Support for Data-Intensive, Variable-Granularity Grid Applications via Distributed File System Virtualization – A Case Study of Light Scattering Spectroscopy

Jithendar Paladugula
ACIS Laboratory
University of Florida
jithenda@acis.ufl.edu

Ming Zhao
ACIS Laboratory
University of Florida
ming@acis.ufl.edu

Renato J. Figueiredo
ACIS Laboratory
University of Florida
renato@acis.ufl.edu

Abstract

A key challenge faced by large-scale, distributed applications in Grid environments is efficient, seamless data management. In particular, for applications that can benefit from access to data at variable granularities, data management can pose additional programming burdens to an application developer. This paper presents a case for the use of virtualized distributed file systems as a basis for data management for data-intensive, variable-granularity applications. The approach leverages on-demand transfer mechanisms of existing, de-facto network file system clients and servers that support transfers of partial data sets in an application-transparent fashion, and complement them with user-level performance and functionality enhancements such as caching and encrypted communication channels. The paper uses a nascent application from the medical imaging field (Light Scattering Spectroscopy – LSS) as a motivation for the approach, and as a basis for evaluating its performance. Results from performance experiments that consider the 16-processor parallel execution of LSS analysis and database generation programs show that, in the presence of data locality, a virtualized wide-area distributed file system setup and configured by Grid middleware can achieve performance levels close (13% overhead or less) to that of a local disk, and superior (up to 680% speedup) to non-virtualized distributed file systems.

1. Introduction

A fundamental challenge faced by large-scale applications distributed across computational "Grids" that span multiple administrative domains is data management. There are important data-intensive applications that can benefit from the availability of distributed, computational "Grids", and require not only high-performance computing resources, but also seamless, high-performance access to distributed data. In several instances, data-intensive applications benefit from the capability of operating on their data sets at different granularities – for example, by sampling down a dataset for a coarse-grain first-order analysis, and considering the entire dataset for high-resolution analyses. This paper presents a case for the use of virtualized, Grid-wide distributed file systems [1] to support data-intensive, variable granularity applications. The analysis is based on experiments conducted based on an emerging medical imaging application that aims at quantitative, non-invasive analysis of tissue for the detection of pre-cancerous lesions – Light Scattering Spectroscopy (LSS) [2][3][4].

The approach presented in this paper provides seamless access to data, facilitates application development by using file system abstractions and implementations available in existing O/Ss, and provides efficient support for variable-granularity programs by leveraging on-demand and application-transparent transfer of data using NFS (Network File System). It addresses limitations of conventional approaches to data management in Grid environments: these typically rely on mechanisms for data transfer that either require explicit naming of files to be transferred by users and/or applications (GASS [12], GridFTP [18]) or special libraries linked to applications to support remote I/O [14]. The approach also addresses limitations of conventional distributed file systems: non-virtualized native NFS implementations that rely on single-domain, local-area network environments for authentication and user identification [19] are not well-suited for a cross-domain Grid deployment; wide-area implementations that rely on whole-file transfers (e.g. AFS [20], CODA [21]) are not widely deployed, and do not support on-demand partial transfers in accesses to file data at different granularities. These limitations are addressed via user-level, middleware controlled distributed file system proxies that intercept, modify and forward NFS remote procedure calls. File system proxies enable Grid-oriented extensions implemented completely at the user level, without requiring any kernel-level modifications to existing O/Ss; these extensions include Grid-controlled identity mapping for cross-domain authentication, locality enhancements via per-application disk-based caching policies, and data communication privacy via per-session encrypted channels.

The applicability of the virtual distributed file system approach to data-intensive, variable-granularity applications is considered in the case study of a representative, nascent medical imaging application. Nonetheless, the approach is not particular to this case, and can be applied to arbitrary, unmodified applications that access file-system based data. The LSS analysis application is based on the processing of the backscattered spectral image of a region of tissue using a Mie-theory based inversion procedure. For each pixel in the image, the size and refractive indices of the scatterers that best fit the data among those stored in a database of Mie-theory generated spectra are found using a least-square error minimization approach. For high-accuracy analysis, large databases with fine-grain high resolution of sizes and refractive indices must be considered. Low-accuracy, coarse-grain analysis can be achieved with smaller (or sampled-down) databases, thereby reducing both the data transfer and computational requirements of the application.

