Chapter 4

Coordination of Access to Large-scale Datasets in Distributed Environments

Tevfik Kosar, Andrei Hutanu, Jon McLaren

Louisiana State University, Baton Rouge, LA 70803

Douglas Thain

University of Notre Dame, Notre Dame, IN 46556

Abstract

The data needs of scientific as well as commercial applications has increased drastically over the recent years. This increase in the demand for large-scale data processing has necessitated collaboration and sharing of data collections among the world's leading education, research, and industrial institutions and use of distributed resources owned by collaborating parties. In a collaborative distributed computing environment, data is often not locally accessible and has thus to be remotely retrieved, processed, and stored. While traditional distributed system solutions work well for computation that requires limited data handling, they may fail in unexpected ways when the computation accesses, creates, and moves large amounts of data especially over wide-area networks. In this chapter, we provide some state of the art solutions for handling and coordination of access to large-scale datasets in widely distributed computing environments.

4.1 Introduction

Modern scientific applications and experiments become increasingly data intensive. Large experiments, such as high-energy physics simulations, genome mapping, and climate modeling generate data volumes reaching hundreds of terabytes [42]. Similarly, remote sensors and satellites are producing extremely large amounts of data for scientists [20, 83]. In order to process these data, scientists are turning towards distributed resources owned by the collaborating parties to provide them the computing power and storage capacity needed to push their research forward. But the use of distributed resources imposes new challenges [53]. Even simply sharing and disseminating subsets of the data to the scientists home institutions is difficult. The systems managing these resources must provide robust scheduling and allocation of storage and networking resources, as well as efficient management of data movement.

One benefit of distributed resources is that it allows institutions and organizations to gain access to resources needed for large-scale applications that they would not otherwise have. But in order to facilitate the sharing of compute, storage, and network resources between collaborating parties, middleware is needed for planning, scheduling, and management of the tasks as well as the resources. Existing research in this area has mainly focused on the management of compute tasks and resources, as they are widely considered to be the most expensive. As scientific applications become more data intensive, however, the management of storage resources and data movement between the storage and compute resources is becoming the main bottleneck. Many jobs executing in distributed environments fail or are inhibited by overloaded storage servers. These failures prevent scientists from making progress in their research.

According to the 'Strategic Plan for the US Climate Change Science Program (CCSP)', one of the main objectives of the future research programs should be *"Enhancing the data management infrastructure"*, since *"The users should be able to focus their attention on the information content of the data, rather than how to discover, access, and use it [19]."* This statement by CCSP summarizes the goal of many cyberinfrastructure efforts initiated by DOE, NSF and other federal agencies, as well as the research direction of several leading academic institutions.

Accessing and transferring widely distributed data can be extremely inefficient and can introduce unreliability. For instance, an application may suffer from insufficient storage space when staging-in the input data, generating the output, and staging-out the generated data to a remote storage. This can lead to trashing of the storage server and subsequent timeout due to too many concurrent read data transfers, ultimately causing server crashes due to an overload of write data transfers. Other third party data transfers may stall indefinitely due to loss of acknowledgment. And even if transfer is performed efficiently, faulty hardware involved in staging and hosting can cause data corruption. Furthermore, remote access will suffer from unforeseeable contingencies such as performance degradation due to unplanned data transfers, and intermittent network outages.

Efficient and reliable access to large-scale data sources and archiving destinations in a widely distributed computing environment brings new challenges:

Scheduling Data Movement. Traditional distributed computing systems closely couple data handling and computation. They consider data resources as second class entities, and access to data as a side effect of computation. Data placement (i.e. access, retrieval, and/or movement of data) is either embedded in the computation and causes the computation to delay, or performed as simple scripts which do not have the privileges of a job. The insufficiency of the traditional systems and existing CPU-oriented schedulers in dealing with the complex data handling problem has yielded a new emerging era: the data-aware schedulers. One of the first examples of such schedulers is the Stork data placement scheduler that we have developed. Section 4.3 presents Stork and data-aware scheduling.

Efficient Data Transfers. Another important challenge when dealing with data transfers over wide area networks is efficiently utilizing the available network bandwidth. Data transfers over wide area, and in particular over high-capacity network links make the performance limitations of the TCP protocol visible. In section 4.4, we discuss these limitations and various alternatives. Much of the work related to wide area data transfer is focused on file or disk-to-disk transfer. We will also present other types of scenarios, how they are different, what the challenges are, and possible solutions.

Remote Access to Data. In some cases, transferring complete datasets to remote destinations for computation may be very inefficient. An alternate solution is performing remote I/O, where the files of interest stay in one place, and the programs issue network operations to read or write small amounts of data that are of immediate interest. In this model, transfer protocols must be optimized for small operations, and the processing site may need no storage at all. In section 4.5, we discuss advantages and challenges of remote I/O, and present the Parrot and Chirp technologies as a case study.

Co-scheduling of Resources. Distributed applications often require guaranteed levels of storage space at the destination sites, as well as guaranteed bandwidth between compute nodes, or between compute nodes and a visualization resource. The development of a booking system for storage or network resources is not a complete solution, as the user is still left with the complexity of coordinating separate booking requests for multiple computational resources with their storage and network booking(s). We present a technique to co-allocate computational and network resources in section 4.6. This co-scheduling technique can easily be extended to storage resources as well.

4.2 Background

In an effort to achieve reliable and efficient data placement, high level data management tools such as the Reliable File Transfer Service (RFT)[62], the Lightweight Data Replicator (LDR) [52], and the Data Replication Service (DRS) [21] were developed. The main motivation for these tools was to enable byte streams to be transferred in a reliable manner, by handling possible failures like dropped connections, machine reboots, and temporary network outages automatically via retrying. Most of these tools are built on top of GridFTP [2], which is a secure and reliable data transfer protocol especially developed for high-bandwidth wide-area networks.

Beck et. al. introduced Logistical Networking [11] which performs global scheduling and optimization of data movement, storage and computation based on a model that takes into account all the network's underlying physical resources. Systems such as the Storage Resource Broker (SRB) [8] and the Storage Resource Manager (SRM) [76] were developed to provide a uniform interface for connecting to heterogeneous data resources and accessing replicated data sets. SRMs were mentioned in detail in Chapter 3.

GFarm [66] provided a global parallel filesystem withonline petascale storage. Their model specifically targets applications where data primarily consists of a set of records or objects which are analyzed independently. Gfarm takes advantage of this access locality to achieve a scalable I/O bandwidth using a parallel filesystem integrated with process scheduling and file distribution. The Open-source Project for a Network Data Access Protocol (OPeNDAP) enabled software which makes local data accessible to remote locations regardless of local storage format [67]. OPeNDAP is is discussed in detail in Chapter 10.

OceanStore [55] aimed to build a global persistent data store that can scale to billions of users. The basic idea is that any server may create a local replica of any data object. These local replicas provide faster access and robustness to network partitions. Both Gfarm and OceanStore require creating several replicas of the same data, but still they do not address the problem of scheduling the data movement when there is no replica close to the computation site.

Bent et al. [13] introduced a new distributed file system, the Batch-Aware Distributed File System (BADFS), and a modified data-driven batch scheduling system [12]. Their goal was to achieve data-driven batch scheduling by exporting explicit control of storage decisions from the distributed file system to the batch scheduler. Using some simple data-driven scheduling techniques, they have demonstrated that the new data-driven system can achieve orders of magnitude throughput improvements both over current distributed file systems such as AFS as well as over traditional CPU-centric batch scheduling techniques which are using remote I/O.

One of the earliest examples of dedicated data schedulers is the Stork data scheduler [54]. Stork implements techniques specific to queuing, scheduling, and optimization of data placement jobs, and provides a level of abstraction between the user applications and the underlying data transfer and storage resources. Stork introduced the concept that the data placement activities in a distributed computing environment need to be first class entities just like computational jobs. Key features of Stork are presented in the next section.

4.3 Scheduling Data Movement

Stork is especially designed to understand the semantics and characteristics of data placement tasks, which can include: data transfer, storage allocation and de-allocation, data removal, metadata registration and un-registration, and replica location.

Stork uses the ClassAd [72] job description language to represent the data placement jobs. The ClassAd language provides a very flexible and extensible data model that can be used to represent arbitrary services and constraints. This flexibility allows Stork to specify job level policies as well as global ones. Global policies apply to all jobs scheduled by the same Stork server. Users can override them by specifying job level policies in job description ClassAds.

Stork can interact with higher level planners and workflow managers. This allows the users to schedule both CPU resources and storage resources together. We have introduced a new workflow language capturing the data placement jobs in the workflow as well. The enhancements made to the workflow manager (i.e. DAGMan) allows it to differentiate between computational jobs and data placement jobs. The workflow manager can then submit computational jobs to a computational job scheduler, such as Condor or Condor-G, and the data placement jobs to Stork.

