Logistical Computing and Internetworking: Middleware for the Use of Storage in Communication

Micah Beck[†], Dorian Arnold[†], Alessandro Bassi[†], Fran Berman[‡], Henri Casanova[‡], Jack Dongarra[†], Terry Moore[†], Graziano Obertelli[‡], James Plank[†], Martin Swany[†], Sathish Vadhiyar[†], Rich Wolski[†]

Abstract:

The ***Logistical Computing and Internetworking*** (***LoCI***) project is a reflection of the way that the next generation internetworking fundamentally changes our definition of high performance wide area computing. A key to achieving this aim is the development of middleware that can provide reliable, flexible, scalable, and cost-effective delivery of data with quality of service (QoS) guarantees to support high performance applications of all types. The LoCI effort attacks this problem with a simple but innovative strategy. At the base of the LoCI project is a richer view of the use of storage in communication and information sharing.

Introduction

At the base of the LoCI research is a richer view of the use of storage in communication. Current approaches to QoS rely on the standard end-to-end model of communication: the state of network flow is maintained at the end nodes and not in the network. By contrast, our concept of *logistical QoS* is a generalization of the typical model that permits state management *within the networking fabric itself*, via a much more flexible control of message buffering, in order to achieve QoS delivery without difficult end-to-end requirements. For example, whenever data is available to be sent well before it needs to be received, it can be *staged*, i.e. moved in advance and stored in a location "close" to the receiver for later delivery. We define such strategies that employ storage in communication, as *logistical network computing*, and the main purpose of the LoCI project is to investigate and test the central conjecture of logistical network computing:

If *1) distributed network storage is made available as resource an flexibly schedulable and*
2) communication, computational, and storage resources can be predictably allocated for coscheduling,

Then *advanced applications can be implemented on computational grids with higher performance and/or lower overall use of communication, computational, and storage resources.*

The structure of our research in the LoCI program reflects the parts of this conjecture, which in turn represent the fundamental elements of logistical network computing. To create a research-computing environment that enables us to allocate communication, computation, and storage resources for coscheduling, we combine four technologies from the world of computational grids:

- **Internet Backplane Protocol (IBP)** [1] is primitive middleware that supports a layer of network storage, implemented as a system of buffers exposed for direct scheduling, that advanced applications can use to leverage state management for high-performance.
- **Network Weather Service (NWS)** [2] enables us to predict the ability of the network to respond to data movement requests over time.

[†] University of Tennessee, work supported in part by the NSF/NGS GRANT #NSF EIA-9975015, and NSF GRANT ACI-9876895.
[‡] University of California, San Diego, work supported in part by the NSF/NGS GRANT #NSF EIA-9975015.

- **NetSolve [3]** provides a programming environment that facilitates the analysis of program dependences, expressed in the form of dependence flow graphs, to understand an application's inherent communication requirements. A major component of LoCI research is identify and provide opportunities for extracting scheduling information from applications.
- **App**lication **Le**vel **S**cheduling **(AppLeS)** [4] is enabling us to derive an efficient schedule that meets those communication requirements. Once the scheduling information is made available, mapping the computation, network and storage resources of the application to the Grid resources, subject to current and predicted resource conditions, is a difficult problem. AppLeS is the leading instance of a range of approaches we are exploring under LoCI.

These Grid technologies have focused, primarily, on the control of compute and network resources to achieve high-performance distributed execution. Logistical computing adds the control of storage to form a comprehensive Grid infrastructure. By exposing more of the underlying storage structure of the network and maximizing its exploitation in scientific applications, our research is moving network computing towards the physical and logical limits of the underlying technology, as is found in more mature areas of computer engineering.

Logistical Network Computing and Explicit Storage Control

Our architectural analysis of high performance network computing derives from an analogy with the architecture of modern pipelined microprocessors. The fundamental elements of modern processor architecture are:

- Buses and functional units which move and transform data, and
- Memory and cache, registers and pipeline buffers that store data.

With these mechanisms in place, the programming interface can then schedule the execution of a program in a way that achieves maximum performance. Careful control of data at the various stages of an execution pipeline is necessary to ensure high performance levels. It is our belief that Grid programs (or the Grid middleware) must analogously control program state as it traverses the Grid.

