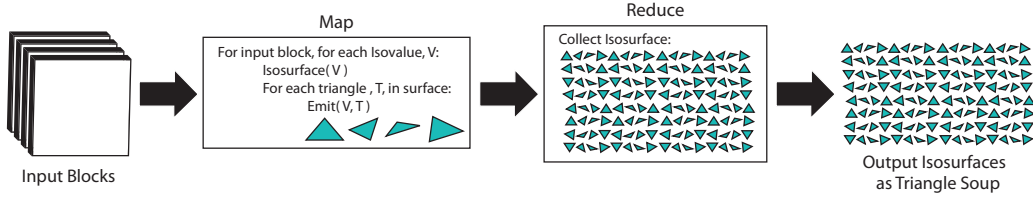
## 4. Visualization Algorithms using MapReduce

In this section we describe MapReduce algorithms for three widely-used and memory-intensive visualization techniques: isosurface extraction, rendering of large unstructured grids, and large model simplification.

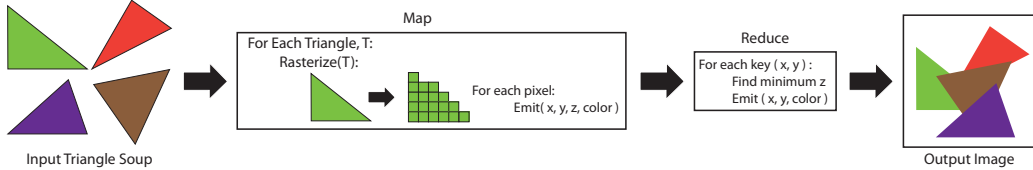
### 4.1. Isosurface extraction

Isosurfaces are instrumental in visual data exploration, allowing scientists to study function behavior in static or time-varying scenarios. Giving an input scalar volume, the core of extraction is the computation of the isosurface as a collection of simplicial primitives that can be rendered using common graphical methods. Our MapReduce-based algorithm for isosurface extraction is based on the Marching Cubes algorithm [LC87], which is the de-facto standard for isosurface extraction due to its efficiency and robustness.

As Figure 3 illustrates, partitioning relies on the natural representation of a scalar volume as a collection of 2D slices. The Hadoop distributed file system uses this strategy to partition data into blocks for each mapper, but imposes some constraints. First, each partition must contain complete



**Figure 3:** MapReduce isosurface extraction. The map phase generates an isovalue as key and triangle data as value. The reduce phase simply outputs the isosurface as a triangle soup to file.



**Figure 4:** MapReduce rasterization. The map phase rasterize each triangle and emits the pixel coordinates as value, and its color and depth as value. The reducer emits the smallest depth value for each location.

slices. Second, it allows the overlap by one slice in only one direction to account for triangles spanning across partitions. Although it may result in duplication of input data, there is no duplication of output triangles since this overlap only occurs in one dimension. In practice, the duplication of input data is small and has no significant effect on the performance of the system. Each mapper computes the triangles of several isosurfaces using the Marching Cubes algorithm as implemented in the Contour Library [Cam99] and emits a (key,value) pair for each isovalue. The key is the isovalue and the value is the triangle data for the each cube in binary format. The reducer receives the data sorted and binned by isovalue, thus, the reduce stage only needs to act as a pass-through, writing the isosurface as a triangle soup to file.

#### 4.2. Rendering

Out-of-core methods have been developed to render datasets that are too large to fit in memory. These methods are based in a streaming paradigm [FS01], and for this purpose the rasterization technique is preferred due to its robustness, high parallelism and graphics hardware implementation. We have designed a MapReduce algorithm for a rasterization renderer for massive triangular and tetrahedral meshes. The algorithm exploits the inherent properties of the Hadoop framework and allows the rasterization of meshes several gigabytes in size and images with billions of pixels.

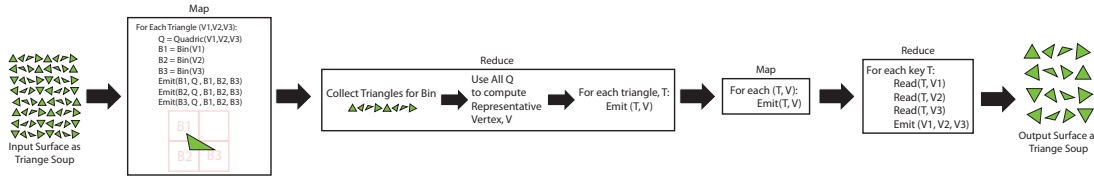
Hadoop partitions the input “triangle soup” among mappers with the constraint that each partition must be a multiple of 36 bytes, thus avoiding the block boundary splitting a triangle. For each triangle, the mapper computes its projection onto the image plane and its corresponding pixels. For each pixel, the mapper outputs a (key,value) pair, with the key being the pixel location in the image plane  $(x, y)$ , and the value being the depth and color of the pixel. The MapReduce framework sorts pixels into the proper order (row-major) to construct the final image. Pixel colors emitted by the map-

per that share the same image plane location are grouped by this sorting. The reducer emits the smallest depth value for each pixel location, therefore accomplishing the z-buffering algorithm automatically. In Figure 4, we give an overview of this algorithm. Mappers and reducers are viewed as geometry and multi-fragment shaders respectively in two distinct phases. Note that this parallels a graphics hardware pipeline and can be similarly extended to handle more advanced visualizations by custom “geometry shaders” and “fragment shaders.” For example, in a volume renderer, each reducer just needs to sort its fragments and composite them instead of a selection based on depth.