Stork also acts like an I/O control system (IOCS) between the user applications and the underlying protocols and data storage servers. It provides complete modularity and extendability. The users can add support for their favorite storage system, data transport protocol, or middleware very easily. This is a crucial feature in a system designed to work in a heterogeneous distributed environment. The users or applications should not expect all storage systems to support the same interfaces to talk to each other. And we cannot expect all applications to understand all the different storage systems, protocols, and middleware. There needs to be a negotiating system between the applications and the data storage systems which can interact with all such systems easily and even translate different data transfer protocols to each other. Stork has been developed to be capable of this. The modularity of Stork allows users to easily insert plug-ins to support any storage system, protocol, or middleware.

Stork can control the number of concurrent requests coming to any storage system it has access to, and makes sure that neither that storage system nor the network link to that storage system get overloaded. It can also perform space allocation and de-allocations to make sure that the required storage space is available on the corresponding storage system. The space allocations are supported by Stork as long as the underlying storage systems have support for it.

4.3.1 Data Placement Job Types

We categorize data placement jobs into seven types. These are:

transfer: This job type is for transferring a complete or partial file from one physical location to another one. This can include a get or put operation or a third party transfer.

allocate: This job type is used for allocating storage space at the destination site, allocating network bandwidth, or establishing a light-path on the route from source to destination. Basically, it deals with all necessary resource allocations pre-required for the placement of the data.

release: This job type is used for releasing the corresponding resource which is allocated before.

remove: This job is used for physically removing a file from a remote or local storage server, tape or disk.

locate: Given a logical file name, this job consults a meta data catalog service such as MCAT [9] or RLS [22] and returns the physical location of the file.

register: This type is used to register the file name to a meta data catalog service.

unregister: This job unregisters a file from a meta data catalog service.

The reason that we categorize the data placement jobs into different types is that all of these types can have different priorities and different optimization mechanisms.

4.3.2 Job Scheduling Techniques

We have applied some of the traditional job scheduling techniques common in computational job scheduling to the scheduling of data placement jobs:

First Come First Served (FCFS) Scheduling: Regardless of the type of the data placement job and other criteria, the job that has entered into the queue of the data placement scheduler first is executed first. This technique, being the simplest one, does not perform any optimizations at all.

Shortest Job First (SJF) Scheduling: The data placement job which is expected to take least amount of time to complete will be executed first. All data placement jobs except the transfer jobs have job completion time in the order of seconds, or minutes in the worst case. On the other hand, the execution time for the transfer jobs can vary from couple of seconds to couple of hours even days. Accepting this policy would mean non-transfer jobs would be executed always before transfer jobs. This may cause big delays in executing the actual transfer jobs, which defeats the whole purpose of scheduling data placement.

Multilevel Queue Priority Scheduling: In this case, each type of data placement job is sent to separate queues. A priority is assigned to each job queue, and the jobs in the highest priority queue are executed first. To prevent starvation of the low priority jobs, the traditional aging technique is applied. The hardest problem here is determining the priorities of each data placement job type.

Random Scheduling: A data placement job in the queue is randomly picked and executed without considering any criteria.

4.3.2.1 Auxiliary Scheduling of Data Transfer Jobs

The above techniques are applied to all data placement jobs regardless of the type. After this ordering, some job types require additional scheduling for further optimization. One such type is the data transfer jobs. The transfer jobs are the most resource consuming ones. They consume much more storage space, network bandwidth, and CPU cycles than any other data placement job. If not planned well, they can fill up all storage space, thrash and even crash servers, or congest all of the network links between the source and the destination.

Storage Space Management. One of the important resources that need to be taken into consideration when making scheduling decisions is the available storage space at the destination. The ideal case would be the destination storage system supports space allocations, as in the case of NeST [1], and before submitting a data transfer job, a space allocation job is submitted in the workflow. This way, it is assured that the destination storage system will have sufficient available space for this transfer.

Unfortunately, not all storage systems support space allocations. For such systems, the data placement scheduler needs to make the best effort in order to not over-commit the storage space. This is performed by keeping track of the size of the data transferred to, and removed from each storage system which does not support space allocation. When ordering the transfer requests to that particular storage system, the remaining amount of available space, to the best of the scheduler's knowledge, is taken into consideration. This method does not assure availability of storage space during the transfer of a file, since there can be external effects, such as users which access the same storage system via other interfaces without using the data placement scheduler. In this case, the data placement scheduler at least assures that it does not over-commit the available storage space, and it will manage the space efficiently if there are no external effects.

Figure 4.1 shows how the scheduler changes the order of the previously scheduled jobs to meet the space requirements at the destination storage sys-

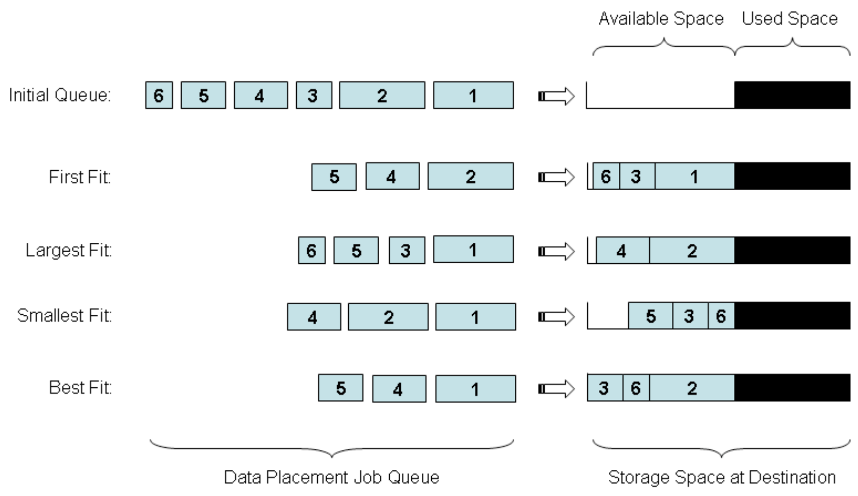

FIGURE 4.1: Storage Space Management: Different Techniques

tem. In this example, four different techniques are used to determine in which order to execute the data transfer request without over-committing the available storage space at the destination: *first fit*, *largest fit*, *smallest fit*, and *best fit*.

First Fit: In this technique, if the next transfer job in the queue is for data which will not fit in the available space, it is skipped for that scheduling cycle and the next available transfer job with data size less than or equal to the available space is executed instead. It is important to point that a complete reordering is not performed according to the space requirements. The initial scheduling order is preserved, but only requests which will not satisfy the storage space requirements are skipped, since they would fail anyway and also would prevent other jobs in the queue from being executed.

Largest Fit and *Smallest Fit:* These techniques reorder all of the transfer requests in the queue and then select and execute the transfer request for the file with the largest, or the smallest, file size. Both techniques have a higher complexity compared with the *first fit* technique, although they do not guarantee better utilization of the remote storage space.

Best Fit: This technique involves a greedy algorithm which searches all possible combinations of the data transfer requests in the queue and finds the combination which utilizes the remote storage space best. Of course, it comes with a cost, which is a very high complexity and long search time. Especially in the cases where there are thousands of requests in the queue, this technique would perform very poorly.

Using a simple experiment setting, we will display how the built-in storage management capability of the data placement scheduler can help improving

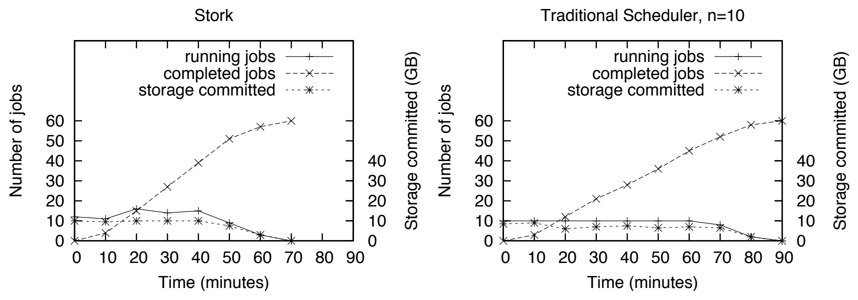

FIGURE 4.2: Storage Space Management: Stork vs. Traditional Scheduler

both overall performance and reliability of the system. In this experiment, we want to process 40 gigabytes of data, which consists of 60 files each between 500 megabytes and 1 gigabytes. First, the files need to be transferred from the remote storage site to the staging site near the compute pool where the processing will be done. Each file will be used as an input to a separate process, which means there will be 60 computational jobs followed by the 60 transfers. The staging site has only 10 gigabytes of storage capacity, which puts a limitation on the number of files that can be transferred and processed at any time.

A traditional scheduler would simply start all of the 60 transfers concurrently since it is not aware of the storage space limitations at the destination. After a while, each file would have around 150 megabytes transferred to the destination. But suddenly, the storage space at the destination would get filled, and all of the file transfers would fail. This would follow with the failure of all of the computational jobs dependent on these files.

On the other hand, Stork completes all transfers successfully by smartly managing the storage space at the staging site. At any time, no more than the available storage space is committed, and as soon as the processing of a file is completed, it is removed from the staging area to allow transfer of new files. The number of transfer jobs running concurrently at any time and the amount of storage space committed at the staging area during the experiment are shown in Figure 4.2 on the left side.