Performance data reported in this paper shows that user-level proxy disk caches allow the proposed approach to achieve performance close to that of local disk (13% slower) and superior to non-virtualized NFS setups in both LAN (83% speedup) and WAN (680% speedup) environments for a 16-processor parallel execution of LSS analysis. Results also show that the implementation of a user-level write-back cache policy allows the parallel generation of LSS databases to achieve performance close to that of local disk (3% slower) and non-virtualized LAN (2% faster), and superior to non-virtualized WAN (280% speedup).

This paper is organized as follows. Section 2 describes the LSS application considered in the case study: motivations, algorithm and a parallel implementation based on MPI and file I/O. Section 3 summarizes distributed virtual file system motivations and the implementation used in the case study, and Section 4 presents results and analysis of experiments that consider the performance of LSS on top of such file system. Section 5 discusses related work, and Section 6 concludes the paper.

2. Application Background

In the past few decades, high-performance computing has driven the development of practical medical applications that are now widely available, such as magnetic resonance imaging and computerized tomography. These solutions have been enabled by sustained performance improvements in the embedded systems that typically support medical applications. However, there are important emerging medical applications for which effective deployments will depend on the availability of high levels of performance that cannot be delivered by an embedded system – thus requiring access to high-performance remote resources. In recent years, information processing is undergoing rapid advances driven by the use of distributed computing systems connected by world-wide networks. Analogous to power grids, "computational grids" have the potential to provide seamless access to high-performance resources (e.g. parallel supercomputers) from ubiquitous, network-enabled devices. The unprecedent levels of computation enabled by this model may foster the development of new medical applications that can improve healthcare and find wide-spread applications in medical facilities of the future.

2.1. Light Scattering Spectroscopy

LSS is a nascent technique that enables the extraction of quantitative information about the structure of living cells and tissues via the analysis of the spectrum of light backscattered from tissues [2][3][4]. The information obtained via LSS analysis consists of the size distribution and refractive index of a spectral image obtained from an area of interest (e.g. skin, colon, oral cavity) by means of a special-purpose optical apparatus.

LSS devices, and the associated analysis techniques that allow non-invasive detection of pre-cancerous changes in human epithelium, have been recently proposed and investigated [3][4]. LSS imaging differentiates from traditional biopsies by allowing in-vivo diagnosis of tissue samples, and by providing an automated, quantitative analyses of parameters related to cancerous changes (e.g. nuclei enlargement) via numerical techniques. As a result, LSS can yield significant advances in healthcare: it has the potential to detect the majority of currently undetectable cancers and significantly reduce cancer mortality (by up to 50%) [4]. However, to achieve high accuracy, the analysis of LSS images requires compute- and data-intensive solutions that perform spectral analyses based on Mie theory inversion procedures.

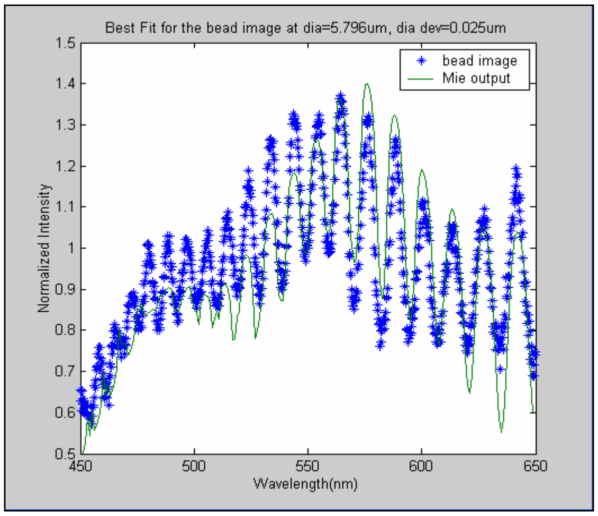

Figure 1. Spectral image obtained from polystyrene beads (diameter=5.8um, stdev = 0.02um) suspended in water, and least-square error fit (diameter =5.796um, stdev = 0.025um)

The LSS technique is based on an optical apparatus that includes a digital image CCD that records spectra of backscattered light for both parallel and perpendicular polarizations [4]. The backscattered spectrum is analyzed based on Mie theory of light scattering by spherical particles of arbitrary size, which enables prediction of the major spectral variations of light scattered by cell nuclei [2]. There are, however, no known analytical closed-form general solutions for inverse Mie functions; LSS analysis requires on the use of Mie theory to generate a database of LSS spectra over a representative range of mean diameters, standard deviations and average relative refractive indices. For each pixel in the image, the size and refractive indices of the scatterers that best fit the data among those stored in the database are found using a least-square error minimization approach. Therefore, the algorithm for quantitative LSS analysis requires two inputs: the spectrum of backscattered light (obtained from an LSS instrument), and one or more databases (obtained from the application of Mie equations across a desired range of diameters and refractive indices).