#### 4.3. Mesh simplification

Despite advances in out-of-core methods for rendering structured or unstructured meshes, it may still not be feasible to use the full resolution mesh. Several mesh simplification techniques have been proposed [GH97, YSZ04, SZL92]. Memory usage is a key aspect of this problem, since techniques often require storage proportional to the size of the input or output mesh. An alternative is given by the OoCSx (improved Out-of-Core Simplification) algorithm [LS01], which decouples this relationship and allows the simplification of meshes of arbitrary sizes. This is accomplished by superimposing a regular grid over the underlying mesh with associations between grid cells and vertices: every grid cell that contains a vertex of the input mesh must also contain a vertex on the output mesh, and every cell must have only one representative vertex. The problem is broken into finding all triangles that span three unique cells, and then finding an optimum representative vertex for each cell. Only a linear pass through the triangle mesh to hash each vertex is needed to find its representative bin before the output of all triangle indices.

The linear pass of OoCSx is not suitable for a parallel implementation. However, if we do not use the minimal error criteria for optimal representative vertices, we are able



**Figure 5: MapReduce mesh simplification.** The first map phase generates the bin coordinate of a vertex as key and a quadric error measure along the three triangle indices as value. The first reduce emits the representative vertex for each triangle. A second map and reduce is applied since multiple current reduce phases are not currently supported.

to implement the algorithm using only the Map phase of Hadoop. This not only avoids the expensive sorting and merging phases of MapReduce but also allows the simplification performance to scale directly with the number of mappers at our disposal. We use two MapReduce jobs to implement the full algorithm since it requires two sorting phases (see Figure 5). The first Map phase bins each vertex into a regular grid to ensure that all triangles contributing vertices to a particular bin arrive on the same node in the Reduce phase. It also computes the quadric measure vector associated with the contributing triangle. For each triangle, three (key, value) pairs are emitted, one for each vertex. The key is the bin coordinate that contains the vertex, and the value is a concatenation of the quadric measure vector with the three indices of the triangle.

The first Reduce phase receives the same (key, value) pair from the Map phase, but sorted and grouped by key. It reads each unique key (bin), and uses the quadric measures of all triangles falling into that bin to compute the representative vertex. If the indices of all vertices of a triangle contributing to a representative vertex are unique, the Reduce phase emits the indexed triangle as key, and the current grid cell and vertex. Thus, across all reducers, there will be exactly three (key,value) pairs with the same key (triangle), each storing a different representative vertex and corresponding bin as its value. Since multiple Reduce phases are currently not supported, we use a second MapReduce job to complete the dereference. This second Map phase merely reads and emits the data output from the first Reduce job. Keyed on triangle index, the second Reduce phase receives the exact three bin-vertex pairs, and emit as final output the simplified mesh.

## 5. Experimental Analysis

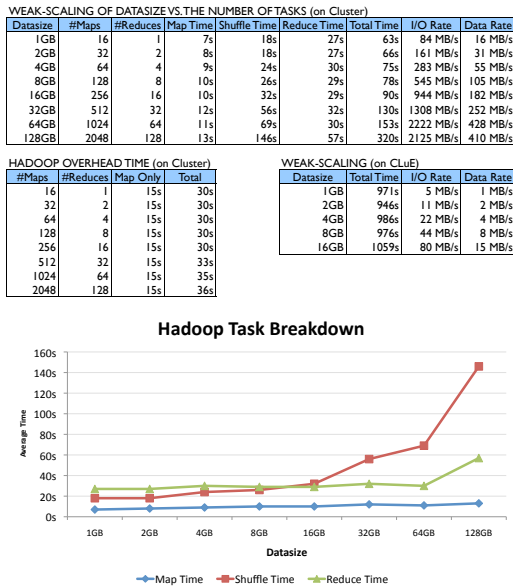
We have performed an in-depth analysis of the algorithms presented in the previous section. We designed our experiments to evaluate, for each algorithm, its ability to scale up and down, as well as the overhead introduced by Hadoop. The first series of tests shows the cost of data transfer through a MapReduce job without any computation, followed by a detailed evaluation of each individual algorithm. By default, the number of mappers that Hadoop launches for a job is a multiple of the number of data blocks, taking care to not exceed the actual number of blocks on its HDFS (counting all replications). On the other hand, the number of

reduce tasks can be specified. To simplify comparison, in our tests we maximize the number of reducers to the system capacity while keeping its ratio to the number of mappers equal to 1. The number of mappers and reducers always equal in our experiments whenever the number of input data blocks permits.

