ELSEVIER

Computer Communications 22 (1999) 1034–1044

computer communications

Logistical quality of service in NetSolve

M. Beck[a], H. Casanova[a], J. Dongarra[a,*], T. Moore[a], J. Plank[a], F. Berman[b], R. Wolski[b]

[a]*Innovative Computing Laboratory, Computer Science Department, University of Tennessee, Knoxville, TN 37996, USA*
[b]*Department of Computer Science, University of California at San Diego, San Diego, CA 92093, USA*

Abstract

From its inception a principal goal of the Next Generation Internet (NGI) has been to find a way to provide reliable, scalable, cost effective, and deployable delivery of data with quality of service (QoS) guarantees as a foundation for innovative NGI applications. But while schemes to provide reliable end-to-end QoS are being actively pursued on a number of fronts, the well known problems with the scalability, cost, and deployability of end-to-end QoS continue to obstruct progress toward achieving this end. Our research focuses on the nature and potential value of an approach to providing QoS that builds on a strategy for allowing NGI applications to dynamically manage remote storage resources in order to stage data locally for later delivery. We call this strategy logistical Quality of Service (*logistical QoS*).

The concept of logistical QoS is a generalization of the typical end-to-end model for reserving QoS, permitting much more flexible use of buffering of messages in order to achieve QoS delivery without difficult end-to-end requirements. Whenever data is available to be sent well before it needs to be received, it can be *staged*, i.e. *moved in advance to a location close to the receiver for later delivery*. Isolating the act of buffering data as a distinct operation, independent of delivery to the receiver, opens up a new dimension of freedom in the management of communication and storage resources that can offer NGI application developers a wide variety of new opportunities to innovate.

Our project focuses on the development of logistical QoS as enabling network functionality for application-driven staging and scheduling of distributed computation on NGI. It is divided into two parts: (1) research on the basic network functionality that is required to support logistical QoS that is reliable, scalable, cost effective, and easy to use; and (2) research that investigates the integration of logistical QoS and the basic network technology that underlies it with the scheduling of distributed computations using NetSolve and the Network Weather Service.

Our work on logistical QoS focuses on the *Internet Backplane*, as providing a mechanism for managing remote storage resources, and the *Internet Backplane Protocol*, as enabling technology for using that mechanism. The idea underlying the concept of the Internet Backplane is that NGI will enable us to consider the global network as an extension of the processor backplane, if only we have a low overhead mechanism for fine grained naming and access to data, analogous to physical addresses and bus transfers. By this analogy the Internet Backplane is a common namespace for fine-grained management of distributed resources. The IBP provides a flexible interface to enable this functionality, allowing reliable and flexible control of remote storage buffers through a general scheme for naming, staging, delivering and protecting data.

We will test logistical QoS as an enabling technology for NGI computing using NetSolve. NetSolve is a software environment for networked computing designed to transform disparate computers and software components into a unified, easy-to-access computational service; it is being used by NSF's *Partnerships for Advanced Computational Infrastructure* to build high-performance systems for distributed computation on leading edge networks. We will investigate the implementation of logistical QoS within NetSolve and the integration of IBP with the Network Weather Service (used to monitor and forecast the performance of network and computational resources) to build a scheduling capability that maximizes the performance of NetSolve across next generation networks.

Keywords: Logistical quality of service; NetSolve; Next generation Internet; Network weather service

1. Introduction: logistical QoS and its relevance to network design

A kind of bootstrap problem tends to obstruct innovation in basic network design for Next Generation Internet (NGI). The trouble is that the guiding purpose of NGI is to enable revolutionary new types of applications, but those applications will develop slowly so long as the basic network technologies are not available to inform and inspire the work of application designers. The problem is exacerbated by the fact that NGI aims to be not just an incremental improvement on familiar network technologies and applications, but a true watershed in the development of the Internet. Our investigation of the nature and potential value of

* Corresponding author. Fax: + 1-423-974-8296.
E-mail addresses: mbeck@cs.utk.edu (M. Beck); casanova@cs.utk.edu (H. Casanova); dongarra@cs.utk.edu (J. Dongarra); tmoore@cs.utk.edu (T. Moore); plank@cs.utk.edu (J. Plank); berman@cs.ucsd.edu (F. Berman); rich@cs.ucsd.edu (R. Wolski)

0140-3664/99/$ - see front matter
PII: S0140-3664(99)00074-2

what we call logistical Quality of Service (*logistical QoS*) is motivated by the concept of a generic, low level network functionality that can offer NGI application developers a wide variety of new opportunities to innovate.

The focus on quality of service is a natural one. From its inception a principal goal of NGI has been to find a way to provide reliable, scalable, cost effective, and deployable *delivery of data with quality of service guarantees* as a foundation for innovative NGI applications. To achieve such QoS delivery, current approaches rely on standard end-to-end communication, in which the messages are buffered only minimally between the sender and the receiver. While schemes to provide reliable end-to-end QoS are being actively pursued on a number of fronts, the well known problems with the scalability, cost, and deployability of end-to-end QoS continue to trouble the network community.

Our concept of logistical QoS is a generalization of the typical model that permits much more flexible use of buffering of messages in order to achieve QoS delivery without difficult end-to-end requirements. Whenever data is available to be sent well before it needs to be received, it can be *staged*, i.e. *moved in advance to a location close to the receiver for later delivery*. Isolating the act of buffering data as a distinct operation, independent of delivery to the receiver, opens up a new dimension of freedom in the management of communication and storage resources. Decisions about staging can then be driven by heuristics, scheduling, or even policy.

The remainder of this paper is organized as follows: In Section 2 we elaborate the concept of logistical QoS and describe the mechanism for data staging that we will develop called the *Internet Backplane*. In Section 3 we describe our work on application-driven logistical scheduling in distributed computation. This work will use NetSolve [1–3] which is middleware for doing distributed computation, and the Network Weather Service, which provides dynamic information about processor loads and network traffic to NetSolve for the purposes of scheduling. Together with the Internet Backplane, these components allow us to build a system that utilizes logistical QoS to maximize computational performance while conserving network resources. In Section 4, we describe the wider benefits of implementing logistical QoS as a basic network functionality within NGI.