In a traditional batch scheduler system, the user could intervene, and manually set some virtual limits to the level of concurrency the scheduler can achieve during these transfers. For example, a safe concurrency limit would be the total amount of storage space available at the staging area divided by the size of the largest file that is in the request queue. This would assure the scheduler does not over-commit remote storage. Any concurrency level higher than this would have the risk of getting out of disk space anytime, and may cause failure of at least some of the jobs. The performance of the traditional scheduler with concurrency level set to 10 manually by the user in the same experiment is shown in Figure 4.2 on the right side.

Manually setting the concurrency level in a traditional batch scheduling system has three main disadvantages. First, it is not automatic, it requires user intervention and depends on the decision made by the user. Second, the set concurrency is constant and does not fully utilize the available storage unless the sizes of all the files in the request queue are equal. Finally, if the available storage increases or decreases during the transfers, the traditional scheduler cannot re-adjust the concurrency level in order to prevent overcommitment of the decreased storage space or fully utilize the increased storage space.

Storage Server Connection Management. Another important resource which needs to be managed carefully is the number of concurrent connections made to specific storage servers. Storage servers being thrashed or getting crashed due to too many concurrent file transfer connections has been a common problem in data intensive distributed computing.

In our framework, the data storage resources are considered first class citizens just like the computational resources. Similar to computational resources advertising themselves, their attributes and their access policies, the data storage resources advertise themselves, their attributes, and their access policies as well. The advertisement sent by the storage resource includes the number of maximum concurrent connections it wants to take anytime. It can also include a detailed breakdown of how many connections will be accepted from which client, such as "maximum n GridFTP connections, and "maximum m HTTP connections".

This throttling is in addition to the global throttling performed by the scheduler. The scheduler will not execute more than lets say x amount of data placement requests at any time, but it will also not send more than y requests to server a, and more than z requests to server b, $y+z$ being less than or equal to x.

Other Scheduling Optimizations. In some cases, two different jobs request the transfer of the same file to the same destination. Obviously, all of these requests except one are redundant and wasting computational and network resources. The data placement scheduler catches such requests in its queue, performs only one of them, but returns success (or failure depending on the return code) to all of such requests. We want to highlight that the redundant jobs are not canceled or simply removed from the queue. They still get honored and the return value of the actual transfer is returned to them, but no redundant transfers are performed.

In some other cases, different requests are made to transfer different parts of the same file to the same destination. This type of requests are merged into one request, and only one transfer command is issued. But again, all of the requests get honored and the appropriate return value is returned to all of them.

4.4 High Performance Wide Area Data Transfers

An important set of challenges appears when dealing with data transfers over wide area networks. Perhaps the most visible issue in this context is that of the transport protocol. Data transfers over wide area, and in particular over high-capacity network links make the performance limitations of the TCP protocol visible. In this section, we will discuss these limitations and various alternatives in the following sections. Much of the work related to wide area data transfer is focused on file or disk-to-disk transfer. We will present other types of scenarios, how they are different and what are the challenges and applicable solutions. This section concludes with a summary of remaining challenges and possible directions for future research.

4.4.1 TCP Characteristics and Limitations

Traditionally, applications using wide area networks have been designed to use the Transmission Control Protocol (TCP) for reliable data communication.

TCP, defined in RFC 793[1] but with numerous updates published since, provides byte stream semantics for end-to-end data transport. The two communicating entities create a TCP connection that can be used to send bytes in order and reliably (this defines byte stream semantics). Another important feature of TCP is that it implements congestion avoidance. Congestion appears when multiple concurrent streams go through the same link or network element (router) or when a stream traverses a low capacity link. Congestion avoidance is necessary since if congestion occurs, the overall utility of the network would be reduced because of the capacity wasted with retransmissions and transmission of data that eventually is dropped. TCP's congestion avoidance mechanism is based on two algorithms: slow start and congestion avoidance. These algorithms utilize the notion of a *congestion window* whose size is modified in response to packed acknowledgments and indication of packet loss. In TCP packet loss is taken as indication of congestion. These algorithms are described in more detail in RFC 2581[2].

We can summarize that TCP works by increasing the congestion window with one segment each RTT (round-trip time for the connection) when no congestion is detected and decreases the window to half when congestion is detected. This algorithm can be described as AIMD (additive increase multiplicative decrease). This behavior is illustrated in Fig. 4.3.

A long history of utilization has shown that TCP is extremely successful in supporting the Internet traffic while avoiding a crash of the system.

[1]`http://www.faqs.org/rfcs/rfc768.html`

[2]`http://www.faqs.org/rfcs/rfc2581.html`

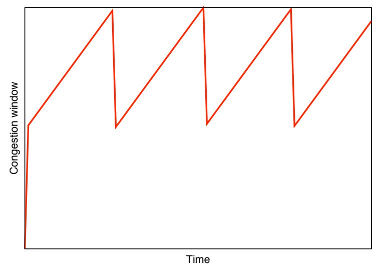

FIGURE 4.3: Theoretical TCP behavior on a Link with no Concurrent Traffic

However its characteristics create performance problems for situations where window sizes become very large, such as when using high-capacity wide area networks. First, the response to a congestion event (reducing the window to half) means that on a network link with no concurrent traffic TCP will not be able to sustain utilization of the full link capacity. Second, the slow increase of the congestion window during the congestion avoidance stage means that on long-distance, high-capacity links the time needed to increase the congestion window to reach the link capacity can be measured in hours. Also in practice, bandwidth utilization as a fraction of the total capacity will be lower on higher capacity networks since the probability of a single packet loss occurrence that is not caused by congestion is higher when the number of packets traversing the network increases and the algorithm treats all packet losses as indication of congestion.

4.4.2 Alternative Transport Protocols

UDP, or the User Datagram Protocol [69] is a thin transport protocol defined on top of IP that provides de-multiplexing of application messages between various applications running on the same host. In effect, UDP offers little more than IP which is addressing, routing and best-effort packet delivery. As such UDP is perfectly suited for implementing various other transport protocols. SCTP or the Stream Control Transmission Protocol is the latest approved standard transport protocol [79]. Similar to TCP it provides reliable transmission and uses the same congestion control mechanism, however the byte stream semantics has been replaced by message delivery. SCTP supports multiple delivery modes (ordered, partially ordered and unordered).

As the number of applications that require high-performance wide area transfers has increased so has the field of research in transport protocols. A survey on the subject [41] classifies newly emerging protocols in three categories: TCP variants, UDP based and protocols requiring router support. A short list of some of these protocols and their distinctive design ideas regarding congestion control follows.

Scalable TCP [51] and HighSpeed TCP [29] use a similar congestion control detection and avoidance mechanism as standard TCP but use a modified response function to packet loss and acknowledgement. In H-TCP [58] the increase function a depends on the time elapsed since the last congestion event, and the decrease function b depends both on throughput and ratio between minimum and maximum delay experienced by a source.

TCP Vegas [16] introduced a congestion avoidance algorithm based on measuring time delays of individual packets. The congestion window is updated based on measured differences between packet round-trip transit times and ideal network round-trip time of the uncongested network. The same approach is used by a newer protocol, FAST TCP [85]. Compound TCP [49] combines these two approaches (delay-based and loss-based) into a single algorithm. Its congestion window has two components: a loss-based window which is computed as in standard TCP and a delay-based computed window added on top of it.

BI-TCP [89] introduces a new technique for determining the available window size called binary search increase. In response to acknowledgment BI-TCP increases its window to the mid-point between the current window size and the maximum known size if this increase does not exceed a certain threshold or with the threshold value if it does (additive increase). When the window exceeds the maximum the available bandwidth is probed using a symmetric response function. The response function was updated in CUBIC [73] from the binary search logarithmic to a strictly cubic function.

TCP Westwood and the updated version TCP Westwood+ [64] use a technique called adaptive decrease which in response to a congestion event tries to guess the available network bandwidth and sets the windows size to the guessed value. The response to acknowledgments remains that of standard TCP.

A different direction is represented by protocols that do not take a window-based approach and thus indirectly control the data transmission rate but rather use a rate-based mechanism directly. These protocols use a fixed or variable data transmission rate and no explicit congestion window.

Reliable Blast UDP or RBUDP [40], is a rate-based protocol that performs reliable data transmission on top of UDP and the data transmission rate is specified by the user. Its successor, LambdaStream [88] automatically modifies the data transmission rate in case of congestion or when available bandwidth is detected. Congestion is detected when the data receiving rate is smaller than the data transmission rate and detection is based on measuring inter-packet delays at both the sender and the receiver.

UDT [37] also uses a rate-based congestion control algorithm. On acknowledgments, each RTT the rate is modified. The increase function depends on the estimated link bandwidth which is probed periodically. On a negative feedback the rate is decreased by a constant factor. UDT also supports a messaging mode which optional ordering and reliability parameters.

