Title: VisIO: Enabling Interactive Visualization of Ultra-Scale, Time Series Data via High-Bandwidth Distributed I/O Systems

Author(s): Christopher J. Mitchell, 229505, HPC-5 / UCF
James P. Ahrens, 113788, CCS-7
Jun Wang, University of Central Florida

Intended for: International Parallel and Distributed Processing Symposium (IPDPS) 2011
Anchorage, Alaska
May 16-20, 2011

VisIO: Enabling Interactive Visualization of Ultra-Scale, Time Series Data via High-Bandwidth Distributed I/O Systems

Christopher Mitchell*, James Ahrens[†], and Jun Wang*
*Department of Electrical Engineering & Computer Science, University of Central Florida, Orlando, Florida 32816–2450
[†]Computer, Computational, & Statistical Sciences, Los Alamos National Laboratory, Los Alamos, New Mexico 87545
Email: {mitchell, jwang}@eecs.ucf.edu*, ahrens@lanl.gov[†]

***Abstract*—Petascale simulations compute at resolutions ranging into billions of cells and write terabytes of data for visualization and analysis. Interactive visualization of this time series is a desired step before starting a new run. The I/O subsystem and associated network often are a significant impediment to interactive visualization of time-varying data; as they are not configured or provisioned to provide necessary I/O read rates.**

In this paper, we propose a new I/O library for visualization applications: VisIO. Visualization applications commonly use N-to-N reads within their parallel enabled readers which provides an incentive for a shared-nothing approach to I/O, similar to other data-intensive approaches such as Hadoop. However, unlike other data-intensive applications, visualization requires: (1) interactive performance for large data volumes, (2) compatibility with MPI and POSIX file system semantics for compatibility with existing infrastructure, and (3) use of existing file formats and their stipulated data partitioning rules.

VisIO, provides a mechanism for using a non-POSIX distributed file system to provide linear scaling of I/O bandwidth. In addition, we introduce a novel scheduling algorithm that helps to co-locate visualization processes on nodes with the requested data. Testing using VisIO integrated into ParaView was conducted using the Hadoop Distributed File System (HDFS) on TACC's Longhorn cluster. A representative dataset, VPIC, across 128 nodes showed a 64.4% read performance improvement compared to the provided Lustre installation. Also tested, was a dataset representing a global ocean salinity simulation that showed a 51.4% improvement in read performance over Lustre when using our VisIO system. VisIO, provides powerful high-performance I/O services to visualization applications, allowing for interactive performance with ultra-scale, time-series data.

***Keywords*-Data Intensive Scientific Computing; Scientific Visualization; Parallel Computing; Distributed Computing; I/O**

I. Introduction

Scientific visualization applications form a core application area within the umbrella of Data-Intensive Computing. These applications have come into prominence in recent years as a direct result of the transition into Petascale class simulations of real-world phenomena as well as in-field deployment of high-resolution sensors. These simulations include nuclear fusion research [1], and cosmology simulations of dark matter theories [2]. Likewise, the deployment of ultra-fast gene sequencing systems [3] and the Large Hadron Collider [4] exemplify the growth and dependence on high-resolution sensors. The scientific demand for transforming these Tera/Petabytes of produced data into meaningful insight has forced these visualization applications to compute interactively while keeping pace with the volumes of produced data [5].

The I/O demands of visualization applications were highlighted in a recent work by Childs, *et. al.* [6] which demonstrated visualization performance at scale on six supercomputers across the United States Department of Energy complex. A key observation made was that while the actual visualization algorithm and rendering times were sufficiently fast for interactive performance even with ultra-scale datasets, I/O time dwarfed these operations by an order of magnitude and in one test by almost two orders of magnitude. One effort to mitigate this effect was proposed by Peterka and Ross, *et. al.* [7], [8] through the use of MPI-IO optimizations and reordering of data within the input files to allow for contiguous reads of needed data. This software based approach, however, is still limited by the scalability of the underlying storage hardware and associated network to handle ever larger datasets.

These visualization applications represent a unique category of applications that perform sophisticated calculation routines but also are data-intensive in that they have to ingest data in the Tera/Petabyte range. These applications are tough to properly serve on both traditional computational-intensive platforms as well as on data-intensive platforms. Traditional computationally intensive platforms are I/O starved [9] and leave these applications waiting for long periods of time while data is read in from a central parallel file system. On the other hand, these applications would struggle on a data-intensive platform due to the lack of high performance CPUs and/or GPUs on which to run the needed calculations.

Therefore, we propose a hybrid approach that can properly support these visualization applications. This approach calls for the use of a traditional High Performance Computing (HPC) cluster with mid to high end CPUs and optional GPU hardware. However, we forgo a traditional centralized parallel file system in favor of a distributed file system with local hard drives attached directly to the individual cluster nodes. Specifically, we deploy the Hadoop Distributed File System (HDFS) [10] in our tests and propose a library, VisIO, which allows for these MPI-based applications to interpret the data locality information exposed by the file system and appropriately schedule computation on nodes locally containing the needed data for that MPI rank.

Motivating this architectural proposal are the limitations inherent in the scalability of todays parallel file systems that make it difficult to handle data-intensive workloads without bottleneck. These centralized systems rely on a single or handful of network links that are statically constructed during installation and are responsible for transferring data to and from the cluster [11]. These links do not scale with the given problem size and while adequate for jobs that are below the maximum available bandwidth, quickly detriment the application with long latencies and slow reads when they are oversubscribed. Conversely, a distributed file system like HDFS, constructed from locally attached disks, can scale with the problem size and node count as needed. Thus, HDFS is able to sidestep the network bottlenecks inherent in todays parallel file systems and allow it to scale from Petascale datasets up towards Exascale datasets. Additionally, visualization applications tend to exhibit N-to-N workloads which differentiates from commonly seen N-to-1 workloads in simulation environments that parallel file systems are designed for [12], [13].

Our contribution, therefore, focuses on laying out the architectural details needed to both define the VisIO library as well as the system details needed to use a distributed file system effectively for this class of applications. Specifically, we use Kitware's ParaView visualization package and the Hadoop Distributed File System (HDFS) on the Texas Advanced Computing Center's (TACC) Longhorn visualization cluster to prove the potential of this system. This can be summarized as:

- Development of the VisIO library for use of distributed file systems within visualization applications working on current ultra-scale and future exascale datasets.
- Development of a new method to leverage file system data locality within an MPI-based program to intelligently co-locate computation and data.

Validation of this concept was done using two representative datasets: the VPIC plasma physics simulation, and an ocean salinity simulation. We experienced between a 50 and 65% improvement in read times for these datasets using VisIO compared to the the current I/O methods. In addition, these performance gains were made while successfully keeping approximately 92% of the data local to the node requiring it. Even though testing was taken to 128 nodes on one of the larger available visualization clusters, we believe that coming larger visualization clusters would yield even better improvements in I/O performance due to increased node count. Thus, we believe that our technique will provide an opportunity to visualize these ever increasing datasets while meeting user expectations for interactivity, currently set at 10 seconds or less for system response [14], [15].

We present the rest of this paper by first detailing some background information in Section II. Section III details the full design of the VisIO library. Section IV discusses the experimental methodology used to prove this system's effectiveness and analyzes the collected results. Finally, Section V examines related works while Section VI discusses our conclusions.

II. Background

A. *ParaView and the VTK Streaming Demand Driven Pipeline*

The core tenant of the design of ParaView is its streaming demand driven pipeline which dictates the data-flow through the individual modules that transform raw data into rendered images. The interworking of the demand driven pipeline is shown in Figure 1. In this example pipeline, the user requests a time step to be loaded and displayed on screen in addition to needed filters to process the data. The client facing GUI reports the request to the executive which sets the parameters for this request on the pipeline for each node and then requests that the renderers produce the picture. Since the renderer does not have any input data, it requests the data range to be displayed from the filter sequence connected to it's input port. Likewise, the filters do not have the data needed to run the filer against and finally report the requested data range to the reader to fetch the needed information from the file system. Once retrieved, the data is propagated back down the pipeline, following the arrows, from the reader to the filter sequence and finally to the renderer for display to the user's screen. This is a simple example of the pipeline in action but can be expanded to a large network of readers, filters, and renderers across numerous nodes to perform this operation in parallel.

