Iteration Aware Prefetching for Remote Data Access

Philip J. Rhodes and Sridhar Ramakrishnan

Department of Computer and Information Science
University of Mississippi
{rhodes, sridhar}@cs.olemiss.edu

Abstract

Although processing speed, storage capacity and network bandwidth are steadily increasing, network latency remains a bottleneck for scientists accessing large remote data sets. This problem is most acute with n-dimensional data. Grid researchers have only recently begun to develop tools for efficient remote access to n-dimensional data sets.

Within the context of the Granite Scientific Database system, we show that latency penalties can be dramatically reduced using explicit knowledge of a user's access pattern represented as an Iterator. *The iterator not only performs an n-dimensional iteration for the user, but also communicates the access pattern to Granite so that a prefetching cache can be constructed that is tuned to the user's access pattern.*

We experimentally evaluate a scenario for incorporating Granite's prefetching mechanism into the Grid, demonstrating extraordinary performance gains. In light of these results, we describe planned additions to existing Grid services to allow selection of datasets according to the user access pattern.

1 Introduction

The size of scientific data sets has grown explosively in recent years, presenting new challenges for researchers and educators without local access to high performance computing resources. The Visible Woman dataset from the National Institutes of Health totals 39GB of anatomical sections. The Sloan Digital Sky Survey contains about 15 terabytes of information [SDSS]. The NCAR Mass Storage System exceeded one petabyte of climate data in 2003, and has recently added a second petabyte [NCAR].

The vast majority of researchers cannot store such datasets on local machines, but remote access to arbitrary subsets of the complete file can make large datasets tractable for most scientists. However, researchers [Radke, Schütt04] have only recently begun to develop efficient methods for accessing subsets of *n*-dimensional datasets, where data exists in a space described by *n* axes. A typical subset request might ask for the data contained in an *n*-rectangular subregion aligned with the dataset axes. Without special support, such a request must be satisfied by a large number of distinct one dimensional requests. Each distinct request must pay network latency costs, which can drastically reduce performance.

Even with an *n*-dimensional access method, penalties associated with network latency can be further reduced if knowledge of the pattern of access is known in advance. For example, if a scientist wants to traverse a 3000x3000 dataset in row-by-row fashion using 3x3 blocks, 1 million 3x3 block accesses are required. Performing so many accesses over a high latency network connection will be unacceptably slow. An alternative is to download the entire file and work on it locally, but this is an unrealistic option for files that are tens or hundreds of gigabytes in size, especially if the scientist is only interested in a subset of the data. A third option is to split the file into chunks or slabs, but such splitting may require assumptions about how the dataset will be accessed. Visualization applications like *ray-casting* and *splatting* may require accessing the dataset in arbitrary directions [Levoy88, Westover90].

However, if the scientist's local workstation has enough memory to store a block of size 500x3000 elements, latency costs are only paid for the 6 separate network transactions necessary to refresh that large block 6 times. If the server and client are sufficiently distant, this can result in dramatic performance gains.

In previous work we described *Iteration Aware Prefetching (IAP)*, a method that increases performance by minimizing the effects of disk latency [Rhodes05b]. IAP uses large *n*-dimensional cache blocks to reorganize the user's access pattern into a comparatively small number of large accesses. IAP is an important feature of the *Granite Scientific Database* [Rhodes01, Rhodes02]. An iterator is used both to perform the desired iteration and to inform the Granite system of the intended access pattern so that the shape of the cache blocks can be tuned to the iteration. Such blocks provide very significant performance gains compared to blocks which are not tuned to the iteration.

We have a growing toolkit of iterators, including rectilinear block, plane, and element iterators, as well an oblique plane iterator that allows arbitrary orientation

This work was supported in part by the National Science Foundation under grants IIS-0082577 and IIS-9871859

within the data volume. In [Rhodes05], we demonstrated the effectiveness of IAP for visualization of large volumetric scientific datasets. In this paper, we concentrate on rectilinear iteration, in which the direction of iteration is aligned with the major axes. The fast fourier transform, wavelet decomposition, feature detection, and other applications involve rectilinear iteration through the data volume in various directions. IAP is particularly effective in mitigating the cost of latency when the direction of iteration would otherwise result in very slow performance.

Although improvements like Internet2, ESNet, and UltraScienceNet [USN] ensure that network bandwidth will increase steadily over the next several years, network latency will continue to hinder remote data access. Since network latency for long distance transfers is often much higher than disk latency, Granite's prefetching mechanism should prove especially useful when researchers work with large datasets stored at distant sites.

