Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

WERNER BENGER(AEI[1] , ZIB[2]), HANS-CHRISTIAN HEGE (ZIB), ANDRÉ MERZKY (ZIB), THOMAS RADKE (AEI), EDWARD SEIDEL (AEI)

Efficient Distributed File I/O for Visualization in Grid Environments

[1] Max-Planck-Institut für Gravitationsphysik Potsdam/Golm Albert-Einstein-Institut
[2] Konrad-Zuse-Zentrum für Informationstechnik Berlin

Contribution to the PDC'99 Conference "Simulation and Visualization on the Grid" in Stockholm

Preprint SC 99–43 (Januar 2000)

Efficient Distributed File I/O for Visualization in Grid Environments

Werner Benger(AEI[‡], ZIB[§]) Hans-Christian Hege (ZIB)
André Merzky (ZIB) Thomas Radke (AEI)
Edward Seidel (AEI)

January 20, 2000

Abstract

Large scale simulations running in metacomputing environments face the problem of efficient file I/O. For efficiency it is desirable to write data locally, distributed across the computing environment, and then to minimize data transfer, i.e. reduce remote file access. Both aspects require I/O approaches which differ from existing paradigms.

For the data output of distributed simulations, one wants to use fast local parallel I/O for all participating nodes, producing a single distributed logical file, while keeping changes to the simulation code as small as possible. For reading the data file as in postprocessing and file based visualization, one wants to have efficient *partial* access to remote and distributed files, using a global naming scheme and efficient data caching, and again keeping the changes to the postprocessing code small.

However, all available software solutions require the entire data to be staged locally (involving possible data recombination and conversion), or suffer from the performance problems of remote or distributed file systems.

In this paper we show how to interface the HDF5 I/O library via its flexible Virtual File Driver layer to the Globus Data Grid. We show, that combining these two toolkits in a suitable way provides us with a new I/O framework, which allows efficient, secure, distributed and parallel file I/O in a metacomputing environment.

1 Introduction

The varied needs of large scale simulations are driving the computational science community to a computing paradigm that utilizes whatever resources can be brought to bear on the problem at hand. Very often local computing resources are insufficient to solve the problem on the desired scale. Further, remotely accessible resources at individual sites may individually be too small as

[‡]Max-Planck-Institut für Gravitationsphysik Potsdam/Golm Albert-Einstein-Institut
[§]Konrad-Zuse-Zentrum für Informationstechnik Berlin

well. However, *combined*, the distributed resources available to a research group, which may itself be distributed across the world, are clearly much larger than any single resource. In this metacomputing environment, one seeks to develop efficient techniques to harness a variety of computing resources across the 'Grid' [GRID], and to use them as a logically single machine. There are many issues to be addressed to make such simulation effective and efficient: the scheduling and acquisition of the different resources must be convenient; the simulation code must run efficiently on the different machines themselves; the algorithms must be appropriate for networked computers; issues of latency and bandwidth must be considered, etc.

In this article, we focus on approaches to file I/O in a metacomputing environment. The kinds of large scale simulations that require a Grid based solution often generate large scale data! The data need to be efficiently handled across the Grid, often by a group of researchers who are themselves distributed across the Grid. This brings up many new issues in I/O, including data layout, writing distributed files, and access to the distributed data for postprocessing.

A simulation running in a metacomputing environment has several ways to handle its file I/O. There are various points to be considered in choosing the most appropriate approach to I/O, e.g:

- One must balance the needs of simulation speed and ease of postprocessing, e.g. at what stage is data conversion accomplished, are expensive postprocessing steps included in the simulation code, etc.?
- Are there remote file systems (NFS, AFS, ...) available on all participating nodes, and are they compatible with each other?
- Have data to be located on one file system finally, or can they be distributed?
- Do we want to end up with a single data file, or can we split data over multiple files?
- Which I/O libraries are available at the different sites?
- Which file format is useful and accepted by postprocessing routines?

Often it turns out that the speed of the simulation is the most important factor[1], and thus everything, including file I/O, is optimized in regard to this. Even in a standard computing environment, I/O can be a major bottleneck in the simulation. In a metacomputing environment, it can become crippling if the standard *local* I/O paradigms are applied. Creating a *single* output file for all nodes necessarily involves a network transfer of major portions of the data, either explicitly by collecting data on one node and writing a single local file, or implicitly by utilizing remote file systems. Remote file systems like NFS and AFS often introduce additional problems: their performance is usually

[1] and usually the most expensive

lower than that of comparable local file systems (due to some administrative overhead), and they presuppose a tight organizational structure between all hosting institutions to allow access over the whole Grid. For systems which need quite some administration like DCE [OSF], this often turns out to be a real problem, and contradicts the flexibility of an ideal Grid environment.

