Active Logistical State Management in GridSolve/L

Micah Beck, Jack Dongarra, Jian Huang, Terry Moore and James S. Plank
{mbeck, dongarra, huangj, tmoore, plan}@cs.utk.edu
Computer Science Dept., University of Tennessee
1122 Volunteer Blvd, Suite 203, Knoxville, TN, 37996-3450

1. Introduction

More than two decades of exponential growth in the supply of the basic computing resources, rapid progress in the deployment of high performance networks, and escalating requirements for distributed collaboration in Computational Science, has caused expectations of the benefits of distributed systems and grid computing environments to soar [1]. It is unsettling, therefore, to recognize that the critical problem of managing the data and process state in such systems [2], especially in the wide area, remains unsolved, and that its effects continue to raise chronic performance and scalability issues for all types of network computing applications. More than five years ago, an NSF workshop on *Distributed State Management* concluded the following:

> "Given the extensive advances in distributed systems technology that have occurred over the last two decades, the number of different distributed applications that have been deployed has been surprisingly small. One reason for this is that many difficult problems (e.g., consistency management, fault tolerance, security, location transparency, scalability, and performance) must be addressed to build a complex distributed application, and there is very little infrastructure available to ease the effort. In addition, the amount of bandwidth being consumed on the Internet for the WWW is increasing exponentially. Both of these observations suggest that a serious effort to build an infrastructure for wide-area state sharing is highly recommended." [3]

A general solution to this problem has yet to be found. Moreover, the advent of data intensive computing exacerbates the situation. To amass the storage and compute services required to manage the flows of state involved in data-intensive applications, current distributed systems must either utilize application specific resources, or, as with most grid applications, leverage a shared "grid fabric" of metacomputing resources [4] via least common denominator middleware. In this context what distinguishes our research program in Logistical Computing and Internetworking (LoCI), and the work on Logistical Networking (LN) [5] at its core, is our insistence on the idea that this fabric itself must be rethought and rearchitected to overcome the constraints imposed by entrenched legacy designs [6].

For the past few years, we have been investigating this idea in a collaborative research effort having two complementary strands. One strand is focused on research and development of LN technology; the other is centered on exploring LN's power to enhance the functionality and performance of NetSolve, a flexible, powerful, and easy to use grid computing environment. Together they provide an ideal vehicle for investigating an approach to grid computing based on a type of grid fabric specially adapted to deal with the logistics of distributed data management, i.e. one that uses LN.

LN brings data transmission and storage within one framework, much as military or industrial logistics treat transportation lines and storage depots as coordinate elements of one infrastructure. Its main challenge has been to realize this idea without sacrificing network-style scalability. Following two enduring examples of successful system architecture — the Internet protocol and the Unix OS kernel interface — we have created a new type of shared network storage that can scale globally [7]. LN builds on a generic, best effort network storage service based on the Internet Backplane Protocol (IBP) [8] to infuse the network itself with storage resources that can be shared, scaled up, and exposed for external scheduling much as IP datagram service is.

The NetSolve grid environment [9, 10] provides an excellent context for exploring the use of LN's storage service to manage the logistics of data and process state in distributed applications [11]. Though NetSolve is more advanced than other "network enabled server" approaches to grid computing in its ease of use and the range of scientific software libraries it supports, the brokered remote procedure call (RPC) mechanism at its core is focused primarily on the selection of trusted computational server, the transfer of arguments from the client to the server, the invocation and monitoring of the remote call-out and the transfer of results back to the client. Taken naively, this mechanism leaves out an important component of distributed applications: the management of state before, during and after any single call.

State management is important when an RPC is not made in isolation, but is part of a larger application. The context of a larger application can allow the call-out to be anticipated before the actual call is made, and perhaps when some arguments are already available. Movement of some arguments in advance of the call can then allow a type of "logistical" quality of service to be obtained [11]. When a sequence of RPCs is made with data dependences between them, the results of one call serves as the input to another call (flow dependence), or the input to one call is identical to the input to another (input dependence). Similar advantages can be gained when a long-running call generates a checkpoint, which is, in effect, a large "early" result that may be passed to another call, which is the resumption of the original call.

In all these cases, state management requires the use of storage (albeit temporary), and this storage must be in proximity to the NetSolve computational servers. Now according to the usual approach to grid computing, which was embodied by the early versions of NetSolve, the only physical storage available in the grid fabric is in that part we distinguish as the *metacomputing fabric*. The metacomputing fabric is composed of those resources in computer centers, and departmental clusters, which are shared in a highly controlled way, among a relatively select community, with authentication, access control and even pure use accounting as leading design issues. By contrast, the elements of the *network resource fabric* are designed to be shared on the model of IP datagram service, i.e. in a relatively unbrokered manner among the entire community, minimizing admission and accounting restrictions as much as possible. On the prevailing model, this part of grid fabric is composed primarily of cables and routers, with no open access to storage resources. The absence of any storage which is shared Internet style, and the corresponding requirement that trusted (and sometimes substantial) storage be available from the metacomputing fabric in the relevant locations, represents a significant inhibitor to the deployability of grid services such as NetSolve.