Tests were performed on two Hadoop-based systems: a local cluster and the NSF CLuE cluster managed by IBM [clu]. The local cluster consists of 60 nodes, each with two quad-core Intel Xeon Nehalem 2.6GHz processors, 24GB of memory and a 320GB disk. The CLuE cluster consists of 410 nodes each with two single-core Intel Xeon 2.8GHz processors, 4GB of memory and a 400GB disk. While still a valuable resource for research, the CLuE hardware is outdated if compared to modern clusters, since it was originally built in 2004 with both low-speed processors and limited memory. Thus, we mostly utilize the performance numbers from the CLuE cluster as a way to validate and/or compare with our results on the local cluster. Since the CLuE cluster is a shared resource among multiple universities, there is currently no way to run experiments in isolation. We made sure to run all of our experiments at *dead* hours to minimize the interference from other jobs. HDFS files were stored in 64MB blocks with 3 replications.

### 5.1. MapReduce Baseline

To evaluate the cost incurred solely from streaming data through the system, several baseline tests were performed. For our scaling tests, we have evaluated our algorithms' performance only for weak-scaling (*i.e.*, scaling the number of processors with a fixed data size per processor). This was chosen over strong scaling (*i.e.*, scaling the number of processors with a fixed total data size) since the latter would require changing a data blocksize to adjust the number of mappers appropriately. The Hadoop/HDFS is known for degraded performance for data with too large or small block-sizes depending on job complexity [Tec], therefore strong scaling is currently not a good indicator of performance in Hadoop. The weak-scaling experiments vary data size against task capacity and proportionally change the number of mappers and reducers. An algorithm or system that has proper weak scaling should maintain a constant runtime. To avoid biasing results by our optimization schemes, we use the default MapReduce job, with a trivial record reader and



**Figure 6:** Hadoop baseline test evaluates data transfer costs. In the local cluster we achieve rates up to 428MB/s

writer. Data is stored in binary format and split into 64-bytes record, with 16 bytes reserved for the key. Map and reduce functions pass the input directly to the output, and are the simplest possible jobs such that the performance is disk I/O and network transfer bounded.

The top table in Figure 6 shows the average cost for map, shuffle and reduce tasks respectively in the local cluster. The last two columns depict the overall disk I/O throughput and data throughput for a particular job. I/O rates were computed by dividing the total number of disk reads and writes including temporary files over the total time, while data rates represent how much input data pass through the job in a second. For map tasks, Hadoop was able to keep the runtime constant, since input files are read in sequence on each node and directed to appropriate mappers. In the reducing step, even though the amount of data is the same as to the map phase and each node write data to its local disk, there is also a local external sorting that incurs in overhead. Nevertheless, both runtimes are still considerably constant, except for the jump from 64GB to 128GB. At this point, the number of reducers guarantees each node has to host at least two reduce tasks if distributed properly, therefore each disk now has double the I/O and seek operations. This can be seen in the disk I/O rates, where the throughput is optimal at 64 tasks on the local cluster with 60 disks and drops while maintaining a relatively high speed for the larger number of tasks.

The shuffle phase of Hadoop is where weak scaling is not linear. This accounts for the data transfer between the map and reduce phase of a job along with sorting. In Hadoop each mapper is likely to contribute a portion of data required by each reducer and therefore is not expected to scale well. The plot in Figure 6 illustrates the breakdown of the three phases.



WEAK SCALING (RESOLUTION)

Resolution	St. MATTHEW (13 GB)				ATLAS (18 GB)			
	#M/R	CLuE time	Cluster time	File Written	#M/R	CLuE time	Cluster time	File Written
1250x1250	256/256	1min 54s	46s	33MB	273/273	1min 55s	46s	41MB
2500x2500	256/256	1min 42s	46s	147MB	273/273	2min 11s	46s	104MB
5000x5000	256/256	1min 47s	46s	583MB	273/273	2min 12s	46s	412MB
10000x10000	256/256	1min 40s	46s	2.3GB	273/273	2min 12s	46s	1.6GB
20000x20000	256/256	2min 04s	46s	10.9GB	273/273	2min 27s	47s	5.5GB
40000x40000	256/256	3min 12s	1min08s	53.14GB	273/273	3min 55s	55s	37.8GB
80000x80000	256/256	9min 50s	2min55s	213GB	273/273	0min 30s	1min58s	151.8GB

WEAK SCALING (RESOLUTION AND REDUCE)

Resolution	St. MATTHEW (13 GB)				ATLAS (18 GB)			
	CLuE #R	256M time	Cluster #R	480M time	CLuE #R	256M time	Cluster #R	480M time
1250x1250	4	1min 13s	8	46s	4	1min 18s	8	46s
2500x2500	8	1min 18s	15	46s	8	1min 19s	15	45s
5000x5000	16	1min 18s	30	46s	16	1min 51s	30	46s
10000x10000	32	2min 04s	60	47s	32	1min 52s	60	47s
20000x20000	64	2min 04s	120	49s	64	2min 34s	120	46s
40000x40000	128	4min 45s	240	1min06s	128	5min 06s	240	55s
80000x80000	256	9min 50s	480	2min14s	256	10min 30s	480	1min41s

**Figure 7:** Rendering results for the St. Matthew (left) and Atlas (right). Performance tables show results for different resolutions. An 80000 x 80000 image of the Atlas (18GB) was generated in 1min and 58s in the local cluster.