If we put aside (for a moment) the desire to have a single compact output file, all compute nodes, regardless of their location, can use optimized local I/O libraries and spread the data over many files, distributed over the Grid, hence speeding up the simulation I/O routines, and subsequently the simulation. But then in postprocessing and visualization, the entire data from all machines must be moved to a single machine where they are recombined into some usual file layout. This would allow preexisting non-metacomputing software to process the generated data in the usual way, but at a huge cost. For datasets of order $10^9 - 10^{12}$ bytes, which can easily be generated in such an environment, this cost may be prohibitive.

It would therefore be desirable and much more efficient to make postprocessing mechanisms aware of the distributed character of the data. It would be even more efficient to then access only data *sub*sets, which really are required for postprocessing, in order to avoid unnecessary network communication. For example, some visualization procedure may only want to operate on a down-sampled dataset or sub-domain, demanding a much smaller network traffic than a complete data staging.

There exist approaches to solve all these problems on the file system level, introducing the concept of distributed parallel file systems. In contrast to merely remote accessible file systems like AFS, NFS and DCE and their client-server concept, distributed file systems are *physically* distributed, and all participating nodes are equivalent. We want to mention *ParFiSys*, a parallel file system from Madrid [PFS], and DPSS, the distributed-parallel storage system from LBL [DPSS]. Such approaches have following features in common:

- To the application, they act as normal *local* Unix file systems, even if parts of the hosted files are located remotely.
- Work load is distributed over multiple I/O nodes, providing very fast I/O even compared to local disks.
- They provide a global name space to all clients, just like NFS, AFS etc. do.
- Often they depend on special hardware infrastructure, and are limited to small local networks/Grids.

With the emergence of Grid infrastructures and corresponding toolkits, also the management of distributed data is assigned by Grid services. The Globus metacomputing toolkit [GLO1, GLO2] for instance provides mechanisms [GASS] to access remote data transparently (by substituting *fopen* and *fclose* calls) on software level. Newer developments like the *Data Grid* API will provide efficient

I/O interfaces to various storage systems, and also will allow the *partial* access of remote and distributed files.

Our goal is to show the usage of such Grid techniques on an application level. For our exemplary distributed computational environment, which is described in the next section, we designed and implemented such a solution based on an additional software layer between our I/O library (HDF5) and the storage system (DPSS and Unix file system), which handles the data distribution over the Grid.

2 Software Components

The DFN Gigabit Testbeds in Germany [GTB] form a unique opportunity to develop and test Grid applications in a network environment, which is going to be the standard environment for German research groups during the next couple of years. One of the pilot projects initiated by this Test Grid is *'TIKSL: TeleImmersion - Kollision schwarzer Löcher'*[2] [TIKSL].

In the course of the TIKSL project, a distributed environment for visualization the results delivered by the Cactus Computational Toolkit [CCT, ALL1] [ALL2, BEN, SEI1] simulations will be developed. Cactus is a environment for collaboratively developing high performance multidimensional numerical simulations, and was developed at AEI Potsdam, University of the Balearic Islands, NCSA, Washington University and other cooperation partners. With the Cactus scientific team itself distributed across the world, it provides an excellent project for partnering in the development of techniques to make use of distributed computing. From its history, it focuses on Numerical Relativity; the scientific part of the TIKSL project is especially focused on the simulation of black hole collisions.

The coalescence of two black holes is considered to be one of the most promising sources of gravitational waves [ANN, SEI2]. This problem is especially important now that LIGO [LIGO] and other physical experiments designed to detect gravitational waves for the first time may be able to detect such events during the next few years. The numerical simulations performed by Cactus are aimed at predictions and calibrations for such gravitational wave detection experiments.

Simulating the complete set of Einstein Equations for black hole collisions numerically requires big computational resources even for 'simple' problems: hundreds of equations and many dozens of grid functions have to be evolved on a sufficiently smooth and dense grid[3]. Computing time for typical problems and boundary conditions are several hours to days. During such simulations, binary output data of about tens to hundreds of GBytes are produced, at a peak rate of several 100 MBytes per second. These data are stored in local HDF5 files.

While Cactus is usually running on MPP systems like T3E and O2K, off-line visualization is usually performed on smaller visualization workstations. The

[2] *'Tele Immersion: Collision of Black Holes'*

[3] sufficiently dense means resolutions of about 256 and more in each spatial dimension

HDF5 files are therefore staged to these remote locations, and for distributed Cactus runs the data are recombined into a single data space and data file. They are then processed by Amira [AMI], an advanced visualization toolkit developed at ZIB Berlin, which directly reads HDF5 files via the HDF5 library and its Unix File Driver. Drawbacks of this solution are obviously the need for recombination of data originating from distributed runs, the (possibly multiple[4]) complete transfer of data, and the locally required storage capacities.