By implementing an Internet-like model of resource sharing for storage [12], IBP enables LN to put a scalable storage service into the network resource fabric. With this foundation, the diverse state management requirements of NetSolve applications are a perfect match for the use of LN. Since LN infrastructure need not be trusted and is inherently temporary in nature, it can be deployed much more scalably. In turn, unbrokered access to ubiquitous storage services makes NetSolve and its applications much more deployable. This natural fit has led to the integration of LN storage mechanisms into the data management layer of recent releases of NetSolve. Substantial results have been obtained from experiments with this grid environment in the prestaging of inputs in advance of calls [13], in the resolution of data dependences [14] and in supporting temporary files of all kinds[15, 16], including checkpoints [17].

But although the success of LN as a scalable mechanism for state management in NetSolve is a starting point, it in no way addresses the complete state management needs of NetSolve or of other grid computing environments. That is because state management requires more than simple storage; it requires active management of data, meaning that stored data must be moved and must be transformed. The active nature of state management is often obscured by the fact that the processing is hidden in conventional storage systems, being performed in operating systems, disk controllers or even in manual or semi-automated administrative procedures. Thus, to the user of file systems and other high level storage-based services, data seems to be maintained through a purely passive but somehow incredibly reliable and high performance service. Examples of active components of storage systems are:

o Redundant encoding of data and striping across disks.

o Recovery of data in the face of failure and restoration of necessary levels of hardware provisioning

o Backup of data to highly reliable and massive storage systems and restoration to fast media on demand.

In distributed applications, the active nature of storage management is more acute because of changing circumstances. For instance, data may not be encoded in the same way when being transferred across the network as when it is stored over long periods on disk.

Grid Computing environments require the active component of state management just as much as conventional Data Centers do, and absent a generic computing service in the network resource fabric, this work must be done by resources in the metacomputing part of the fabric. For example, the specialized storage management components of projects such as Condor can provide this service [18], but this increases the balkanization of services produced by the provisioning of storage in such components. Another approach might be to use available clusters and supercomputers. This is usually overkill, however, as a means of implementing the simple operations needed for active state management; it also requires that data be moved to the locations where trusted computational servers are provisioned. But the most common approach is simply to make use of large, centralized systems such as HPSS. This approach, unfortunately, relegates Grid state management to the same centralized environments from which users were supposed to be liberated by Grid Computing.

In Active Logistical State Management (ALSM), we extend the Internet paradigm from storage to state management, making data transformation a sharable part of the network. Our hypothesis is as follows:

> *If, by adhering to end-to-end principles, a generic, best effort computing service can be integrated into a globally scalable logistical network, then a* ***programmable*** *resource fabric can be created that is capable of providing the kind of active state management that grid applications require.*

To test this hypothesis we have created a new abstraction of processor resources and a primitive computing service, called the *Network Functional Unit* (*NFU*) [19]. Like IP and IBP, the NFU is designed for global scalability so that it can be implemented as part of the enriched network resource fabric that LN has already created. Our test environment is a new version of NetSolve, called *GridSolve/L*. Using GridSolve/L, we are working with a distributed application that is well suited to drive ALSM application requirements, in the area of scientific visualization. Below we explain the background and motivation for ALSM and describe the initial application work we are doing to explore its potential and its limitations.

2. A Generic Data Transformation Service for ALSM

Since computation is inherently more complex than storage because of the variety of services and operations that can be applied, research on ALSM is inherently more challenging than our previous work on LN. It is important to note at the outset, however, the success of LN research on network storage supplies the work on network computation for ALSM with a tremendous advantage that previous efforts in this general area have not possessed — a generic storage service, exposed for application scheduling and scalable to the wide area. The presence of such a storage element in the network resource fabric opens up the possibility of attacking the problem of globally scalable network processing in a new way [19].

LN architecture [7, 20, 21] provides essential context for ALSM for GridSolve/L.In designing a scalable network data transformation service for ALSM, we follow a methodology that is simple, but which reverses the typical order of thought in the design of distributed systems. Rather than starting with an idea of what level of functionality we require of the network, or what sort of intermediate nodes we want to build, we start with the requirement that the system scale, using adherence to the end-to-end principles as the means by which such scalability can be achieved. This requirement is stringent enough to dictate most of the features of the basic service.

In particular, conformity to the end-to-end principles requires that the semantics of such a service be simple and weak [22]. If the semantics are too complex, it will fail the requirement that services implemented at intermediate nodes be generic. Likewise, if the service makes guarantees that are too strong, then it will not compose with a scalable communication network, like the Internet, without breaking. This approach was followed in the design of IP for wide area communication, and in the design of the Internet Backplane Protocol (IBP) for wide area storage services [7]. The crucial step is to define the right basic abstraction of the physical resource to be shared at the lowest levels of the stack.