The Hadoop Overhead Time table shows (only) the overhead of communication across the map and reduce phases. Note that each phase takes about 15 seconds to start on our local cluster. We also include weak-scaling results for the CLuE cluster for comparison. The I/O rates are considerably lower than the local performance, and, as expected, this is likely due to age of the system and the shared usage. From the weak scaling tests we conclude that the MapReduce model can be robust when the number of nodes scales with the data size. Little cost is incurred for using more input data, and the effective transfer rates scale proportionally to the input data size. However, in order to ensure fault tolerance, disk I/O is heavily involved and could bound the overall performance.

## 5.2. Isosurface Extraction


We tested the isosurface MapReduce algorithm on two datasets: a small skull volume (256MB) and a larger ppm Richtmyer-Meshkov instability simulation volume (7.6GB). Since our baseline testing has shown that the amount data produced can effect Hadoop’s performance, we performed tests that varied the number of isosurfaces generated in a single job, since this can have a drastic effect on the amount of data being produced. For few isosurfaces, we expect a small number of triangles to be produced. Conversely, for many isosurfaces, more data will be output to disk than was used for input. We keep the number of maps constant at 256, as this is the largest power-of-two we can use without pigeon holing more than 1 mapper to a node of the CLuE cluster.

We observed that in the extraction of dozens of isosurfaces (as part of parameter exploration) data output increases proportionally to runtime. Jobs are relatively fast for the standard case of fewer isosurfaces, since input and output data are partitioned evenly to mappers and reducers, thus the amount of disk I/O for input and output is relatively small (*e.g.*, approximately 32MB per Mapper). The runtime of this algorithm is mostly affected by the shuffling phase, where triangles emitted from mappers exceed the available buffer and are sorted out-of-core before being transferred to the reducers. The performance of this phase depends on the amount of temporary disk I/O used when the mapper runs out of in-core buffer space.

In the tables of Figure 1, the *File Written* column denotes the amount of temporary data produced (not the HDFS output). For the skull data set, the algorithm runs quite fast up to 8 isosurfaces, close to Hadoop’s overhead. When moving to 16 isosurfaces, the disk I/O starts to increase abruptly causing the algorithm to slow down. This increase denotes the amount of temporary disk storage needed for the Mappers to sort the data. Nevertheless, while the data increases quadratically, the run-time only triples. For the PPM dataset, we also see similar behavior in the disk usage. The run-time increases from 10 minutes to 28 minutes and 2 minutes to 5 minutes when the disk I/O increases from 104GB to 252GB for the CLuE and the local cluster, respectively. Figure 1 also shows the isosurfaces for this dataset. The rendering for the combustion PPM dataset is achieved by chaining the map task of our isosurface extraction with the map and reduce job of our surface rendering described in Section 5.3.

## 5.3. Rendering

Performance of the rasterization algorithm is dependent on the output resolution, camera parameters and geometry size. The impact of geometry and resolution is proportional to the number of triangles to be rasterized and fragments generated, while the impact of camera parameters is more difficult to estimate. For instance, depending on a camera position, pixels may receive no or multiple fragments. Hence, reducers may receive no data or several fragments to com-