2.2. Grid-based LSS Analysis

In current medical applications (e.g. computerized tomography) the physical area covered by the imaging device is large, thus requiring a large and expensive apparatus. Cost and area constraints limit the deployment of such devices to a few units in a medical facility. In addition, since the cost of the imaging device is high, a costly high-performance computing unit attached to the device is justifiable. In contrast, the physical area of tissue analyzed by an LSS imaging apparatus is typically small – of the order of square centimeters. LSS imaging can therefore be performed with smaller, portable devices deployed in larger numbers across medical facilities. In this scenario, the use of a costly high-performance computer attached to the imaging devices is no longer attractive. However, it is important to perform high-performance computation to obtain a quantitative analysis of LSS images in quasi-real-time, allowing feedback to clinicians while the patient is under examination.

Previous efforts in the coupling of instrumentation and distributed infrastructures have considered image-processing applications – such as parallel analysis of data collected from electron microscopes [5][6][7]. Results from previous work motivate the use of network-based computing to solve applications with similar characteristics – computational-intensive and with high degrees of parallelism – including medical applications [8]. LSS imaging is an application that can greatly benefit from a network-computing model, given its substantial computational requirements and amenability to parallelism. This application exhibits tradeoffs between computation time and accuracy: ideally, LSS imaging should be performed in quasi real-time to allow clinical feedback while patients are under examination; however, a detailed analysis may be too expensive – in terms of response time or the cost of utilizing remote resources – to be performed in all cases. A variable-grain solution that seamlessly supports multiple execution models – short response time (and possibly low accuracy), and high accuracy (at the expense of large response time) – is therefore desirable.

2.3. LSS Implementation

The programming approach for LSS applications that implement both Mie database generation and least-square fitting analysis is built on top of a file system. This has allowed for simplicity in the design, reuse of O/S distributed file system implementations, and ease of integration with Grid middleware and Web-based problem-solving environments capable of virtualization [22]. In the resulting system, LSS applications and problem solving middleware are integrated in a way that allows users to (from a web browser) upload or request the generation of LSS databases; upload images; request the execution of LSS analyses; and download output images. The underlying Grid middleware supports user-transparent resource allocation and dynamic setup of virtual file system sessions. The web-based LSS Grid environment is available in the main In-VIGO portal at the University of Florida (http://invigo.acis.ufl.edu); courtesy accounts are available.

2.3.1. Database Generation. The LSS analysis relies on Mie theory inversion to determine the spectral variations of the light scattered by cell nuclei. Closed-form solutions to this problem are not available; hence, the inversion procedure is implemented numerically through least-square minimization against a database of spectra generated using the Mie function. The diameter, diameter deviation and relative refractive index are used as inputs for the generation of the lookup database (Figure 2); values chosen for these parameters need to be derived from the expected ranges of sizes and refractive indices of cells under investigation. The Mie function returns the scattering intensity (ΔI) as a function of wavelength and scattering angle.

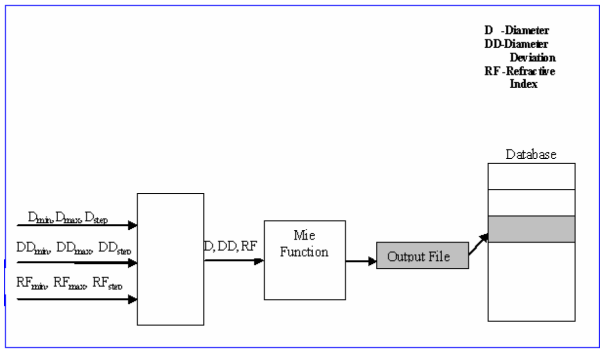

Figure 2. LSS database generation. A range of diameters, diameter deviations and refractive indices are provided as inputs to a program that coordinates the execution of Mie function module and construct database records for each data point.

The implementation of the program that generates LSS databases takes minimum, maximum and step values for diameter, diameter deviation and refractive index and generates a sequence of input files. For each input, it runs a separate executable that calculates Mie function spectra and writes results to output files. The Mie function output file is averaged across scattering angles, normalized, and appended as a record to the database file. Data parallelism can naturally be exploited by using different files for independent databases.