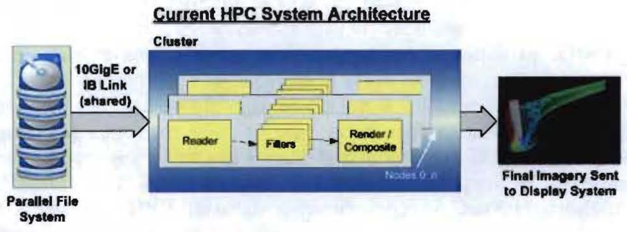
Current HPC System Architecture
Cluster
10GigE or IB Link (shared)
Reader
Filters
Render / Composite
Nodes 0..n
Final Imagery Sent to Display System
Parallel File System

Fig. 1. VTK/ParaView visualization demand driven pipeline. In this version of the pipeline, a globally accessible network attached file system feeds input data files to nodes as they are requested by the pipeline. The demand driven pipeline then proceeds to process and render the input files producing the final composited image.

This layering approach allows for complete modularity within the visualization process. The pipeline enforces a standardized input and output port specification and an executive performs the needed operations to call various methods within the modules at the correct time to keep data moving through the pipeline. This modularity allows for an arbitrary set of readers to interact with an arbitrary set of filters which in turn work with an arbitrary set of rendering engines. Thus, it is possible to swap out components as needed to gain desired results within the visualization pipeline. In our case, the stock version of the reader for MultiBlock data files will be replaced with a custom version designed to use the VisIO system and associated distributed file system. This swap process does not impact any other piece of the pipeline thus allowing continued use of existing filters and rendering engines without additional modification being needed.

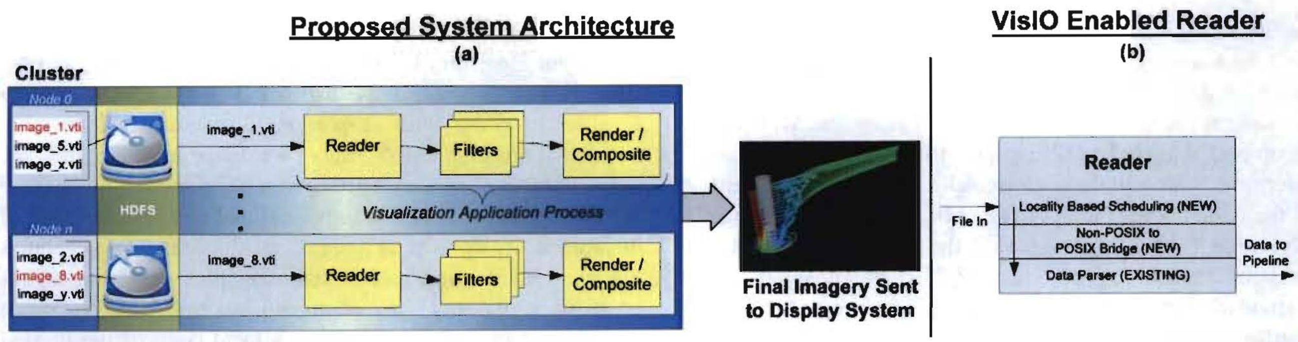

Fig. 2. Proposed Visualization Pipeline with VisIO enabled reader. (a) Each node is responsible for a non-overlapping sub-extent denoting a piece of the total picture to render. Each node has an independent visualization pipeline which only coordinates with the other nodes for final compositing and in special cases for some filters. Each node is scheduled a single input file out of the set based on replica placement. (b) The VisIO enabled reader, within the pipeline, uses two new modules to access the distributed file system and intelligently schedule read operations based on data locality.

B. The Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is the open source community's response to the Google File System (GFS) for use with MapReduce style workloads. HDFS is designed with the idea that it is quicker to ship the computational executables to the stored data and process in-situ rather than to pull the needed data from storage and send it via a network to the compute node that has the executable loaded. Thus, HDFS calls for local hard drives to be installed in each compute node (known as a DataNode) and distributes stored data, broken down into chunks, across the nodes using a pseudo-random placement system. To combat hard drive failure rates, HDFS calls for these chunks to be replicated, by default, a total of 3 times within the cluster. A chunk, by default, represents up to 64MB of a file - a decision that reflects HDFS's design goal of handling large files. All of the metadata associated with mapping a given chunk to a DataNode is contained within a memory based map loaded on the NameNode (HDFS's metadata server) and backed up to the Secondary NameNode. Thus, this distributed file system presents the applications with a chunk oriented view of the files stored, with local access to a given chunk from three of the nodes within a cluster.

While originally designed for use with the MapReduce framework, HDFS can be interfaced with traditional parallel programs (including those based on MPI) via it's libHDFS library for C/C++ based programs. By using libHDFS, a client program is able to make connections to the HDFS's NameNode, and request locality information from the file system for a given chunk (specified by a file name, offset within the file, and length of the request). The NameNode will respond to the client with a listing of the DataNodes (the actual storage locations for the chunks in question) where the requested chunks are stored. Reads requested from clients that are executing on nodes with locally stored chunks can deliver data directly to the application, bypassing the network, and improving I/O performance. Based on this information, the application can make decisions on where to execute a given piece of code.

III. Design and Implementation of VisIO

A. Design of a Distributed File System Enabled Reader

As discussed, currently ParaView relies on an I/O model that is represented in Figure 1 and calls for a centralized parallel file system to service data requests for the cluster. Based on the foundations set fourth above, a move towards the I/O model shown in Figure 2 is desired to improve I/O performance for current and next generation datasets being produced. This new model calls for ParaView to be told which files are available locally via the Distributed File System (DFS) and assign processing responsibility for that data to the given node. Implementation of this idea requires a rework of the ParaView reader modules within the visualization pipeline.

Our VisIO enabled reader for ParaView is currently designed to operate on VTK MultiBlock files to allow for flexability in the underlying data type (structured or unstructured). MultiBlock files are self-describing with XML headers and are composed of an index file with a ".vtmb" extension. Data files, one per process in the group that wrote the original data, are stored as raw binary representing a given type of data (image, rectilinear grid, etc.). As such, when placed on the target DFS, HDFS, the index file is replicated equal to the number of nodes in the cluster to enable all nodes rapid access to the needed metadata about the data to be visualized. The data files are loaded with a standard replication factor of 3 and the chunk size specified such that the entire data file fits within a single chunk. This last requirement of fitting the entire data file into a single chunk is done to ensure that each pipeline is assigned a single, contiguous (in terms of the contained extents) work unit. This allows for easier scheduling of the data to a pipeline while improving performance by not requiring seeking between multiple chunk locations on the hard drive.

One key issue that had to be addressed is interfacing ParaView's POSIX view of the I/O system with HDFS's non-POSIX compliant access methods. The primary reader code invoked within the VTKOpenFile() method handles this translation. This code starts by intercepting the requested file name from the pipeline and if the request is for a ".vtmb" index file, the reader calls the assignNodes() method to execute the scheduling algorithm, detailed in the next section, on the group of data files represented by the given index file. This schedule is computed on the first rank within the MPI process group and scattered to all nodes within the group for use to determine which node reads which data file. The remainder of the code within the VTKOpenFile() method is outlined in Algorithm 1. This code checks if the file exists on the HDFS as claimed and then proceeds to get information about the file, particularly the file's size for use by the actual read command, similar to what is done with a POSIX stat() call. A character buffer is allocated to hold the file's binary contents and calls are made to the HDFS to open and then read the entire data file into the character buffer. This buffer is then converted to an STL istringstream so that the remainder of the reader code can be used unchanged as it is expecting a variable of type ifstream from this method. This conversion to an istringstream is the key to allowing the rest of the existing code, which is expecting a POSIX friendly access to the file, to continue working without modification. Finally, the HDFS handles are closed and the allocated memory reclaimed prior to handing the istringstream off to the rest of the reader infrastructure to be parsed and converted into VTK data structures for use by the rest of the visualization pipeline.