2. Logistical QoS and mechanisms for data staging

2.1. Background: logistical QoS in the context of NGI

The idea of logistical QoS arises from a basic distinction in the requirements that certain applications have for moving data across the network. Consider, for example, an application of collaborative telemedicine that is likely to become commonplace with NGI. Two doctors, Dr. S and Dr. J, who live in different cities want to meet to discuss a particular patient's diagnosis and the treatment alternatives. Though it would obviously be convenient to hold their meeting on the network, there is a substantial store of information of various kinds—the patients medical records, X-rays, CAT scans, MRI and PET images, etc.—that they may need to reference during the course of their discussion. This is a situation for which one of the emerging forms of "networked collaboration", i.e. synchronous communication among distributed participants with shared viewing and control of documents and other digital objects, would seem to be ideal, especially if the session is supported by high-performance networking with QoS.

But when we examine such an on-line, multimedia consultation with a view to providing QoS support, there seems to be a broad division between two different components of the doctors' communication and a clear distinction between the QoS requirements associated with each. On the one hand, there is the CONTEXT for the communication, i.e. all of those data objects (e.g. MRI and PET images) that exist prior to communication and so can be referred to and talked about during the communication. On the other hand, there is the CONVERSATION that will occur during meeting itself, including not only what they say to each other over the audio stream, but also perhaps what they type in a chat window or what their faces express across a two-way video feed. From a QoS point of view, these two components of the overall communication can be handled very differently:

- *Delivery of contextual data can be staged*—Since the contextual data exists prior to the communication, the data can be moved in advance from an end node to some intermediate point (which we call a *data depot*) and stored there in a *data staging step*. Then the data is *delivered* from the depot to the appropriate node as called for and with the required quality of service. Both movements of the data will require network resources, but the data staging step will be able to use best effort, class of service, or reservation at very low levels. The final delivery step does have to meet the QoS specified by the receiver, but this operation can now be performed through an over-provisioned network, across a single switch [4], or within a local ATM cloud. This is logistical QoS.
- *Delivery of conversational data needs standard end-to-end QoS*—Data cannot be staged when it either has to be received in real time as its created or when the decision to send it cannot be anticipated in advance. Conversation is made up of both these types of data. The typical approach to achieving QoS for these cases is *end-to-end reservation* [5,6] The network resources necessary to send the data with required quality of service are declared in advanced by the respective senders and guaranteed by the network to be available.

The difference between logistical QoS and end-to-end QoS is made clear in Fig. 1. Here, A is the sender, B the

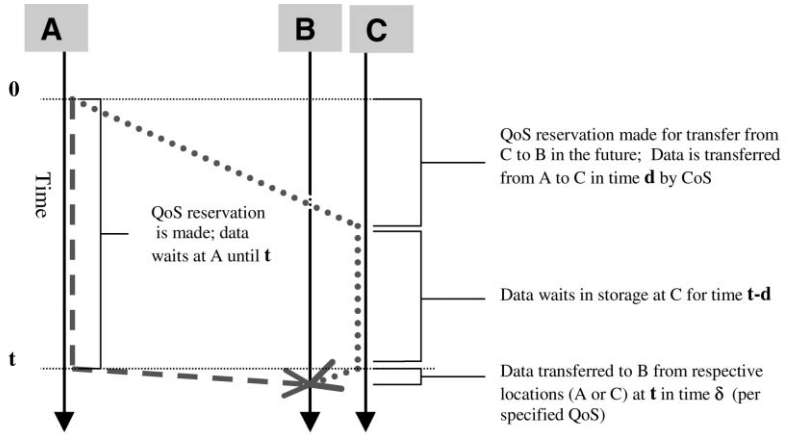

Fig. 1. Scheduled delivery of data in a standard QoS (dashed line) and logistical networking (dotted line).

receiver, and C represents a data depot. The figure compares the two forms of data delivery for a simple case in which a communication is scheduled at time 0 to occur at time *t*:

Standard QoS (dashed line) moves *all* the data components of the communication in a single step. By contrast, logistical QoS (dotted line) is composed of a data staging step and a communicative delivery step. For those data elements that can be staged, the result at the receiving end can be just the same quality as end-to-end delivery in one step, but the work of the sender and the network has been spread over time and storage space. In fact, some QoS requirements which are impossible to achieve through end-to-end communication may be possible through logistical QoS. An example is latency requirements lower than the limit imposed by the speed of light, which is particularly important when using satellite links. Logistical QoS can bring the data closer to the receiver and push perceived latency below this limit, whereas end-to-end QoS treats communication as if it were all "conversation" and no "context" and so is limited by the physics of real-time interaction.

Logistical QoS can be characterized as *application-driven* because advanced prediction of the resource requirements for the transfer and final delivery of stagable data depends on the nature of the application. Allowing QoS to be application driven in this way has a number of advantages. For instance, logistical QoS can hide communication "problems" like latency variance (commonly called jitter) and bandwidth variance from the user which would otherwise have to be ensured via end-to-end QoS mechanism. By knowing the response time requirements of the application, the system can use logistical QoS in place of end-to-end QoS where appropriate to provide service of identical quality from the user's point of view. Making this substitution helps to conserve resources by unburdening end-to-end QoS reservation mechanisms from having to handle this immense class of data. It thereby frees these resources for other applications with real-time response requirements. Furthermore, it allows our data staging protocol (described in Section 2.2.2) to optimize its use of the resources. The staging software can decide how, when, and where to move the data so that the *best* (e.g. cheapest or most reliable) resources are used to accomplish the job. This optimization process allows the protocol to leverage existing end-to-end QoS mechanisms only where it is profitable to do so.