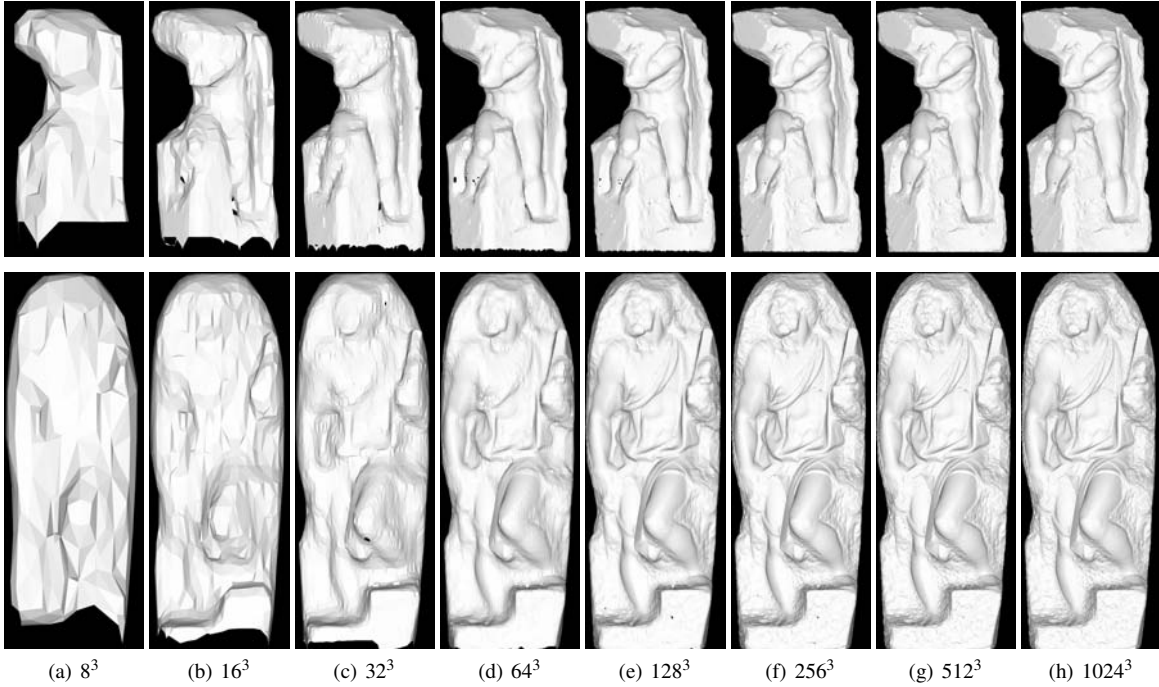
TETRAHEDRAL MESHES VOLUME RENDERING (on Cluster)						
Model	#Tetrahedra	#Triangles	Time	#Fragments	Bytes Read	Bytes Write
Spx	0.8 millions	1.6 millions	3min 29s	9.8 billions	320 GB	473 GB
Fighter	1.4 millions	2.8 millions	2min 20s	5.3 billions	172 GB	254 GB
Sfl	14 millions	28 millions	6min 53s	16.8 billions	545 GB	807 GB
Bullet	36 millions	73 millions	4min 19s	12.7 billions	412 GB	610 GB

**Figure 8:** Volume rendering of the earthquake dataset (SF1) using a 100MP image. Table shows volume rendering statistics for other tetrahedral meshes. The Bullet dataset with 36 million tetrahedra is rendered in 4min and 19s.

pute depth ordering. Figure 7 shows rendering results and weak scaling tests with run times and temporary disk usage. For the CLuE cluster, the cost for rendering images of 100MP or less is insignificant compared to the Hadoop overhead. For our local cluster, this threshold is more than 1GP. For images of this size, the cluster is stretched to its limit and performance is limited by the amount of data written to disk. There is a significant increase in the data size due to the large amount of temporary caching by the system due to insufficient buffer space for the shuffling phase.

Figure 8 illustrates a volume rendering pipeline modified from our surface rendering pipeline. Tetrahedral meshes are broken down into a triangle soup, with each tetrahedron face split into a separate triangle. The reduce phase is modified to perform color, opacity mapping and compositing of the fragments. The accompanying table shows results for a variety of additional meshes. As also shown in the table, the most time-consuming image to render at 100MP is not the largest dataset (Bullet) but the earthquake dataset (SF1). This is due to the many large (and flat) tetrahedra that define empty regions at the bottom of the model. Scalar values of these triangles rarely contribute to the final image, but generate a large number of fragments which causes a more expensive shuffle phase.





**Figure 10:** Simplified meshes using volumes from  $8^3$  to  $1024^3$  using decimation rates under 5%.

CLuE time		ATLAS (18 GB)			St MATTHEW (13 GB)		
#Gridsize	#Maps/ #Reduces	Job 1 Time	Job 2 Time	Output Size	Job 1 Time	Job 2 Time	Output Size
$8^3$	256/256	5min 45s	52s	22 KB	5min 45s	52s	23 KB
$16^3$	256/256	3min 54s	49s	98 KB	3min 54s	49s	105 KB
$32^3$	256/256	3min 51s	49s	392 KB	3min 51s	49s	450 KB
$64^3$	256/256	3min 40s	49s	1.6 MB	3min 40s	49s	1.9 MB
$128^3$	256/256	4min 12s	49s	6.4 MB	4min 12s	49s	7.5 MB
$256^3$	256/256	3min 50s	49s	26 MB	3min 50s	49s	30 MB

Cluster time		ATLAS (18 GB)			St MATTHEW (13 GB)		
#Gridsize	#Maps/ #Reduces	Job 1 Time	Job 2 Time	Output Size	Job 1 Time	Job 2 Time	Output Size
$8^3$	377/377	58s	56s	22 KB	54s	55s	23 KB
$16^3$	377/377	58s	55s	98 KB	54s	54s	105 KB
$32^3$	377/377	55s	54s	392 KB	51s	52s	450 KB
$64^3$	377/377	57s	54s	1.6 MB	55s	55s	1.9 MB
$128^3$	377/377	55s	58s	6.4 MB	52s	52s	7.5 MB
$256^3$	377/377	55s	55s	26 MB	55s	55s	30 MB
$512^3$	377/377	55s	55s	102 MB	55s	55s	119 MB
$1024^3$	377/377	55s	57s	399 MB	55s	53s	461 MB

**Figure 9:** Simplification results. The algorithm requires two map-reduce jobs (jobs 1 and 2 in the table). The local cluster process both datasets in roughly 55s per job.