Algorithm 1 VTK MultiBlock Reader

```
Input: File name to open.
Output: istringstream with requested file for XML parsing.
Steps:
 1: Call hdfsExists to check for existence of file.
 2: if File Exists == FALSE then
 3:     Return error to pipeline and exit method.
 4: end if
 5: Call hdfsGetPathInfo to get file statistics.
 6:     →Store file's chunk size.
 7: Allocate a character buffer the size of the file's chunk.
 8: Call hdfsOpenFile to get a handle to the file.
 9: Call hdfsPread to read the chunk from beginning to end
    into the character buffer.
10: Copy the character buffer to an STL string.
11: Assign the STL String to an istringstream
12: Set the istringstream to be pointed to by the reader's
    stream pointer for use by the XML parser.
13: Call hdfsCloseFile to close file handle and free memory
    allocations for the character buffer.
```

B. Node Assignment

The key to gaining optimal read performance when using HDFS is to assign computational processes such that the data they need is local to the node's hard drive. This avoids the need to remotely pull data from hard drives on other nodes which themselves are tasked with processing another subset of the data. However, this scheduling of computational tasks to nodes needs to be done with consideration of HDFS's placement algorithm.

As previously mentioned, HDFS uses a pseudo-random placement algorithm to store chunks of data. Thus, it is possible to have data unevenly distributed on the cluster leaving a subset of nodes with proportionally more data than other nodes that are available. As such, we define a metric, *scarcity*, as the number of available nodes which can serve a file that are currently not allocated to a computational task. Through use of this metric, we devised an assignment algorithm that schedules the scarcest files first before handling files that have higher levels of availability. Any computational tasks that could not be assigned to a node containing a local copy of the needed data are directly paired with the remaining nodes with the expectation that the needed data for this small percentage of tasks will be pulled over the cluster network. The number of files that must request data over the network is tracked in our testing and evaluated in section IV.D.

In comparison, a naive implementation of the assignment algorithm, using a first-in, first-out implementation on a list of files to be scheduled, would have a higher potential to leave files scheduled to non-optimal nodes without local copies of the needed data. This is because nodes that can be used to service files that are scarce in the cluster are likely to be assigned to process another eligible file on that node. Thus the optimal solution from this naive algorithm would be entirely dependent on the ordering of the input lists.

To achieve scheduling via scarcity, we implement a version of the stable marriage algorithm [16] as shown in Algorithm 2. We select the next file to match to a node based on the number of nodes available at the start of the iteration to service that file - it's scarcity value. However, in the original algorithm, the members of each set to be matched amongst have a weighting system to determine preferability to a given potential match. Since there is no preference between any node containing a replica of the original data, we simply match the file with the first available node within the set of nodes containing the file. Thus, the weighting values from the original algorithm are reduced to a "1" or "0" for a file's preference for a node based on if the node has the data or not.

C. HDFS Use Considerations

To achieve the best possible performance from the HDFS, a few considerations had to be taken into account. First, should the file size exceed the configured chunk size for the HDFS instance (64 MB by default), the file will be broken up into multiple chunks with each chunk being replicated independently of the other related chunks. However, if a process only has a fraction of the overall assigned piece local and the other parts must be pulled from the network, the locality performance advantage disappears due to the network overhead and possible congestion at the nodes the chunks were requested from. Thus, the chunk size for the file system must be set to be larger than the largest data file present. This ensures that a process has all of the needed data for its assigned extents locally available. Additionally, we also size the data files such that each node only has to handle a single piece of the entire dataset for a given time step. This prevents

Algorithm 2 Node Assignment Algorithm

Input:

- A set C of nodes available.
- A set F denoting all files making up a time step.
- A list R of tuples, $< node, file >$, denoting where each file and its replicas are stored. There may be more than one instance of a given node or file within R.

Output:

- A list A of tuples, $< node, file >$, denoting which file has been assigned to a given node. The values node and file may only be used once in A.

Steps:

```
1: while there exists an instance: $F' \bigcap R \neq 0$ do
2:     Find $F'$ with $min(F' \bigcap R)$.
3:     $C'$ = first node in set $min(F' \bigcap R)$.
4:     $A_{C'} = F'$
5:     Remove $C'$ from $R$.
6:     Remove $F'$ from $R$.
7: end while
8: for all files $F'$ that have not been assigned do
9:     Find first $A_{C'} == NULL$
10:    Set $A_{C'} = F'$
11: end for
```

costly context switches between multiple processes on a node to handle multiple pieces as well as prevents even costlier seeks on the hard drive to service multiple chunk requests to the HDFS.

Finally, we also override the default replication of 3 times for the metadata files (one per time step, ".vtmb" extension) and replicate these files n ways where n is the number of DataNodes in the job allocation. This mitigates an issue where every ParaView process at once tries to access the metadata file for the time step at the beginning of the time step. A hotspot would occur within the cluster amongst the three nodes storing the metadata file if the defaults were left as is. A future version of the reader could be updated to have a node containing the metadata file locally read in the file and broadcast it to the remaining nodes. An acceptable solution considering the metadata file is usually tiny (a few hundred kilobytes) compared to a data file.

D. Fault Tolerance

The notion of being able to run this system at scale brings the issue of fault tolerance to the forefront. Traditionally, if the file system were to fail, the entire ParaView application would either crash or would render incorrect results (missing pieces of the image for example) depending on the severity of the failure. However, when using HDFS, the loss of a DataNode process or its underlying storage infrastructure poses no threat to the application or the integrity of the rendered results. Due to HDFS's replication system, a reader in this situation can still request the data on the failed node and the HDFS will deliver the requested chunk from one of the replica nodes. So, while a performance hit will be taken for the remote transfer of the data, ParaView can still continue to run and still produce correctly rendered results. In comparison, parallel file systems rely on hardware level resiliency measures such as RAID and hot spares to handle faults. While providing a level of fault tolerance, this mechanism does not cover issues such as an entire RAID group of LUN going offline. Additionally, RAID rebuilds can slow down I/O operations to any file on a RAID group while transfer of a replica from another node in the HDFS only impacts performance on the requesting and the servicing node.

IV. Test and Validation

A. Test Cluster Setup

Testing of our HDFS enabled ParaView reader was conducted at the Texas Advanced Computing Center (TACC) on their Longhorn visualization cluster [17]. This system consists of 256 total nodes comprised of Dell PowerEdge R610 (240 nodes) and R710 (16 nodes) servers with dual Intel Nehalem Xeon E5540 processors. Each node has either 48GB (R610) or 144GB (R710) of RAM and all nodes have a 73GB Seagate Savvio 15K.2 local hard drive (model: ST973452SS) [18]. The node interconnect is Mellanox QDR InfiniBand and has interconnections to a 210TB Lustre parallel file system. All nodes were running CentOS 5.4 at the time of testing.

Testing was conducted using ParaView 3.8.0 and Hadoop version 0.20.2. Additionally, Hadoop's configuration was kept as close to default as possible to eliminate unfair advantages given to Hadoop via tuning that would not be possible to provide to Lustre. TACC's storage infrastructure was running version 1.8.3 of Lustre. Lustre was left to its defaults of using a stripe count of four and stripe size of 1 MB. The Mesa 3D graphics library, version 7.6.1, was used for off-screen rendering by ParaView. Finally, the supplied MVAPICH2 version 1.4.1 MPI library was used to enable access to the IB interconnect within the cluster.

It should also be noted that testing was setup such that reported results are based off of the performance ParaView sees from the given file system. Time taken to load the HDFS instance from archive is not counted nor is the time to transfer data from archive to Lustre for testing. This data migration issue is outside of the scope of this paper and treated as a cost inherent in working with a system where simulations or data producing sensors are not internal to the cluster running the visualization application.

B. Test Datasets

1) VPIC - Plasma Physics Simulation: This data set is the result of a Magnetic Reconnection experiment run using VPIC (a Los Alamos National Laboratory developed Particle-In-Cell code) [19] that was run during the LANL Roadrunner Open Science Runs. This data set was produced on a full system run on Roadrunner and represents approximately 4.68 TB of raw data. Due to the need to move this data set to the test system, from LANL, for our experiments, a subset of this data was selected, consisting of just the magnetic field vector, and converted to ParaView's MultiBlock format. It should be noted that the data was written out in "append" mode to allow for the least amount of time to be spent on parsing of the data as binary data in this format can be directly read into memory by the ParaView readers. Additionally, of the total 342 time steps, 20 were selected (steps 0 through 190 in increments of ten). As a result, our test set was approximately 135 GB in total size and 6.9 GB per time step.