2.1.1. The Internet backplane as an architecture for NGI computing

Our approach to data staging uses the idea of the *Internet Backplane*. The concept of the Internet Backplane is a synthesis of trends in distributed operating systems [7], from the World Wide Web, and from the development of the Network Computer and other appliances which rely on network resources for basic services. Current work growing out of seminal work of David Tennenhouse in Active Networks at MIT LCS augments the network with compute capability provided as part of the communicative fabric rather than attached at the periphery. However, the compute power of Active Networks is dedicated to enhancing the functionality of the network and is not available for application level computing. The development of the PC-based computational infrastructure and the growth of the Web show that the most powerful combination is for state managed by end user to actually reside within end user organizations. At the same time, the World Wide Web has shown the phenomenal power of allowing uniform access to resources located throughout the network. The Web was designed for the commodity Internet, and so supports only loosely coupled operations, but NGI provides reliability. Reliability gives us control from a distance, and control enables management of resources shared by closely coupled distributed operations.

Despite its shortcomings, the World Wide Web has become the distributed file system of the commodity Internet. NGI allows us to not only access files better, but to consider new forms of access which require remote management of resources. NGI enables us to consider the global network as an extension of the processor backplane, if we

only had a low overhead mechanism for fine-grained naming and access to data, analogous to physical addresses and bus transfers. It is this analogy which gives rise to the notion of an Internet Backplane: a common namespace for fine-grained management of resources. The thrust of our project is the use of the Internet Backplane to implement data staging, but we are convinced that the development of such a facility will have far reaching applications beyond the implementation of logistical QoS. We discuss those implications further in Section 4.

2.2. Mechanisms for data staging

In order to implement Logistical QoS we must have a mechanism for data staging. An important element of our project is the development of such mechanisms. Data staging can be layered on top of the protocol stack, and then requires the overhead of being received and processed by an intermediate process before being delivered. Another approach is to integrate it lower in the protocol stack, modifying existing protocols or designing new ones. In this section we describe the functionality required for various approaches to data staging, and we specify operations which would be supported, regardless of the specific implementation approach. Deployable, well-engineered approaches to adding such functionality to concrete protocols will be one product of our research.

2.2.1. Logistical connections

The most straightforward, but least general way to allow message staging is to extend the semantics of a TCP/IP connection to allow explicit management of buffers that reside at the receiving node or at unspecified depots along the connection's path. In such a protocol, the sender may send data to a buffer, which may be triggered for delivery to the receiver by either the sender or the receiver. Note that the buffer may be reused, overwritten or re-sent, thus enabling some of the logistical QoS services described above.

The benefit of this formulation is that there are no issues of buffer naming or protection, since buffers are not visible except to the participants. However, the limitation is that a single TCP/IP connection must be maintained, and it is not possible to stage data at a depot that is not part of the connection. To allow freer control of buffer management at depots, we must formulate a more general scheme for naming, staging, delivering and protecting data. We call this the Internet Backplane Protocol (IBP).

2.2.2. The Internet backplane protocol

Consider the infrastructure requirements for staged delivery of a message from sender A to receiver B using depot C. A must be able to stage a message at C and arrange for it to be delivered at a later date during an interaction with B. If there is no end-to-end component to the interaction between A and B, then a simple timer might be used at C. However this would be unusual—it would be much more common for a staged message to be delivered as part of an interaction which also included end-to-end communication. In order for the staged message to be coordinated with the overall interaction, there must be some coordinating signal from either A or B to trigger its delivery.

A minimal protocol to support staged delivery would have two operations: storing (or *staging*) a message at depot C and triggering delivery to node B of a message stored at depot C. One problem with this simple scheme is that messages are often used to implement transfer of a large byte array, and so are sent in packet flows (streams of packets with the same source and destination). The need to send a flow with guaranteed QoS rules out a protocol that requires a separate trigger for each message in the flow. We must be able to stage the entire flow and trigger its delivery with a single operation. Another problem is the need for security. If access to stored messages is not secure then communication can be intercepted at any depot. Finally, we must be able to flexibly allocate resources at a depot to different connections.

One direct solution to these problems is to allow large byte arrays to be stored at the depot, and allowing the entire array to be turned into a packet flow and delivered to the recipient with a single operation. The original sender or the recipient may trigger this delivery. Other important functionalities that must be present are remote allocation and management of byte arrays, and copying of byte arrays from one remote node to another.

We have implemented the following API for IBP version 1.1

```
Cap_set = IBP_allocate(C, size, attributes)
```

This allocates a byte array with maximum size *size* at depot C. A *capability set*, composed of a read capability and a write capability, is returned to the caller. These are used by IBP clients to access the byte array. Note that the clients as such do not choose a name for the byte array. The name is implicitly part of the capability set returned from the `IBP_allocate call`. The *attributes* contain information about how the IBP server should store the byte array. Currently, there is only one attribute, *volatility*, which can take one of two values: *non-volatile*, which means that the server will store the byte array until it is explicitly deleted, and *volatile*, which means that the IBP server may delete the byte array at its own discretion.

```
IBP_store(write_cap, ptr, size)
```

This stores *size* bytes of data, starting at the pointer *ptr* to the byte array referenced by the write capability *write_cap*. Note that the location of the depot is encoded as part of *write_cap*. `IBP_store` has append semantics, meaning that multiple `IBP_store` calls may be made to the same byte array.

```
IBP_copy(read_cap, write_cap)
```