The Group Transport Protocol or GTP [87] is a rate-based protocol design that uses receiver-based flow management. Its design considers the problem of multiple senders – single receiver flows explicitly. GTP uses flow statistics to estimate the capacity of each flow and allocates bandwidth to each according to the max-min fairness criteria.

The eXplicit Control Protocol (XCP) [50] is a router-supported transport protocol. Based on the observation that packet loss is not a reliable indication of congestion, XCP uses feedback from the network (routers) in order for the sender to correctly determine the degree of congestion in the network.

We should also note that with the advent of wide area optical network links high-speed protocols for wide area transport are being researched and deployed outside the TCP/IP world. For example InfiniBand [3], a high-speed interconnect protocol originally designed for local area networks can be expanded to wide area networks using specialized hardware (Longbow by Obsidian Research[4]). Fibre Channel extension to wide area networks was also demonstrated by ADVA[5]. Fibre Channel is an interconnect technology originally designed for local area networks[6].

4.4.3 Current Status and Options

Link utilization for dedicated or long-distance connections and fairness between concurrent transfers are two of the many possible metrics that can be used to evaluate a transport protocol. There is no clear consensus (but there is work in progress in IETF [7] on the list of metrics that should be used for evaluating a transport protocol. However it is becoming clear that every transport protocol represents a trade-off point in the vast multi-dimensional space of evaluation metrics. There is no clear answer to the general question: *"What is the best protocol?"* as this answer depends on at least three important factors: what is the definition of *best* for the given scenario, what is the application type and what are the deployment, network and traffic constraints for the scenario.

Many evaluations and comparisons of transport protocols use the transfer of large data files over wide area networks as a motivating application scenario. File transfers need reliable transmission, are not highly sensitive to latency or throughput variations, are usually long-lived, use disk-to-disk transmission and the data is organized as a linear sequence. The important measure for a file transfer application is the total time needed to transfer a file from the source to the destination.

[3]`http://www.infinibandta.org/specs/`

[4]`http://www.obsidianresearch.com/`

[5]`http://www.advaoptical.com/`

[6]`http://www.fibrechannel.org/technology/overview.html`

[7]`http://tools.ietf.org/html/draft-irtf-tmrg-metrics`

One the most utilized data transfer system for grid and distributed applications today is GridFTP [4, 63]. One approach that can be taken to overcome some of the limitations of TCP is to use parallel connections. GridFTP [3] and PSockets [77] are early adopters of this mechanism. On public or shared links, the improvement in data transfer rate may come at the cost of other streams reducing their transfer rate which is not always acceptable. Also, on dedicated connections other protocols provide better performance. Recently, the Globus GridFTP implementation was modified to accept various data transport plug-ins. One of the currently available plug-ins uses the UDT transport protocol [17] mentioned previously.

One of the practical issues of high-speed protocols is that for some of them implementations are only available as kernel patches and switching TCP variants in the kernel on a shared resource is in most situations undesirable. An alternative to kernel patches is the implementation of transport protocols in the user space, using UDP. This is the approach utilized for implementing transport protocols such as RBUDP or UDT. The UDT library also provides a framework for custom user-space implementations of various congestion-control algorithms [36]. The advantage of user space protocol implementations is that they can be used in any application. The disadvantage is that they are more resource intensive, the CPU utilization on the machines is higher than that of kernel implementations for the same protocol. The packetization effort is also becoming more significant. The application or the transport protocol implementation must process each individual packet as it is sent or received from the network. If the packet size is small the number of packets becomes too large for the application to be able to obtain high transmission rates. Many applications now require the network to support "jumbo" (or 9000 bytes) packets in order to obtain high transmission rates and to reduce CPU utilization.

The increasing cost of protocol operations has been recognized by network controller providers who as one possible solution provide offload engines that move the processing from the CPU to the network controller [23].

4.4.4 Other Applications

Although not traditionally seen as a high-performance data transport applications, interactive applications using video transmission over long distance networks have complex requirements regarding network transmission. Videoconferencing applications [43] require minimal latency in the video transmission path. Using uncompressed, or low-rate compressed video are options that can be applied to video conferencing. The data streams carrying uncompressed video can use in excess of 1Gbps transmission rates and are highly sensitive to network latency and jitter. Fortunately, limited data loss is in some cases considered acceptable for video streams because for interactivity it is less costly to lose data than to retransmit it. Video transmission generally uses the standard Realtime Transmission Protocol[8] on top of plain UDP. Videoconferencing technology can be used for interactive remote visualization [48, 46] as these two problems have similar challenges and requirements.

Another application is that of visualization of remote data. Often, scientists need to locally visualize datasets generated on remote supercomputers by scientific simulations. These datasets can be very large and it is not desirable to transfer the entire data file locally for visualization. As a user is only interested in certain parts of the dataset at any given time, one alternative is to transfer only the section(s) of interest to the local machine. When the data is received, the visualization is updated and the user can move to another section of interest [71]. An important difference to file transfer is that the user may access different sections of the data object, depending on the type of data and analysis. The data transfer is not continuous. The user may analyze a certain section for a while then move on to the next one. Data transfer is thus bursty and a single data access may take as little as one or two seconds as the application needs to remain responsive. This is a significant difference to the file transfer scenario: for example it is a major issue for remote visualization (but not necessary for file transfer) if the transport protocol takes several seconds to reach the maximum transfer rate. One solution that is applicable to dedicated network connections are transmission protocols with fixed data transmission rate. The data transmission rate needs to be computed based on network, compute and storage resource availability [18].

The destination of the transfer is the main memory of a local machine and network utilization can be improved by using remote main memory to store (cached or prefetched) data of interest. We thus have disk-to-memory or memory-to-memory transfers. When disk is utilized as a data source (or destination), it is usually the disk that is the bottleneck in the system. For memory-to-memory transfers we see that the end hosts, and the efficiency of application and protocol implementations are becoming the bottleneck. In addition to the main memory optimization, multiple remote resources (data servers) may be utilized in parallel to improve data transfer performance.

4.4.5 Challenges and Future Directions

With increasing availability of high-speed wide area network connections we expect the amount of interest in high-speed transfer protocols to increase. Although a significant number of transport protocols have been proposed, the availability of implementations for the end user is still very limited. We believe that the problem of finding efficient transport mechanism for applications will continue to be a difficult one in the near future. The need for developing new protocols and analyzing the existing ones will continue to increase as new requirements are formulated and new applications are emerging.

[8]`http://www.faqs.org/rfcs/rfc3550.html`

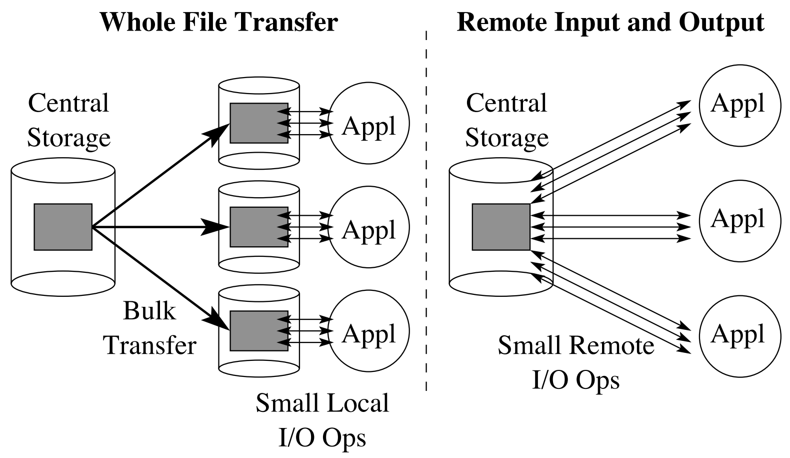

FIGURE 4.4: Whole File Transfer vs. Remote Input and Output

Efficient implementations will be needed for these protocols to be adopted. A framework for evaluating transport protocols as well as frameworks for implementing new transport protocols will help with some of the issues. We believe that in the future different applications will use different transport protocols. Also a single application will use different protocols when executed in different network environments. Defining a set of application benchmarks and a set of representative network environments may help protocol designers in the future.

4.5 Remote Input and Output

An alternate form of access to distant data is remote I/O, where the files of interest stay in one place, and the programs issue network operations to read or write small amounts of data that are of immediate interest. In this model, transfer protocols must be optimized for small operations, and the processing site may need no storage at all. Figure 4.4 shows both of these modes of file access. There are several potential benefits for employing remote I/O:

- **Remote I/O simplifies usability.** It is not always easy for the end user to identify exactly what files he or she needs. A complex program or data set can easily consist of hundreds of files and directories, of which not all are needed for any given task. In an interactive application, it may not even be possible to identify the needed files until run-time.

When using remote I/O, the system fetches the files needed at run-time, and the user is freed from this burden.

- **Remote I/O exploits selectivity.** When users explore very large data sets interactively, they may not need the entire contents of the repository. When using remote I/O, only the data that is actually needed for the current application is retrieved.