We call the new abstraction of computational resources, which is to be added as an orthogonal extension to the functionality of IBP, the *Network Functional Unit* (*NFU*). The name "Network Functional Unit" was chosen to fit the pattern established by other components of the LN infrastructure. This pattern expresses an underlying vision of the network as a primitive computing platform with exposed resources that are externally scheduled by endpoints. The archetype here is a more conventional network: the system bus of a single computer (historically implemented as a backplane bus), which provides a uniform fabric for storing and moving data. This analysis inspired the name for LN's fundamental protocol for data transfer and storage, the "Internet Backplane Protocol." Extending the analogy to include processing, we looked for that component of a computer that has no part in data transfer or storage, serving only to transform data placed within its reach. The Arithmetic Logic Unit (ALU) seemed a good model, with its input and output latches serving as its only interfaces to the larger system. Hence the component of an IBP depot that transforms data stored at that depot is called the Network Functional Unit.

Just as IP is a more abstract service based on link layer datagram delivery, and IBP is a more abstract service based on "access layer" block storage, the NFU is a more abstract service based on "execution layer" computational fragments (e.g. OS time slices) that are managed as generic "operations." "Execution layer" is our term for the data transformation service at the local level, including all the different possible execution platforms and all their various individual attributes, e.g. fixed time slice, differing failure modes, local architecture and operating system. In all three cases, the generic service is made independent of the particular attributes of the underlying link/access/execution layer services beneath it by adopting semantics that hide the particularities of the units of service, failure models, and addressing schemes which belong to the services in the lower layers.

Just as IP datagram service and IBP byte-array service allow a uniform model to be applied globally to transmission and storage resources respectively, the higher level "operation" abstraction allows a uniform NFU model to be applied to processor resources globally. This step is essential to creating the most important difference between execution layer computation slices and NFU operation service: Through the use of the NFU operation service, any participant in an ALSM network can make use of any execution layer computational resource in the network regardless of who owns it. The use of IP networking, supplemented by IBP storage, to access NFU processor resources creates a global data transformation service.

2.1 A Best-effort Data Transformation Service for ALSM

Whatever the strengths of this application of the Internet paradigm to data transformation, however, it leads directly to two problems. First, the chronic vulnerability of IP networks and Logistical Networking to Denial of Service (DoS) attacks on bandwidth and storage resources respectively applies equally to the NFU's computational resources. Second, the classic definition of a time slice execution service is based on execution on a local processor, so it includes strong semantics that are difficult to implement in the wide area network. Following the example of the Internet and LN, we address, or at least partially address, both of these issues through special characteristics of the IBP/NFU allocation strategy: by default the NFU offers only a time limited unit of service and provides only best effort availability. In specifying the service semantics of the different dimensions of NFU service, we can use corresponding design aspects of IP and IBP as guides because they are all based on the same model. Accordingly, the five dimensions of the NFU's weakened service semantics to be addressed are fragmentation, availability, statelessness, correctness and security.

- *Fragmentation* — Scalable services for data transmission, storage, and transformation in wide area systems must place limitations on the maximum size of their unit of service, and therefore inherently involve service fragmentation. In all these cases, the reason for limiting the maximum unit of service is that otherwise it is much more difficult to *share* the resources of the intermediate node, and hence much more difficult to make the service scale up in terms of the number of nodes and users. Unlike the traditional model of computation, fragmentation means that we must view our ALSM service as transforming stored state within the NFU enabled depot, or *active depot*, perhaps completing only a part of the "job" or "call" ultimately intended by the end user. To complete extended operations, multiple computational fragments will have to be applied, either at a single active depot or using many depots, with attendant movement of state for fault tolerance and possibly parallelism. Dealing with the consequences of fragmentation, in particular in overcoming operation latencies through *caching of control structures and pipelining calls*, forms a significant part our research challenge.

- *Availability* — As with IP and IBP, a best-effort data transformation service has to allow for the unavailability of a given service node. In order for the work to proceed when a particular node becomes unavailable, redundant resources must be used to perform the transformation on some other node. This means that the data on which the service acts must be accessible even in the presence of failures in nodes that may both store and process; such state-management strategies fall into the domain of LN. Although the need to find alternate active nodes and retrieve stored data from remote sources means that applications cannot rely on predictable delays, the service seen by end-users must be acceptably reliable

- *Statelessness* — The ability of Internet endpoints to use the best-effort services of routers to construct stronger services end-to-end relies partly on the fact that intermediate nodes are stateless, so that the construction of an alternate route affects only performance characteristics seen at the end-points. Following a similar philosophy, the only state a depot maintains is the state required in order to implement its basic services —allocation of storage, writing and reading. It maintains no additional state visible to the endpoint, so that depots appear interchangeable to the endpoint, except for the data that they actually store. Endpoints carry only the burden of keeping track of data stored at depots, not of managing any other visible state. Like storage, the transformation of stored state can only be performed at an intermediate node that can maintain persistent state. This is the reason that the NFU is a natural extension of the IBP depot. To minimize control state, however, the active services implemented there must not maintain any visible state other than that which they transform.