This copies the byte array referenced by *read_cap* to the

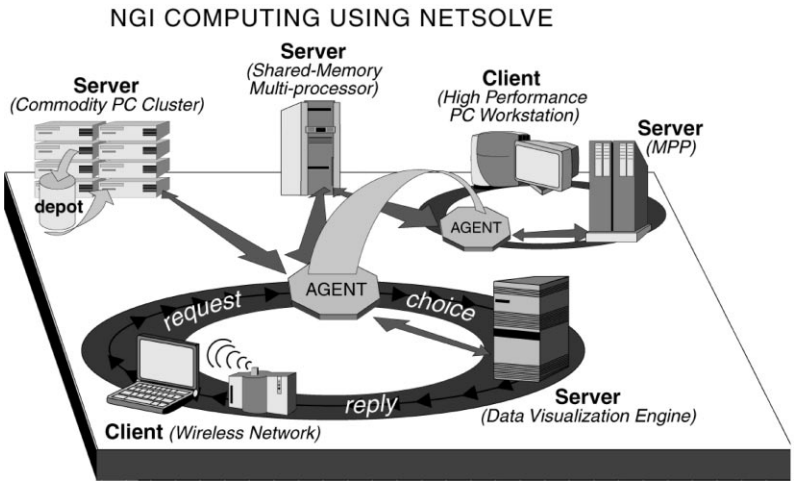

Fig. 2. Netsolve enabled computational grid with IBP servers providing state management.

byte array referenced by *write_cap*. Like `IBP_store`, this has append semantics. Note that the source and destination byte arrays can be on different hosts, which means that `IBP_copy` triggers a remote copying of data.

`IBP_deliver(read_cap, host, port)`
This delivers the byte array referenced by *read_cap* to the named host and port over a TCP/IP socket. Once again, the host of the byte array, and the target host for delivery may be different, meaning that `IBP_deliver` triggers a remote delivery of data from IBP to a client.

`IBP_decrement(capability), IBP_increment(capability)`
These allow the clients to maintain reference counts with a byte array's capabilities. `IBP_allocate` initially sets these reference counts to one for both read and write capabilities. If a client decrements a write capability's reference count to zero, then the associated byte array becomes read-only. If a read capability's reference count becomes decremented to zero, the associated byte array is deleted. Note that volatile byte arrays may be deleted arbitrarily by the IBP server, but while this affects the success of `IBP_store`, `IBP_copy` and `IBP_deliver` calls, it does not affect the values of the reference counts.

`IBP_probe(capability)`
This call allows clients to determine the state of a byte array, for example, whether it exists, how large it is, and the reference count of the capability.

Thus, a fundamental feature of IBP is the use and management of capabilities instead of names. In IBP 1.1, each capability is simply a character string, which may be copied from client to client if so desired. If additional security on top of IBP is desired, the capabilities may be encrypted.

The `IBP_copy` operation allows a node to remotely move staged data from one depot to another for reasons of scheduling, proximity or reliability. Thus, IBP provides a flexible interface for logistical QoS. IBP has been implemented in on top of TCP/IP connections using a daemon at each depot. The one unfortunate aspect of connection-based delivery is that it is awkward to intersperse staged and end-to-end communication. A UDP protocol would be more flexible in this regard. Another approach would be to modify TCP/IP to allow the participation of proxy senders such as our depots.

3. Scheduling for logistical QoS in NGI computing

In the preceding section we have motivated and described primitive mechanisms for data staging in support of logistical QoS. While various mechanisms could be used, IBP is the most general of these and for the purposes of this discussion we will assume that it is the one in use.

Like other such mechanisms, IBP requires that communicating applications take responsibility for making decisions about when and where to stage and then deliver data. But data staging decisions are non-trivial, so not every application will be able to make them effectively. Distributed computing in the dataflow model is a special case that lends itself especially well to scheduling and resource allocation. In the dataflow model, computation is performed by functional components which take input values and produce output values but have no side effects. This means that a functional component can execute on any network node that has the available resources as long as inputs and outputs can be transmitted with the quality of service required for high performance computing [8].

The dataflow model, as implemented in the Netsolve system described in the next section, is particularly appropriate to *logistical scheduling* (i.e. scheduling for logistical QoS) because data staging decisions add another dimension to the space of possible schedules but otherwise is compatible with the Application Level Schedulers (AppLeS) [9–11] scheduling paradigm. In addition, when the schedule involves several functional steps, the dependence graph

between these steps forms a run time dependence graph that can be analyzed to find the critical path through the computation. This analysis allows us to identify communications with quality of service requirements that are not on the critical path. We can use logistical QoS rather than end-to-end QoS on these communications with no loss of overall performance.

3.1. NGI computing with Netsolve and the network weather service

3.1.1. NetSolve

NetSolve makes an excellent test environment for experiments with logistical scheduling. It is a software environment for networked computing designed to transform disparate computers and software components into a unified, easy-to-access computational service. It aggregates the hardware and software resources of any number of computers that are loosely connected across a network and offers up their combined power through client interfaces that are familiar from the world of uniprocessor computing (e.g. MATLAB, simple procedure calls). It uses a client–agent–server paradigm (Fig. 2) to deliver the power of distributed resources while hiding the complexity of the underlying system.

As shown in the figure, a NetSolve server may be arbitrarily complex: a uniprocessor, a MPP (Massively Parallel Processor), a Shared-Memory Multiprocessor, or a cluster of workstations. Moreover, there is no upper bound on the number of servers that can be aggregated to form a NetSolve resource pool, and new servers can be added on the fly with little effort. This quality of dynamic composability contributes to NetSolve flexibility.

When a user wants a certain computational task to be performed, he/she can use any one of a number of familiar software clients to contact an agent with the request. A NetSolve agent keeps track of information about all the servers in its resource pool, including their availability, load, network accessibility, and the range of computational tasks that they can perform. The agent then selects a server to perform the task, and the server responds to the client's request. The client's computations are performed remotely at the server. When the agent informs the client software of the best available resource to service the request, the client's data is sent to the server and the server uses its installed software components to perform the computation. When the server finishes the computation, it sends the result back to the client and the agent is notified of the completion of the task.