#### 5.4. Mesh Simplification

To analyze the out-of-core simplification algorithm in the MapReduce model, we use two large triangle meshes as input: the Atlas statue (18GB) and the St Matthew statue (13GB) from the Digital Michelangelo Project at Stanford University. In these tests, we are interested in seeing the effects of scaling the simplifying grid size. The amount of work done in the Map phase should be very consistent, as each triangle must always compute a quadric measure vector and bin its three vertices. Smaller grid sizes force more vertices to coincide in any particular bin, thus changing the

grouping and potentially reducing the parallelism in the Reduce phase.

However, the decrease in performance from this should be amortized by the decreased output of the Reduce phase, as fewer triangles will be generated. In the tables of Figure 9 we observe that this is exactly what occurs. Since our method must be a two pass algorithm in Hadoop, we have included the run times for both jobs (Job 1 and Job 2). Rendered images of simplified models of the Atlas and the St Matthew statue are also shown in Figure 10 with the grid sizes varying from  $8^3$  to  $1024^3$ . Decimation rates for these results are all under 5% and they were all rendered using the renderer proposed in Section 5.3.

## 6. Discussion

In this section, we discuss some of the “lessons learned” from our experience with MapReduce and Hadoop. For users of visualization techniques, it is difficult to know when the results or workload will push beyond the cluster limits and severely increase runtimes. It was clear from our experiments that keeping the number of maps and reduces below the maximum task capacity is the first step towards avoiding such pathological cases. While nodes can run multiple tasks, we find that increasing the number of nodes in proportion to the data size provides the most reliable and consistent scalability, suggesting that the overhead to manage additional nodes is not prohibitively expensive.

Additional tasks may improve performance in certain

cases, but should not be a default assumption. The results from our exploratory implementations are encouraging and match the scalability we expected, up to a limit. When the size of the output data is unpredictable, as in the case of isosurface extraction, memory requirements can quickly exhaust available resources, leading to disk buffering and ultimately increasing runtime. Scalability, in our experience, is only achieved for data reductive tasks — tasks for which the output is smaller than the input. Most visualization tasks satisfy this property, since they typically render (or generate) data that is smaller than the input mesh or volume. It should also be pointed out that this cost is insignificant when compared to today's standard practice of transferring data to a client, and running a local serial or parallel algorithm. Indeed, the cost of transferring the data to a local server alone dwarfs the cost of any such MapReduce job.

For those interested in developing visualization algorithms for MapReduce systems, our experience has shown that even naïve implementations can lead acceptable results. Implementing the MapReduce algorithms was relatively simple. However, as with any highly-parallel system, optimization can be painstaking. In the case of MapReduce, we found that the setup and tuning of the cluster itself was just as important, if not more important, than using the right data format, compressor, or splitting scheme. To analyze the suitability of existing algorithms to the MapReduce model, attention should be paid to where and how often sorting is required. As the model only allows a single sort phase per job, multi-pass algorithms can incur significant overhead when translated naïvely into MapReduce. Specifically, a MapReduce implementation will rarely be competitive with state-of-the-art methods in terms of raw performance, but the simplicity and generality of the programming model is what delivers scalability and extensibility. Further, the degree of parallelism in the Reduce phase is determined by both the intended output of the algorithm, as well as the distribution of data coming from the Map phase. Careful consideration of the hashing method may or may not have a dramatic effect on the algorithm performance.

We summarize below observations and conclusions we made in our work with the Hadoop system:

- Results from our scaling tests show Hadoop alone scales well, even without introducing optimization techniques.
- Considerations about the visualization output size are very important. Visualization techniques should decrease or keep relatively constant the size of the data in the pipeline rather than increase it. MapReduce was not designed to handle enormous intermediate datasets, and performs poorly in this context.
- From a qualitative standpoint, we found the MapReduce model easy to work with and implement our solutions. Optimization, in terms of compression and data reader/writers required thought and experimentation. Configuring job parameters and cluster settings for optimal performance was very challenging. We feel that this complexity is always inherent in a large distributed environment, and therefore is acceptable. Further, this work can potentially be performed once per cluster, and the cost can therefore be amortized over many MapReduce jobs.
- The inability to chain jobs makes multi-job algorithms such as the mesh simplification slightly cumbersome to execute, and more difficult to analyze. Projects such as Pig [ORS\*08] and Hive [TSJ\*09] that offer a high-level yet extensible language on top of MapReduce are promising in this regard.
- The Hadoop community could greatly benefit from better progress reporting. Uneven distribution of data across reducers may result in display of near completion (*e.g.*, 98%) when in fact the bulk of the work remains to be completed. This is problematic if the user does not know *a priori* what a good reducer number should be, and arbitrarily chooses a high value.
- While at any particular time, job runtimes are fairly consistent, they vary as a whole from day to day. This is most likely due to the state of the HDFS and movement of replicated data. Being aware of these effects is important in order to make meaningful comparisons of performance results. On that note, all data within any one table was generated within a short time span.