2.3.2. Least-square error minimization. The LSS analysis program uses MPI for coordination and for determining the global minimum from independently computed local minima, while file I/O is used by each MPI process to independently access its databases. As the fit for each database can be processed independent of each other, the program is parallelized across the database files in the directory. A master-slave strategy as shown in Figure 3 is used to parallelize the program. The master receives the input directory name and counts the number of database files in the directory, then assigns files to each processor to balance their load. Each processor calculates the fit for its own set of databases, given by number of files/number of processors. The processors send the local least square error and the corresponding diameter, diameter deviation and refractive index to the master processor in the form of an array. The master receives the local least square errors from each processor and in turn calculates the global least value among the errors received and finally returns the corresponding diameter, diameter deviation and the refractive index for the image.

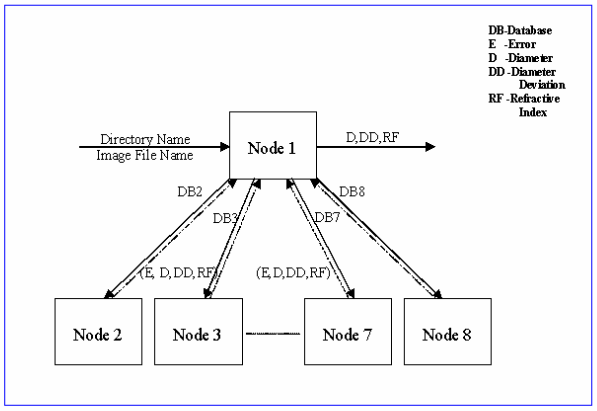

Figure 3. LSS parallelization across database records. Node 1 is the master. The master node determines which databases are assigned to each node; the actual databases are accessed independently from each node's file system.

The use of multiple independent lookup databases that are accessed through a conventional file system interface allows for 1) the partition of large datasets across multiple nodes, 2) the seamless execution of the program in conventional local-area and cluster-based environments used for MPI-based parallel executions, and 3) seamless integration with distributed Grid environments that are built on top of virtual file systems [1][22]. These issues are addressed in Section 3.

3. Distributed File System Virtualization

Previous work on the PUNCH distributed virtual file system [1] has considered a virtualization layer on top of NFS to allow data to be transferred on-demand between storage and compute servers for the duration of a computing session. This functionality is realized via user-level extensions to existing NFS implementations that allow reuse of unmodified clients and servers of conventional operating systems; implementations use middleware-controlled proxies to map identities between dynamically-allocated logical accounts and transparently broker a user's access to files across administrative domains. It leverages NFS implementations and does not require any modifications to either operation systems or applications. Furthermore, data transfer in the distributed virtual file system is on demand and transparent to the user. This behavior is inherited from the underlying NFS protocol, which allows for partial transfer of files on a block-by-block basis (typically 4K to 32Kbytes in size). This property is important when supporting applications that access large files, but not necessarily in their entirety – for example, accesses to the virtual disk of a "classic" VM are typically restricted to a working set that is considerably smaller (<10%) than the large virtual disk file [23][24].

In addition to supporting on-demand transfers and dynamic identity mappings, middleware-controlled proxies support performance and functionality extensions that make the virtual file system suitable for wide-area, Grid applications. Large read/write disk-based user-level caches can be employed on a per-application basis to complement typical kernel memory buffers; write policies can be determined (e.g. write-back vs. write-through) also on a per-application basis; privacy and session-key authentication can be established without requiring modifications to underlying NFS implementations via the use of SSH tunnels [25]. The resulting solution, with enhancements at user-level while preserving unmodified O/S file system abstractions, is applicable to a wide variety of applications – even commercial, binary legacy codes for which there is no possibility for a user to alter source code and/or re-link to specialized libraries. In particular, it is well-suited for variable-granularity applications. The next section presents an analysis of its performance for the LSS application described in Section 2.

4. Performance Analysis

This section summarizes an analysis of the performance of LSS in a distributed virtual file system environment. Subsection 4.1 describes the experimental setup, while Subsection 4.2 presents results and analyses.