As shown in Fig. 2, there may be multiple instances of NetSolve agents on the network, and different clients may contact different agents depending on their locations. The agents exchange information about their different servers and allow access from any client to any server if desirable. NetSolve can be used either on the open Internet or on a private Intranet (e.g. inside a department at a university) without participating in any wide area computation.

3.1.1.1. Current state of NetSolve development. At present NetSolve has been deployed and is being used in several research institutions. The current version implements NetSolve's client–agent–server model and includes a suite of software for a variety of clients and servers. The source code is available at http://www/cs/utk.edu/netsolve. Windows 95/NT versions of the client (C, FORTRAN and MATLAB [12]) are available and an NT server version is being developed. The NetSolve team has integrated a variety of numerical and scientific packages into NetSolve computational servers. These numerical libraries cover several fields of computational science, including Linear Algebra, Optimization, and Fast Fourier Transforms. Moreover NetSolve's highly composable architecture allows anyone to easily add resources to the system. The most significant advances in the recent development of NetSolve include the integration of ScaLAPACK (Scalable Linear Algebra Package) [13] to exploit the power of multiprocessor NetSolve servers using PVM [14] or MPI [15] and an interface to the Condor [16–18] system. Applications currently being developed using NetSolve come from diverse fields, including image processing and neuroscience simulation.

3.1.1.2. NetSolve on the NGI. NetSolve is a novel approach to designing middleware for building computational grids across high performance networks such as NGI. Its philosophy is to give access to distributed computational resources via simple and unified interfaces. This approach brings a great level of flexibility to the NetSolve users, and allows NetSolve to adapt to different classes of users. Indeed, the NetSolve user community is composed not only of people who use NetSolve as a computational engine, but also those who use it as an operating environment for building domain-specific metacomputing applications. As a result, NetSolve is experiencing a rapid increase in popularity and is likely to generate substantially higher demand on the network as it is used for more complex and diverse applications. There are many situations in which NGI will be essential to enable NetSolve to satisfy these new requirements. Examples include:

- Real-time applications that enable visualization by transforming the user's input data into a stream of video images.
- Some computing environments may provide a NetSolve system access to CPUs only for limited period of time. In such settings, it is crucial to be able to transfer large amounts of data to the CPU in time, before it is preempted.
- We have started experimenting with the use of multiple NetSolve servers that take part in the parallel execution of large iterative algorithms. Such applications require a

very low latency since the servers need to be synchronized at every iteration.

A standard way of addressing the question of how to provide NetSolve with the necessary performance guarantees is by using end-to-end QoS mechanisms to reserve bandwidth. However, not every example has an absolute need for end-to-end QoS. If the communication involved is not on the critical path of computation, logistical QoS can be used instead. The advantages to the Netsolve user are many: the use of possibly expensive wide area QoS services are minimized, as is the chance that the network will have to reject the communication and thus halt the computation due to congestion. The problem then becomes one of designing schedulers that can find opportunities for using logistical QoS through data staging and can take account of them in finding high performance, low cost schedules. The implementation and design of such schedulers is the object of Section 3.2.1, and they are clearly leveraged by IBP as it has been defined in Section 2.2.2.

3.1.2. Network weather service

The other key component for this NGI computing strategy is the Network Weather Service (NWS) [19]. NWS periodically monitors the performance that is deliverable to an application from system resources, and forecasts what that performance will be for the time frame in which the application will execute. By working with forecasts of resource performance, scheduling agents can craft a schedule which makes the best use of the resources that *will be* available at the time the application will execute.

To generate dynamic performance forecasts, the NWS operates a distributed set of *sensors* from which it gathers readings of the instantaneous conditions. Sensors return measurements of deliverable resource performance such as network bandwidth and latency between hosts, or CPU availability as a percentage. These measurements may either be based on available performance data or may result from a small probe that the system conducts of each resource. For example, to measure CPU availability, the NWS periodically runs local performance monitoring utilities such as *uptime* and *vmstat* (on Unix systems) and attempts to calculate from the resulting data the fraction of the CPU that is available as a percentage of time. Alternatively, to measure network bandwidth, the NWS moves a small amount of data between hosts and times the transfer. This latter approach potentially introduces some load on the network being measured. Active performance experiments such as the bandwidth sensor must be tuned so that the amount of load they introduce is relatively insignificant compared with the available resource capacity. The current supported NWS sensors are CPU availability, end-to-end network bandwidth, and latency. While end-to-end network measurements often imply scalability problems, the NWS organizes its network performance experiments hierarchically so that only representative host pairs are queried.

The NWS then treats any series of measurements taken from a sensor as a statistical time series for the purposes of forecasting. To make forecasts, the system applies a variety of different models to the relevant series simultaneously. Each separate model then produces a forecast of what the next measurement in that series will be. When the next measurement is available from the sensor, the NWS calculates the forecasting error associated with each of the time series models. The forecast generated by the model that yields the lowest cumulative forecasting error at any given time is reported as *the* forecast.

3.2. Scheduling with logistical QoS for NGI computing

To model our scheduling problem, we consider a computing environment that can be viewed as a completely connected graph where the vertices are called *locations* and the edges are called communication *links*. The links can be used to exchange data between two locations, and the performance of the communication along one link at a certain time is measured by a metric that may be multidimensional. This model is of course applicable to the Internet, with varying communication latencies and bandwidths between different sites. Each location can be the host to a *processing unit* (workstation, cluster of workstations, MPP, etc.) and to any number of *storage units* (memories, disks, etc.) that are accessible to the processing unit. Some of the storage units will play the role of *depots* where data can be staged for logistical QoS. Each processing unit runs a *server* (e.g. a NetSolve server) that can grant users access to CPU time on the unit. *Software components* that can be used to perform users' computations on processing units are stored on some of the storage units in the network. Finally, the input data to a user's computation are also stored on some storage units (e.g. on the user's host).