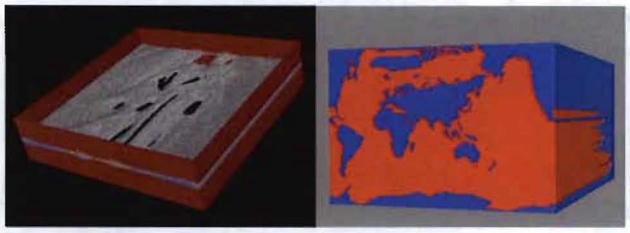
Fig. 3. Representative images of the datasets tested. The image on the left is from the Los Alamos National Laboratory's VPIC Plasma Physics simulation of magnetic reconnection. The image on the right is from the Los Alamos National Laboratory's Ocean Modeling Simulation showing ocean salinity.

2) Ocean Salinity - Ocean Simulation: The ocean salinity data set was generated by LANL's Climate, Ocean and Sea Ice Modeling project [20]. The model this data represents is a time series of the world's ocean salinity from surface to ocean floor ranging from February 2001 to January 2004. This represents 36 total time steps, one per month in the given range. Again, due to constraints on moving data from LANL to our test cluster at TACC, only one variable worth of files were moved. Once located on the test cluster, the data files were pre-processed to clone the original variable ten times to match the original number of variables tracked by the simulation. The final VTK MultiBlock files, containing the data in rectilinear grid format, averaged 25 GB per time step. Thus, the entire data set, across all time steps, encompassed 887 GB worth of data.

C. Validation of VisIO under Simulated Workload

Before implementing a VisIO compatible reader, a benchmark was performed with a similar workload as the proposed reader to validate that HDFS would be capable of reading with the bandwidth needed for use by ParaView. The benchmark consisted of the same sequence of I/O calls as the reader would make but executed against dummy files. These dummy files were setup such that each node in the test would be given a unique file that was 1 GB in size, thus making this a weak scaling test. Node count was increased from 1 to 128 nodes, in powers of 2, for testing. The benchmark would then proceed to have each of the nodes open and read into memory the contents of its assigned file with the process being timed. The results of this test are shown in Figure 4.

The file systems tested included the Lustre parallel file system, and HDFS in two different configurations. The Lustre installation is globally shared amongst the nodes in the cluster and all test files were stored on the same shared volume. The HDFS variant 1 test was setup such that the HDFS chunk size was equivalent to the file size so that the file would not be split into chunks (and placed on various nodes) while the replication factor was set to equal the number of nodes in the cluster. This test represents an ideal case where all of the file is present on a given node that the MPI job could need it on. The variant 2 test used the HDFS defaults where the chunk size was set to 64 MB and the replication factor set to 3. This test represents the expected performance of HDFS without tuning and with blocks transfered over the network to reassemble the file for use by the MPI program. Finally, we plot the Max Theoretical Hard Drive Bandwidth to show the aggregate bandwidth possible from the given number of raw hard drives based on the maximum quoted data sheet bandwidth [18] for transferring data from the platter to the disk buffer. This sets the maximum upper threshold theoretically possible and allows for comparison of the efficiency of HDFS.

Two key observations can be made from these results that are important considering the nature of the application and its workload. First, it can be seen that both HDFS tests follow the same linear growth trend as the theoretical hard drive bandwidth but at a scaled rate. This linear growth in available bandwidth as the number of nodes increases indicates that HDFS could be used to scale I/O capacity on demand as larger datasets are presented to the visualization system. Second, it can be seen that Lustre, while able to produce considerable bandwidth, peaks in its ability to deliver data at 32 simultaneous clients and begins falling off as the node count continues to increase. This is due to Lustre having a finite network connection in which to serve data to the cluster that saturates and then hampers read performance once this saturation point has been reached. This behavior is not scalable as dataset size increases or more nodes are added to the cluster and limits Lustre's ability to deliver data on demand in a visualization style workload. Thus these results motivate our exploration of a VisIO enabled reader.

D. Testing of ParaView using VisIO Reader

1) Test Setup: Once it was proven that HDFS could provide the needed read bandwidth to sustain ParaView, the VisIO system was implemented per the discussion in Section III. Testing was conducted such that ParaView would read and process a complete time series for a given dataset and report the read times for every file opened for processing. A python batch script was written to setup the visualization environment and needed filters to create a reproducible test. The script then instructed ParaView to iterate through each of the time steps and produce a JPEG image of the rendered screens. This script was submitted to the ParaView server via the provided pvbatch utility to produce a test run on a given node count with the desired file system.

For testing, ParaView with the test script was run on 16, 32, 64, and 128 nodes on TACC's Longhorn cluster. 16 nodes was selected as the minimum number of nodes to test on due to the need for enough aggregate hard drive space to store the datasets and their replicas. The largest value, 128 nodes, was chosen as it was the largest number of nodes a single

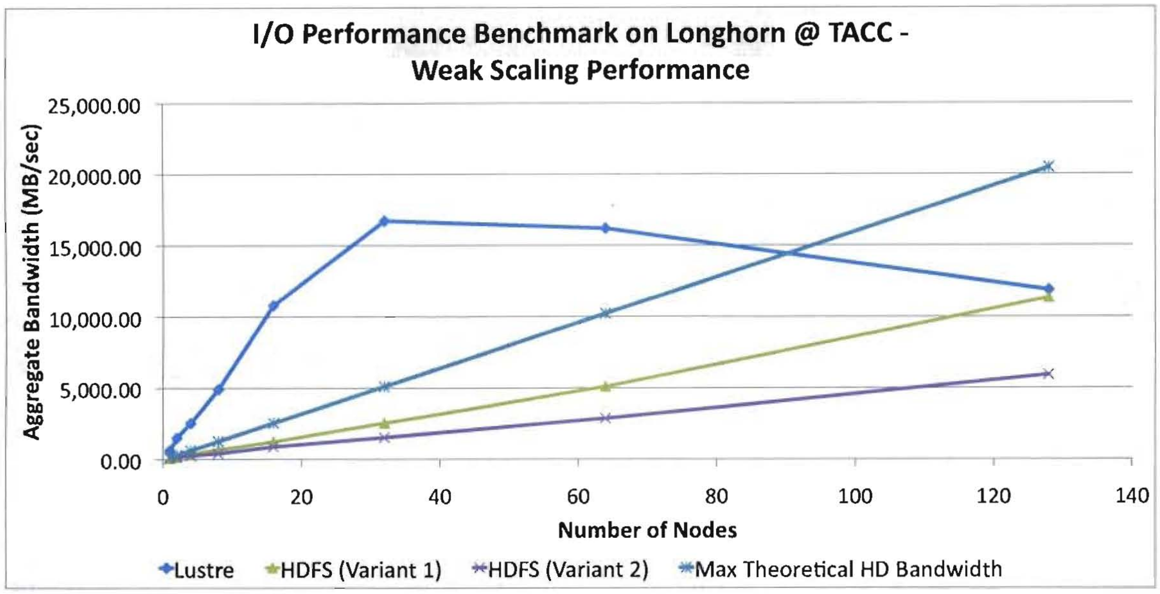

Fig. 4. Benchmark test of Lustre and two HDFS varients showing motivation for persuing a VisIO enabled reader for ParaView. Variant 1 tests performance when all files are local to the node requesting them while variant 2 represents the impact of chunking the files and replicating 3-ways thus causing network transfers between the DataNodes. Maximum theoretical hard drive bandwidth is also plotted for comparison to the HDFS results. The key observation from these results is that HDFS scales linearly for a visualization workload while Lustre peaks in its ability to deliver data and then tapers off in bandwidth as the number of clients increases.

user would be allowed to acquire in a job allocation on a regular basis without needing intervention from the system administrators.

The provided Lustre instance was directly accessed and only the number of clients accessing it was increased with each test set. HDFS was run such that the number of DataNodes was equal to the number of ParaView server nodes thus allowing the file system to scale with allocation size. Test results reported for both file systems represent the average read performance, per file, seen over each file opened for processing and across three retests for a given node count. In reporting average times, we can present a fairer picture of what the user will see over time when using the system rather than reporting maximum read times which show the worst case for a particular run as well as highlighting system noise induced variability.