- **Remote I/O minimizes use of local storage.** The storage available where programs actually run – on user's desktops and in computing clusters – is not likely to have the capacity or performance of an institutional data storage system. By employing remote I/O, the local disk is removed from the system, improving performance, and increasing the possible execution sites for a given program.

- **Remote I/O minimizes initial response time.** When using file transfer for a large workload, no processing begins until an entire input file is transferred, and no output becomes available until some output file transfers complete. For long running jobs, this can be a problem, because the user cannot even verify that the program is running correctly until the workload completes. When using remote I/O, outputs become immediately available to the end user for verification or further processing.

On the other hand, a remote I/O system may result in a large number of network round trips in order to service each small I/O request. If the remote I/O is performed over a high latency wide area network, the result may be very low CPU utilization because the CPU is constantly waiting for a network operation to complete. In high latency networks, it is more practical to perform whole file transfers. In practice, remote I/O is best used when the desired application is interactive, selectively uses data, or the execution nodes are storage constrained.

4.5.1 Challenges of Remote I/O

An ideal remote I/O system allows an unmodified application to access remote files in the exact same way as local files, differing perhaps only in performance and naming. That is, a user ought to be able to take any existing application and attach it to a remote I/O system without requiring any changes to the code or the local computing system.

In order to design a suitable remote I/O system for an application, we must address each of the following design challenges:

How should the application be attached to the remote I/O system? Distributed file systems such as NFS [75] and AFS [45] rely on kernel-level drivers and configuration in order to redirect applications to remote services. As a result, these traditional systems cannot be used in large scale distributed systems where the end user is harnessing machines from multiple independent domains without any help from the local administrator. Instead, we must find a user-level mechanism for attaching the application to the remote I/O system.

One way to perform this attachment is by modifying libraries. If the application is written to use a middleware library like MPI-IO [82], then that library can be modified to perform remote I/O as in RIO [32]. Another technique is to replace the standard C library. This can be done by recompiling the application, as in Condor [59], or by preloading dynamic libraries as in Bypass [80] or XUnion [84]. Each of these techniques is effective on a subset of applications, but none applies to all possible binaries. For these reasons, we recommend the use of the `ptrace` interface, described below, for attaching applications to a remote I/O system.

What protocol should be used for remote I/O? There exist a variety of data transfer protocols in use on the Internet today. Unfortunately, few are directly suitable for carrying small I/O operations over the network. HTTP [27] is designed for moving entire files, and has no facilities for querying or managing directories. FTP [70] and variants such as GridFTP [5] provide some directory capabilities, but are also focused on large scale data movement, requiring a new TCP connection for every open file. For the typical application, which opens hundreds or thousands of files to access executables, libraries, and data files, one TCP connection for each file results in poor performance and possibly resource exhaustion. NFS [75] and its many variants [44, 6, 28, 10, 33] would appear to be more well suited, but the NFS protocol is difficult to disentangle from a kernel level implementation, due to the use of persistent numeric file handles to identify files. For these reasons, we recommend that remote I/O requires a custom network protocol that corresponds more closely to the Unix I/O interface.

What security mechanisms are needed for remote I/O? A traditional Unix filesystem has very limited security mechanisms, in which users are identified by integers, and a total of nine mode bits are used to specify access controls. A remote I/O system used in the context of a larger distributed system must have more sophisticated kinds of access control. Multiple users form virtual organizations [31] that wish to share data securely across the wide area. Some users require elaborate encryption and authorization mechanisms, while others are content to identify users by simple hostnames. A suitable security mechanism for remote I/O must be flexible enough to accommodate all of these scenarios without overly burdening the user that has simple needs. For these reasons, we recommend the use of multiple authentication techniques with full text subject names and access control lists.

4.5.2 Case Study: Parrot and Chirp

As a case study of a complete system for remote I/O, we present a discussion of the tools Parrot and Chirp. Figure 4.5 shows how Parrot and Chirp work

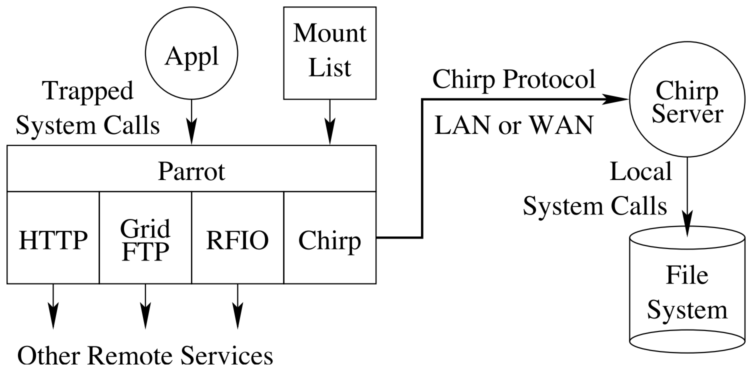

FIGURE 4.5: Using Parrot and Chirp for Remote I/O

together to provide seamless access to remote storage using the Unix I/O interface. These tools have been used to provide remote I/O services for a variety of applications in bioinformatics [15], biometrics [65], high energy physics [81], and molecular dynamics [86].

Parrot is a general purpose tool for attaching applications to remote I/O systems. Parrot does not require changes to applications or special privileges to install, so it is well suited for use in large distributed systems such as computing grids. Parrot works by trapping an application's system calls through the debugging interface. System calls unrelated to I/O (such as `sbrk`) are allowed to execute unmodified. System calls related to I/O (such as `open`, `read`, and `stat`) are handled by Parrot itself, and the results returned to the application. Parrot is not limited to single process programs: it can run complex multi-process scripts and interpreted languages such as Perl and Java. An interactive session using Parrot can be started by simply invoking:

```
% parrot tcsh
```

Parrot can be thought of as a user-level operating system. It contains multiple drivers that implement access to remote services. Each driver is represented in the root of the file namespace visible to applications run within Parrot. For example, the HTTP driver makes HTTP servers visible under files named like `/http/server.somewhere.edu/mydata`. Drivers are provided for a variety of protocols such as GridFTP [5], Nest [14], and RFIO [7]. However, for the reasons described above, none of these protocols is perfectly suited for performing remote I/O.

To address this problem, we created the Chirp protocol and server in order to provide the precise Unix I/O semantics required by Unix applications. A Chirp server is a user-level process that can be deployed by any user without special privileges. It exports a local filesystem to remote users via a protocol that closely resembles the Unix I/O interface, containing operations such as `open`, `read`, `stat`, and so forth. Like the other protocols, data on a Chirp server can be accessed under the pathname `/chirp/server.somewhere.edu/mydata`.

Each Chirp server periodically makes itself known to a global catalog server by sending a UDP packet listing its name, address, and other vital information. Information about all known Chirp servers can be obtained by querying the catalog server via a web browser, or by executing `ls /chirp` within Parrot.

Chirp provides a flexible security model that is suited to the cooperative nature of large scale distributed computing. The user may choose from a variety of mechanisms, depending on the degree of security required.

Upon connecting to a Chirp server, the client may authenticate with Kerberos [78], the Globus Grid Security Infrastructure [30], or by simple hostnames. Depending on what method is chosen, a given user might be known by any of the following subject names:

```
kerberos:smith@somewhere.edu
globus:/O=Somewhere/CN=Smith
hostname:client.somewhere.edu
```

Each directory in a Chirp server is protected by an access control list that lists acceptable subjects and access rights. Clearly, this access control list cannot be stored in the standard nine bits reserved for access control in Unix. Instead, each directory has a hidden file `.__acl` which can be manipulated with the commands `parrot_getacl` and `parrot_setacl`. For example, the following access control list grants read access to any client at the campus computing center, any client holding Globus credentials issued by the university, and all other rights to a single Kerberos user:

```
hostname:*.hpc.somewhere.edu    RL
globus:/O=Somewhere/*           RL
kerberos:smith@somewhere.edu    RWLDA
```

Using Parrot and Chirp together, a user can run any sort of program on one machine and use it to access data on any other machine on the Internet as if it were in the local filesystem, while protected by strong security mechanisms. The system can easily be deployed into an existing computational grid. The user must simply adjust their submission scripts to execute e.g. `parrot myjob.exe` instead of just `myjob.exe`.

To redirect access to remote files, the user may either adjust the program arguments manually, or provide Parrot with a *mountlist*, which is a text file that redirects file names to other locations. For example, the following mount list would cause an application to load its files from Chirp server `alpha`, libraries from `beta`, and use temporary space on `gamma`. In this way, a program or script with hard-coded path names can be employed without modification.

```
/bin   /chirp/alpha.somewhere.edu/bin
```

```
/lib   /chirp/beta.somewhere.edu/lib
/tmp   /chirp/gamma.somewhere.edu/tmp
```

4.5.3 Open Problems in Remote I/O

There are several interesting avenues of future research in remote I/O:

Combining Remote I/O and Directed Transfer. As noted above, remote I/O is most useful when an application has highly selective behavior on a large data set. However, there are many cases where the needs of an application may be practically known. For example, it may be clear that a job will need certain executables and libraries, but the exact input data is unknown. In these cases, it would be advantageous to perform an efficient bulk transfer of the known needs, and then rely upon remote I/O for the unknown needs at runtime. This is related to the idea of push-caching [38].