There are two main points to this paper. First, we demonstrate the advantages of Granite's IAP method when applied to the problem of remote access to *n*-dimensional scientific datasets. Second, we outline our plans to augment existing Grid metadata and tools to support IAP in Grid Computing environments. The next section discusses background and related work, followed by a brief description of our implementation of IAP for array based data in section 3. Section 4 describes a scenario for IAP on the Grid which we evaluate experimentally in section 5. Section 6 outlines our plan for bringing IAP to Grid computing. We end with future work and conclusions.

2 Background

Grid computing has evolved in recent years in response to the need for coordinated sharing of resources, data, and knowledge among geographically separated groups of scientists and institutions [Buyya01, Cannataro03, Chervenak01, Foster01]. Grid computing environments are built on a layer of basic services that support the resource management, data transfer, authentication, and instrumentation functions necessary to support a collaborative computing environment [Chervenak01, Foster97, Laszewski00]. These include *storage systems*, an abstraction for the various tools which provide efficient access to the large datasets used by Grid scientists, and *metadata repositories*, which store descriptive information about files and network resources [Chervenak01]. Typical choices for storage systems include HPSS (High Performance Storage System), and HDF5, a format and API for scientific data access [Allen00, Watson95]. Metadata is made available through services such as LDAP (Lightweight Directory Access Protocol) [Howes97, Wahl97].

On top of these basic services are systems that support access to distributed data resources through the addition of replica management and selection [Chervenak01, Csajkowski98, Raman98, Vazhkudai01]. Allowing grid applications to choose which of several copies of the same data can be accessed most efficiently is an important part of replica selection [Vazhkudai02]. The access itself can be accomplished with the help of tools such as GASS (Global Access to Secondary Storage), RIO (Remote IO), SRB (Storage Resource Broker), and GridFTP [Baru98, Bester99, Foster97, Allcock05]. Nallipogu, et al. extend SRB through pipelining, improving performance partly by minimizing latency [Nallipogu02].

2.1 Communications

Development of low level protocols for transferring very large datasets has been an active field of research. Both TCP and UDP based methods have been developed.

GridFTP is a commonly used method of transferring large files in a Grid environment [Allcock05]. It uses striped and parallel data transfer to maximize the use of available bandwidth. GridFTP's architecture is modular, allowing the development of extensions for new protocols and functionality. An example can be found in [Schütt04]. Currently, GridFTP transfers data using TCP, but its open architecture allows for other options.

UDP based methods often have the advantage of higher overall performance compared to TCP methods because they do not pay the overhead associated with TCP's built-in mechanism to guarantee completeness of the received data. However, in most scientific applications it is imperative that all data arrive safely, so UDP based methods must provide their own mechanism for ensuring completeness [Grossman04, Rao04].

The work described in this paper relies upon our own implementation of a simple UDP method that avoids the latency penalties associated with TCP, yet guarantees the completeness of the received data. Although this method is robust enough to demonstrate the effectiveness of IAP for Grid applications, our eventual goal is to take advantage of GridFTP and other established protocols.

2.2 Remote n-dimensional access

Grid support for multidimensional data sets is largely provided using underlying scientific data APIs such as HDF5 [Allen00]. Only recently has work begun on ways of querying remote sources of data within a multidimensional paradigm. For example, Radke et al. describe an extension to GridFTP that adds support for remote n-dimensional queries to files stored in HDF5 [Radke].

Schütt et al. describe *nested FALLS,* a method of specifying a series of one dimensional accesses in a single compact representation. By mapping an *n*-dimensional block to the underlying one dimensional file, nested FALLS can be used to send a single query over the network, rather than a large number of one dimensional queries. In addition, some elements in a volume can be skipped, allowing basic multiresolution functionality through subsampling.

In order to demonstrate the effectiveness of IAP for remote access, we designed and implemented a Granite server that allows a remote client to request an arbitrary n-dimensional rectangular subset of a dataset.

2.3 Prefetching and Caching

Efficient access to data stored on local disk has been a topic of intense interest for decades. For spatial scientific datasets, perhaps the best known method is *chunking* [Sarawagi94]. Chunking reorganizes a dataset into n-dimensional chunks according to the expected access pattern. However, for extremely large datasets it is impractical to make a copy of the dataset for each expected access pattern. In this context, access patterns are defined by the shape of rectangular block queries, so the number of variations is very large. An alternative is to use chunks that have the same length in each direction, providing a reasonable speedup regardless of the access pattern used. HDF5 can produce and use chunked files, and can also use chunks as its basic unit of I/O and caching.