Let us suppose that the user performs computation on such a system during a *session*. At the beginning of the session, an initial set of input data is available on some of the storage units. The user performs possibly several computations, each of them generating output data that can be input to a subsequent computation. More generally, a computation can be viewed as having a "context" that consists of user data (original input, intermediate data) and possibly of computation specific information (checkpoint files, temporary files). That data can be used by different components of the computation and can be *replicated* across different data depots in advance to increase the benefits of locality. This is what we call *staging* in the context of this document. In this section, we explain what research is involved in the design of the staging-capable schedulers.

We postulate that a computation can be segmented in the four following steps:

1. choosing a processing unit,
2. making sure that all the input data and the software components are available to the processing unit,

3. performing the actual computation, thereby generating output data,
4. possibly moving output data to other locations.

Note that in our model, the software components can be seen as read-only user data.

To provide a model of this process it is convenient to define the *data map* abstraction. A data map is a function that associates each remote storage unit with the set of software components and user data present at that unit during a session. Before any computation has been performed, the data map is such that no location contains any data that is the output of a computation. Several depots can contain the same user data, modeling replication. The two *basic operations* that can be performed by the system can now be viewed as functions that operate on a data map to generate another data map. The first is a computation that generates output at the location where the chosen processing unit for the computation is located. The second is a *transfer* that moves data around among distributed storage units. One can associate a cost to each operation. The cost of a transfer, for instance can be the estimated by the aggregate time that it takes to communicate data over the different links. The cost of a computation can be represented by the estimated time for a processing unit to access the software components and input data and to generate output data. A session can be seen as a succession of basic operations, and the sum of all the costs of these operations is the cost of the entire session. We want to emphasize here that even though time is the most intuitive cost, the notion of cost is fairly general and the cost may be in fact be multi-dimensional.

3.2.1. Building a scheduler

The objective of a scheduler in this setting is to minimize the cost of a user's session, resulting in increased performance for the end-user. Minimizing costs can be achieved by data staging on the appropriate data depots in conjunction with corresponding choices of the computing units to be used during a user's session. The scheduler must therefore have some knowledge of what the user intends to do during his/her session. Different levels of knowledge are available depending on the paradigms of the application the user is using to access the computational environment (e.g. NetSolve). For instance the user may send requests to the scheduler one by one at run-time. This is the current NetSolve paradigm. In such a case, we say that the scheduler must perform a *one-step* decision to assign a processing unit to the user's computation. One can envision a more sophisticated paradigm where the user specifies *simultaneous* computations at once and several processing units must generally be assigned to each computation. It is also possible that the user can specify a sequence of groups of simultaneous computations and provide that information for the scheduler. In that case, we say that the scheduler must make a *multi-step* decision. The most general cases would be for the user to specify an entire data-dependency graph for his/her session and let the scheduler process the graph to decide on a schedule in a multi-step fashion.

In order to make decisions about a particular schedule and evaluate the corresponding costs, the scheduler must *predict* the state of the computing environment in terms of network performances and processor loads. We intend to use the NWS and integrate it with the IBP protocol for such predictions (see Section 3.2.1). Depending on the confidence in the prediction, the scheduler will choose the appropriate level of sophistication for the schedules. Indeed, the time spent to improve a scheduling decision must be chosen according to the expected level of accuracy of the prediction.

The AppLeS project proved that the use of schedulers embedded in agents can take distributed applications to new levels of performance in dynamically changing distributed environments. However, that project also showed that not only the design, but also the implementation of such scheduling agents is very difficult. NetSolve's framework hides the low-level details of distributed computing, but it offers only rudimentary scheduling strategies for the moment, especially since it does not try to perform data staging. Combining the NetSolve framework with the scheduling philosophy of AppLeS is well positioned to provide an enhanced environment for the efficient execution of distributed applications. The deployment of such an environment on the NGI will be leverage the IBP protocol of Section 2.2.2.

3.2.2. Cases of interest

In order to experiment with the IBP and to better understand the scheduling issues with staging in the computing environment described in the previous section, it is possible to implement a succession of schedulers that correspond to cases of increasing complexity. We will first consider *one-step schedules* that consist of evaluation of a single functional computation and then consider *multiple-step schedules*, which consist of multiple functional computations connected by data dependencies.

NetSolve's current model

This is the most direct implementation, since the current NetSolve model requires only a one-step schedule and there is no data staging. In the data dependence graph of this case, all data flows between the client and the server. There is already a widely used implementation of such a paradigm. However, re-implementing such a simple scheduler provides a perfect testbed for the IBP and would undoubtedly provide feedback to the IBP implementers.

NetSolve with basic staging

Many members of the NetSolve user community have expressed a desire for "caching capabilities" at the server side. Instead of implementing an ad hoc solution, using a combination of IBP and a real scheduler makes it possible to allow the servers to keep copies of some of the user's input and output data. This is really a sub-case of data staging where each server is also a data depot, and in the data

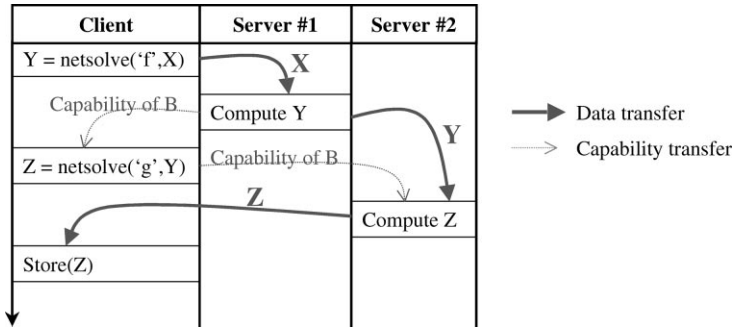

Fig. 3. Netsolve using IBP to achieve performance via data staging.

dependence graph all data flows either between the client and the server or is a loop from the server to itself (in the case of consuming cached data). This scheme can lead to replication of a piece of data on more than one server, but a server can only copy data from the client. In order to manage the cache the scheduler has to make some assumptions about the probability of using of the data in future computations at that server, but this is still be a one-step schedule for the scheduler. The performance of NetSolve would be greatly improved by such a scheduler, as it would allow NetSolve to use data locality within a computation for the first time.