Exploiting Common Access Patterns. Many programs perform complex but predictable patterns of access across a filesystem. For example, a search for an executable or a library requires a predictable search through a known list of directories. An `ls -l` requires a directory listing followed by a metadata fetch on each file. While each of these cases can be individually optimized, a more general solution would allow the client to express a complex set of operations to be forwarded to the server and executed there, in the spirit of active storage [74].

4.6 Co-scheduling Compute and Network Resources

The increasing availability of high-bandwidth, low-latency optical networks promises to enable the use of distributed scientific applications as a day-to-day activity, rather than simply for demonstration purposes. However, in order to enable this transition, it must also become simple for users to reserve all the resources the applications require.

The reservation of computational resources can be achieved on many supercomputers using advance reservation, now available in most commercial and research schedulers. However, distributed applications often require guaranteed levels of bandwidth between compute resources. At the network level there are switches and routers that support the bandwidth allocation over network links, and/or the configuration of dedicated end-to-end lightpaths. These low-level capabilities are sufficient to support the development of prototype middleware solutions that satisfy the requirements of these applications.

However, the development of a booking system for network resources is not a complete solution, as the user is still dealing with the complexity of coordinating separate booking requests for multiple computational resources with their network booking(s). Even if there is a single system available that can reserve all the required compute resources, such as Moab, or GUR (which can reserve heterogeneous compute resources), this does not address the need to coordinate the scheduling of compute and network resources. A co-allocation system that can deal with multiple types of resources is required.

4.6.1 HARC: The Highly-Available Resource Co-allocator

HARC, the Highly-Available Resource Co-allocator [61, 39], is an open-source software infrastructure for reserving resources for use by distributed applications. From the client's perspective, the multiple resources are reserved in a single atomic step, so that all of the resources are obtained if possible; if not, no resources are reserved. We call this all-or-nothing reservation of multiple resources *co-allocation* or *co-scheduling*. Our definition is slightly more general than that commonly found in the literature, in that we do not mandate that all the reservations have the same start and end times. Rather, HARC allows a different time window to be specified for each resource being reserved; the case where all of these windows are the same is just a special case.

Here, we will briefly sketch the architecture of HARC, showing how the co-allocation process works, and explaining how the system achieves its high availability. We will then show how HARC can be used to co-schedule Compute and Network resources, and also show how HARC can be extended to cover more types of resource. Finally, some results of our experiences using HARC to reserve network resources, as part of both special demonstrations, and on a more production-like basis are presented.

4.6.2 Architecture

To provide the required atomic behavior, HARC treats the co-allocation process as a *transaction*, and a *transaction commit* protocol is used to reserve the multiple resources. When designing HARC, we wanted to provide a co-allocation *service*, but without introducing a single point-of-failure into the architecture. For these reasons, HARC was designed around the *Paxos Commit* Protocol [35] by Gray and Lamport. Paxos Commit replaces the single Transaction Manager process that you find in the classic *Two-Phase Commit* Protocol [57][9] with multiple, replicated processes known as *Acceptors*.[10] The overall system makes progress as long as a majority of Acceptors remain operational. By deploying sufficient Acceptors, it is possible to construct systems

[9]Or see: `http://en.wikipedia.org/wiki/Two-phase-commit_protocol`

[10]This terminology comes from the Paxos Consensus algorithm [56], upon which Paxos Commit is based.

which have a very long Mean-Time-To-Failure, even if the Mean-Time-To-Failure of each individual Acceptor is quite short.[11]

The HARC Architecture is shown in Figure 4.6. To reserve resources using HARC,

1. The client makes a request, e.g. from the command line, via the Client API,

2. The request goes to the HARC Acceptors, which manage the co-allocation process.

3. The Acceptors talk to individual Resource Managers, which make the individual reservations by talking to the underlying scheduler for their resource.

HARC was designed so that it could be extended by the Grid community to handle new types of resources, without needing modifications to the existing code. Specifically, the required steps for extending HARC to handle a new resource type are: designing the description of the new resource type (XML), adding client code that provides a method for encoding requests as XML, and adding a Resource Manager module which decodes requests, and talks to the underlying scheduler for the resource, to make reservations. The scheduler needs to support reservations.

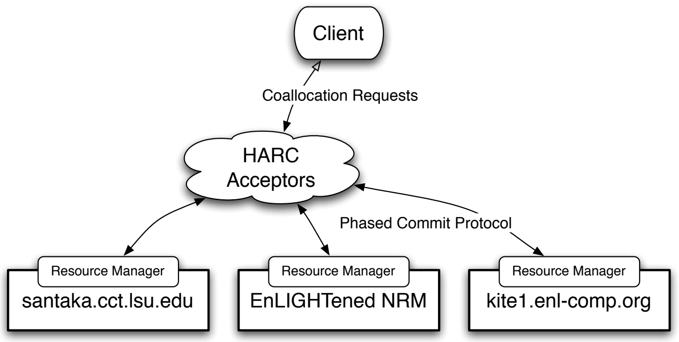

FIGURE 4.6: HARC Architecture. NRM = Network Resource Manager

[11]Let's conservatively assume that the Mean-Time-To-Failure (MTTF) of a single Acceptor is 1 week, and that following a failure, it takes an average time of 4 hours to bring the Acceptor back online. Then, a deployment with seven Acceptors would have a MTTF of over 10 years [61, Sec. 2.3].

4.6.3 Reserving Network Resources using HARC

One of the main goals of the EnLIGHTened Computing Project [24](2005-2007) was to enable coordinated use of compute and network resources. This required network resources such as those in the EnLIGHTened Testbed (shown in Figure 4.7 (a)) to be managed using middleware; previously this task required sending emails to network administrators. To enable this, a HARC Network Resource Manager (NRM) component was developed [60].

In the case of compute resources, there are a number of schedulers which HARC can communicate with, e.g. Moab, LoadLeveler, etc.; but there is no equivalent product for optical networks. Currently, there are a number of prototype network schedulers, e.g. [68, 25], however these are often site-specific so we wrote our own simple scheduler for the EnLIGHTened network.

The software on the optical elements (Layer 1 optical network switches from Calient) in the testbed supports GMPLS, and connections across the testbed can be initiated by sending a command to a switch at either end of the connection. A dedicated lightpath can be set-up between any two entities at the edge of the cloud shown in Figure 4.7 (b). These are either routers (UR2, UR3) or compute nodes (RA1, VC1, CH1).

The NRM accepts requests for network connections on a first-come, first-served basis. Requests specify the start and end points of the connection (using the three letter acronyms shown in the figure). The following command-line example requests 8 cores on machines at LSU and MCNC as well as a dedicated lightpath connecting them:

```
$ harc-reserve -c santaka.cct.lsu.edu/8 \
               -c kite1.enlightenedcomputing.org/8 \
               -n EnLIGHTened/BT2-RA1 -s 12:00 -d 1:00
```

4.6.4 Experiments Using HARC

HARC has been successfully used to co-schedule compute and network resources in a number of large-scale demonstrations, most notably in the high-profile EnLIGHTened/G-lambda experiments at GLIF 2006 and SC'06, where compute resources across the US and Japan were co-scheduled together with end-to-end optical network connections [26].[12] The HARC NRM has also been used on a more regular basis, to schedule a subset of the optical network connections used to broadcast lectures from Louisiana State University as part of an education experiment carried out in 2007. [47][13]

[12]Also see: `http://www.gridtoday.com/grid/884756.html`

[13]This was the first distance learning course ever offered in uncompressed high definition video. Participating locations included other sites in Louisiana, Masaryk University in the Czech Republic, University of Arkansas, and North Carolina State University

HARC Compute Resource Managers are now deployed on several machines in the TeraGrid[14], LONI[15] and UK NGS[16] infrastructures. This growing, supported infrastructure was used in demos by the GENIUS Project [34] at SC'07.

Although the early results are encouraging, if the advance scheduling of lightpaths is to become a production activity, then the network scheduling service(s) need to be properly integrated with the other control/management plane software to ensure that these activities do not interfere with the pre-scheduled lightpaths (and vice-versa).