## 7. Conclusions and Future Work

The analysis performed in this paper has shown that the MapReduce model provides a suitable alternative to support large-scale exploratory visualization. The fact that data transfer alone is more expensive than running such a job *in-situ* is sufficient justification, and will become more evident as datasets grow in size. The availability of a core set of visualization tools for MapReduce systems will allow faster feedback and learning from new and large datasets. Additionally, as these systems continue to evolve, it is important for the visualization community to periodically re-evaluate their suitability. Here, we provide a baseline for such a comparative analysis. We have shown how three visualization techniques can be adapted to MapReduce. Clearly, many additional methods can be adapted in similar ways, in particular memory-insensitive techniques or inherently parallel techniques. What remains to be investigated is how to combine visualization primitives with conventional data management, query, and processing algorithms to construct a comprehensive scalable visual analytics platform.

## References

- [amaa] Amazon Simple Storage Service (Amazon S3). <http://www.amazon.com/gp/browse.html?node=16427261>. 2
- [amab] Amazon SimpleDB. <http://www.amazon.com/SimpleDB-AWS-Service-Pricing/b?ie=UTF8&node=342335011>. 2
- [aws] Amazon web services - elastic mapreduce. <http://aws.amazon.com/elasticmapreduce/>. 1

- [Bor07] BORTHAKUR D.: The Hadoop distributed file system: Architecture and design. [http://lucene.apache.org/hadoop/hdfs\\_design.pdf](http://lucene.apache.org/hadoop/hdfs_design.pdf), 2007. 2
- [Cam99] CAMAHORT E.: 1999. The Contour Library <http://www.ticam.utexas.edu/ccv/software/libcontour/>. 4
- [CDG\*06] CHANG F., DEAN J., GHEMAWAT S., HSIEH W. C., WALLACH D. A., BURROWS M., CHANDRA T., FIKES A., GRUBER R. E.: Bigtable: a distributed storage system for structured data. In *Proc. of the 7th USENIX Symp. on Operating Systems Design & Implementation (OSDI)* (2006). 2
- [CJL\*08] CHAIKEN R., JENKINS B., LARSON P.-A., RAMSEY B., SHAKIB D., WEAVER S., ZHOU J.: Scope: easy and efficient parallel processing of massive data sets. In *Proc. of the 34th Int. Conf. on Very Large DataBases (VLDB)* (2008), pp. 1265–1276. 1, 2
- [clu] Nsf cluster exploratory (nsf08560). <http://www.nsf.gov/pubs/2008/nsf08560/nsf08560.htm>. 1, 5
- [DG04] DEAN J., GHEMAWAT S.: MapReduce: simplified data processing on large clusters. In *Proc. of the 6th USENIX Symp. on Operating Systems Design & Implementation (OSDI)* (2004). 1, 2
- [DG08] DEAN J., GHEMAWAT S.: MapReduce: simplified data processing on large clusters. *CACM* 51, 1 (2008), 107–113. 1
- [DHJ\*07] DE CANDIA G., HASTORUN D., JAMPANI M., KAKULAPATI G., LAKSHMAN A., PILCHIN A., SIVASUBRAMANIAN S., VOSSHALL P., VOGELS W.: Dynamo: Amazon’s highly available key-value store. In *Proc. of the 21st ACM Symp. on Operating Systems Principles (SOSP)* (2007), pp. 205–220. 2
- [DPR\*08] DEWITT D. J., PAULSON E., ROBINSON E., NAUGHTON J., ROYALTY J., SHANKAR S., KRIOUKOV A.: Clustera: an integrated computation and data management system. In *Proc. of the 34th Int. Conf. on Very Large DataBases (VLDB)* (2008), pp. 28–41. 2
- [FS01] FARIAS R., SILVA C. T.: Out-of-core rendering of large, unstructured grids. *IEEE Comput. Graph. Appl.* 21, 4 (2001), 42–50. 4
- [GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *SIGGRAPH ’97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), ACM Press/Addison-Wesley Publishing Co., pp. 209–216. 4
- [goo] Google App Engine Datastore. <http://code.google.com/appengine/docs/datastore/>. 2
- [had] Hadoop. <http://hadoop.apache.org/>. 1, 2
- [HG05] HEBER G., GRAY J.: *Supporting Finite Element Analysis with a Relational Database Backend; Part 1: There is Life Beyond Files*. Tech. rep., Microsoft MSR-TR-2005-49, April 2005. 2
- [Hiv] Hive. <http://hadoop.apache.org/hive/>. Accessed March 7, 2010. 3
- [IBM05] IBM SYSTEMS AND TECHNOLOGY GROUP: *IBM Deep Computing*. Tech. rep., IBM, 2005. 2, 3