The goal of the TIKSL project and the solution presented here therefore include the capability to

- read data files remotely, at the server(s) hosting the simulation
- recombine the data on the fly
- write/read directly via the HDF5 library (for minimal code changes on simulation and visualization side)
- read data selectively, or stage partial files (usually not all data are visualized, thus there is no need to transfer all data!)
- read data without significant performance penalties (compared to local reads).

The next sections give a short overview about the crucial software components used in our environment, and describe our new approach for file I/O in detail.

2.1 The HDF5 Data Format and I/O Library

The *basic* file I/O in Cactus is focused on HDF5 [HDF1, HDF2], the *Hierarchical Data Format, release 5.* HDF5 is a file format and associated I/O library, designed for efficient, platform independent file I/O. HDF5, like its predecessors, has been developed at the National Center for Supercomputing Applications.

HDF5 was chosen as I/O library for several reasons:

- The HDF5 storage model is very simple – it consists of only two fundamental object types: *datasets*, which contain N-dimensional arrays of arbitrary data, and *groups*, that may be used to arrange related datasets in a hierarchical structure (similar to a Unix file system).
- Datasets are self-describing, their size and data type are stored, along with grouping information and optional user-supplied data (via attributes attached to datasets or groups), as metadata in the HDF5 file. That makes it possible to interpret structure and contents of a file without any external information.

[4]if multiple scientists want to visualize the data at different locations, the complete dataset has to be transfered to each of these locations.

- The library offers a variety of predefined standard data types as well as the capability of mapping arbitrary types or even structures into a portable file format representation.

- Its platform independence allows one to use HDF5 data files without the need of any explicit data conversion. Necessary type conversions are done on the fly when reading datasets – completely transparent to the application. This is especially useful on the Grid. HDF5 is ported to a wide variety of OS/machines and supports a large number of different hardware platforms.

- The HDF5 library implements sophisticated techniques for efficient and flexible data access/transfer, e.g. data compression, parallel I/O using the MPI-2 file I/O extensions, and mapping of '*hyperslabs*' as arbitrary subspaces into the global data space. The availability of hyperslab selections for read and write operations allows one to minimize the amount of data actually transferred from or to the storage system, which is particularly important for remote file accesses.

- The library layout is very modular and clearly structured, file I/O is encapsulated in a *Virtual File Driver* (VFD) layer. This layer provides the mapping of basic I/O routines used by the HDF5 library onto the respective API of the underlying storage systems. Users can add their own VFD, e.g. to implement transparent access to remote and distributed data, as shown in later sections.

- HDF5 already provides several approaches to process split (and potentially distributed and remote) files, like *External File Lists* (EFL), a split file driver, and a Unix file system-like mounting mechanism.

All in all HDF5 is a powerful file format and library. The library already provides a lot of functionality not available in other I/O libraries. It is quite likely, that HDF5 will become a widely accepted standard in the scientific computing domain like its predecessors CDF and HDF4. By enhancing the HDF5 library with support for efficient remote file access, we will enable many application programs to transparently perform I/O with full HDF5 functionality also on remote (and possibly distributed) files.

2.2 Globus and Data Grid

There are many possible ways to develop and enroll an application in a metacomputing environment. And there are many tools and toolkits available [LEG, PVM, CON, UNI], which are designed to support this task for programmers and users. The Globus metacomputing toolkit is certainly one of the most advanced toolkits available today. It provides a fundamental metacomputing infrastructure (like a global security layer, resource management tools and communication libraries) as well as a flexible and secure administrative framework[5].

The I/O scheme presented in this paper relies on one of the Globus components, the *Data Grid* [DG1, DG2]. The term 'Data Grid' describes distributed systems that *"support the integrated management and analysis of large (petabyte-scale) distributed data collections"*. The *Data Grid* Storage Client-API (SC-API) is designed to ease the development of Data Grid applications exploiting sophisticated techniques (e.g. network striping, parallel I/O, network protocol tuning) to achieve very high performance. The Globus *Data Grid* toolkit is supposed to provide platform independence[6] for such applications.

The *Data Grid* cannot be seen as a stand alone toolkit. To build a Data Grid Application, the programmer will typically use functions from the *Data Grid* Storage Client-API as well as from other *Data Grid* or Globus components (e.g., mechanisms for access to metadata catalogs, security, and computing resources) to implement a particular application.

While a file usually is regarded as an uninterpreted sequence of bytes which is located in a storage system, the term *logical file* as used by the *Data Grid* refers to an entity that has a globally unique name and that may also have associated with it metadata and one *or more* physical file instances. Logical files may be organized in collections. In this context, a storage system is an entity that responds to requests to read and/or write named file instances. Note that the term 'storage system' as used here denotes a logical construct and need not map directly to low-level storage. Additionally, the *Data Grid* provides the (semi automatic) possibility to create replicas and to buffer logical files on faster or closer storage sites. Such operations cause updates of metadata catalogs, like the replica catalog.