4.1. Experimental Setup

The experiments have been performed on a 32-node Linux-based cluster. Each physical node is configured as follows: 2.4GHz Pentium-4, 1.5GB RAM, 18GB disk, Gbit/s Ethernet, Red Hat Linux 7.3. Experiments have been conducted in a virtual-machine based Grid [23] with VMware GSX 2.5 VMs (256MB memory, 4GB virtual disk, Red Hat Linux 7.3). The implementation of MPI for the cluster is based on LAM/MPI 6.5.9. The directory containing database files is mounted via a virtual file system [1] on all nodes. The LAN file server is a dual Pentium-3 1.8GHz server with 1GB RAM and 8-disk/500GB SCSI RAID5 array. The WAN file server is a dual Pentium-3 1GHz server with 1GB RAM and 46GB IDE RAID0 array. Experiments consider both local-area and wide-area virtual file systems. The WAN experiments are based on connections between University of Florida and Northwestern University through Abilene. The NFS traffic in both local area and wide area environments is tunneled through an SSH based private virtual file system. The proxy cache uses NFS version 2 with 8KB buffer size. The cache at the client side is configured with 1GB size, 512 file banks which are 16-way associative.

A database of LSS spectra is generated over a range of diameters (5.65 to 5.97um in steps of 0.0005um), diameter deviations (0.005 to 2.5um in steps of 0.005um) and constant refractive index (0). This results in a database with 320000(640*500*1) records approximately. The Mie function takes on average 20s to compute one record (the sequential generation of the full database would take approximately 74 days). As each record is independent of each other, the database can be generated in parallel. Using 32 parallel processes, the generation of this database takes 3 days, resulting in 32 data files and 1.9GB disk space. With respect to the location of the databases, the following scenarios have been considered:

Local: The databases are stored in a local-disk file system.

LAN: The databases are stored in a directory NFS-mounted from a LAN server.

WAN: The databases are stored in a directory NFS-mounted from a WAN server. File system proxies are used to forward RPC calls.

WAN+C: The databases are stored in a directory NFS-mounted from a WAN server. File systems proxies are used to forward RPC calls and support client-side disk caching.

In the NFS-mounted cases, two scenarios are considered with respect to the state of the kernel client buffer memory cache: one where the file system is re-mounted (1st run), and one where the file system is not remounted (2nd run). All execution times cover the entire run of an application, and are measured at physical machines. The execution time for LSS analysis against each of the 32 databases is, on average, 46.5s. The best data fit for the experiment image is obtained at a diameter of 5.796um, diameter deviation of 0.025um and refractive index of 0. Figure 1 shows the image scattering intensity as a function of wavelength and the corresponding Mie theory fit.

4.2. Results and Analysis

Table 1 shows the execution times for parallel LSS application in different scenarios; Figure 4 shows the corresponding speedup plots. In the table, 1st run and 2nd run represent the first and second executions following an NFS mount. For WAN+C 2nd run, the proxy cache is "warm" in both mount and unmount configurations. The results show that the parallel LSS analysis achieves speedups of up to 13.5. The advantage of client-side disk caching (WAN+C) in the presence of temporal locality becomes apparent when the number of processors is increased. The performance overhead of WAN+C with respect to local disk reduces significantly from 459% in case of a single processor to 12.5% with 16 processors. This can be explained by the increase in aggregate cache capacity stemming from the availability of independent proxy caches in each node. As the number of processors is increased, the working set size per each node is reduced and fits the proxy cache, resulting in high hit rate in the client-side disk cache.

Table 1. Execution times (seconds) for LSS analysis. Scenarios where databases are stored in local disk, LAN and WAN files servers are considered.

#Proc	Local Disk	LAN		WAN		WAN +C		
		1st run	2nd run	1st run	2nd run	1st run	2nd run	
							mount	unmount
1	1318	1404	1396	13473	11860	12465	7001	7369
2	664	735	718	5961	5883	5979	2204	2225
4	333	432	397	2992	2986	3044	674	1496
8	172	301	269	1993	1482	1580	228	317
16	99	234	203	817	755	804	111	183

The results summarized in Figure 4 support important conclusions. First, the hit-time overhead (with respect to local disk) introduced by the proxy-based virtual file system layer is small for the 16-processor case when the application exhibits temporal locality. Second, it can be observed from the performance difference between the WAN+C and WAN 2nd run scenarios that that the kernel-level buffer cache does not have sufficient capacity to hold the working dataset of an LSS database. The proxy-level disk cache behaves as a second-level cache to the kernel buffers; in fact, the 16-processor WAN+C scenario also achieves better performance than both LAN cases because kernel buffer misses are served by proxy disk cache accesses. It is important to point out that the design of the NFS call-forwarding file system proxy allows for a series of proxies, with independent caches of different sizes, to be cascaded between client and server, supporting scalability to a multi-level cache hierarchy (e.g. a two-level hierarchy with GBytes of cache storage space in a node's local disk, and TBytes of storage space available from a LAN disk array server).