Another important difference between modern RISC and VLIW architectures and the CISC architectures of the 70s and 80s is that instructions are predictable because they model the processor pipeline very directly. All elements of the pipeline behave in a completely deterministic fashion except for the cache, which is statistically predictable.

In our model of logistical network computing, the fundamental elements are

- Predictable networking and computation which move and transform data, and
- Storage that is accessible from the network.

Using these elements, the programming interface can then schedule the execution of a program in a way that achieves maximum performance. One important difference between logistical network computing and traditional methods is that it is based on global scheduling expressed at the programming interface but implemented by local allocation throughout the network. Traditional approaches express at the programming interface only complex higher-level operations defined in terms of the endpoints, encapsulating the complexity of the network. The result is that it is much harder to implement predictable operations.

The Internet Backplane as Middleware for Next Generation Software

In order to experiment with logistical network computing, some mechanism for the management of storage is required. Staging can be implemented at many possible levels in the application or operating system, and as with all software architecture decisions, the tradeoffs are complex, involving many factors including compatibility of interfaces, administrative convenience and performance.

Most network computing environments are fairly self-contained in the sense that data flows only between processors which host compute servers, and so it is possible to implement data depots and

storage management as part of the compute server. Under this approach staging is accessible only to a single network computing domain, so that the management of storage is not shared between environments (e.g. NetSolve [5], Globus [6], and Legion [7]) or between instances of single environment. Such sharing is important because it allows storage to be managed as an aggregate resource rather than as several smaller pools, and because it allows performance-enhancing services such as caching to be implemented in an application- and environment-neutral manner.

The middleware approach is to abstract a model of state management from the particular computing environment and to define it to be a lower level service. It is possible to implement that service in a user-level library, in a daemon process or in kernel network drivers that reach lower into the protocol stack. In fact, the functionality may ultimately be spread across these architectural levels, and could ultimately be supported by modifications to the network infrastructure itself.

A key innovation of the LoCI project is the implementation of a software mechanism for distributed data staging, called the ***Internet Backplane Protocol (IBP),*** a middleware service implemented by TCP/IP connections to a daemon processes, in the style of FTP and NFS.

An Overview of the Internet Backplane Protocol (IBP)

Fundamentally, IBP is designed to allow much freer control of buffer management at distributed storage depots through a general, but non-traditional scheme for naming, staging, delivering and protecting data. To address the needs of new Grid applications IBP diverges from the standard storage management systems (e.g. distributed file systems or databases) in three fundamental ways, which we consider in turn.

IBP serves up both *writable* and readable storage to anonymous clients as a wide-area network resource

The Internet is a mainly stateless communication substrate that serves up two kinds of network resources to its generic and unauthenticated clients: read-only storage through anonymous FTP and the Web, and remote processing servers that connect to clients via two-way ASCII communication pipes with Telnet. There are projects that are trying to enlarge this resource space, such as Jini, NetSolve, and active disk and network movement. IBP enlarges this resource space by focusing on storage, namely writable storage. The benefits of offering writable storage as a network resource are numerous:

- Quality of service guarantees for networking can be met more easily when the intermediate routing nodes can store the communication buffers.
- Resource schedulers can include the staging of data near the processing resources for better resource utilization and better scheduling.
- Content services can be enhanced with both client and server-driven replication strategies (including, but not limited to caching, content push, multicast support, and replica management) for improved performance.
- A ubiquitous foundation for achieving fault-tolerance may be achieved.

Currently, most strategies for achieving the above benefits are *ad hoc* workarounds of the existing Internet architecture.

IBP allows for the remote control of storage activities

Storage managed by IBP may be viewed as files or buffers, located on reliable storage, in RAM, or perhaps on an active disk. IBP allows a user or processing entity to both access and manage these storage entities *remotely*, without being involved in the actual manipulation of the bytes. We present three general categories of how this improves application performance and flexibility below.