- *Correctness and Security* — Trustworthiness of intermediate nodes in network services is an important issue. The fewer assumptions that are made about the trust placed in intermediate nodes, the more scalable the resulting network. An IP network represents an extreme case: it makes only probabilistic correctness assumptions and no security assumptions, but then checks correctness through end-to-end checksums and assures privacy through end-to-end encryption. In some cases, these same techniques can apply to ALSM; but in cases where complex data transformations are implemented at the active depot, some level of trustworthiness must be assumed. This will reduce the scalability of the LN component of our system to some degree.

We have several ways to minimize this effect. One approach is to try and push data transformations onto a GridSolve server that is authenticated and trusted. Another is to replicate data and perform transformations redundantly, checking the results; this is done in some peer-to-peer computing systems [23, 24]. Sometimes, applications-specific checks can be applied, but this approach places a burden on the application programmer. All of these techniques allow us to work with less trustworthy depots when that is an important consideration.

If the key to developing scalable ALSM is fragmentation and weakening of the data transformation service provided at an active depot, then the complementary research program must consist in building up the necessary strong services and guarantees at the endpoints. The fundamental tools of such aggregation are the reduction of high level application algorithms to sets of operations and necessary communication, respecting the dependences between them. Further research topics involve increasing reliability through redundant operations. Overall, the limits of expressiveness and performance must be probed and mapped, to ultimately discover the limitations as well as the promise of a scalable ALSM.

3. Active Depot and Middleware Research

3.1.1 Design of a Computational Intermediate Node: The NFU-enabled Depot

The NFU is a module that is added to an IBP storage depot in order to transform stored data (Figure 1). We call a depot so modified an *active depot*. Restricting an operation to data held in RAM forces any necessary movement between disk and RAM to be explicitly directed by the end-point using IBP, just as in data movement between depots. In this section we give a conceptual overview of a possible NFU interface.

The NFU implements a single additional operation, `IBP_nfu_op`:

```
IBP_nfu_op(depot,
  opcode, nPara,
 *paras, timeout)
```

The `IBP_nfu_op` call in this simplified form is used to invoke an operation at the IBP depot, specified by the IP address and port it binds to. The operation is specified as an integer argument, whose meaning is set by a global registry of operation numbers. The arguments to the operations consist of a list of IBP capabilities (cryptographically secure names) for storage allocations on the same depot on which the operations are being performed. Thus, there is no implicit network communication performed by a given depot in response to an `IBP_nfu_op` call. A flag indicates whether the operation is soft (using only idle cycles). The capabilities specified in this call can enable reading or writing, and the limitations of each are reflected in the allowed use of the underlying storage as input or output parameters. The number and type of each capability are part of the signature of the operation, specified at the time the operation number is registered. Any violation of this type information (for instance, passing a read capability for an output parameter) may cause a runtime error, but it is not checked by the implementation of `IBP_nfu_op` at the client side. Such effects as aliasing between capabilities are also not detected.

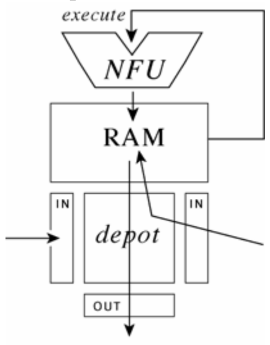

Figure 1: Active IBP Depot

There are a number of important and obvious refinements to this call that give it more structure and can in some instances make correct use and efficiency more likely (such as handling scalar arguments and storage capabilities differently). They have been omitted for brevity and clarity. An important difference between `IBP_nfu_op` and remote procedure call mechanisms is that since data is assumed to already be stored in capabilities, all issues of data representation and type are pushed onto

the end points and operations themselves, rather than being part of the `IBP_nfu_op` call.
The behavior of the active depot in response to an `IBP_nfu_op` operation is to map the specified capabilities in to an address space, look up the operation in a library of dynamically invoked calls, and invoke it. The set of operations implemented at each depot is determined by local policy. Because the storage allocations are local to the depot, simple memory mapping operations can be used, obviating the need for any data movement when invoking an operation. This optimization limits the implementation of `IBP_nfu_op` to active depots implemented in RAM or running on operating systems that can map disk storage directly to a process address space. The fragmented nature of the operation can be enforced in two ways: by refusing to add to the active depot library any call that does not provide a strict limit on resource use, and/or by monitoring resource use and terminating execution when the limit is reached.