HDFS was tested in two configurations: with and without the locality awareness algorithm activated. Each HDFS variant was configured slightly different to produce a fair test based on how the file system was going to be used. Testing of HDFS without the locality awareness was done by directly loading HDFS with the test data files using the defaults of a 64 MB chunk size and three way replication. The ".vtmb" index files were replicated by the number of nodes in the cluster so that each node could directly look up the metadata for the dataset to be visualized without waiting for a network transfer (per section III.C). For the HDFS test with locality, test files were loaded into the HDFS exactly as described in section III.C - using a chunk size equal to file size and three way replication.

2) Scalability of VisIO Reader: The central advantage of leveraging HDFS rather than a central parallel file system is the promised ability to scale as needed to accommodate larger I/O demands. Proving this called for a strong scaling test where a given real simulation dataset was processed by ParaView with increasing node counts from 16 to 128 nodes (in powers of two) using the VisIO based reader (with and without the locality algorithm). The results were compared to the baseline Lustre installation using the same node count. The VPIC dataset was chosen due to its size which permitted scaling down to 16 nodes, with the data set fitting into the HDFS, while also allowing scaling to 128 nodes with a non-trivial per node file size.

Similar to the weak scaled synthetic benchmark, previously discussed, the strong scaling test run with ParaView showed that the VisIO enabled reader was capable of continuously improving I/O performance as node count was increased. Shown in Figure 5, read times for the VPIC dataset exponentially decreased as the node count (and by extension hard drive count) was increased by a power of two. HDFS's performance, follows the same trend regardless of the reader's use of the locality algorithm, but as shown, the locality algorithm does shift the read times consistently downward with the improved read performance. In comparison, Lustre's read performance held approximately constant regardless of node count.

Thus, from this trending, we can reasonably expect that given more nodes, the VisIO enabled reader would be able to continue to linearly gain in bandwidth allowing for still faster reads of the given dataset or capability to read in still larger datasets with acceptable performance. Additionally, we

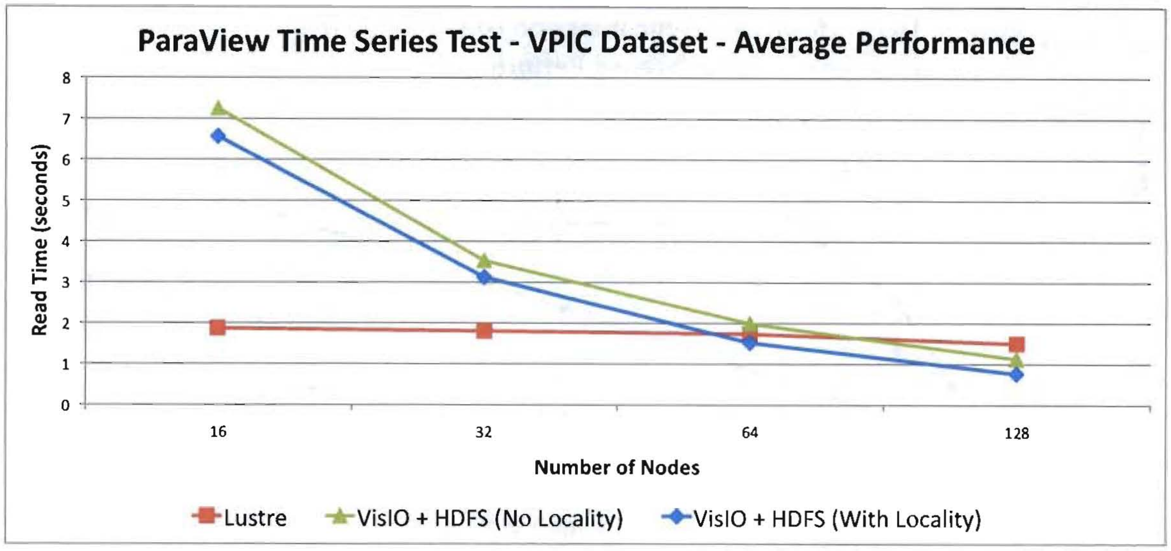

Fig. 5. Average read times as reported by ParaView for the VPIC time-series dataset. Tests run to compare performance on Lustre with the HDFS enabled reader both with and without the ability to sechedule MPI ranks based on file system locality information.

notice that without the locality algorithm, HDFS is able to surpass Lustre in read performance for this particular dataset at between 64 and 128 nodes. Using the locality algorithm, the read performance improves enough such that HDFS matches and outperforms Lustre at a node count just under 64.

3) Closer Look: Reader Performance at 128 Nodes: While Figure 5 clearly indicates the scaling trend expected from the VisIO based reader using HDFS compared to the standard version using Lustre, it does not clearly show the performance benefits to ParaView of using HDFS as compared to Lustre. Taking a closer look at the 128 node test runs, we see that Lustre is able to read a given file out of the files needed to construct the entire time series in 1.512 seconds while HDFS takes either 1.134 seconds or 0.776 seconds depending on if the locality algorithm is used or not. These differences in read times are illustrated in Figure 6. This translates into a 28.57% improvement in read performance if the VisIO enabled reader without locality is used compared to Lustre and a 64.38% improvement if the locality algorithm is used.

As a check, the ocean salinity data, was run as a second dataset at 128 nodes to see how it performed with the various readers. These results, also illustrated in Figure 6, showed a significant drop in read times when using HDFS. Lustre was able to read in the given data in 5.320 seconds per file while the HDFS reader was able to read in the given data at a rate of 3.480 seconds per file without the locality algorithm and 2.509 seconds with the locality algorithm. This represents a 41.82% and a 51.43% improvement respectively in per file read times.

4) Locality Algorithm Effectiveness: Use of a distributed file system by VisIO provides the ability to remove the bottleneck of a centralized parallel file system's network which is usually a fraction of the cluster interconnect fabric's capability. However, while this does provide a marked improvement in I/O performance, further improved times can be realized when

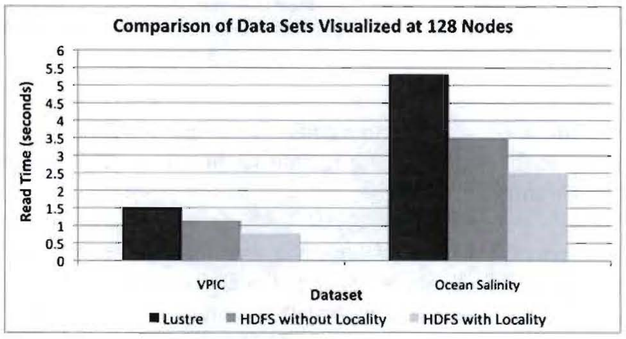

Fig. 6. Comparison of results collected on 128 nodes of TACC's Longhorn Visualization cluster. Two datasets were tested: VPIC and Ocean Salinity. These datasets were tested using the Lustre Parallel File System as well as using HDFS with its locality algorithm enabled.

processes are scheduled to nodes which locally contain the data needed, as discussed in Section III.B. Looking at the test data detailed above, we see that the VPIC dataset, for example, achieves a 35.81% improvement in using the locality aware VisIO based reader over the version that is not locality aware.

During testing, the number of requests for files by processes not able to access the data locally was tracked. This number was then converted to a ratio of the number of remotely accessed files to the total number of files read over all of the time steps. This ratio was used to track the effectiveness of the locality algorithm in its ability to schedule processes to nodes containing local copies of the data. This remote pull ratio calculated for the VPIC and ocean salinity datasets was 7.42% and 7.68% respectively. This relatively small proportion of remotely pulled files is a result of the HDFS's pseudo-random placement algorithm which presents a situation where the needed files for a given time step may not be evenly distributed such that across all nodes there is at least one