- [IBY\*07] ISARD M., BUDIU M., YU Y., BIRRELL A., FETTERLY D.: Dryad: Distributed data-parallel programs from sequential building blocks. In *Proc. of the European Conference on Computer Systems (EuroSys)* (2007), pp. 59–72. 2
- [iri] Incorporated Research Institutions for Seismology (IRIS). <http://www.iris.edu/>. 3
- [Kos07] KOSMIX CORP.: Kosmos distributed file system (kfs). <http://kosmosfs.sourceforge.net>, 2007. 2
- [Law] LAWRENCE LIVERMORE NATIONAL LABORATORY: VisIt: Visualize It in Parallel Visualization Application. <https://wci.llnl.gov/codes/visit> [29 March 2008]. 3
- [LC87] LORENSEN W., CLINE H.: Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics* 21, 4 (1987), 163–169. 3
- [LS01] LINDSTROM P., SILVA C. T.: A memory insensitive technique for large model simplification. In *VIS ’01: Proceedings of the conference on Visualization ’01* (Washington, DC, USA, 2001), IEEE Computer Society, pp. 121–126. 4
- [lss] Large Synoptic Survey Telescope. <http://www.lsst.org/>. 3
- [Ma09] MA K.-L.: In situ visualization at extreme scale: Challenges and opportunities. *Computer Graphics and Applications, IEEE* 29, 6 (nov.-dec. 2009), 14–19. 3
- [mic] Azure Services Platform - SQL Data Services. <http://www.microsoft.com/azure/data.mspx>. 2
- [ORS\*08] OLSTON C., REED B., SRIVASTAVA U., KUMAR R., TOMKINS A.: Pig latin: a not-so-foreign language for data processing. In *SIGMOD’08: Proc. of the ACM SIGMOD Int. Conf. on Management of Data* (2008), pp. 1099–1110. 1, 2, 3, 9
- [PAB\*08] PAUL B., AHERN S., BETHEL E. W., BRUGGER E., COOK R., DANIEL J., LEWIS K., OWEN J., SOUTHARD D.: Chromium Renderserver: Scalable and Open Remote Rendering Infrastructure. *IEEE Transactions on Visualization and Computer Graphics* 14, 3 (May/June 2008). LBNL-63693. 2, 3
- [par] Paraview. <http://www.paraview.org> [29 March 2008]. 3
- [PDGQ05] PIKE R., DORWARD S., GRIESEMER R., QUINLAN S.: Interpreting the data: Parallel analysis with Sawzall. *Scientific Programming* 13, 4 (2005). 2
- [PPR\*09] PAVLO A., PAULSON E., RASIN A., ABADI D. J., DEWITT D. J., MADDEN S. R., STONEBRAKER M.: A comparison of approaches to large scale data analysis. In *SIGMOD* (Providence, Rhode Island, USA, 2009). 1
- [RSFWH98] RICHARDSON T., STAFFORD-FRASER Q., WOOD K. R., HOPPER A.: Virtual network computing. *IEEE Internet Computing* 2, 1 (1998), 33–38. 2
- [Sil] SILICON GRAPHICS INC.: OpenGL vizserver. <http://www.sgi.com/products/software/vizserver>. 2, 3
- [slo] Sloan Digital Sky Survey. <http://cas.sdss.org>. 3
- [SME02] STEGMAIER S., MAGALLÓN M., ERTL T.: A generic solution for hardware-accelerated remote visualization. In *VIS-SYM ’02: Proceedings of the symposium on Data Visualisation 2002* (Aire-la-Ville, Switzerland, Switzerland, 2002), Eurographics Association, pp. 87–ff. 2
- [SZL92] SCHROEDER W. J., ZARGE J. A., LORENSEN W. E.: Decimation of triangle meshes. In *SIGGRAPH ’92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1992), ACM, pp. 65–70. 4
- [Tec] TECHNOLOGIES I.: Hadoop performance tuning - white paper. 5
- [TSJ\*09] THUSOO A., SARMA J. S., JAIN N., SHAO Z., CHAKKA P., ANTHONY S., LIU H., WYCKOFF P., MURTHY R.: Hive - a warehousing solution over a map-reduce framework. *PVLDB* 2, 2 (2009), 1626–1629. 9
- [Yaha] YAHOO! RESEARCH: PNUTS - Platform for Nimble Universal Table Storage. <http://research.yahoo.com/node/212>. 2

- [yahb] Yahoo! expands its m45 cloud computing initiative, adding top universities to supercomputing research cluster. <http://research.yahoo.com/news/3374>. 1
- [YDHP07] YANG H., DASDAN A., HSIAO R.-L., PARKER D. S.: Map-reduce-merge: simplified relational data processing on large clusters. In *SIGMOD'07: Proc. of the ACM SIGMOD Int. Conf. on Management of Data* (2007), pp. 1029–1040. 2
- [YIF\*08] YU Y., ISARD M., FETTERLY D., BUDIU M., ER-LINGSSON U., GUNDA P. K., CURREY J.: DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In *Proc. of the 8th USENIX Symp. on Operating Systems Design & Implementation (OSDI)* (2008). 1, 2
- [YSZ04] YAN J., SHI P., ZHANG D.: Mesh simplification with hierarchical shape analysis and iterative edge contraction. *IEEE Transactions on Visualization and Computer Graphics* 10, 2 (2004), 142–151. 4