3.1.2 A Data Structure for the Flexible Aggregation of Network Computation

From the point of view of the Network Computing community, it is likely that one of the most striking features of ALSM is the way it appears to simply jettison the well known methods of usage for computation, viz. processes invoked to run programs or execute procedure calls to completion. These familiar abstractions can be supported in the logistical paradigm, but that support must conform to its “exposed-resource” design philosophy embodied in the end-to-end principles. As with IBP-based storage, implementing computing abstractions with strong properties — reliability, fast access, unbounded size, etc.— involves creating a construct at a higher layer that aggregates more primitive NFU operations below it, where these operations and the state they transform are often distributed at multiple locations. For example, when data is stored redundantly using a RAID-like encoding, it is fragmented and stripped across depots and parity blocks are computed and stored also (see Section 3.1.3). All of the storage allocation and metadata representing their structure is stored in an XML encoded data structure called an *exNode* [7]. In the event that data is lost, the exNode is consulted and data movement and XOR operations are issued to reconstruct it.

To apply the principle of aggregation to exposed computations, however, it is necessary to maintain control state that represents such an aggregation, just as the sequence numbers and timers are maintained to keep track of the state of a TCP session. In the case of logistical storage allocations, we followed the traditional, well-understood model of the Unix file system’s inode, the data structure used to implement aggregation disk blocks to create files, in designing a data structure called the external node, or *exNode*, to manage aggregate IBP allocations [25]. Rather than aggregating blocks on a single disk volume, the exNode aggregates byte arrays in IBP depots to form something like a file. (ExNodes, for example, lack user metadata that files typically have.) The exNode supports the creation storage abstractions with stronger properties, such as a network file, which can be layered over IBP-based storage in a way that is completely consistent with the exposed resource approach.

The case of aggregating NFU operations to implement a complete data transformation service is similar because, like a file, a process has a data extent that must be built up out of constituent storage resources. In a uniprocessor operating system like Unix, the data structure used to implement such aggregation is the memory map within the process control block, which includes both page addresses for RAM resources and block addresses for disk resources. In fact, process extents and files are very closely related, as can be seen by the existence of system calls, like `mmap`, that identify the storage extent of a file with part of the data extent of a process.

Exposing the primitive nature of RAM and disk as storage resources has the simplifying effect of unifying the data extent of a file with the data extent of a process;

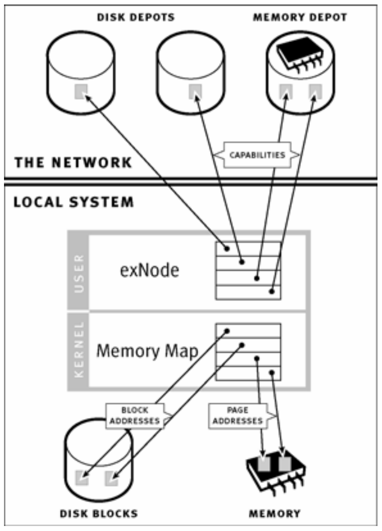

Figure 2: The exNode provides a uniform view of data and process state in ALSM

both can be described by the exNode (Fig.2). However, each must be augmented with additional state to implement either a full-fledged network file or a full-fledged network process. Thus, the closely related services of file caching/backup/replication and process paging/checkpoint/migration can be unified into a single set of state management tools.

3.1.3 Encoding for Fault-tolerance

Assuming that the NFU *Logistical Runtime System* (*LoRS*) tools may involve partitioning a task into discrete units with temporal and data dependencies between them, executing them in a fault-tolerant manner can be straightforward or complex. As with functional or idempotent remote procedure calls, failed (or indeterminately slow) executions may be aborted and retried [26, 27]. While this is straightforward and general, it is prone to be inefficient, since the period between initial invocation and retry is useless to the progress of the computation.

One solution to this problem that fits naturally in the ALSM paradigm is checkpointing. Since NFU operations are required to be time-limited, one natural output parameter of such an operation will be a checkpoint. For example, a matrix multiply operation may be defined to complete partially, and record its progress as an output parameter so that the same NFU may continue the operation, or so that the entire operation may be migrated to a different NFU.

Building further, execution units may be replicated among multiple NFUs, much like data may be replicated among multiple IBP depots. The LoRS tools scheduling the computation may perform replication aggressively (e.g. every execution is attempted simultaneously) or use some kind of bounded retry to strike a balance between computational progress and overuse of resources. Interestingly, this bears resemblance to the LoRS download operation from replicated sources, which may partition the download into many smaller downloads that can be replicated or retried [28]. We anticipate that as in the download operation, the bounded retry will perform best in practice.

If the computation is partitioned into a graphical representation of discrete execution units linked by temporal and data dependencies, we may leverage the work on the fault-tolerant execution of such graphs in [29-32]

Finally, our exposed approach can also support more application-specific solutions to fault-tolerance. For example, consider the act of creating a parity encoding for an exNode. As long as the integrity of the entire data set is maintained through a checksum, then the correctness and success of creating the parity encoding is not vital. The *probability* of correctness and success merely needs to be high. Similarly, consider a structured computation such as matrix factorization, which can not only be partitioned among multiple execution nodes, but which may also be enriched with encoding information so that as long as some subset of execution instances complete correctly, the answer may be both reconstructed and verified to a degree of error tolerance. Again, as long as the probability of each instance's correct completion is high enough, the computation may be performed efficiently [33].