The *Data Grid* concept for logical files does not only provide a global name space for these files, but allows physically distributed files as well, since a logical file access is no longer restricted to *local* physical files, but may also refer to one or more files *somewhere* in the Grid. Accessing a certain part of this logical file results in the access of the associated file instance, whose true location or storage layout remains completely hidden from the application[7].

To enhance the HDF5 library with the ability to transparently access remote and distributed files, which are located somewhere in the Data Grid, we provide an HDF5 Virtual File Driver interfacing to the Globus Data Grid.

2.3 Amira Visualization System

Amira is a 3D visualization and volume modeling system for interactive applications in natural sciences, engineering and medicine [AMI]. The software system

[5]For more information on the many different parts of the Globus Toolkit see `http://www-fp.globus.org/details/`.

[6]that is: storage system independency

[7]The Data Grid toolkit is currently under development. Its functionality as statet in this paper is only partly implemented at the current state. This refers in particular to the replication and buffing mechanisms.

has been developed at the Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) and serves as a software platform for the development of working environments for specific applications domains (so-called virtual labs). A professional version of Amira is distributed commercially.

Amira is built in a modular, object-oriented fashion. It does not adhere to the data-flow concept. After loading data objects, they can be displayed or modified simply by selecting (from context menus) appropriate display or computational modules. Instantiated objects and their mutual dependencies appear automatically as a graphical network. By modifying this representation and selecting suitable parameters for active modules, the visualization process can be controlled with minimal amount of user interaction

The system's 3D graphics have been implemented using Open Inventor and OpenGL. Graphics hardware, if available, can be used to display large datasets at interactive speed. Amira displays its visual results in one or several graphics windows that allow users to interact directly, in three dimensions, with the depicted data objects. Different display techniques can be arbitrarily combined in a single window. All functions are script-able via a TCL-command interface.

The system currently comprises about 25 different data-object types and more than 100 modules that can be dynamically loaded at runtime. It can be easily extended by own additional modules. Currently, Amira intrinsically supports various grid types like hexahedral, curvilinear, and unstructured tetrahedral grids. AMR data types are currently under development.

At present Amira supports access to remote data by retrieving data carrying geometric information via a TCP socket connection. This method allows to inspect data from a running simulation program. E.g. the Cactus code is equipped with such an interface that allows sending of surfaces[8] or line information like particle paths. This was demonstrated during the SC99 conference in Portland, OR.

The current implementation uses some proprietary socket communication library and a data type specific API, which needs to be extended when the need for transfering new data types arises. A much more general concept is desireable for future online visualization of intermediate simulation results.

Several basic aspects of *streaming* of huge datasets on a Grid are comparable to the file I/O issues addressed in this paper:

- Self describing datasets based on simple primitive data types are favoured to provide a future-safe coupling of simulation and online visualization programs.
- Data conversion has to be done transparently, fast and on the fly, providing platform independency.
- Selection of partial data, like e.g. hyperslabs, can reduce the actual network traffic.

[8] Actually implemented are iso-surfaces which are computed on the supercomputer in parallel during the simulation run.

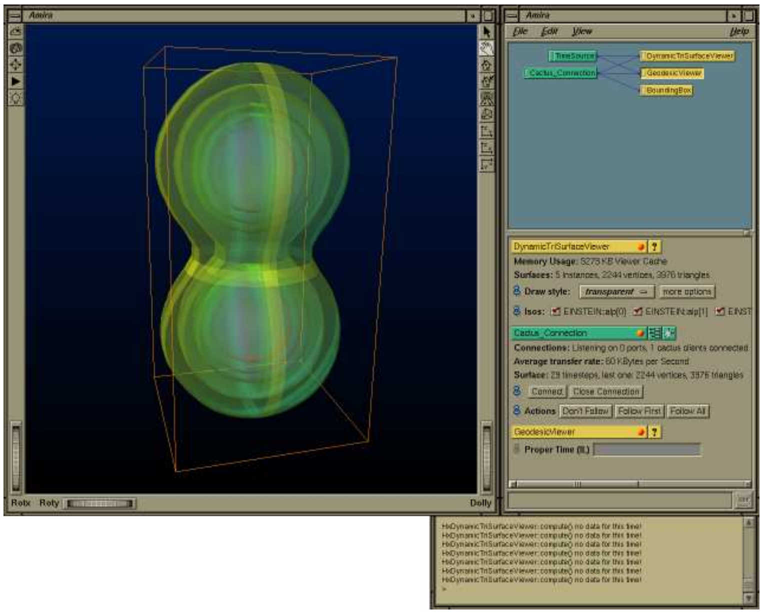

Figure 1: Amira workspace visualizing iso-surfaces, which are streamed from a remote Cactus simulation run.