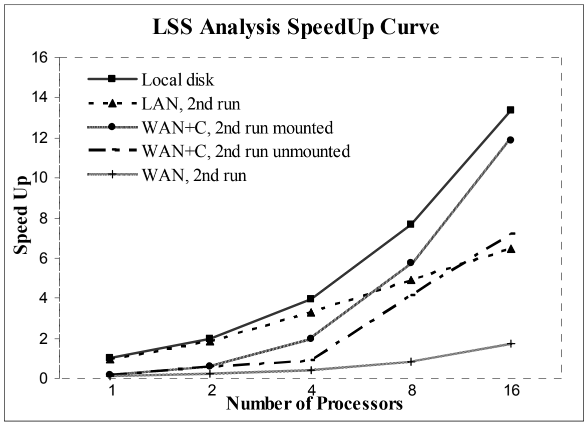

Figure 4. Speedup plot for parallel LSS analysis application.

Figure 5 shows speedups for the parallel database generation process. The database generation is a computational-intensive process that constantly exercises the file system – the legacy program that computes the Mie function reads from input files, and generates output files that are processed to generate each database entry. Therefore, this process generates many write requests that are subsequently invalidated by a file's removal. This experiment thus considers proxy-based configurations (WAN+C) that implement both a typical NFS write-through (WT) policy and an alternative write-back policy (WB) of blocks in the proxy disk cache.

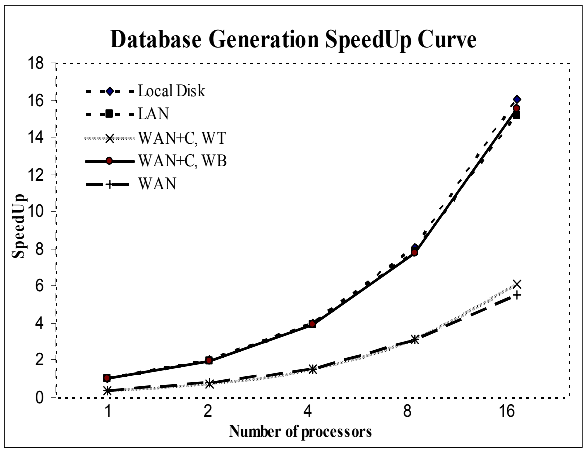

Figure 5. Speedup plot for parallel database generation. Only 1st run results are shown. The difference between 1st and 2nd runs is very small because data is mostly written.

The speedup plot shows that performance is close to linear with respect to number of processors for local-disk, LAN and WAN+C/WB. The performance gained from using a write-back cache policy in user-level proxies (as opposed to native, write-through schemes) is evident from the figure. The performance overhead in WAN+C scenario with local write back cache varies from 2% to 3% relative to the local disk configuration. The WAN+C write-back cache scheme performs slightly better than LAN scenario because it avoids network transfers for data that is written and then removed (such as the output Mie function files).

Table 2. Error, WAN execution time and number of NFS data blocks transfers for database sampling.

Sampling Interval	LSS Error	Time (s)	Number of Blocks
1	2.899	793	14666
5	2.900	700	14662
10	2.902	432	6894
20	2.916	323	3622
40	2.934	152	1856

Table 2 shows experimental results that consider variable-granularity executions of the LSS application. Low accuracy analysis is obtained by down-sampling the databases: an interval of *n* indicates that *n* records are skipped in the database before reading another record for analysis. The results shown in the table are based on the executions performed on 16 nodes in the WAN configuration. It can be seen that the least-square error has increased and the execution time has decreased as the sampling interval is increased. The 4th column in the table indicates the number of NFS blocks being transferred from the file server. It can be seen that the virtual file system transfers the data partially on demand for the low-accuracy case, reducing the overall transfer size by a factor of 8 and execution time by a factor of 5.2 (with respect to whole-file transfer). The reduction in the number of blocks transferred does not follow a linear relationship with respect to the sampling interval due to NFS client-side read-ahead (prefetching) implemented in the kernel. Nonetheless, the reduction in transfer requirements is substantial, and is handled by the virtual file system in an application-transparent way.