Much of the prefetching work relating to a network environment is geared toward web applications. [Wang99] provides a survey of caching schemes for this area. For Grid applications, Perez et al. [Perez02, Perez03] describe *MapFS*, a system that uses hints to help decide which data to prefetch and retain in a cache.

2.4 Access Patterns and the Grid

The designers of the Data Grid [Chervenak01] recognize the need for applications to convey hints about access pattern to the underlying storage system. Researchers have also used a history of the files accessed by an application to decide replication policy [Bell03, Ranganathan02]. However, to the best of our knowledge, no attempt has been made to convey the expected multidimensional pattern of access within a file to the remote server. Awareness of the access pattern allows fewer, larger data requests to be made across a network, thereby reducing total latency costs.

3 The Granite Model

The Granite model accommodates multiresolution representations, datasets combined from multiple sources, and formats ranging from simple arrays to unstructured meshes. Unstructured meshes are addressed by the *lattice* component. The work presented here is done within the context of the *datasource* component of the Granite system, which handles array-based data.

A datasource is conceptually an *n*-dimensional array containing a set of sample points. The array indices define the *index space,* also called a *data volume*. Each index space location has a collection of associated data values, called a *datum*.

Datasources must handle two basic kinds of queries. A *datum query* specifies a single index space location, and is satisfied by the return of a single datum. A *subblock query* specifies an n-dimensional rectangular region of the index space, and is satisfied by the return of a data block, which is conceptually an n-dimensional array of datums.

3.1 Storage Orderings

While a datasource has an index space that is n-dimensional, the file is a one dimensional entity. The datasource is responsible for satisfying queries expressed in its index space by reading data from the file. It must therefore map its index space to file offsets. It does this with the help of a kind of *axis ordering*. An axis ordering is simply a ranking of axes from outermost to innermost. "Innermost" and "outermost" suggest position in a set of nested for loops. The innermost axis changes most frequently and is called the *rod axis* when referring to storage orderings. Axes are labeled with numbers, so an axis ordering is really just a list of integers. For example, the storage ordering for Figure 1 would be {0,1} if axis 0 is vertical and axis 1 is horizontal. When an axis ordering denotes the order in which data is stored on disk, it is called a *storage ordering*. The innermost axis of a storage ordering is known as the *rod axis*, where *rods* are series of elements

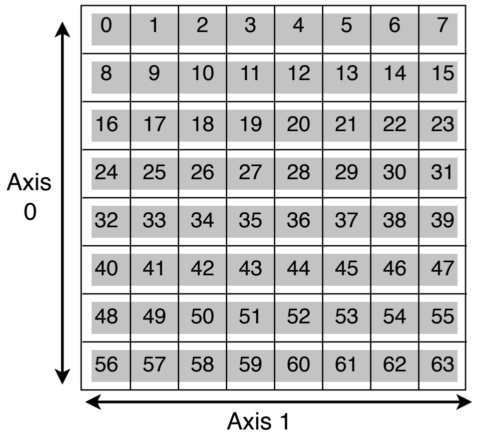

Figure 1. The numbers indicate the ordering of elements in the one dimensional file. The storage ordering here is {0,1}, and the shaded regions indicate the rods for the file.

that are contiguous in both the data volume and the one dimensional file.

3.2 Iterators

Since our system aims to improve I/O performance for particular access patterns, we use iterators to represent the access pattern, as well as to perform the actual iteration through the datasource index space. Iterators have a value that changes with each invocation of the iterator's next() method. This value might denote a single location in the index space, or perhaps an entire region. In either case, the iterator value can be used directly in both datum and sub-block queries.

An axis ordering is used to help represent the behavior of iterators that proceed through the index space in rectilinear fashion. Generally, the performance of an iteration over data stored on disk depends upon the relationship between the storage ordering and the iteration ordering. Performance is highest when the two are the same. In this ideal case, the path of the iterator through the data volume maps to a straightforward "left to right" path through the file on disk. In this situation, the filesystem caching provided by the operating system does very well in caching and prefetching the correct data. Performance is considerably worse when the storage ordering and iteration ordering don't match. Here, the path through the data volume maps to a pattern that skips around in the file. This makes the filesystem cache much less effective, since it tends to cache and prefetch the wrong data. This problem is worst when the storage and iteration orderings are the reverse of each other. For example, a {2,1,0} iteration yields the worse performance on a {0,1,2} file. A {1,2,0} iteration is better, while a {0,1,2} iteration is best, since it matches the storage ordering.