3.2 ALSM and Distributed Visualization

3.2.1 Visualization in a Grid Environment

As a main driver application for exploring ALSM and GridSolve/L as an effective computing paradigm, we are working on distributed visualization. Visualization users routinely deal with data sets which require large-scale parallel computing in order to be analyzed and rendered. On the cutting edge of complexity, volume data sets now come with a number of scalars, vectors and even tensor matrices on each voxel. The output size of some routine simulation or data captures already frequently goes beyond gigabytes. Unfortunately, computing resources capable of dealing with such data sets are not commonly available to the researchers who need them. To this end, the Grid computing paradigm as implemented by ALSM and GridSolve/L provides the proper resource suite and computing methodology.

To give an example of a driving visualization scenario author Huang has close collaboration with Vanderbilt Medical School on human brain research. Understanding the morphology, structure, and function of the human brain and their underlying relationship is a grand goal of medical research, with far-reaching potential impact on all human beings. As a cornerstone, advanced imaging modalities including Diffusion Tensor MRI (DT-MRI) [34] and Functional MRI (fMRI) [35] have been developed. DT-MRI provides the first and only non-invasive way to obtain physical measurements to distinguish nerve bundles within the brain's gray matter and possibly reveal the interconnections bewteen various functional areas on the cerebral cortex. fMRI is the sole approach to quantitatively distinguish in vivo the functional differences of various areas on the cerebral cortex. Combining DT-MRI and fMRI measurements, groundbreaking scientific discovery of how brain function relates to brain morphology and structure are expected. However, unfortunately, processing and visualization of DT-MRI and fMRI data sets for such study require a great amount of computing and storage resources that medical schools around the globe do not have.

To better appreciate the computation involved, we here briefly introduce DT-MRI and fMRI. In DT-MRI, on each voxel a 3×3 matrix is stored [34]. On each voxel, dozens of correlation coefficients are stored. Such fMRI information is used in the visualization process to allow hypothesis driven queries to establish the existence and nature of connections among ROIs   The production of a typical visualization takes about 10 minutes on 30 parallel processors. Obviously, when real medical hypothesis-driven study is underway, scores of these visualizations will be required on each member of a study, and studies typically consist of dozens of members.

Additionally, there is a second and potentially direr problem. The radiologists and neurologists need to have an easy to use computational tool for their research, and Matlab is currently the universal tool they use. Indeed, for many of them, Matlab is the sole programming tool they know how to use. C programming, system libraries, TCP/IP, and parallel processing are beyond their specialization. Thus, unless the grid computing environment can be invoked from a problem solving interface such as Matlab, it will be useless for this class of researchers.

3.2.2 The Visualization Pipeline and Our Methodology

The visualization pipeline associated with this process has three natural stages: selection and preconditioning of data before it is processed by the parallel computer, the main computational phase, and then postprocessing to match the specific visualization environment and requirements of the user, including implementation of a prioritized data movement algorithm, such as occlusion culling or level-of-detail (LoD), to ensure minimum latency for the viewer. This pipeline maps naturally into GridSolve/L, with the pre- and post-processing stages being mapped onto depots, with data transformation being performed using the Network Functional Unit, and the major computational stage being implemented on a cluster using the NetSolve server as its front end. The entire pipeline will in fact appear to the user as a GridSolve/L call, with the NetSolve server taking responsibility for directing the functions of the depots used in all stages except the initial upload and prioritized download of data, which are the responsibility of the user application.

With the advent of ALSM, different distributed visualization algorithms can be categorized in a straightforward way, as follows:

- o those that pre-compute all results that may be requested and use the service provider as a static database. A simple movie or static iso-surface streaming algorithms are good examples.
- o those that maintain raw data on-the-fly, and, compute view/predicate-dependent query results, which are then transmitted to clients during run-time. Examples include view-dependent streaming techniques, such as view-dependent LoD iso-surfaces and image-based streaming, or schematic non-photo-realistic rendering methods using strokes or stippling.
- o those that transform raw data into various intermediate forms, which then get converted to other forms useful to a specific client interaction during run-time. Examples include new transfer function sensitive streaming methods and temporal-spatial coherence based wavelet compression/reconstructions, etc.

Just using simple LoRS tools, the first category can be implemented indexing or addressing, as we have shown in our paper [21]. Pre-computed results can be staged on a local depot so close to client that the network performance between the depot and client never fails the requirement. The second category requires some relatively complicated operations, such as occlusion computation using depth sorting, a software opacity buffer (a 2D array), and array operations like summation, averaging, etc. Image-based techniques may require a numerical integration operation using the Riemann sum. The shading procedures, used in non-photo-realistic drawings, would use dot product, cos() and pow() functions. All of these operations are ideal as NFU data transformation primitives that are part of ALSM.