5. Related Work

Grid data management has been investigated in previous efforts in the context of distributed instrumentation [9][10][11]. However, proposed techniques have focused on a model where support for communication is explicitly provided by the application and/or grid middleware – typically, the data is "staged" from instrument to a remote computing node (and back). In addition, existing techniques are geared towards cases where substantial computational infrastructure (hardware and network capacity, and software expertise) is available at the site where data collection is performed (e.g. a national research center). In contrast, medical applications such as LSS imaging benefit from a different model, where support for communication is handled transparently from applications, hence reducing programming complexity, and where the computational infrastructure support consists of commodity computers and networks typical of a medical facility.

Current grid solutions typically employ file staging techniques to transfer files between user accounts in the absence of a common file system. Examples of these include Globus [12] and PBS [13]. As indicated earlier, file staging approaches require the user to explicitly specify the files that need to be transferred, or transfer entire files at the time they are opened. This poses additional application programming challenges (the programmer must explicitly identify all data that may be necessary to perform computation so that it can be transferred prior to execution) and may lead to unnecessary data transfers (e.g. of data needed for high-accuracy analysis that is not used in a low-accuracy computation). These requirements hinder the deployment of solutions that can dynamically adapt computation based on run-time requirements (since the choice of the working data set is statically determined before execution). In contrast, the architecture based on a Grid virtual file system allows for transfers of data on-demand, and on a per-block basis. Hence, the amount of data transferred is determined by the amount of data actually used in computation, and decisions regarding the data used in computation can be efficiently performed at run-time. This is important in applications such as LSS, where the size of working sets used in computation can vary dynamically based on accuracy requirements.

Some systems (e.g., Condor [14]) utilize remote I/O mechanisms from special libraries to allow applications to access remote files. Kangaroo [15] also employs RPC-based agents. However, unlike VFS, Kangaroo does not provide full support for the file system semantics commonly offered by existing NFS/UNIX deployments (e.g. delete and link operations). Legion [16][17] employs a modified NFS daemon to provide a virtual file system. From an implementation standpoint, this approach is less appealing than NFS call forwarding: the NFS server is customized, and must be extensively tested for compliance, performance and reliability.

6. Conclusions

Data management is a key challenge to be addressed in the context of Grid environments. Traditional approaches to Grid data management rely on application and/or middleware knowledge of file names for whole-file transfers, APIs that expose support for partial file transfers to an application developer, and/or customized libraries for remote I/O, requiring that support for on-demand transfers be either customized via application programming efforts or via library re-linking. A Grid file system approach to data management supports on-demand transfers at the O/S layer, requiring no application modifications. Such support is especially important for applications that access data at different granularities.

This paper presents a case for the use of a virtualized Grid file system for applications of this kind. A nascent application from the medical imaging domain (LSS) is used as a basis for this study; its performance is analyzed for different virtual file system scenarios. Results show that, in addition to leveraging native NFS client/server support for on-demand block transfers, the virtual file system can improve upon the performance of native implementations by means of per-session user-level disk caches and write policies. For an application that exhibits locality (e.g. in the case of LSS imaging, when multiple images are analyzed against the same set of databases, or when temporary file system data is invalidated locally by a write-back cache avoiding costly network transfers), the proxy caches can deliver performance levels close to that of a local disk.

7. Acknowledgements

Effort sponsored by the National Science Foundation under grants EIA-0224442, ACI-0219925 and NSF Middleware Initiative (NMI) collaborative grant ANI-0301108. The authors also acknowledge a gift from VMware Corporation and a SUR grant from IBM. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF, IBM, or VMware. The authors would like to thank Vadim Backman and his group at Northwestern University for valuable discussions and for providing access to LSS resources and data, and Peter Dinda for providing access to computational resources for the experiments.

8. References

[1] R. Figueiredo, N. Kapadia and J. A. B. Fortes, “The PUNCH Virtual File System: Seamless Access to Decentralized Storage Services in a Computational Grid”, Proc. IEEE International Symposium on High Performance Distributed Computing (HPDC), August 2001.

[2] Backman V, R. Gurjar, K. Badizadegan, I. Itzkan, R. R. Dasari, L. T. Perelman, M. S. Feld, “Polarized light scattering spectroscopy for quantitative measurement of epithelial cellular structures”, IEEE J Sel Top Quant. Elec., 5, 1019 (1999).

[3] Backman V, et al. "Detection of preinvasive cancer cells in situ", Nature, 406, 35-36 (2000).