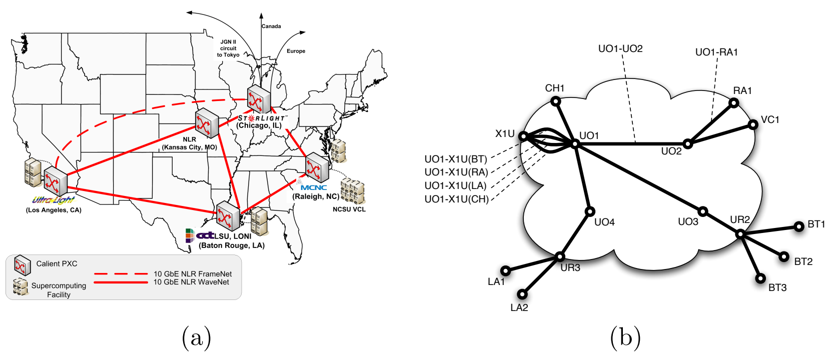

FIGURE 4.7: The EnLIGHTened Testbed: a) Geographically b) as seen by HARC NRM

4.7 Conclusions

The increase in the demand for large-scale data processing has necessitated collaboration and sharing of data collections among the world's leading education, research, and industrial institutions and use of distributed resources owned by collaborating parties. Efficient and reliable access to large-scale data sources and archiving destinations in a widely distributed computing environment brings new challenges such as scheduling data movement and

[14]`http://www.teragrid.org/`
[15]`http://www.loni.org/`
[16]`http://www.ngs.ac.uk/`

placement, efficient use of wide area network links, remote access to partial data, and co-scheduling of data storage, network and computational resources. In this chapter, we tried to address the challenges and possible solutions in each of these areas with specific examples and case studies. We believe that these examples are only some of the initial steps taken towards the solution of these problems, and there is still a lot to be done to break the barriers in front of the domain scientists to easily and efficiently access and use distributed datasets.

4.8 Acknowledgments

The work presented in this chapter is in part sponsored by the National Science Foundation under award numbers CNS-0846052 (Career-Kosar), CNS-0643229 (Career-Thain), CNS-0619843 (PetaShare), EPS-0701491 (CyberTools), CNS-0509465 (Enlightened), by the SURA SCOOP Program (ONR N00014-04-1-0721, NOAA NA04NOS4730254) and by the Board of Regents, State of Louisiana, under Contract Numbers DOE/LEQSF (2004-07), NSF/LEQSF (2007-10)-CyberRII-01, and LEQSF(2007-12)-ENH-PKSFI-PRS-03.

References

[1] NeST: Network Storage Technology. http://www.cs.wisc.edu/condor/nest/.

[2] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S.Tuecke. Secure, efficient data transport and replica management for high-performance data-intensive computing. In *Proceedings of IEEE Mass Storage Conference*, April 2001.

[3] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in high-performance computational grid environments. *Parallel Comput.*, 28(5):749–771, 2002.

[4] W. Allcock, J. Bester, J. Bresnahan, S. Meder, P. Plaszczak, and S. Tuecke. GridFTP: Protocol extensions to FTP for the Grid. *GWD-R (Recommendation)*, April 2003.

[5] W. Allcock, A. Chervenak, I. Foster, C. Kesselman, and S. Tuecke. Protocols and services for distributed data-intensive science. In *Proceedings of Advanced Computing and Analysis Techniques in Physics Research*, pages 161–163, 2000.

[6] P. Andrews, P. Kovatch, and C. Jordan. Massive high-performance global file systems for grid computing. In *Supercomputing*, Seattle, WA, November 2005.

[7] O. Barring, J. Baud, and J. Durand. CASTOR project status. In *Proceedings of Computing in High Energy Physics*, Padua, Italy, 2000.

[8] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The sdsc storage resource broker. In *Proceedings of CASCON*, 1998.

[9] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In *Proceedings of CASCON*, Toronto, Canada, 1998.

[10] A. Batsakis and R. Burns. Cluster delegation: High-performance fault-tolerant data sharing in NFS. In *High Performance Distributed Computing*, 2004.

[11] M. Beck, T. Moore, J. Blank, and M. Swany. Logistical networking. In *Active Middleware Services, S. Hariri and C. Lee and C. Raghavendra, editors. Kluwer Academic Publishers*, 2000.

[12] J. Bent. Data-driven batch scheduling. Ph.D. Dissertation, University of Wisconsin-Madison, 2005.

[13] J. Bent, D. Thain, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. Explicit control in a batch-aware distributed file system. In *Proceedings of the First USENIX/ACM Conference on Networked Systems Design and Implementation*, 2004.

[14] J. Bent, V. Venkataramani, N. LeRoy, A. Roy, J. Stanley, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Flexibility, manageability, and performance in a grid storage appliance. In *IEEE Symposium on High Performance Distributed Computing*, Edinburgh, Scotland, July 2002.

[15] C. Blanchet, R. Mollon, D. Thain, and G. Deleage. Grid deployment of legacy bioinformatics applications with transparent data access. In *IEEE Conference on Grid Computing*, September 2006.

[16] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. Tcp vegas: new techniques for congestion detection and avoidance. In *SIGCOMM '94: Proceedings of the conference on Communications architectures, protocols and applications*, pages 24–35, New York, NY, USA, 1994. ACM.

[17] J. Bresnahan, M. Link, G. Khanna, Z. Imani, R. Kettimuthu, and I. Foster. Globus gridftp: What's new in 2007 (invited paper). In *Proceedings*

of the First International Conference on Networks for Grid Applications, 2007.

[18] Jr. C. Toole and A. Hutanu. Network flow based resource brokering and optimization techniques for distributed data streaming over optical networks. In *MG '08: Proceedings of the 15th ACM Mardi Gras conference*, pages 1–8, New York, NY, USA, 2008. ACM.

[19] CCSP. Strategic plan for the us climate change science program. CCSP Report, 2003.

[20] E. Ceyhan and T. Kosar. Large scale data management in sensor networking applications. In *Proceedings of Secure Cyberspace Workshop*, Shreveport, Louisiana, November 2007.

[21] A. Chervenak, R. Schuler C. Kesselman, S. Koranda, and B. Moe. Wide area data replication for scientific collaborations. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, November 2005.

[22] L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf. Performance and scalability of a Replica Location Service. In *Proceedings of the International Symposium on High Performance Distributed Computing Conference (HPDC-13)*, Honolulu, Hawaii, June 2004.

[23] A. Currid. Tcp offload to the rescue. *ACM Queue*, 2(3), May 2004.

[24] EnLIGHTened Computing: Highly-dynamic Applications Driving Adaptive Grid Resources. `http://www.enlightenedcomputing.org`.

[25] A. Takefusaa et al. G-lambda: Coordination of a Grid Scheduler and Lambda Path Service over GMPLS. *Future Generation Computing Systems*, 22:868–875, 2006.

[26] S. R. Thorpe et al. G-lambda and EnLIGHTened: Wrapped In Middleware, Co-allocating Compute and Network Resources Across Japan and the US. In *Proceedings of Gridnets 2007: First International Conference on Networks for Grid Applications (Lyon, France, October 2007).*

[27] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol (HTTP). Internet Engineering Task Force Request for Comments (RFC) 2616, June 1999.

[28] R. Figueiredo, N. Kapadia, and J. Fortes. The PUNCH virtual file system: Seamless access to decentralized storage services in a computational grid. In *IEEE High Performance Distributed Computing*, San Francisco, CA, August 2001.

[29] S. Floyd. Highspeed tcp for large congestion windows. Internet draft, work in progress ftp://ftp.rfc-editor.org/in-notes/rfc3649.txt, December 2003.

[30] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. In *ACM Conference on Computer and Communications Security*, pages 83–92, San Francisco, CA, November 1998.

[31] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150, 2001.

[32] I. Foster, D. Kohr, R. Krishnaiyer, and J. Mogill. Remote I/O: Fast access to distant storage. In *Workshop on I/O in Parallel and Distributed Systems (IOPADS)*, 1997.

[33] A. Ganguly, A. Agrawal, P. O. Boykin, and R. J. Figueiredo. WOW: Self organizing wide area overlay networks of workstations. *Journal of Grid Computing*, 5(2), June 2007.

[34] GENIUS: Grid Enabled Neurosurgical Imaging Using Simulation. `http://wiki.realitygrid.org/wiki/GENIUS`.

[35] J. Gray and L. Lamport. Consensus on transaction commit. Technical Report MSR-TR-2003-96, Microsoft Research, January 2004. http://research.microsoft.com/research/pubs/view.aspx?tr_id=701.

[36] Y. Gu and R. Grossman. Supporting configurable congestion control in data transport services. In *Supercomputing 2005*, November 2005.

[37] Y. Gu and R. L. Grossman. Udt: Udp-based data transfer for high-speed wide area networks. *Comput. Networks*, 51(7):1777–1799, 2007.

[38] J. Gwertzman and M. Seltzer. The case for geographical push-caching. In *Hot Topics in Operating Systems*, 1995.

[39] HARC: The Highly-Available Resource Co-allocator. `http://www.cct.lsu.edu/~maclaren/HARC`.

[40] E. He, J. Leigh, O. Yu, and T. A. DeFanti. Reliable blast udp: Predictable high performance bulk data transfer. In *CLUSTER '02: Proceedings of the IEEE International Conference on Cluster Computing*, pages 317–324, Washington, DC, USA, 2002. IEEE Computer Society.

[41] E. He, P. V., and M. Welzl. A survey of transport protocols other than standard tcp. http://www.ggf.org/documents/GFD.55.pdf, November 2005.