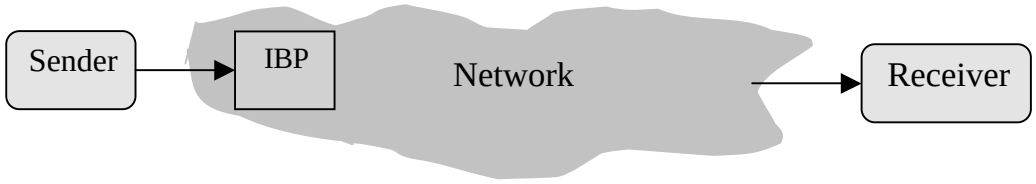

Figure 1: IBP: Keeping Data Close to Sender

As an illustration in Figure 1, consider the generation of sensor data in NWS. NWS generates a tremendous about of performance data in order to make its predictions. It is not clear when the data is being collected whether or not it will be used (i.e. clients might not request predictions for a few minutes). Therefore it is optimal to store the data in a location close to the sender so that the storing is optimized. Sending the data to clients is less optimal, but that is a more infrequent operation. Ideally, of course, the data is stored on the machine being monitored, but that may not be possible. Storing it nearby in IBP is the next best alternative.

A similar example is checkpointing computations within NetSolve for fault-tolerance [8]. Since checkpoints may never be used, NetSolve would like to optimize the act of checkpointing. Obviously, it's not a good idea to store the checkpoint on the compute server, because if the server fails, the checkpoint may not be available (since the server is down). IBP thus allows the servers to checkpoint "nearby," which allows for an optimal balance of performance and fault-tolerance.

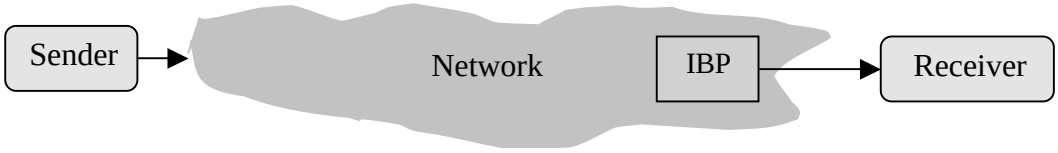

Figure 2: IBP: Place Data Close To Receiver

In Figure 2, the data is put close to the receiver so that the overhead of receiving is low. Standard performance optimizations such as staging and caching fall into this category, and are well-known enough to require no further elaboration.

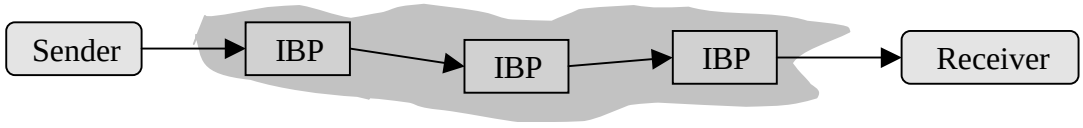

Figure 3: IBP: Utilizing Storage Throughout

In Figure 3, storage is used in the network to explicitly route a message. This obviously improves the performance of broadcast messages. Additionally, it helps with intermediate link failures. With standard end-to-end networking, one has to resend packets from the sender if *any* link fails. With intermediate

storage, the resend only has to happen on the failed link. Finally, with intermediate storage, a user can do explicit routing, which may be much more effective than standard Internet routing [9].

IBP decouples the notion of user identification from storage

Typically, storage systems require authentication for any access that uses a persistent resource, whereas networking has no persistent resources and so can rely on security implemented at the end-points. IBP treats all storage as if it were a communication buffer by offering up writable storage on the network to *unauthenticated* clients. That clients are unauthenticated does not mean that the system is chaotic or without safeguards. IBP allows the owner of a storage server to define how much storage to serve for IBP and how that storage should be served. In particular, IBP file allocation includes the notion that files may have a limited lifetime before they are removed by the IBP system. Each file is accessed through a unique storage capability so that access can be restricted without authentication. In addition, an IBP file may be allocated as *volatile*, meaning that the IBP server may revoke the storage at any time. Such a system strikes a balance between offering the benefits of writable storage on the network, and making sure that the owner of such storage has the ability to reclaim it when desired.