All these issues are handled by the HDF5 library, but for file I/O only. It is therefore a reasonable consideration to use the HDF5 API as well for live data streaming as it is used for file access. In our work we also plan to provide an HDF5-compatible network streaming interface, either implemented as HDF5 replacement (which might still make use of some functionality of the original HDF5 library), or again as Virtual File Driver.

Some of the various problems we are facing in developing such a generic remote streaming library are possible metadata inconsistencies inside the HDF5 library, the design of a sensible *bi*-directional protocol when employing a VFD-based solution, and the general fact that HDF5 was designed with file I/O in mind, not for socket connections and asynchroneous data requests. This work will be the topic of a future report.

For now, the presented enhancement of HDF5 on *file* I/O level makes Amira and every other HDF5 compatible postprocessing tool Grid-aware in a sense, that it allows the simple and efficient handling of huge distributed data sets.

3 Adding Distributed File I/O to HDF5

As described before, the HDF5 library in its current distribution provides access to local files only. For use in a metacomputing environment we want to extend its file I/O interface in order to handle *remote* data as well. Because of the clear functional structure of the HDF5 library this means simply to add another Virtual File Driver that is capable of performing efficient remote file I/O. This driver would make use of the Globus Data Grid API and thus provide the user with an interface to access HDF5 files located anywhere in the Data Grid.

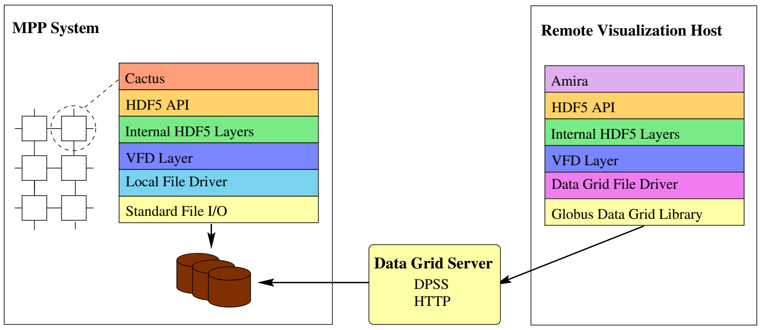

Figure 2: Files written with some local File Driver of HDF5 are off-line accessible via the Globus/Data Grid File Driver from remote hosts, hiding their possibly splitted and distributed nature. The VFD Layer in HDF5 completely conceals the underlying I/O infrastructure to the application.

A second task is to support *distributed* file I/O. Under the premise that the simulation should be allowed to write datasets in the most efficient way (i.e. write chunked data into local files), the visualization must then be able to establish a global view on the complete data space as originally seen by the simulation program. There exist two possible approaches to reconstruct this gobal data space. Both can be implemented using the features of the HDF5 library and the Globus Data Grid:

- The simulation handles the physical distribution of datasets within the global data space as a set of *segments* (offsets and sizes), which are written to one or more files local to that processor. These are then referred to in a HDF5 *External File List* (EFL).

  The EFL contains the filenames as URLs to qualify their distribution over the Data Grid.

  Once the EFL is completely specified, it is automatically stored as meta-data information in the HDF5 files, and can be immediately used by the visualization program to read the distributed remote data via a Data Grid VFD – no additional preparations to address remote files are needed here.

  The disadvantage of this approach is the potential negative impact on the I/O performance when writing chunks of data in a heterogeneous environment: since external files do not contain any self-describing type information, the simulation must use a common data type representation throughout the whole application, which would make data conversions necessary on those nodes with a differing byte order. Right now, files specified in EFLs do also not support hyperslab selections, and are reopened for each access. This needs to be modified.

- Another possibility is to write each chunk of data independently as a single dataset into a separate HDF5 file. This gives the simulation the freedom to choose the easiest and most efficient method of outputting its chunked data (eg. local optimizations can be applied by using parallel I/O for a shared file on all nodes that reside on the same machine).

  Of course, the simulation has also to provide the visualization with some description of the physical distribution of the files (analogous to the EFL). This can be done by storing the filenames as complete URLs in attributes attached to each of the HDF5 files or its datasets.

  On the visualization side all chunked files are logically linked together to rebuild the original global data space. For linking the *mounting* feature of HDF5 can be used, which allows to mount external HDF5 files into the group hierarchy of the currently opened file (similar to mounting file systems in a Unix environment).