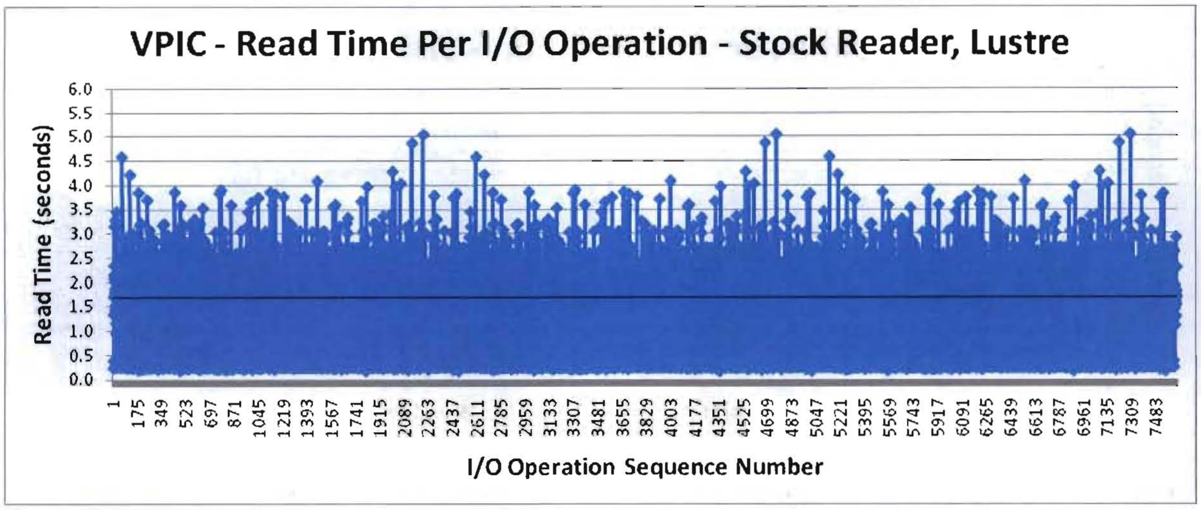

Fig. 7. Trace of time taken for each call to vtkFileSeriesReader with the stock reader in ParaView 3.8.0 and using Lustre as the file system. This represents a baseline of what ParaView currently experiences in terms of I/O read performance.

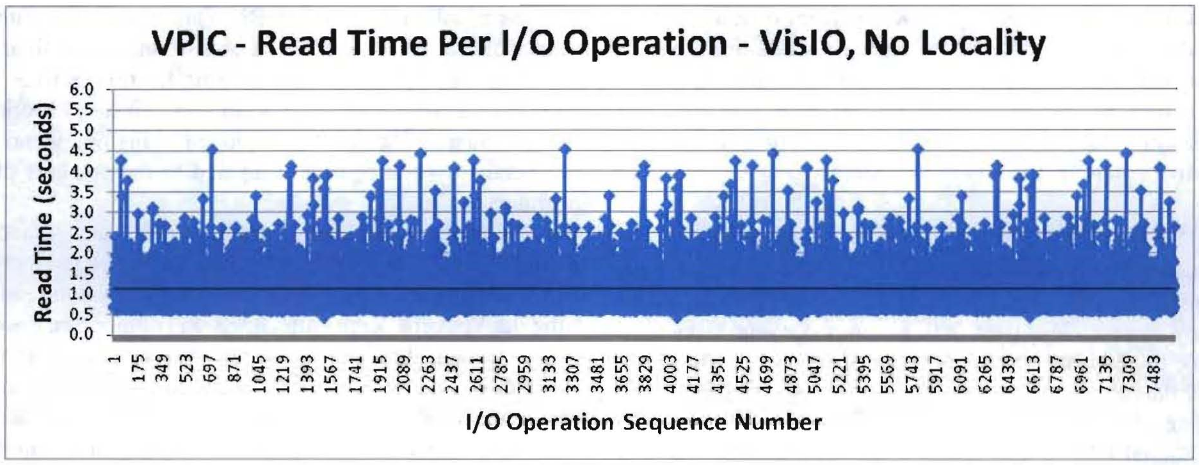

Fig. 8. Trace of time taken for each call to vtkFileSeriesReader with VisIO support embedded but without the locality algorithm in operation. The number of spikes in operation time and the wide spread of results around the trend line indicate wider variability in read times as a result of I/O not being contained to nodes that locally store the data needed.

unique file for that time step.

In addition to tracking the remote file request percentage, a trace of the request time for each call into vtkFileSeriesReader was captured and plotted in Figure 7 for the stock reader in ParaView 3.8.0 being used with Lustre. As shown, the variation in read times around the trend line is fairly wide which complements the computed standard deviation of 0.70 seconds. In addition, frequent bursts with read times several seconds higher than the mean are seen indicating high congestion to the parallel file system thus bottlenecking the application while the read request waits to return.

In comparison, Figure 8 plots our VisIO enabled reader but with the locality algorithm disabled. This shows strictly the benefit a distributed file system can provide compared to a parallel file system in terms of overall read times and the variability in the individual operations. A look at this trace shows that without the locality algorithm in operation, there are still many instances of spikes in read time that are significantly higher than the average but not as long in duration indicating improved bandwidth. While not as substantial as the peaks seen using Lustre, this still represents the longer times needed to transfer large portions of the time step over the network to the requesting nodes from the nodes storing the data. In addition, the standard deviation for this particular test run is 0.53 seconds showing the variability of the non-outlier read times is less than Lustre's standard deviation of 0.70 seconds.

In comparison to the plot in Figure 8, the plot shown in Figure 9 shows the same VisIO enabled reader but with the locality algorithm operational. In this case, it can be seen that the number of outliers has been reduced to a sporadic few indicating the minimization of longer running network reads.

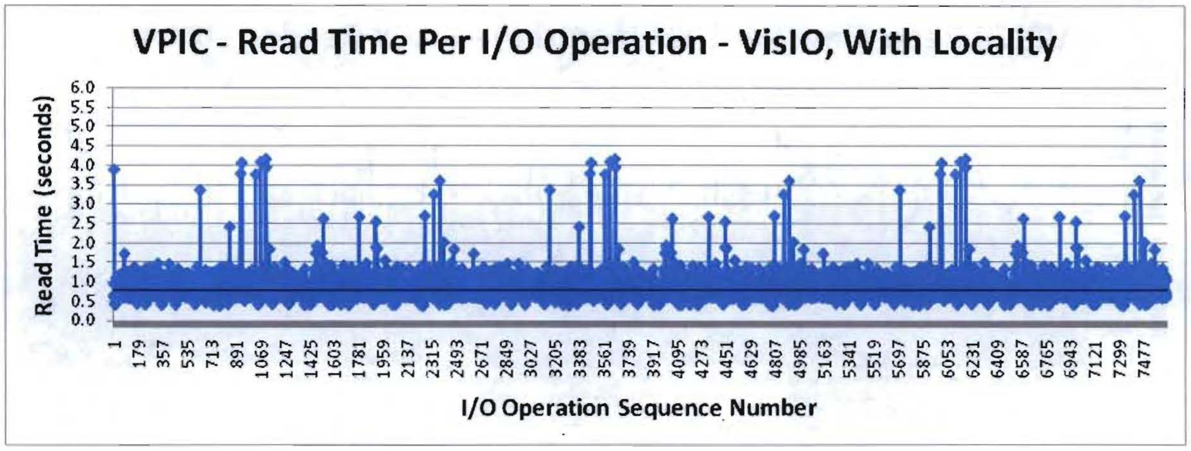

Fig. 9. Trace of time taken for each call to vtkFileSeriesReader with VisIO support enabled and using the locality algorithm. Compared to Figure 8, the number of spikes in read time are diminished and there is a tighter variability around the trend line when computation is kept predominantly to nodes containing local copies of the needed data.

This also corresponds with the lower percent of remote pulls detailed above. Additionally, the standard deviation drops to 0.28 seconds from 0.53 seconds without the locality algorithm. This in turn will yield more consistent read times for the visualization application; a preferable condition due to the interactive nature of the application's use.

E. Multi User Environments

While not common in traditional HPC environments, a multi user environment may be a possible deployment path for a data intensive compatible cluster. In this setup, nodes run tasks from multiple users either simultaneously or in a context switching manner thus allowing multiple user's jobs to run and access the distributed file system simultaneously. This is in stark contrast to traditional HPC setups where a user is the sole user of a subset of nodes in the cluster for the duration of the job.