Logistical Mechanisms

The two key logistical mechanisms that we are designing are the Internet Backplane Protocol (IBP), which allows us to express logistical data movement, and the Network Weather Service (NWS) that allows us to predict the effects of future requests for data movement.

The Internet Backplane Protocol API

We have defined and implemented a client API for IBP consisting of seven procedure calls, and a server daemon software that makes local storage available for remote management. Currently, connections between clients and servers are made through TCP/IP sockets.

IBP client calls may be made by any process that can connect to an IBP server. IBP servers do not require administrative privileges to install and operate, so IBP has the flavor of software such as PVM [10] that can leverage the privileges of ordinary users to create a distributed computing platform. IBP servers can implement various storage allocation policies in order to control the local impact. For example, the IBP server may be allowed to allocate spare physical memory, or it may by directed to only allow the allocation of unused disk space and to revoke that allocation in favor of local use when necessary. Alternatively, the IBP server may enforce only time-limited allocations, where the storage is automatically revoked after a set time period. These features manage the local impact of allowing allocation of local resources through IBP.

Each IBP server allocates storage in the form of append-only byte arrays. There are no directory structures or file names (this structure can be layered on top of IBP through the use of a directory server such as Globus' MDS). Clients initially allocate storage through a request to an IBP server. If the allocation is successful, the server returns three *capabilities* to the client, one for reading, one for writing, and one for management. These capabilities can be viewed as names that are assigned by the server and are meaningful only to IBP. The contents of the capability can be obscured cryptographically in order to implement a basic level of security. In order to achieve high performance, applications can pass and copy capabilities among themselves without coordinating through IBP.

IBP's API and several logistical network computing applications are described in detail in other documents [11],[12].

The Network Weather Service: Monitoring Resources for Logistical Scheduling

While IBP provides the mechanisms that allow applications to exploit logistical network computing, resource usage must be carefully scheduled or application performance will suffer. To make these decisions the scheduler must *predict* the *future* performance of a set of resources. We use the ***Network***

Weather Service (***NWS***) [13] to make these predictions based on the observed performance history of each resource.

The Network Weather Service (NWS) periodically **monitors** available resource performance by passively and actively querying each resource, **forecasts** future performance levels by statistically analyzing monitored performance data in near-real time, and **reports** both up-to-date performance monitor data and performance forecasts via a set of well-defined interfaces.

The monitor interface is easily extensible; currently implemented monitors include TCP/IP latency and bandwidth, CPU load, and Globus GRAM process start times [14]. Monitor traces are presented as time series to a set of NWS forecasting models that make short-term performance predictions levels. Both forecast values and accuracy measures are reported for each resource and performance characteristic. Using this accuracy information, schedulers can gauge the value of individual forecasts and use this valuation to exploit different risk strategies. Forecast and forecast-quality data is published via C-language interfaces for access by dynamic schedulers.

It is the function of logistical scheduling to compose intermediate network and storage resources into an end-to-end "path" that supports a specified quality-of-service (QoS). Our schedulers will rely on NWS performance forecasts to identify, dynamically, resource compositions that meet the QoS specifications of different, and potentially competing, Grid applications. A key research question that we are addressing concerns the degree to which NWS predications may be effectively composed to produce an overall predication.

Logistical Scheduling and the AppLeS Project

The APST project (AppLeS Parameter-Sweep Template) targets the efficient deployment and scheduling of large-scale parameter-sweep applications over the Computational Grid. In that work, we designed and evaluated an adaptive scheduling algorithm [15] that makes decisions concerning data location and transfer. Experiments were conducted that ran over a cross-continent Grid. IBP was used as an underlying mechanism to implement scheduling decisions. This experience proved that IBP provides the adequate model, API, and mechanism to easily implement sophisticated scheduling algorithms in wide-area computing environments available today and will be described in the full paper.