The *iteration space* is the space traversed by the iterator. It may be the entire index space of a datasource, or some subset of that space. We also represent the starting point and the stride through the iteration space in cases where the iterator skips over some locations. Along with the axis ordering, all this information is useful and available when the system creates a prefetching cache tuned to the iteration.

3.3 Spatial Prefetching

Most caching methods for local data view files as one dimensional entities, but this view of the data is not adequate for scientific applications involving multidimensional datasets because it misses the neighborhood relationships that the user needs. The problem becomes even more acute as the dimensionality of the dataset increases. To address this issue we have designed a multidimensional cache that corresponds to the user's dimensional view of the data.

In this section we present a brief conceptual overview of a form of IAP called *spatial prefetching (SP)*. For a more thorough and formal discussion of spatial prefetching and IAP in general, see [Rhodes05b].

3.3.1 Choosing Cache Block Shape

Because our multidimensional cache model is aimed at supporting multidimensional array data, we organize the cache itself as a collection of data blocks, called *cache blocks*, with the same dimensionality as the data. A significant component of the caching strategy is to determine how to shape the cache blocks to most effectively improve I/O performance.

If the pattern of future accesses is already known, however, we can choose a cache block shape that guarantees that all the needed contents will be used before being discarded. We call such a cache block shape *well formed* with respect to the iteration. Caches with blocks well formed for an iteration do not reload discarded blocks when the iteration is performed. This is particularly valuable in a remote context because of the expense of retransmitting data across a network.

Given some amount of memory with which to construct a cache block, a well formed block can be built by examining the iterator ordering from right to left. For each axis, we extend the cache block as far as we can along that axis until we either run of out memory, or hit the end of the iteration space. If there is still memory available, we repeat this process with the next axis, terminating when either memory or the list of axes is exhausted.

For example, consider a {1,2,0} iteration over a 512^3 datasource. First, we initialize the cache block shape to {1,1,1}. We start with axis 0, at the right side of the or-

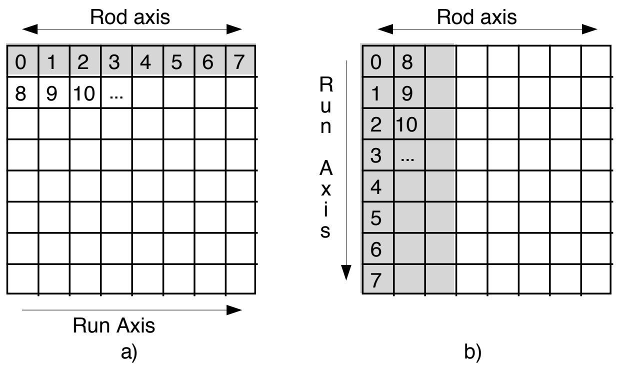

Figure 2. The numbers indicate the order in which elements are visited by the iterator, while the shaded regions represent an effective cache block shape. a) Both the storage and iterator ordering are {0,1}. b) The storage ordering is {0,1}, but the iterator ordering is {1,0}.

dering, and extend the block shape to the end of the iteration space along this axis, yielding a shape of {512,1,1}. Next, we extend the shape along axis 2, but perhaps available memory constrains us to a shape of {512,1,128}.

This cache block shape is well formed with respect to a {1,2,0} iteration, so once the iterator leaves this block it need not be reloaded. For datum iterations and most block iterations, this means that only one cache block need be kept in memory at a time, since the iterator never hops back and forth between blocks. The shaded regions in figure 2 are examples of well formed cache block shapes for {0,1} and {1,0} iterations, respectively.

If the cache is being used to directly access a disk, we must allocate enough memory for the cache block so its constituent rod segments are long enough to provide a speedup. The memory required is partially determined by the relationship between the storage and iterator orderings. For example, the cache block shape shown in figure 2b requires more memory than the one shown in figure 2a because the storage and iterator orderings are orthogonal.

4 Remote Access and IAP

We see two important ways in which Granite can contribute to Grid computing. First, IAP can be used during remote data access to reduce the penalties associated with network and disk latency. Second, if grid tools and applications are aware of the storage ordering of datasets, better decisions can be made when choosing datasets and access patterns. In this section we describe a scenario for using IAP in a remote environment. Section 5 presents an experimental evaluation of this scenario, while section 6 discusses our plans for bringing IAP to grid environments in light of these results.