The third category is, the most challenging and most useful. A GridSolve server library needs to developed on the whole infrastructure. This library will need to collect intermediate results from peer depots before performing their main computations, and the computations will need to be performed, storing results on the IBP depots for post-processing. A goal of our on-going research is to design and build a suite of core GridSolve libraries for each of the major area of visualization, including medical visualization, volume rendering, iso-surfacing, flow visualization, as well as time-varying multi-variate visualization.

Using GridSolve/L, this visualization paradigm can be applied in order to implement the DT-MRI and fMRI visualization pipeline required to support author Huang’s Vanderbilt collaborators. After a subject has been scanned, a radiologist opens up his Matlab client and invokes a Netsolve function to construct the nerve network among a dozen regions of interest (ROIs). The data moves to computation nodes, where the brain nerve bundles connecting the ROIs are computed in parallel. The raw imaging data sets are not necessarily large, ranging between 50 to 100 MB per patient. However, the visualization models of the nerve fibers can be so detailed that more than 500 MB of visualization data are generated. Matlab is not a tool that can comfortably handle geometric data sets of such sizes. This then becomes a remote visualization problem as discussed above. View-dependent and image-resolution dependent data streaming, level-of-detail streaming, occlusion culling, etc. would all be performed using ALSM as the data is streamed back to radiologists at Vanderbilt. The use of ALSM provides the application developer significant control over how data is transferred and buffered within the network. The visualization component of the whole pipeline can be the Matlab visualization toolkit or a custom-made plug-in viewer.

4. Bibliography

[1] I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure," Morgan Kaufman Publishers, 1999, pp. 677.

[2] J. Ousterhout, "The Role of Distributed State," in *CMU Computer Science: a 25th Anniversary Commemorative*, R. F. Rashid, Ed.: Addison-Wesley, 1991, pp. 199--217.

[3] J. Carter, P. Cao, M. Dahlin, M. Scott, M. Shapiro, and W. Zwaenepoel, "Distributed State Management," Report of the NSF Workshop on Future Directions for Systems Research,, July 31, 1997.

[4] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of SuperComputer Applications*, vol. 15, no. 3, 2001.

[5] M. Beck, T. Moore, J. Plank, and M. Swany, "Logistical Networking: Sharing More Than the Wires," in *Active Middleware Services*, vol. 583, *The Kluwer International Series in Engineering and Computer Science*, S. Hariri, C. Lee, and C. Raghavendra, Eds. Boston: Kluwer Academic Publishers, 2000.

[6] M. Beck, T. Moore, and J. S. Plank, "Exposed vs. Encapsulated Approaches to Grid Service Architecture," in *Grid Computing -- GRID 2001*, *LNCS 2242*, C. Lee, Ed. Denver, CO: Springer Verlag, 2001, pp. 124-132.

[7] M. Beck, T. Moore, and J. S. Plank, "An End-to-end Approach to Globally Scalable Network Storage," in *Proceedings of ACM Sigcomm 2002*. Pittsburgh, PA: Association for Computing Machinery, 2002, pp. 339-346.

[8] J. S. Plank, A. Bassi, M. Beck, T. Moore, M. Swany, and R. Wolski, "Managing Data Storage in the Network," *IEEE Internet Computing*, vol. 5, no. 5, pp. 50-58, September/October, 2001.

[9] H. Casanova and J. Dongarra, "Applying NetSolve's Network Enabled Server," *IEEE Computational Science & Engineering*, vol. 5, no. 3, pp. 57-66,1998.

[10] D. C. Arnold and J. Dongarra, "The NetSolve Environment: Progressing Towards the Seamless Grid," in *Proceedings of the 2000 ICPP Workshops*, P. Sadayappan, Ed. Toronto, Canada: IEEE Computer Society, 2000, pp. 199-206.

[11] M. Beck, F. Berman, H. Casanova, J. Dongarra, T. Moore, J. Plank, and R. Wolski, "Logistical Quality of Service in NetSolve," *Computers and Communications*, vol. 22 pp. 1034-1044,1999.

[12] A. Bassi, M. Beck, G. Fagg, T. Moore, J. Plank, M. Swany, and R. Wolski, "The Internet Backplane Protocol: A Study in Resource Sharing," in *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2002)*. Berlin, Germany: IEEE, 2002.

[13] W. Elwasif, J. Plank, and R. Wolski, "Data Staging in Wide Area Task Farming Applications," in *IEEE International Symposium on Cluster Computing and the Grid*. Brisbane, Australia, 2001, pp. 122-29.

[14] D. C. Arnold, S. S. Vahdiyar, and J. Dongarra, "On the Convergence of Computational and Data Grids," *Parallel Processing Letters*, vol. 11, no. 2, pp. 187-202, June/September, 2001.