NetSolve as an Environment for Experimentation in Logistical Scheduling

To explore logistical QoS using IBP we are applying logistical scheduling to the management of state in the NetSolve network computing environment. Dataflow graphs describe functional computation in terms of arguments and return values, so they can be used to represent NetSolve client programs. In order to model the optimization of NetSolve using state management, it is necessary to introduce imperative operations that model IBP's store and deliver calls. If we augment our dataflow graphs with such imperative nodes, it is then necessary to represent the dependences between them [16]. We use a Directed Acyclic Graph (DAG) representation of computational modules. Dependence flow graphs are a very natural tool for optimizing the use of state. In addition, NetSolve can be use IBP to stage data close to the point of the computation as in Figure 3.

Preliminary Results

For our experiments, we wanted to show how this relatively simple use of LoCI facilities could yield benefits and help motivate the investigation of more complicated techniques that either make the system easier to use or give even better performance. To mimic the geographical expanse of the Grid, we placed

a NetSolve client application at the University of California, San Diego and experimented with requests to a pool of computational and LoCI servers at the University of Tennessee. We used a set of matrices from the Harwell-Boeing collection of the Matrix Market repository [17] to solve systems of equations using the MA28 [18] solver library.

Figure 4 shows the results we obtained when varying the number of accesses (cache hits) made to the data from the Harwell-Boeing set. For various data sizes, we found the average times of 10 runs using traditional calls to NetSolve transmitting data over the network. We then made another set of runs with the same dataset, this time storing the data in LoCI storage and having the server retrieve the data from the storage server. During these runs we collected the time expended for compute cycles, NetSolve overhead, network transmissions and LoCI overhead. We used this collected data to deduce what the turn-around time would be as we increased the number of times the client application requested the computation. The two graphs of figure 4 show the results for datasets of size 16.1KB and 2.68 MB, respectively. These represent both the smallest and largest data sizes with which we experimented. We also collected data for a range of sizes in between these points (21.4KB, 35.7KB, 55.4KB, 247KB, 302.4KB, 995KB, and 1.01MB) and testify that they bear similar results. The presented graphs show a worst case of 7 accesses (in the 16.KB case) and 2 accesses (in the 2.68MB) needed before the overhead added by the LoCI is outweighed by the reduction of network activity caused by cache reuse. The graphs reach an asymptotic level that represent the points at which computational capacity, and not network bandwidth and latency, becomes the system bottleneck. For the 2.68MB sample, this occurs at a point when the enhanced system is operating at more than three times (3x) faster than the unenhanced system.

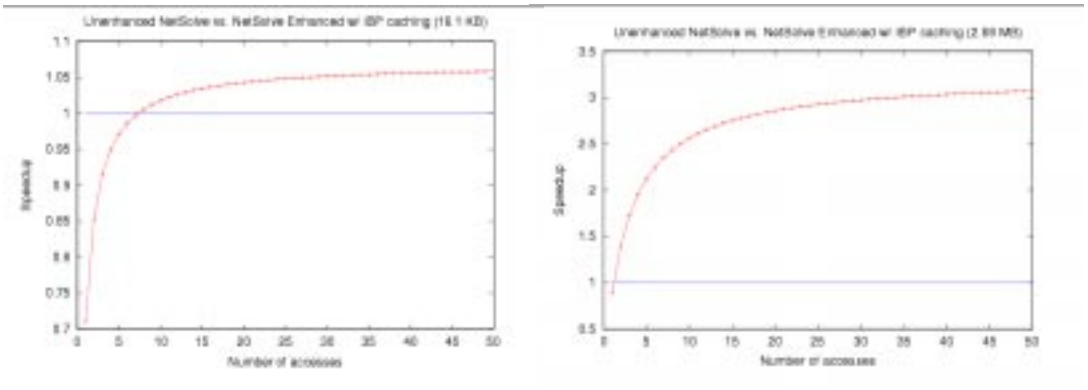

Figure 4: Results of Improved Computational Efficiency with IBP Caching Is Used With NetSolve

Conclusions and Future Work

By exposing intermediate communication state to application or middleware control, Logistical Computing forms a comprehensive approach to Grid computing. Process resources, network resources, and storage resources can be explicitly controlled and scheduled to ensure performance in the face of fluctuating resource availability. In particular, the Internet Backplane Protocol allows distributed applications to break with the end-to-end communication model achieving better quality of service levels

through explicit state control. By combining this innovative approach to dynamic storage management with NetSolve and the Network Weather Service we have been able to deliver high-performance distributed computing to the end user through the familiar RPC programming model. Our intention is to continue our development of Logistical Computing and to deploy a campus-wide testbed using the Scalable Intracampus Research Grid (SInRG) at the University of Tennessee. Designed to develop a University Grid user community, we are developing a Logistical Computing environment for SInRG both as a means of validating our results, and easing the Grid programming burden.