4.1 Reducing Latency Costs

Figure 3 shows an overview of a remote access implementation using IAP. On the left, an iterator specifies a user access pattern that accesses elements in the order indicated by the numbering. Using IAP, the client constructs a cache block that can contain a large number of the elements required by the user. Instead of sending a query for each element in the iteration, the client may now send a query for each cache block. Since each cache block contains a large number of elements, the number of times network latency costs must be paid is greatly reduced. Also, since the cache blocks are well formed with respect to the iteration, once the iteration leaves a cache block, the block will not be needed again. For the example shown in the figure, this means that sixteen separate accesses have been transformed into four. In practice, we use cache blocks that are hundreds of megabytes in size, yielding a much more dramatic reduction in the number of network transactions.

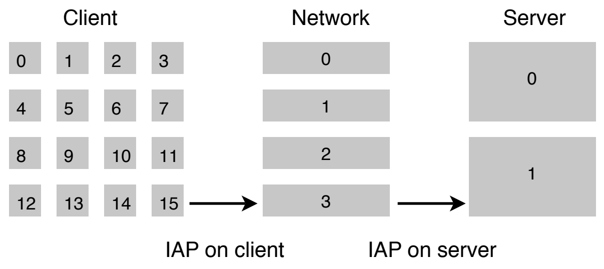

Figure 3. IAP transforms the user access pattern on the client (specified using an iterator) into a comparatively small number of network transactions with the server, thereby reducing network latency penalties. The server may then elect to use IAP to further transform the access pattern to reduce disk latency penalties.

In section 5 we investigate whether it is beneficial for the server to apply IAP once more to the access pattern that it sees coming in over the network. On the right side of figure 3, we show that the four block queries are transformed into two larger block queries using an IAP cache on the server itself.

4.2 Choosing Iteration Order and Data Sets

Although client-side IAP helps to address disk latency as well as network latency, the results in the next section show that it would be best if the user iteration ordering closely matches the storage ordering.

There are two ways to achieve this. If the user application doesn't strictly require a particular iteration ordering, then it may elect to use one that closely matches the dataset storage ordering on the most convenient server. If a particular iteration ordering is required, then a server should be chosen that holds a copy of the dataset with a storage ordering that most closely matches the required iteration ordering.

5 Results

To examine the benefits of IAP and the Granite model for remote data access, we ran a series of tests that transferred data between a client at the University of Mississippi and a server at the University of New Hampshire. The server is a single processor Pentium 4 machine with a 2.4Ghz CPU and 2GB of RAM running the Linux operating system. The disk on this machine is a fast 15,000 RPM SCSI disk with a 3.6ms average read seek time. The client is a 2.5Ghz dual processor Power Macintosh G5 and 2GB of RAM running OS X. Both machines are connected to the local network with 100Mb Ethernet. The institutions are connected via Internet2 with a round trip time of ap-

	Remote Datum Iteration			Remote 64^3 Block Iteration			Local Datum Iteration		Local Block Iteration	
Client Cache	128MB LRU	128MB SP	128MB SP	128MB LRU	128MB SP	128MB SP				
Server Cache	none	none	256MB SP	none	none	256MB SP	none	256MB SP	none	256MB SP
Ordering	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**
{0, 1, 2}	$5.9\text{x}10^7$ (est)	543	495	$1.9\text{x}10^4$	317	275	3380	469	125	102
{1, 2, 0}	$5.9\text{x}10^7$ (est)	874	816	$1.9\text{x}10^4$	326	276	6013	665	128	114
{2, 1, 0}	$5.9\text{x}10^7$ (est)	1194	978	$2.0\text{x}10^4$	618	423	$2.2\text{x}10^6$ (est)	764	674	238

Table 1. Results for a traversal of a 1GB subset of an 8GB file. All execution times are in seconds.

proximately 48ms. All transfers were done using our own UDP protocol using a packet length of 61440 bytes and an inter-packet delay of 10ms. This protocol allows the client to request either a single datum or an n-dimensional block of data from the server. In preliminary tests, we found that traversing the data volume using single datum requests would take several years, so we concentrate here on tests that use block requests only.

Table 1 shows our results for traversals of a 1024x 1024x1024 (1GB) subset of an 8GB file of bytes stored in {0,1,2} order using various caches and iterator orderings. For each test, the filesystem cache on the server was flushed to remove the effects of operating system caching.