NetSolve with general staging

In this case, the scheduler can perform any transfer it deems appropriate in order to minimize the cost of the user's computations. For instance, a server might acquire one part of a user's input from the client host, and one part of the input from another nearby server. The data dependence graphs are quite general in this case. This can still be done in a one-step schedule. In Fig. 3, we give an example of how the IBP mechanisms can be used by NetSolve to achieve performance via data staging. In this example, the user wants to perform two computations during his/her session and the output of the first computation is the input to the second computation. However, the first computation can only be performed on *Server* 1, whereas the second computation can only be performed on *Server* 2. This might be due to the presence of essential software components that cannot be exchanged among the servers (e.g. for architecture incompatibility reasons). Such cases are very common and reported by many NetSolve users. The left part of the figure shows the instruction flows on the three participant hosts as well as the data transfers. The output to the first computation, *Y*, stays staged on *Server* 1 (which is also a data depot) and is delivered to *Server* 2 when it is needed for the second computation. Thanks to IBP, it is possible to exchange capabilities instead of actual data, and to use those capabilities to access staged data. A scheduler decides that the computations should be performed as it is shown in the example, and then uses IBP to implement such a schedule.

More advanced models

The natural extension to the three previous schemes is to add multi-step decision making to the scheduler. This greatly increases the number of possibilities and parameters in the decision making process, especially when one considers the full dynamic data-dependency graph. The implementation of such schedulers will face difficult theoretical problems as highlighted in Section 3.2.3. At the moment, NetSolve does not provide any user interface that could make use of a multi-step scheduler because data staging cannot be conveniently implemented. Therefore, multi-step schedules must return all data to the client at every step, and so present no advantage over one-step schedules.

Constraint-based models

One issue we have not mentioned so far is the fact that there might be several constraints on the system. One example is limited disk space in the storage units. There may be situations where there is a shortage of disk space before a user's session is terminated and the scheduler must then choose to discard some of the cached/replicated user data. Another example would be in the case of non-free resources, where a certain amount of money is charged for the use of a resource (storage or processing). The user may then ask the scheduler to minimize the time it takes to perform the session while staying below a fixed monetary cost.

3.2.3. Implementation of schedulers

As we have already mentioned, the schedulers will be implemented on top of IBP. The use of IBP frees the implementers from the hassle of data logistics and lets them concentrate on the important theoretical issues. We assume that schedulers will be making decisions at run time, using predictions from the NWS to make scheduling decisions. However, we have seen that for general paradigms (multi-step, data-dependency graph), finding the optimal schedule can require extensive computations. In fact, the general cases are bound to be NP-complete. The schedulers will then use heuristics, still being developed, to approximate an optimal schedule. The trade-off between the additional time spent to improve a scheduling decision and the corresponding gain in session cost is a very important issue and will be decided based on the confidence in the NWS predictions. For a highly predictable environment, it is worth spending some extra time refining the schedule, whereas a more naive schedule would be sufficient in a more uncertain

environment. In addition to the use of prediction confidence, and in the case of multi-step decision making, the scheduler should periodically re-evaluate its schedule to steer and adapt it to changing network and processor conditions.

In the NetSolve framework, such schedulers will be implemented within the NetSolve agent. IBP, however, will be integrated with every component of the software, including client interfaces, agents, and servers.

3.2.4. Extensions for fault tolerance

A subsidiary use of IBP and logistical scheduling is to extend NetSolve's support for fault tolerance. The NetSolve system is that can already be used to deliver fault tolerance transparently through the use of checkpointing mechanisms implemented completely on the server. However, such checkpointing produces data that must be stored outside the server in order to allow it to be retrieved in case of failure. In cases where the storage containing the checkpoint can fail, more than one replica of this data must be maintained. This is particularly true in cases where storage is "borrowed" from servers which can reclaim it if needed.

As is well known in checkpointing community, the time taken to produce and store a checkpoint can be huge in the case of parallel computations on large data sets. For this reason, location of the checkpoint data is of crucial importance. Logistical scheduling can be used as a way to achieve high performance through the use of data locality, but also as a way to implement fault tolerance across different computing resources. One can envision storage servers whose only role is to store checkpoint files until corresponding computations have completed successfully. Such servers fit perfectly in our model since the checkpoint file can be modeled as an output that is not visible to the user.

3.2.5. The NWS and the Internet backplane

We plan to integrate the NWS and the Internet Backplane so that the NWS is aware of the existence of storage resources at IBP depots and can track and predict their availability for any particular computation. Programs written to use IBP can then be made *resource aware* and hence optimize their resource usage, including their use of distributed storage, "on-the-fly."

Our initial experiences with NetSolve show that software infrastructure can shield the user from the complexities of network programming while delivering good performance (see Section 3.1.1). Similarly, experiences with the NWS and AppLeS indicate that scheduling agents which consider dynamically varying resource performance levels perform well in network environments [9–11]. To combine the benefits of these two separate approaches, we must enhance the current NWS functionality by developing new resource sensors for the IBP and extending the current forecasting system.

Since the application interface can often impact performance dramatically, it is important to monitor performance levels from the IBP layer itself. To do so, we are developing IBP performance sensors to be integrated into the overall system. In particular, the storage performance of potential staging sites will be an important parameter for an IBP scheduler to consider. We will develop both passive storage performance sensors based on current monitoring technology and active storage performance experiments as part of this work. Similarly, the availability of "in-core" memory space at potential staging and execution sites will be important to IBP scheduling. The current NWS memory sensors are rudimentary and can make only a gross approximation of available real memory.