The software for IBP, NWS, NetSolve, and AppLeS can be found at the following url's.
http://icl.cs.utk.edu/ibp/
http://nws.cs.utk.edu/
http://icl.cs.utk.edu/netsolve/
http://apples.ucsd.edu/

References:

1. Plank, J., et al. *The Internet Backplane Protocol: Storage in the Network*. in *NetStore99: The Network Storage Symposium*. 1999. Seattle, WA.
2. Wolski, R., N. Spring, and J. Hayes, *The Network Weather Service: a Distributed Resource Performance Forecasting Service for Metacomputing.* Future Generation Computer Systems, 1999. **15**(5-6): p. 757-768.
3. Casanova, H., et al., *Application-Specific Tools*, in *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, Editors. 1998, Morgan Kaufman Publishers: San Francisco, CA. p. 159-180.
4. Berman, F., et al., *Application-level scheduling on distributed heterogeneous multiprocessor systems*, in *Proceedings of Supercomputing '96*. 1996.
5. Casanova, H. and J. Dongarra, *Applying NetSolve's Network Enabled Server.* IEEE Computational Science & Engineering, 1998. **5**(3): p. 57-66.
6. Foster, I. and K. Kesselman, *Globus: A Metacomputing Infrastructure Toolkit.* International Journal of Supercomputer Applications, 1997. **11**(2): p. 115-128.
7. Grimshaw, A., W. Wulf, and e. al., *The Legion vision of a worldwide virtual computer.* Communications of the ACM, 1997. **40**(1): p. 39-45.
8. Agbaria, A. and J.S. Plank. *Design, Implementation, and Performance of Checkpointing in NetSolve*. in *International Conference on Dependable Systems and Networks (FTCS-30 & DCCA-8)*. 2000.
9. Swany, M. and R. Wolski, *Data Logistics in Networking: The Logistical Session Layer*. 2001, University of Tennessee: Knoxville, TN.
10. Geist, A., et al., *PVM : Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*. 1994, Cambridge, MA: The MIT Press.
11. Bassi, A., et al., *Internet Backplane Protocol : API 1.0*. 2001, University of Tennessee, Computer Science Department.
12. Elwasif, W., et al. *IBP-Mail: Controlled Delivery of Large Mail Files*. in *NetStore '99: Network Storage Symposium*. 1999: Internet2, http://dsi.internet2.edu/netstore99.
13. Wolski, R., *Forcasting network performance to support dynamic scheduling using the Network Weather Service*, in *Proceedings of the 6th IEEE Symposium on High Performance Distributed Computing*. 1997, IEEE Computer Society Press: Los Alamitos, CA. p. 316-325.

14. Foster, I. and C. Kesselman, eds. *The Grid: Blueprint for a New Computing Infrastructure*. 1998, Morgan Kaufman Publishers: San Francisco, CA. 677.
15. Casanova, H., et al. *The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid}*. in *Proceedings of SuperComputing 2000*. 2000. Dallas, TX: IEEE.
16. Arnold, D.C., D. Bachmann, and J. Dongarra. *Request Sequencing: Optimizing Communication for the Grid*. in *Euro-Par 2000 Parallel Processing, 6th International Euro-Par Conference*. 2000. Munich, Germany: Springer Verlag.
17. Boisvert, R.F., et al., *Matrix Market : A Web Resource for Test Matrix Collections*, in *The Quality of Numerical Software: Assessment and Enhancement*. 1997, Chapman & Hall: London. p. 125-137.
18. Duff, I., A.M. Erisman, and J.K. Reid, *Direct Methods for Sparse Matrices*. 1986, Oxford: Clarendon Press.