3.1 Current Implementation

As a first step towards the realization of this global picture, the HDF5 library was made Data Grid-aware by adding a new virtual File Driver to the VFD layer. This Data Grid VFD maps the basic I/O operations onto the Data Grid SC-API routines. Depending on the type of URL passed to the open call, the Data Grid Storage Client library automatically selects the appropriate low level protocol driver for remote data transfers. The Data Grid supports several protocols which can be addressed by the URLs passed to the Data Grid VFD:

- `file://localhost/<path>`
  for processing files located on the local machine's Unix file system

- `x-dpss://<dpss_host>/<dpss_path>`
  for processing remote files that reside on a a scalable, high-performance, distributed-parallel data storage system (DPSS)

- `http://<www_host>/<www_path>`
  for processing files that are accessable from any Web server.
  The Data Grid HTTP driver can read files only sequentially by now but will provide random file access also in its next version. For efficient partial file accesses (which are important for hyperslab selections) the driver will then make use of the extended features offered by HTTP 1.1.

The Ftp protocol is supported as well by the Grid Storage API but cannot be used for our purposes because it does not allow random file access.

Before opening a remote HDF5 file, the Data Grid VFD has to be selected by modifying the default file access property list passed to the HDF5 file open call. Subsequently all I/O operations on this file like opening, closing, reading, and writing of both metadata and raw data are directed to this VFD, which in turn calls the appropriate Storage Client API routines according to the file's URL and the low level protocol specified therein.

As discussed above, we are interested not only in remote reading of distributed datasets, but we want complete flexibility in choosing general subsections of files (downsampling, choosing arbitrary subregions, etc.), which are called hyperslabs. This may be important not only to reduce data transfer over a network, but also because for scientific or visualization reasons it maybe necessary to examine arbitrary subparts of the dataset. When performing I/O on a hyperslab selection, it may happen that a single HDF5 API operation actually consists of many basic reads or writes, which are all passed to the VFD layer *individually*; e.g. a hyperslab read of a 10x10 dataset with down-sampling applied to every second element in each dimension would cause the VFD to execute 50 individual reads of single elements (compared to only one block read when accessing the full dataset). Such hyperslab operations would probably show very poor performance if done in a conventional way because the latencies of each individual read/write passed over the network would sum up to unacceptable delays. Therefore the hyperslab selection code inside the HDF5 library was extended to pass hints down to the VFD layer, which indicate if the current read/write call belongs to a certain hyperslab selection, and more *associated* reads/writes will follow. The Data Grid VFD now can make use of these hints and optimize remote I/O operations by accumulating all subsequent individual reads/writes into a single transaction, representing the complete hyperslab I/O transfer. Such a transaction is then passed to the Grid Storage client en bloc, which in turn sends it to the Grid Storage server where it is executed at once, and the results are sent back to the client in a single data stream.

As soon as the Grid Storage API can handle such transaction requests in addition to individual reads/writes, the already implemented code modifications in the HDF5 library can be enabled to improve I/O performance for hyperslab selections.

A prototype of the Data Grid VFD was demonstrated at the SC99 conference at Portland, OR in October 1999 (see Fig. 3). Amira used this driver to visualize datasets from a simulation of colliding neutron stars and collapsing gravitational waves. The datasets in gigabyte range had been previously generated by Cactus. The HDF5 output files were located on a Distributed Parallel Storage System (DPSS) at the Argonne National Laboratory, but had also been transfered to the local visualization workstation (driving an I-Desk on the show-floor at Portland).

It was demonstrated that during the read of individual time steps one could switch between the remote file and its local copy – thus proving complete transparency of remote and local I/O at application level, with only network delay added when fetching the data from remote. Due to the available bandwidth of the OC-12 network connection between Portland (OR) and Argonne (IL), and the overall latency minimization due to optimization of HDF5's buffer usage for receiving and converting data, it was even possible to successfully present a smooth animation of the gravitational wave collapsing to a black hole.

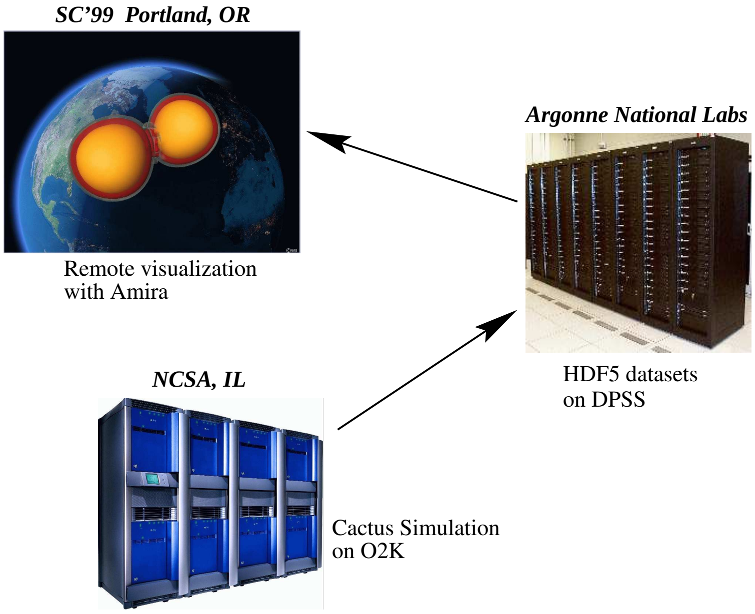

Figure 3: The remote visualization demo as presented at SC99.