5.1 IAP vs. LRU Caching

In order to establish the effectiveness of *n*-dimensional cache blocks with shapes tuned to the iteration pattern, we compare our method with a simple LRU cache containing cubic blocks. Cubic blocks are not well formed, as discussed in section 3.3.1, so the iterator will cause the cache to reload the same block many times during the course of the traversal. We tested LRU performance using both datum and block iterations. The datum iteration requests a series of single elements, while a block iteration requests a series of contiguous 64^3 blocks from the server. Performance for the datum iteration was very poor, so we ran tests on a subset of the 1024^3 volume used in table 1. Using a single cache block of 128MB on the client, we project that a complete traversal of the 1024^3 volume would take approximately 692 days for a datum iteration, as shown in column *a*. Column *d* shows that the time for a block iteration is much more reasonable (about 5.4 hours), since the number of network transactions is smaller. Columns *b* and *e* show that a similar test for a 128MB SP cache accomplishes the same traversals in times ranging from 317 to 1194 seconds, depending on the type of iteration and its ordering. Even for block iterations, the 128MB SP cache yields speedups of 59.9, 58.3, and 30.7 for each ordering. The SP cache produces speedups of about 100,000 for datum iterations.

We also ran a test using an LRU cache with 4096 cache blocks of dimensions 32^3 totaling 128MB, and found performance was far worse than in the other LRU test. The projected time for a complete {0,1,2} traversal using this cache was $1.2\text{x}10^7$ seconds (138 days). Although using smaller blocks means that each block is reloaded fewer times, the data volume is divided into many more blocks, so many more requests are made to the server as the iterator traverses the space.

It should be clear from these results that shaping the cache block to suit the iteration is extremely effective. Because such cache blocks are well formed we can make the block as large as possible, greatly reducing the number of network transactions but completely avoiding the problem of repeatedly reloading data from the server.

5.2 Iterator Ordering and Performance

Columns *g* and *i* show execution times for datum and block iteration conducted locally on the server machine without any caching except the filesystem caching mechanism. The results for the datum iteration in column *g* show extreme sensitivity to the relationship between the storage and iterator orderings. This relationship determines locality of access and the number of reads made to disk during the traversal. The times for block iteration shown in column *i* show much less sensitivity and are much faster overall because block requests allow Granite to make fewer reads and to perform those reads in an order that maximizes locality in the file on disk.

For similar reasons, it is harder for an SP cache to improve upon performance with block iteration. The results for a 256MB cache shown in column *j* show very marginal gains compared to column *i*, with the exception of the {2,1,0} case. Because this last case shows marked improvement, however, the results in column *j* show less sensitivity to iteration ordering.

The results for remote access in columns *b* and *e* show that even when accessing data over a high latency network connection, performance is still sensitive (by a factor of roughly 2) to iteration ordering. Columns *c* and *f* show the results of our attempt to reduce this sensitivity and improve overall performance by adding a 256MB SP cache on the server. The results for the block iteration show some improvement, but the datum iteration results show relatively little gain in performance or reduction in sensitivity. This is largely because the SP cache on the client is already reorganizing the user access pattern in a manner that increases disk performance. This explains why remote performance exceeds local performance in columns *b* and *g*. In order to produce further gains, the server must use even more memory than the client.

6 Granite and the Grid

The results in the preceding section show that although IAP is enormously effective for remote access, the performance is still sensitive to the datasets' storage ordering on disk. Switching from our own transport protocol to the more efficient GridFTP will make this sensitivity more apparent. Server-side IAP is not a cost effective solution for our scenario, since a large separate cache would need to be maintained for every client connection.

An attractive alternative is to make available replicas of important datasets with different storage orderings. Although this requires duplication of the data, replication is already an important part of Grid computing. Replicas with different orderings may be purposefully seeded on the Grid or produced incidentally by a scientist iterating through an existing replica and saving the result.

This scheme requires the modification of several existing Grid components, including data transport, metadata, and resource management.

To take advantage of the efficiencies of GridFTP, we must extend it to support the retrieval of *n*-dimensional subsets of remote Granite datasets, similar to Radke's work for HDF5 [Radke]. For server-side IAP, no further modification of GridFTP is necessary, since the server does not need to know the access pattern in advance.

We must attach to replicas additional metadata describing the *n*-dimensional bounds and storage ordering of each particular copy. This metadata is static, and should be conceptually paired with the data stored in the storage system. Granite currently uses XML to describe both bounds and orderings as lists of integers, and similar representations should be possible in other systems.

If replicas with various orderings are available, a client may either choose an iteration which matches the most convenient copy, or the client or broker may choose a copy which best suits the user's intended access pattern. If no replica has a storage ordering that exactly matches the iteration ordering, we must choose the best alternative, a process which requires us to *rank* the available orderings from best to worst match.

A simple ranking can be constructed using the storage ordering and dataset bounds to determine for which replica the iteration would have the best locality in the one dimensional file space.