We will also need to develop longer range forecasting techniques for the IBP. Current NWS forecasting methods yield one-step-ahead forecasts. A scheduler using NWS forecasts must consider them valid for only a single future sampling period, typically 10–60 s. Since the time frames of interest to IBP applications may be large, we will require longer range forecasting as well. In addition, current network infrastructure provides high-fidelity capacity monitoring facilities. It is possible, for example, to get accurate packet arrival rates from various network routers and gateways. We will develop *combinational forecasters* that correlate existing monitor data, particularly for network resources, with end-to-end performance measurements to derive more accurate forecasts. Our initial experiences with such hybrid techniques indicate that they can yield better forecasts than their univariate counterparts.

4. Conclusion: expanding our definition of the network

The network is often thought of as consisting of resources which hold no application state. There is no model of storage use other than file servers and file caches. The Network Computer is distinguished from a PC by the lack of stored application state, and so is often characterized as "diskless". But since a diskless NC cannot achieve adequate performance, developers have already added a disk for the purpose of caching application state held in file servers. Caches are transparent, but they are not flexible. A flexible, low overhead-distributed storage model is needed, but there is no appropriate standard.

Adding storage to the definition of the network will enable a new generation of application designers to make much stronger assumptions about the level of control they can exercise over remote resources. The new communicative capabilities of the NGI will be multiplied by the possible ways in which information can be staged and delivered.

One effect of increasing control over storage at the network's periphery is that the storage itself can be implemented in the network core. The application-specific decisions about exactly where to place data and when to move it can be made by applications using facilities such as the NWS or even offloaded to services implemented completely

within the network. In this scenario, a network can model flows that persist over weeks or years as well as flows that are over in a few milliseconds. Information will flow at an appropriate pace without creating artificial boundaries, and the network will manage all of it. Other consequences include the following:

The power of global naming

When a pool of resources is named by a set of names that have a global interpretation throughout the network, then we have the power of pointers. A name can be stored or communicated at will without any danger of leaving its scope. The Web created global names (URLs) for files accessed by application-level protocols. This created a community of users. IBP creates global names for files accessed by a very lightweight protocol, and so will create a community of applications.

Flexible storage management

The key to creating a high performance interface to any architecture is flexibility: the ability to make it enter any state and exhibit any behavior that is possible at the lower level. It is very difficult to create a high performance interface that also hides basic resources. The TCP/IP interface hides buffer management, and so cannot model staging except by the introduction of unnecessary high level overhead such as setup of multiple connections. IBP expresses storage explicitly, and thus allows a much greater degree of flexibility in its management.

Storage enables scalability

Since a stored file can be copied, each copy can serve a large number of user and can also be used to create subsequent copies. Thus the dissemination of any file defines a multicast tree which exists over time as well as space. If there is adequate support for the convenient location and creation of subsequent copies, then this is the most scalable approach to data dissemination possible. Using multicast delivery of messages when storage dissemination will do is convenient but destroys scalability. File dissemination can be made convenient instead.

Storage vs. network: the trade-off changes

Storage does not trade off well against the local area network because LAN bandwidth (100–1000 Mb/s) has overtaken disk bandwidth (40–80 Mb/s). Disk compares very well to typical flows in a next generation wide area network (10–20 Mb/s).

References

[1] H. Casanova, J. Dongarra, K. Seymour, Client User's Guide to NetSolve, 1996.

[2] H. Casanova, J. Dongarra, NetSolve: a network server for solving computational science problems, The International Journal of Supercomputer Applications and High Performance Computing 11 (3) (1997) 212–223.

[3] H. Casanova, J. Dongarra, NetSolve's network enabled server: examples and applications, IEEE Computational Science and Engineering 24 (12–13) (1998) 57–67.

[4] J. Crowcroft, et al., A rough comparison of the IETF and ARM service models.

[5] A. Banerjea, et al., The Tenet real-time protocol suite: design, implementation and experiences, IEEE/ACM Transactions on Networking 4 (1996).

[6] L. Zhang, et al., RSVP: A new resource ReSerVation Protocol, IEEE Network September (1993).

[7] J.H. Morris, et al., Andrew: a distributed personal computing environment, Communications of the ACM 20 (3) (1986) 184–201.

[8] A. Beguelin, et al., Graphical development tools for network-based concurrent computing, in: Supercomputing '91, Albuquerque, 1991.

[9] F. Berman, et al., Application-level scheduling on distributed heterogeneous networks, in: Proceedings of the Supercomputing '96, Pittsburgh, 1996.

[10] F. Berman, R. Wolski, Scheduling from the perspective of the application, High-Performance Distributed Computing. Proceedings of HPDC, August 1996.

[11] N. Spring, R. Wolski, Application level scheduling of gene sequence comparison on meta-computers, in: ACM 1998 International Conference on Supercomputing, 1998.

[12] MathWorks Inc., MATLAB Reference Guide, 1992.

[13] L.S. Blackford et al., ScaLAPACK Users' Guide, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[14] A. Geist, et al., PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing, MIT Press, Cambridge, MA, 1994.

[15] M. Snir, MPI: The Complete Reference, MIT Press, Cambridge, MA, 1996.

[16] M. Litzkow, M. Livny, Experience with the Condor Distributed Batch System, in: Proceedings of the IEEE Workshop on Experimental Distributed Systems, 1990.

[17] M. Litzkow, M. Livny, M.W. Mutka, Condor—A hunter of idle workstations, in: Proceedings of the 8th International Conference of Distributed Computing Systems, June 1988.

[18] J. Pruyne, M. Livny, A worldwide flock of Condors: load sharing among workstation clusters, Journal on Future Generations of Computer Systems 12 (1996).

[19] R. Wolski, Forcasting network performance to support dynamic scheduling using the Network Weather Service, *Proceedings of the 6th IEEE Symposium on High Performance Distributed Computing*, IEEE Computer Society Press, Los Alamitos, CA, 1997 pp. 316–325.