3.2 Future Work

The current implementation is capable of accessing single remote files, but the datasets are still required to be written as unchunked data. To overcome this limitation, our future work will focus on an I/O scheme that allows the simulation to output its data as chunks into distributed files, using parallel I/O for node-local files. For the visualization this means that it must be able to reconstruct the global data space from all chunks, without the need for staging any physical data. Read accesses are then performed on this global data space.

Since none of the approaches described in section 3 leading to such distributed I/O schemes has a clear advantage over the other, we decided to implement both of them. They require enhancements within the HDF5 library itself as well as additional code functionality on top of it in order to establish a consistent view on the complete data space as seen by the simulation and the visualization.

Currently, an external file list may contain only local files, because all operations on an EFL file are mapped directly onto standard Unix I/O calls, bypassing the VFD layer of HDF5. This has to be changed in the future to allow EFL entries to refer to Data Grid URLs instead of local files.

Currently I/O operations on external files are implemented quite inefficient: for each access the file is reopened, and closed afterwards. Adding functionality for managing state information of already opened external files will help to improve performance dramatically.

The second approach of mounting independent chunked HDF5 files together requires a new abstraction layer for transfering distributed data, because the HDF5 data mapping can be applied to *single* datasets only. Since the global data space is now distributed over several data sets (chunks), the visualization would no longer be able to transparently select data from *somewhere* in this global data space, but would have to take its distribution into account. Thus an interface is necessary that is able to operate on such groups of chunked datasets, iterating over each chunk while applying the appropriate partial hyperslab selection.

Ideally such functionality would be integrated in a HDF5 wrapper on top of the HDF5 library. Most of the implementation work will probably comprise the re-implementation of the HDF5 hyperslab selection code. We hope that much of the existing code from the HDF5 library can be reused here.

4 Summary

We presented a solution to the problem of efficient file I/O in Grid Environments, where a large-scale scientific application running in a distributed computing environment produces huge amounts of data, that are to be postprocessed.

Our approach is to allow the simulation program to output its data locally, using well known efficient local I/O paradigms. This results in a data layout that is distributed over several physical files within the Data Grid[9]. For post-processing a global view on the original data is re-established, without the need of explicitly staging all data locally. Hyperslab selections can be applied on this global data space in order to fetch only those data which are really needed locally, e.g. for visualization purposes.

We use the Globus Metacomputing Toolkit with its Data Grid component to manage the remote files, and HDF5 as the underlying I/O library and data format for accessing such distributed data. A new driver was added to the Virtual File Driver layer of the HDF5 library which provides efficient and transparent access to remote files, based on the Data Grid library. A prototype version of this VFD was presented at the SC99 conference in Portland (OR), where we successfully demonstrated the visualization of datasets from a huge recombined remote HDF5 file located on a high-performance storage system at Argonne (IL).

Our future work will enable this version to handle data sets distributed over multiple files, using HDF5's features to specify a global data space either via an External File List, or by mounting separate HDF5 files together.

[9]In terms of the Data Grid the data are stored in a single logical file, which refers to multiple (distributed) file instances.

Acknowledgments

It is a pleasure to thank many colleagues and collaborators who contributed to this work, both directly and indirectly. Mike Folk and Quincey Koziol from the HDF5 development team have worked closely with us on this project, and hosted one of us (TR) for an extended visit to NCSA in advance of SC99. Ian Foster, Brian Toonen, Steve Tuecke, and others from the Globus team at ANL provided much help, particularly on the Data Grid work, and also hosted a visit of TR. Gabrielle Allen, Tom Goodale, Gerd Lanfermann, and Joan Massó of the Cactus development team at AEI and the University of the Balearic Islands have been essential in providing time and energy in Cactus support for the project. Detlef Stalling and Malte Zöckler of the Amira development team have been important collaborators in the visualization work. John Shalf at NCSA has been an excellent collaborator, providing many ideas and expertise in all areas, but especially related to this project, on general I/O issues. Jason Novotny at NLANR has also contributed, especially on issues involving Globus and user interfaces. Computing resources and general support, financial and otherwise, have been provided by AEI, ANL, NCSA, RZG and ZIB. Finally, and most importantly, the project is directly supported by DFN/BMBF, both through financial support and provision of the Gigabit Testbed.