[15] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments," in *9th Heterogeneous Computing Workshop (HCW'00)*. Cancun, Mexico, 2000, pp. 349-363.

[16] H. Casanova, F. Berman, G. Orbertelli, and R. Wolski, "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid," in *Proceedings of Supercomputing 2000 (SC2000)*. Dallas, TX, 2000.

[17] A. Agbaria and J. S. Plank, "Design, Implementation, and Performance of Checkpointing in NetSolve," in *International Conference on Dependable Systems and Networks (FTCS-30 and DCCA-8)*: IEEE, 2000, pp. 49-54.

[18] D. Thain, J. Basney, S.-C. Son, and M. Livny, "The Kangaroo Approach to Data Movement on the Grid," in *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)*, 2001.

[19] M. Beck, T. Moore, and J. S. Plank, "An End-to-End Approach to Globally Scalable Programmable Networking," presented at Future Directions in Network Architecture (FDNA-03), an ACM SIGCOMM 2003 Workshop, Karlsruhe, Germany, August 25-29, 2003.

[20] J. S. Plank, S. Atchley, Y. Ding, and M. Beck, "Algorithms for High Performance, Wide-area Distributed File Downloads," *Parallel Processing Letters*, vol. 13, no. 2, pp. 207-224, June, 2003.

[21] J. Ding, J. Huang, M. Beck, S. Liu, T. Moore, and S. Soltesz, "Remote Visualization by Browsing Image Based Databases with Logistical Networking," in *Proceedings of SC2003*. Pheonix, AZ, 2003.

[22] D. P. Reed, J. H. Saltzer, and D. D. Clark, "Comment on Active Networking and End-to-End Arguments," *IEEE Network*, vol. 12, no. 3, pp. 69-71, May/June, 1998.

[23] L. F. G. Sarmenta and S. Hirano, "Bayanihan: building and studyingWeb-based volunteer computing systems using Java," *Future Generation Computer Systems*, vol. 15, no. 5-6, pp. 675-686,1999.

[24] L. F. G. Sarmenta, "Sabotage-tolerance mechanisms for volunteer computing systems," *Future Generation Computer Systems*, vol. 18, no. 4, pp. 561-572,2002.

[25] A. Bassi, M. Beck, and T. Moore, "Mobile Management of Network Files," in *Third Annual International Workshop on Active Middleware Services (AMS 2001)*. San Franscisco, CA: Kluwer Academic Publishers, 2001, pp. 106-115.

[26] J. S. Plank, H. Casanova, M. Beck, and J. Dongarra, "Deploying Fault Tolerance and Task Migration with NetSolve," *Future Generation Computer Systems*, vol. 15, no. 5-6, pp. 745-755,1999.

[27] S. Sekiguchi, M. Sato, H. Nakada, S. Matsuoka, and U. Nagashima, "Ninf : Network based Information Library for Globally High Performance Computing," presented at Proc. of Parallel Object-Oriented Methods and Applications (POOMA), Santa Fe, NM, 1996.

[28] J. S. Plank, S. Atchley, Y. Ding, and M. Beck, "Algorithms for High Performance, Wide-Area, Distributed File Downloads," Department of Computer Science, University of Tennessee, Knoxville, Technical Report, UT-CS-02-485, October, 2002.

[29] A. Beguelin, J. J. Dongarra, G. A. Geist, R. Manchek, and V. S. Sunderam, "HeNCE: Graphical Development Tools for Network-Based Concurrent Computing," in *Proceedings SHPCC-92*, 1992, pp. 129-136.

[30] D. Cummings and L. Alkalaj, "Checkpoint/rollback in a distributed system using coarse-grained dataflow," presented at 24th International Symposium on Fault-Tolerant Computing, Austin, TX, 1994.

[31] C. R. Johnson, S. Parker, D. Weinstein, and S. Heffernan, "Component-Based Problem Solving Environments for Large-Scale Scientific Computing," *Journal on Concurrency and Computation: Practice and Experience*, vol. 14 pp. 1337-1349,2002.

[32] W. Bausch, C. Pautasso, R. Schaeppi, and G. A. . "BioOpera: Cluster-aware Computing," in *Proc. of the 4th IEEE International Conference on Cluster Computing (Cluster 2002)*. Chicago, IL, 2002.

[33] D. Boley, G. H. Golub, S. Makar, N. Saxena, and E. J. McCluskey, "Floating Point Fault Tolerance with Backward Error Assertions," *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 302-11, February, 1995.

[34] P. J. Basser, J. Mattiello, and D. Le Bihan, "MR diffusion tensor spectroscopy and imaging," *Biophysics Journal*, vol. 66 pp. 259-267,1994.

[35] D. J. Heeger and D. Ress, "What does fMRI tell us about neuronal activity?," *Nature Reviews Neuroscience*, vol. 3 pp. 142-151,2002.