We could also model expected performance by taking into account the physical characteristics of the disk the dataset is stored on, including average seek time, I/O bandwidth, disk buffer size, and disk block size. However, it may be more effective and accurate to collect actual disk performance statistics for various orderings whenever possible. This approach has the advantage of providing numbers that can be be directly compared to the end-to-end network statistics gathered in some grid systems [Vazhkudai01].

In any case, replica selection must be performed with both disk and network performance in mind. If all replicas are nearby, storage ordering will dominate, but if the replica with the ideal storage ordering is very distant, it may be better to choose a nearby replica with a storage ordering that is at least a reasonable match.

While the specification and dissemination of storage ordering and bounds information should be fairly straightforward, the implementation of the matching process is more complex. However, the ClassAd system [Raman98] contains a ranking mechanism that could be augmented to support replica selection that takes n-dimensional information such as storage ordering and bounds into account.

7 Conclusions and Future Work

We have shown that spatial prefetching, an example of iteration aware prefetching, is extremely effective in reducing the costs of network latency when iterating through sub-volumes of large *n*-dimensional datasets. In addition, we have shown that remote access performance is sensitive to the relationship between iterator and storage ordering, even when network latency is high.

Because of this sensitivity, we suggest that for important datasets useful to many researchers, multiple copies with different storage orderings should be available for remote access. The dataset bounds and storage ordering must then be made available to Grid services to allow clients to select a replica that best suits the user's intended access pattern. Alternatively, the user access pattern might be selected based upon the storage ordering of the most convenient server.

We will begin the process of bringing IAP to Grid computing by adapting Granite to take advantage of GridFTP. We must then augment the metadata associated with a particular file instance to include the bounds and storage ordering. Lastly, we must design and implement a mechanism for selecting the replica which will give the best performance for the access pattern desired by the application. As a result of this work, scientists will have effective remote access to the very large n-dimensional datasets that are an increasingly important part of modern science.

8 Acknowledgments

The authors would like to offer their sincere thanks to R. Daniel Bergeron, Gerry Pregent, Gina Ross, Ted Sparr, and most especially Andrew Foulks for allowing us to use UNH facilities to conduct our experimental evaluation. We would also like to thank Sudarshan Vazhkudai for his helpful comments.

9 References

[Allcock05] B.Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, "The Globus Striped GridFTP Framework and Server", *HPDC 2005 (Submitted)*

[Allen00] G Allen, W Benger, T Goodale, H Hege, G Lanfermann, A Merzky, T Radke, E Seidel, J Shalf, "The Cactus Code: A Problem Solving Env. for the Grid", *HPDC 2000*

[Baru98] C Baru, R Moore, A Rajasekar, M Wan, The SDSC Storage Resource Broker Proc. CASCON'98 Conference, 1998

[Bell03] W. Bell, D. Cameron, R. Carvajal-Schiaffino, A. Paul Millar, K. Stockinger, F. Zini, "Evaluation of an Economy-Based File Replication Strategy for a Data Grid", *Int. Wrk. on Agent based Cluster and Grid Computing at CCGrid 2003*

[Bester99] J. Bester, I. Foster, C. Kesselman, J. Tedesco, S. Tuecke, "GASS: A Data Movement and Access Service for Wide Area Computing Systems", *Workshop on Input/Output in Parallel and Distributed Systems*, 1999

[Buyya01] "Economic Models for Management of Resources in Peer-to-Peer and Grid Computing", *ITCom 2001*

[Cannataro03] M. Cannataro and D. Talia,"The Knowledge Grid", *CACM*, January 2003

[Chervenak01] A Chervenak, I Foster, C Kesselman, C Salisbury, S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets", *Jnl. of Network and Comp. Applications*, 2001

[Czajkowski98] K. Czajkowski, I. Foster, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. "A Resource Management Architecture for Metacomputing Systems", *4th Workshop on Job Scheduling Strategies for Parallel Processing*, 1998

[Foster01] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid", *Intl J. Supercomputer Applications*, 2001

[Foster97] I. Foster, C Kesselman, "Globus: A metacomputing infrastructure toolkit", *Int. Jnl of Supercomputing App.*, 1997

[Grossman04] R. Grossman, Y. Gu, D. Hanley, X. Hong, B. Krishnaswamy, "Experimental studies of data transport and data access of earth-science data over networks with high bandwidth delay products", *Comp. Networks, Vol. 46(3),* 2004

[Howes97] T.A. Howes and M.C. Smith. "LDAP Programming Directory-Enabled Application with Lightweight Directory Access Protocol". *Technology Series*. MacMillan, 1997.