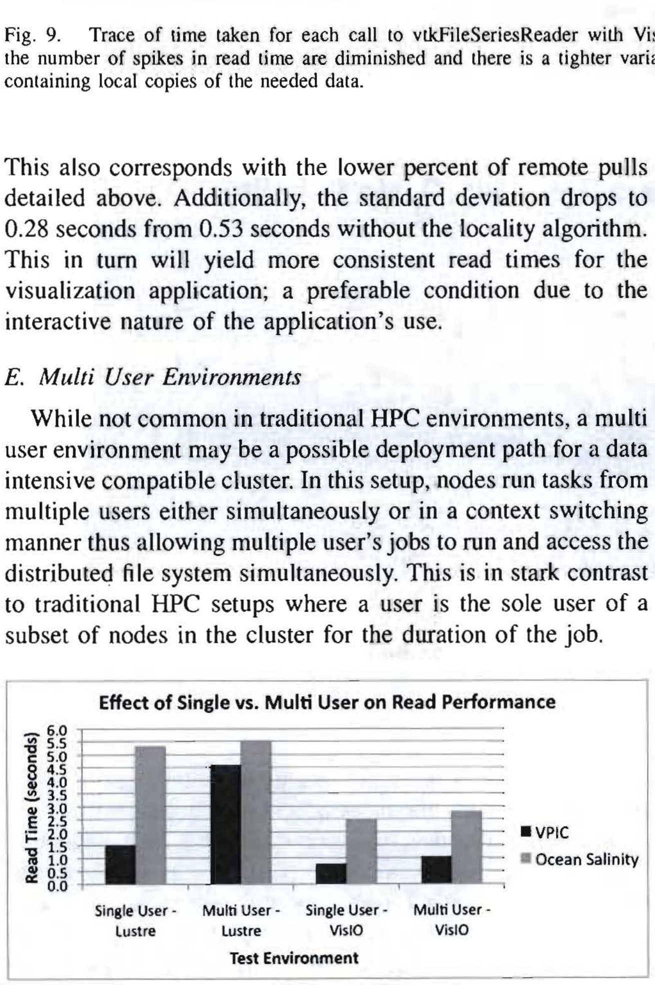

Fig. 10. Comparison of single and multi user test runs. Single user runs had the VPIC and Ocean Salinity data run independently of each other while the multi user runs had both run together. As VPIC is the shorter running job, due to having fewer time steps, the impact of multiple jobs running is seen more prominently in its runs.

Thus, to see how VisIO and HDFS would handle in an environment where it is not the sole user, we devised a test where two instances of ParaView would be launched at the same time and asked to load two different datasets (VPIC and ocean salinity) from HDFS. This would simulate two users working on two problem sets at the same time. The same test was also performed against Lustre with the stock ParaView reader to show how similar increased workloads would compare. The results are plotted in Figure 10 and show the average over three runs compared to the numbers plotted in Figure 6.

It should be noted that there is a 3.08 second increase in time to read the VPIC set from Lustre under increased load while we see only a 0.275 second increase in read time using the VisIO package with HDFS. In comparison, the ocean salinity visualization showed increases in read time but not as substantial. Lustre showed a 0.221 second increase in read time while VisIO with HDFS showed a 0.293 second increase from their single user values. The relatively minor increases in the ocean salinity average read times can be attributed to the fact that the ocean salinity dataset runs for a longer time (12 time steps longer) than the VPIC dataset this giving the tail end of the test similar performance characteristics as the single user tests thus dampening the effect of having multiple processes running.

Finally, we noted the changes in the standard deviations of the runs' individual I/O operation times and found that between the multi user runs with Lustre and VisIO a standard deviation of 2.298 seconds and 0.690 seconds respectively were seen for VPIC. Ocean salinity showed a Lustre standard deviation of 1.880 seconds while the VisIO standard deviation was 1.595 seconds. Particularly illustrated with the VPIC runs, it can be seen that VisIO manages to keep variability in individual read times closer to constant than the standard reader was able to using Lustre especially under increased loads present in a multi user environment.

V. Related Works

Several visualization packages exist for scientific data analysis and are in widespread use within the High Performance Computing (HPC) arena. VTK [21] is a framework that is used to abstract away the OpenGL calls needed to display graphics thus allowing the scientist to focus on their data rather than the system. Parallel VTK [22] and its front-end, ParaView [23] are higher level implementations of the VTK framework which allow the user to run VTK jobs in parallel as well as to build and control VTK programs from an easy to use GUI, respectively. VisIt [24], like ParaView, is a GUI front-end to VTK that allows the user to control the visualization process without needing detailed, hands-on programming. EnSight [25] is an alternate, commercially developed, tool and environment that supports scientific visualization at scale.

Visualization at scale, however, presents its own set of challenges. Childs, *et. al.* [26] detail how they needed to apply a subset of possible problem specific optimizations dynamically to the VisIt pipeline to allow it to perform acceptably at scale. These optimizations are specified in the notion of a contract which is passed along the pipeline detailing what optimizations are needed from the end of the pipeline back to the beginning. These optimizations include such ideas as minimizing disk reads and details on how to manipulate the data within the pipeline. Childs, *et. al.* [6] also detail the performance bottlenecks experienced when visualizing at scale. This work, as discussed earlier, showed that I/O times dwarfed the time needed to run the visualization algorithms (isosurfacing in this case) and render the final image. This result is the motivating force behind our VisIO solution to rework I/O within the visualization pipeline.

Considering the demand for visualization at scale and evidence of an I/O bottleneck, work has been done to alleviate the problems associated with large amounts of data. Parallel File Systems have been made available that are capable of allowing multiple nodes access to the same file or subset of files at significant bandwidths. The most prominent of these file systems include: PVFS2 [27], Panasas [28], Lustre [29], and GPFS [30]. However, even with the success of these file systems, improvements have needed to be proposed and implemented to accommodate common patterns seen in HPC workloads, particularly N-to-1. Carns, *et. al.* [12] detail five techniques to help handle small file accesses within a parallel file system (PVFS2 in this case). Small file access poses a performance problem for parallel file systems which are designed to best handle large I/O operations. Similarly, Thakur, *et. al.* [13] detail a method for handling noncontiguous I/O requests from a single process and multiple processes within a cluster to the same file (N-to-1 access pattern). For the single process case, data sieving is used to have a process read in a large chunk of a file and filter out of various smaller parts needed. For the multiple process case, collective or two-phase I/O calls for each process to read a contiguous region of the file and then use inter-process communication to redistribute the information read to the requesting process. While these techniques work well for HPC simulations, a visualization application working with a N-to-N pattern and without MPI-IO support will not be able to benefit.

While these techniques are general in nature, some work has been done to specifically address I/O performance within scientific visualization workflows. Yu, *et. al.* [31], discuss a method of using input processors to handle the data fetching from the file system using optimized MPI-IO routines. Their strategy, while showing alleviation of the I/O bottleneck, requires dedicated nodes to be set aside for the task of acting as input processors and the number required grows proportional to data set size. Perterka and Ross, *et. al.* [7], [8] explored running volume rendering applications at scale directly on their BlueGene/P system rather than on a dedicated visualization system. Their work shows that using MPI-IO operations as well as reorganization of the simulation results within the file assisted in providing needed I/O performance for large datasets. Their technique, however, relies on the presence of a parallel file system and interconnect network that is sufficiently fast as to not bottleneck the visualization workflow.

Complementing this work on parallel file systems, distributed file systems were developed to address the issues of scaled out datacenters with large data volumes. The seminal distributed file system currently is the Google File System (GFS) [32] used to run almost all of Google's internal infrastructure. The GFS calls for hard drives to be locally installed in their servers rather than using a centralized file system to allow for the co-location of computation on nodes where the needed data is stored. Since GFS is a propietary system, there exists a couple of open source implementations which strive to replicate GFS's functionality. The most mature of which is Hadoop's Distributed File System (HDFS) [10] and upon which our proposed improvements are based. Also available, are the CloudStore [33] and Ceph [34] distributed file systems. Finally, D.E. Shaw Research recently proposed a new system called Zazen [35] which migrates data from compute resources to caches on an analysis cluster to allow for local access to data for post-processing. While similar in spirit to our employed method, Zazen requires specially built analysis applications which integrate with the cache system to determine delegation of tasks to nodes where the data locally resides in a cache. In contrast, our VisIO system allows a general purpose visualization application to leverage the data locality provided by the HDFS (and without the limitations of a cache) with just changes to the data reader code which is a modular component to be replaced as needed.

VI. Conclusion

In this paper we have proposed and developed an I/O system that is optimized for handling scientific visualization applications working with ultra-scale datasets. Our system, VisIO, allows traditionally MPI and POSIX based visualization applications to leverage the increased bandwidth possible from a distributed file system. In addition, we further stregthen the ability of these applications to benefit from a DFS by providing a data locality aware scheduling algorithm that is used to schedule individual process ranks on nodes that contain the needed data for the operation to be performed.