References

[ALL1] G. Allen, T. Goodale, E. Seidel, *The Cactus Computational Collaboratory: Enabling Technologies for Relativistic Astrophysics, and a Toolkit for Solving PDEs by Communities in Science and Engineering.* IEEE 7th Symp. on the Frontiers of Massively Parallel Computation-Frontiers 99 (1999)

[ALL2] G. Allen, W. Benger, C. Hege, J. Massó, A. Merzky, T. Radke, E. Seidel, J. Shalf, *Solving Einstein's Equations on Supercomputers.* IEEE Computer (in press) (1999)

[ANN] P. Anninos, J. Massó, E. Seidel, W.-M. Suen, *Numerical Relativity and Black Holes.* Physics World **9 (7)** (1996) 43-48

[AMI] *Amira - an Advanced 3D Visualization and Volume Modeling System,* `http://www.amiravis.com` or `http://amira.zib.de`, (1999)

[BEN] W. Benger, I. Foster, J. Novotny, E. Seidel, J. Shalf, W. Smith, P. Walker, *Numerical Relativity in a Distributed Environment.* Proc. Ninth SIAM Conference on Parallel Processing for Scientific Computing (1999)

[BH] P. Anninos, D. Hobill, E. Seidel, L. Smarr, and W.-M. Suen, *When Black Holes Collide,* Proc. Sixth Canadian Conference on General Relativity and

Relativistic Astrophysics, Fields Institute Comunications **54** (1997) 151

[CCT] *The Cactus Code Server,* `http://www.cactuscode.org`, (1999)

[CON] M. Litzkow, M. Livny, and M. W. Mutka, *Condor - A Hunter of Idle Workstations,* Proc. 8th Int. Conf. of Distributed Computing Systems (1988) 104-111

[DG1] 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,* (1999) Submitted to NetStore '99

[DG2] A. Cervenak, I. Foster, C. Kesselmann, C. Salisbury, S. Tuecke, *Storage Client API Specification,* unpublished (1999)

[DPSS] B. Tierney, J. Lee, B. Crowley, M. Holding, J. Hylton, F. Drake, *A Network-Aware Distributed Storage Cache for Data Intensive Environments,* Proc. IEEE High Performance Distributed Computing (HPDC-8) (1999)

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

[GLO1] I. Foster, C. Kesselman, *Globus: A Metacomputing Infrastructure Toolkit,* Int. J. Supercomputer Applications, **11(2)** (1997) 115-128

[GLO2] I. Foster, C. Kesselman, *The Globus Project: A Status Report,* Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop, 4-18 (1998)

[GRID] I. Foster, C. Kesselman (Editors), *The Grid: Blueprint for a Future Computing Infrastructure,* Morgan-Kaufmann (1999)

[GTB] *DFN Gigabit Testbeds,* `http://www.dfn.de/projekte/gigabit/home.html`, (1999)

[HDF1] *A User's Guide for HDF5,* `http://hdf.ncsa.uiuc.edu/HDF5/doc/H5.user.html`, (1999)

[HDF2] *HDF5: API Specification Reference Manual,* `http://hdf.ncsa.uiuc.edu/HDF5/doc/RM_H5Front.html`, (1999)

[LEG] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, P. F. Reynolds Jr, *A Synopsis of the Legion Project,* UVa CS Technical Report CS-94-20 (1994)

[LIGO] A. A. Abromovici, et. al, *Ligo: the laser interferometer gravitational wave observatory,* Science, **256** (1992) 325

[OSF] The Open Software Foundation *Introduction to OSF DCE,* PTR Prentice Hall, Englewood Cliffs, New Jersey (1988 - 1991)

[PFS] J. Carretero, F. Pérez, P. de Miguel, F. Garca, L. Alonso, *ParFiSys: A Parallel File System for MPP,* ACM SIGOPS **30(2)** (1996) 74-80

[PVM] V. S. Sunderam, *PVM: A Framework for Parallel Distributed Computing,* Concurrency: Practice and Experience, **2(4)** (1990) 315–339

[SEI1] E. Seidel, *Technologies for Collaborative, Large Scale Simulation in Astrophysics and a General Toolkit for solving PDE's in Science and Engineering.* in T. Plesser, P. Wittenburg (Editors), *Forschung und wissenschaftliches Rechnen.* Heinz-Billing-Preis Essay, Max-Planck-Gesselschaft München (1999)

[SEI2] E. Seidel, *Black Hole Coalescence and Mergers: Review, Status, and "Where are we heading?"* Progress of Theoretical Physics (in press) (2000)

[TIKSL] *Tele Immersion: Collision of Black Holes - A GTB Project,* `http://www.zib.de/Visual/projects/TIKSL/`, (1999)

[UNI] M. Romberg, *The UNICORE Architecture: Seamless Access to Distributed Resources,* Proc. IEEE High Performance Distributed Computing (HPDC-8) (1999)