[Laszewski00] G. von Laszewski, I. Foster, J. Gawor, P. Lane, "A Java Commodity Grid Kit", *Concurrency and Computation: Practice and Experience*, 2001

[Levoy88] M. Levoy, "Display of Surfaces from Volume Data" *IEEE Computer Graphics and Applications, Vol. 8(3)*, 1988

[Nallipogu02] E. Nalligopu, F. Ozguner, M. Lauria,"Improving the Throughput of Remote Storage Access through Pipelining", *3rd Int'l Wrk. on Grid Computing (GRID 2002)*

[NCAR] "NCAR's MSS exceeds 1 petabyte", news item at http://www.scd.ucar.edu/news/03/features/ 0227.petabyte.html

[Perez02] M. Perez, R. Pons, F. Garcıa, V. Robles, J. Carretero, "A proposal for I/O access profiles in parallel data mining algorithms", *3rd ACIS International Conf. on SNPD*, June 2002

[Perez03] Perez, M., Carretero, J., Garcia, F., Pena, J.M., and Robles, V., "MAPFS: A Flexible Infrastructure for Data-Intensive Grid Applications", *Annual CrossGrid Project Workshop and 1st European Across Grids Conference*, Spain, 2003

[Radke] Globus Striped Ftp Server with HDF5 plugin http://www.cactuscode.org/VizTools/ Radke.html

[Raman98] R. Raman, M. Livny, M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing", *HPDC 1998*

[Ranganathan02] K. Ranganathan and I. Foster, "Identifying Dynamic Replication Strategies for a HighPerformance Data Grid", *International Workshop on Grid Computing*, 2001

[Rao04] N. S. V. Rao, Q. Wu, S. M. Carter, and W. R. Wing. "Experimental results on data transfers over dedicated channels", 1st International Workshop on Provisioning and Transport for Hybrid Networks: *PATHNETS*, 2004

[Rhodes01] P.J. Rhodes, R.D. Bergeron, and T.M. Sparr, A Data Model for Distributed Multisource Scientific Data, *Hierarchical and Geometrical Methods in Scientific Visualization*, Springer-Verlag, Heidelberg, 2001

[Rhodes02] P.J. Rhodes, R.D. Bergeron,, and T.M. Sparr, "A Data Model for Distributed Multiresolution Multisource Scientific Data", *Hierarchical and Geometrical Methods in Scientific Visualization*, Springer-Verlag, Heidelberg, Germany, 2002

[Rhodes05] P.J. Rhodes, X. Tang, R.D. Bergeron, and T.M. Sparr, "Out of core visualization using Iterator Aware Multidimensional Prefetching", *Proc. SPIE Vol. 5669, p. 295-306, Visualization and Data Analysis 2005*

[Rhodes05b] P.J. Rhodes, X. Tang, R.D. Bergeron, and T.M. Sparr, "Iteration Aware Prefetching for Large Scientific Datasets", *Proc. SSDBM 2005*. pp. 45-54

[Sarawagi94] S. Sarawagi, M. Stonebraker, "Efficient Organizations of Large Multidimensional Arrays", *Proc. of the Tenth International Conference on Data Engineering*, Feb. 1994

[Schütt04] T. Schütt, A. Merzky, A. Hutanu, F. Schintke, "Remote Partial File Access Using Compact Pattern Descriptions", *Proc. of the 4th Int. Symp. on Cluster Computing,* 2004

[SDSS] Sloan Digital Sky Survey, http://www.sdss.org/

[USN] DOE UltraScienceNet: Experimental Ultra-Scale Network Testbed for Large-Scale Science, http://www.csm.ornl.gov/ultranet

[Vazhkudai01] S. Vazhkudai, S. Tuecke, I. Foster, "Replica Selection in the Globus Data Grid", *IEEE International Symp. on Cluster Computing and the Grid (CCGrid),* May 2001

[Vazhkudai02] S. Vazhkudai, J. M. Schopf, I. Foster, "Predicting the Performance of Wide Area Data Transfers", *16th International Parallel and Distributed Processing*, 2002

[Wahl97] M. Wahl, T. Howes, and S. Kille, Lightweight Directory Acces Protocol (v3), RFC 2251, IETF 1997

[Wang99] J. Wang, "A Survey of Web Caching Schemes for the Internet", *Proc. of ACM SIGCOMM '99,* 1999

[Watson95] R.W. Watson, R.A. Coyne,"The parallel I/O architecture of the high-performance storage system (HPSS)", Proc. 14th *IEEE Symposium on Mass Storage Systems*, 1995

[Westover90] K. Westover, Footprint Evaluation for Volume Rendering, *Computer Graphics,* vol. 24, 1990, pp. 367-376