A VisIO enabled ParaView reader was put into operation on TACC's Longhorn visualization cluster and used with the

Hadoop Distributed File System. Testing was conducted on data from the VPIC plasma physics simulation, and the ocean salinity simulation. Results showed that use of HDFS, if allowed to scale with the allocated node count, will linearly improve in read bandwidth available to the application and can exponentially decrease the amount of time needed to read a data file by a given MPI rank. Compared to the provided, statically-provisioned, Lustre parallel file system, HDFS is capable of dynamically allocating storage resources (as easily as CPU resources are allocated for a compute-bound job) and given enough nodes can over take Lustre in read performance. Testing showed a between 50 and 65% read performance improvement amongst the datasets when read via our VisIO enabled reader.

Overall, this system has proven that it is a possible path forward for the ever increasing demands being placed on scientific analysis visualization applications which are being constantly challenged to interactively deliver insight to current Petascale and future Exascale simulations and experiments.

ACKNOWLEDGMENT

The authors would like to thank John Patchett, Joshua Wu, and Patricia Fasel of Los Alamos National Laboratory. We would also like to thank Berk Geveci, Utkarsh Ayachit, and David DeMarle of Kitware, Inc. Additionally, we thank William Daughton for access to his VPIC datasets, and the COSIM team for the ocean salinity dataset. Finally, we would like to thank the NSF for sponsoring this work under grants CCF-0811413 and CAREER CCF-0953946. Additionally, we thank the US Department of Energy for sponsoring us under Early Career Principal Investigator Award: DE-FG02-07ER25747. The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing visualization resources that have contributed to the research results reported within this paper. URL: http://www.tacc.utexas.edu.

REFERENCES

[1] G. S. Jones, "Fusion gets faster," August 2009. [Online]. Available: http://www.nccs.gov/2009/08/17/fusion-gets-faster/

[2] S. Habib, A. Pope, Z. Lukic, D. Daniel, P. Fasel, N. Desai, K. Heitmann, C.-H. Hsu, L. Ankeny, G. Mark, S. Bhattacharya, and J. Ahrens, "Hybrid petacomputing meets cosmology: The roadrunner universe project," *Journal of Physics: Conference Series*, vol. 180, p. 012019 (10pp), 2009. [Online]. Available: http://stacks.iop.org/1742-6596/180/012019

[3] M. C. Schatz, B. Langmead, and S. L. Salzberg, "Cloud computing and the dna data race," *Nat Biotech*, vol. 28, no. 7, pp. 691–693, 07 2010. [Online]. Available: http://dx.doi.org/10.1038/nbt0710-691

[4] G. Lo Presti, O. Barring, A. Earl, R. Garcia Rioja, S. Ponce, G. Taurelli, D. Waldron, and M. Coelho Dos Santos, "Castor: A distributed storage resource facility for high performance data processing at cern," sep. 2007, pp. 275 –280.

[5] A. Larzelere, "Delivering insight: The history of the accelerated strategic computing initiative (asci)," Lawrence Libermore National Laboratory, Tech. Rep., 2009. [Online]. Available: https://asc.llnl.gov/asc_history/Delivering_Insight_ASCI.pdf

[6] H. Childs, D. Pugmire, S. Ahern, B. Whitlock, M. Howison, Prabhat, G. Weber, and E. Bethel, "Extreme scaling of production visualization software on diverse architectures," *Computer Graphics and Applications, IEEE*, vol. 30, no. 3, pp. 22 –31, may. 2010.

[7] T. Peterka, H. Yu, R. Ross, K.-L. Ma, and R. Latham, "End-to-end study of parallel volume rendering on the ibm blue gene/p," sep. 2009, pp. 566 –573.

[8] R. B. Ross, T. Peterka, H.-W. Shen, Y. Hong, K.-L. Ma, H. Yu, and K. Moreland, "Visualization and parallel i/o at extreme scale," *Journal of Physics: Conference Series*, vol. 125, no. 1, p. 012099, 2008. [Online]. Available: http://stacks.iop.org/1742-6596/125/i=1/a=012099

[9] R. Ross, "Parallel i/o and computational parallel i/o and computational science at the largest scales," November 2009, slides from presentation.

[10] "Apache hadoop project," March 2010. [Online]. Available: http://hadoop.apache.org/

[11] G. Grider, H. Chen, J. Nunez, S. Poole, R. Wacha, P. Fields, R. Martinez, P. Martinez, S. Khalsa, A. Matthews, and G. Gibson, "Pascal - a new parallel and scalable server io networking infrastructure for supporting global storage/file systems in large-size linux clusters," april 2006, pp. 10 pp. –340.

[12] P. Carns, S. Lang, R. Ross, M. Vilayannur, J. Kunkel, and T. Ludwig, "Small-file access in parallel file systems," may 2009, pp. 1 –11.

[13] R. Thakur, W. Gropp, and E. Lusk, "Data sieving and collective i/o in romio," feb 1999, pp. 182 –189.

[14] B. Shneiderman, "Response time and display rate in human performance with computers," *ACM Comput. Surv.*, vol. 16, no. 3, pp. 265–285, 1984.

[15] J. A. Hoxmeier and C. Dicesare, "System response time and user satisfaction: An experimental study of browser-based applications," in *Proceedings of the Association of Information Systems Americas Conference*, 2000, pp. 10–13.

[16] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. pp. 9–15, 1962. [Online]. Available: http://www.jstor.org/stable/2312726

[17] "Texas advanced computing center," April 2010. [Online]. Available: http://www.tacc.utexas.edu

[18] "Seagate savvio 15k.2 data sheet," September 2010. [Online]. Available: http://www.seagate.com/docs/pdf/datasheet/disc/ds_savvio_15k_2.pdf

[19] K. J. Bowers, B. J. Albright, L. Yin, W. Daughton, V. Roytershteyn, B. Bergen, and T. J. T. Kwan, "Advances in petascale kinetic plasma simulation with vpic and roadrunner," *Journal of Physics: Conference Series*, vol. 180, no. 1, p. 012055, 2009. [Online]. Available: http://stacks.iop.org/1742-6596/180/i=1/a=012055

[20] "Los alamos national laboratory climate, ocean, and sea ice modeling project," September 2010. [Online]. Available: http://climate.lanl.gov/

[21] "The visualization toolkit (vtk)," March 2010. [Online]. Available: http://www.vtk.org

[22] J. Ahrens, C. Law, W. Schroeder, K. Martin, and M. Papka, "A parallel approach for efficiently visualizing extremely large, time-varying datasets," Los Alamos National Laboratory, http://www.vtk.org/VTK/img/pvtk.pdf, Tech. Rep., 2000.

[23] "Paraview," March 2010. [Online]. Available: http://www.paraview.org

[24] "Visit visualization tool," March 2010. [Online]. Available: www.llnl.gov/VisIt/

[25] "Cei - ensight visualization software," March 2010. [Online]. Available: http://www.ensight.com/

[26] H. Childs, E. Brugger, K. Bonnell, J. Meredith, M. Miller, B. Whitlock, and N. Max, "A contract based system for large data visualization," oct. 2005, pp. 191 – 198.

[27] "Parallel virtual file system 2 (pvfs2)," March 2010. [Online]. Available: http://www.pvfs.org/

[28] "Panasas," March 2010. [Online]. Available: http://www.panasas.com/

[29] "Lustre file system," March 2010. [Online]. Available: http://sun.com/lustre

[30] "General parallel file system (gpfs)," March 2010. [Online]. Available: http://www.almaden.ibm.com/StorageSystems/projects/gpfs/

[31] H. Yu and K.-L. Ma, "A study of i/o techniques for parallel visualization," *Journal of Parallel Computing*, vol. 31, no. 2, 2005.

[32] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 29–43, 2003. [Online]. Available: http://labs.google.com/papers/gfs-sosp2003.pdf

[33] "Cloudstore - high performance scalable storage," March 2010. [Online]. Available: http://kosmosfs.sourceforge.net/

[34] "Ceph - petabyte scale storage," March 2010. [Online]. Available: http://ceph.newdream.net/

[35] T. Tu, C. A. Rendleman, P. J. Miller, F. Sacerdoti, R. O. Dror, and D. E. Shaw, "Accelerating parallel analysis of scientific simulation data via zazen," in *Usenix File and Storage Technologies (FAST) 2010*, 2010.