[42] T. Hey and A. Terefethen. The data deluge: An e-science perspective. In *Grid Computing - Making the Global Infrastructure a Reality, chapter 36, pp. 809-824. Wiley and Sons*, 2003.

[43] P. Holub, L. Matyska, M. Liška, L. Hejtmánek, J. Denemark, T. Rebok, A. Hutanu, R. Paruchuri, J. Radil, and E. Hladká. High-definition mul-

timedia for multiparty low-latency interactive communication. *Future Generation Computer Systems*, 22(8):856–861, 2006.

[44] P. Honeyman, W. A. Adamson, and S McKee. GridNFS: Global storage for global collaboration. In *Local to Global Data Interoperability*, 2005.

[45] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[46] A. Hutanu, G. Allen, S. D. Beck, P. Holub, H. Kaiser, A. Kulshrestha, M. Liška, J. MacLaren, L. Matyska, R. Paruchuri, S. Prohaska, E. Seidel, B. Ullmer, and S. Venkataraman. Distributed and collaborative visualization of large data sets using high-speed networks. *Future Generation Computer Systems*, 22(8):1004–1010, 2006.

[47] A. Hutanu, M. Liška, P. Holub, R. Paruchuri, D. Eiland, S. R. Thorpe, and Y. Xin. Uncompressed hd video for collaborative teaching - an experiment. In *Proceedings of CollaborateCom*, 2007.

[48] B. Jeong, L. Renambot, R. Jagodic, R. Singh, J. Aguilera, A. Johnson, and J. Leigh. High-performance dynamic graphics streaming for scalable adaptive graphics environment. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 108, New York, NY, USA, 2006. ACM.

[49] Q. Zhang K. Tan, J. Song and M. Sridharan. Compound tcp: A scalable and tcp-friendly congestion control for high-speed networks. In *Fourth International Workshop on Protocols for Fast Long-Distance Networks*, 2006.

[50] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 89–102, New York, NY, USA, 2002. ACM.

[51] T. Kelly. Scalable tcp: improving performance in highspeed wide area networks. *SIGCOMM Comput. Commun. Rev.*, 33(2):83–91, 2003.

[52] S. Koranda and M. Moe. Lightweight data replicator. http://www.ligo.caltech.edu/docs/G/G030623-00/G030623-00.pdf.

[53] T. Kosar. A new paradigm in data intensive computing: Stork and the data-aware schedulers. In *Proceedings of Challenges of Large Applications in Distributed Environments (CLADE 2006) Workshop*, Paris, France, June 2006.

[54] T. Kosar and M. Livny. Stork: Making data placement a first class citizen in the Grid. In *Proceedings of the 24th Int. Conference on Distributed Computing Systems (ICDCS 2004)*, Tokyo, Japan, March 2004.

[55] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, 2000.

[56] L. Lamport. Paxos made simple. In ACM SIGACT news distributed computing column 5. *SIGACT News*, 32(4):18–25, 2001.

[57] B. W. Lampson and H. E. Sturgis. Crash recovery in a distributed data storage system. Technical Report (Unpublished), Xerox Palo Alto Research Center, 1976, 1979. `http://research.microsoft.com/Lampson/21-CrashRecovery/Acrobat.pdf`.

[58] D. Leith and R. Shorten. H-tcp: Tcp for high-speed and long-distance networks. In *Second International Workshop on Protocols for Fast Long-Distance Networks*, 2004.

[59] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Eighth International Conference of Distributed Computing Systems*, June 1988.

[60] J. MacLaren. Co-allocation of Compute and Network resources using HARC. In *Proceedings of "Lighting the Blue Touchpaper for UK e-Science: Closing Conference of the ESLEA Project*. PoS(ESLEA)016, 2007. `http://pos.sissa.it/archive/conferences/041/016/ESLEA_016.pdf`.

[61] J. MacLaren. HARC: The Highly-Available Resource Co-allocator. In Robert Meersman and Zahir Tari, editors, *OTM Conferences 2007, Part II: Proceedings of GADA'07*, volume 4804 of *Lecture Notes in Computer Science*, pages 1385–1402. Springer Verlag, 2007.

[62] R. K. Madduri, C. S. Hood, and B. Allcock. Reliable file transfer in grid environments. In *Proceedings of 27th Annual IEEE Conference on Local Computer Networks*, November 2002.

[63] I. Mandrichenko, W. Allcock, and T. Perelmutov. Gridftp v2 protocol description. *GWD-R (Recommendation)*, May 2005.

[64] S. Mascolo, L. A. Grieco, R. Ferorelli, P. Camarda, and G. Piscitelli. Performance evaluation of westwood+ tcp congestion control. *Perform. Eval.*, 55(1-2):93–111, 2004.

[65] C. Moretti, J. Bulosan, D. Thain, and P. Flynn. All-Pairs: An abstraction for data intensive cloud computing. In *International Parallel and Distributed Processing Sympsoium*, 2008.

[66] Y. Morita, H. Sato, Y. Watase, O. Tatebe, S. Segiguchi, S. Matsuoka, N. Soda, and A. DellAcqua. Building a high performance parallel file system using grid datafarm and root i/o. In *Proceedings of the 2003 Computing in High Energy and Nuclear Physics (CHEP03)*, 2003.

[67] OPeNDAP. Open-source Project for a Network Data Access Protocol. http://www.opendap.org/.

[68] C. Palansuriya, M. Büchli, K. Kavoussanakis, A. Patil, C. Tziouvaras, A. Trew, A. Simpson, and R. Baxter. End-to-End Bandwidth Allocation and Reservation for Grid applications. In *Proceedings of BROADNETS 2006*. http://www.x-cd.com/BroadNets06CD/pdfs/87.pdf, October 2006.

[69] J. Postel. User datagram protocol, 1980.

[70] J. Postel and J. Reynolds. File Transfer Protocol (FTP). Internet Engineering Task Force Request For Comments 959, October 1985.

[71] S. Prohaska, A. Hutanu, R. Kahler, and H. Hege. Interactive exploration of large remote micro-ct scans. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 345–352, Washington, DC, USA, 2004. IEEE Computer Society.

[72] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, Chicago, Illinois, July 1998.

[73] I. Rhee and L. Xu. Cubic: A new tcp-friendly high-speed tcp variant. In *Third International Workshop on Protocols for Fast Long-Distance Networks*, February 2005.

[74] E. Riedel, G. A. Gibson, and C. Faloutsos. Active storage for large scale data mining and multimedia. In *Very Large Databases (VLDB)*, 1998.

[75] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the Sun network filesystem. In *USENIX Summer Technical Conference*, pages 119–130, 1985.

[76] A. Shoshani, A. Sim, and J. Gu. Storage resource managers: Middleware components for grid storage. In *Proceedings of Nineteenth IEEE Symposium on Mass Storage Systems*, 2002.

[77] H. Sivakumar, S. Bailey, and R. L. Grossman. Psockets: the case for application-level network striping for data intensive applications using high speed wide area networks. In *Supercomputing '00: Proceed-*

ings of the 2000 ACM/IEEE conference on Supercomputing (CDROM), page 37, Washington, DC, USA, 2000. IEEE Computer Society.

[78] J.G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the USENIX Winter Technical Conference*, pages 191–200, 1988.

[79] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream control transmission protocol, 2000.

[80] D. Thain and M. Livny. Multiple bypass: Interposition agents for distributed computing. *Journal of Cluster Computing*, 4:39–47, 2001.

[81] D. Thain, C. Moretti, and I. Sfiligoi. Transparently distributing cdf software with parrot. In *Computing in High Energy Physics*, 2006.

[82] R. Thakur, W. Gropp, and E. Lusk. Data sieving and collective i/o in ROMIO. In *Symposium on the Frontiers of Massively Parallel Computation*, 1999.

[83] S. Tummala and T. Kosar. Data management challenges in coastal applications. *Journal of Coastal Research*, 2007.

[84] E. Walker. A distributed file system for a wide-area high performance computing infrastructure. In *USENIX Workshop on Real Large Distributed Systems*, Seattle, WA, November 2006.

[85] D. X. Wei, C. Jin, S. H. Low, and S. Hegde. Fast tcp: motivation, architecture, algorithms, performance. *IEEE/ACM Trans. Netw.*, 14(6):1246–1259, 2006.

[86] J. Wozniak, P. Brenner, D. Thain, A. Striegel, and J. Izaguirre. Generosity and gluttony in GEMS: Grid enabled molecular simulations. In *IEEE High Performance Distributed Computing*, July 2005.

[87] R. X. Wu and A. A. Chien. Gtp: Group transport protocol for lambda-grids. In *Cluster Computing and the Grid*, pages 228–238, April 2004.

[88] C. Xiong, J. Leigh, E. He, V. Vishwanath, T. Murata, L. Renambot, and T. A. DeFanti. Lambdastream – a data transport protocol for streaming network-intensive applications over photonic networks. In *Third International Workshop on Protocols for Fast Long-Distance Networks*, February 2005.

[89] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control for fast long distance networks. In *INFOCOM 2004*, volume 4, pages 2514–2524, March 2004.