[4] Backman V, Gurjar R, Perelman LT, Georgakoudi I, Badizadegan K, Itzkan I, Dasari RR, Feld MS, "Imaging human epithelial properties with polarized light-scattering spectroscopy", Nature Medicine, 7, 1245-1248 (2001).

[5] G. von Laszewski et al., "Real-time Analysis, Visualization, and Steering of Tomography Experiments at Photon Sources", Proc. 9th SIAM Conf. on Parallel Processing for Scientific Computing, Apr 1999.

[6] S. Smallen, H. Casanova and F. Berman, "Applying Scheduling and Tuning to On-line Parallel Tomography", Prof. of Supercomputing, Denver, Nov 2001.

[7] S. Smallen et al., "Combining Workstations and Supercomputers to Support Grid Applications: The Parallel Tomography Experience", 9th Heterogeneous Computing Workshop, May 2000.

[8] A. Apostolico et al, "Requirements for Grid-Aware Biology Applications", DataGrid WP10 Workshop, DataGrid-10-D10.1-0102-3-8, Sept 2001, http://marianne.in2p3.fr/datagrid/wp10

[9] A. Chervenak, I. Foster, C. Kesselmann, C. Salisbury, S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets", to appear, Journal of Network and Computer Applications, 23(3) p187-200 July 2000.

[10] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger and K. Stockinger, "Data Management in an International Data Grid Project", IEEE/ACM Intl. Workshop on Grid Computing (Grid'2000), Dec. 2000.

[11] J. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany and R. Wolski, "The Internet Backplane Protocol: Storage in the Network", Network Storage Symposium (NetStore), Seattle, WA 1999.

[12] J. Bester, I. Foster, C. Kesselman, J. Tedesco and S. Tuecke, "GASS: A Data Movement and Access Service for Wide Area Computing Systems", Proc. 6th Workshop on I/O in Parallel and Distributed Systems, May 1999.

[13] R. Henderson and D. Tweten, "Portable Batch System: Requirement Specification", Technical Report, NAS Systems Division, NASA Ames Research Center, Aug. 1998.

[14] M. Litzkow, M. Livny and M. W. Mutka, "Condor: a Hunter of Idle Workstations", Proc. 8th Int. Conf. on Distributed Computing Systems, pp104-111, June 1988.

[15] D. Thain, J. Basney, S-C. Son, and M. Livny, "The Kangaroo Approach to Data Movement on the Grid", Proc. 10th Intl. Symp. on High Performance Distributed Computing (HPDC), pp325-333, Aug. 2001.

[16] B. White, A. Grimshaw, and A. Nguyen-Tuong, "Grid-based File Access: the Legion I/O Model", in Proc. 9th IEEE Int. Symp. on High Performance Distributed Computing (HPDC), pp165-173, Aug 2000.

[17] B. White, M. Walker, M. Humphrey, A. Grimshaw, "LegionFS: A Secure and Scalable File System Supporting Cross-Domain High-Performance Applications", Proceedings of Supercomputing (SC), Nov 2001.

[18] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke. Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing, IEEE Mass Storage Conference, 2001.

[19] B. Callaghan, *NFS Illustrated*, Addison-Wesley, 2002, ISBN 0-201-32570-5.

[20] J. Morris, M. Satyanarayanan, M. Conner, J. Howard, D. Rosenthal and F. Smith, "Andrew: A Distributed Personal Computing Environment", Communications of the ACM, 29(3) pp184-201, March 1986

[21] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, D. Steere, "Coda: A Highly Available File System for a Distributed Workstation Environment", IEEE Transactions on Computers, 1990, 39(4), 447-459.

[22]. S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu. "From Virtualized Resources to Virtual Computing Grids: The In-VIGO System", to appear, Future Generation Computing Systems, special issue, Complex Problem-Solving Environments for Grid Computing, David Walker and Elias Houstis, Editors.

[23]. R. Figueiredo, P. Dinda and J. Fortes, "A Case for Grid Computing on Virtual Machines", Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS), May 2003

[24]. C. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. Lam and M. Rosenblum, "Optimizing the Migration of Virtual Computers", Proceedings of the 5th Symposium on Operating Systems Design and Implementation, 2002.

[25]. R. J. Figueiredo, "VP/GFS: An Architecture for Virtual Private Grid File Systems". In Technical Report TR-ACIS-03-001, ACIS Laboratory, Department of Electrical and Computer Engineering, University of Florida, 05/2003.