Proc. of 2nd International Workshop on Grids and Advanced Networks, Chicago (USA), May 2004

The File Mover: An Efficient Data Transfer System for Grid Applications

Cosimo Anglano, Massimo Canonico
Dipartimento di Informatica
Università del Piemonte Orientale, Alessandria (Italy)
email:{cosimo.anglano,massimo.canonico}@unipmn.it

Abstract

In this paper we present the File Mover, *a data transfer system designed to optimize the transfer of potentially very large files. The File Mover relies on an* overlay network *architecture, where a set of machines cooperate in the transfer by forwarding among them portions of the files being transferred. Data transfer times are minimized by choosing, for each transfer, the set of relays that maximize the expected throughput. Preliminary experiments show that the File Mover is able to profitably exploit existing network paths not chosen by IP routing algorithms, thereby enhancing file transfer performance.*

1 Introduction

The scientific exploration in many disciplines, like High Energy Physics, Climate Modeling, and Life Sciences, requires the processing of massive data collections, whose size is in the order of Terabytes (and sometimes even Petabytes) [22], For these applications, the creation of *Data Grids*[8], that pool geographically distributed storage and computing resources, seems a promising solution. In order to enable the achievement of satisfactory performance, Data Grids require the availability of a system able to transfer potentially huge files in the shortest possible amount of time [9]. As a matter of fact, the completion time of typical Data Grid applications is given by the sum of their execution time and of the time taken to transfer the data they need [26], and is often dominated by the data transfer time.

To the best of our knowledge, all the file transfer systems we are aware of [2, 7, 15, 16, 30, 32] rely on transport-level protocols (namely TCP and UDP) to move data from their source to their destination. Although these systems exploit highly sophisticated techniques to increase the transfer rate, they suffer from a common drawback, namely their reliance on the IP routing protocols, that has as consequence the fact the throughput they achieve is limited by the bandwidth available on the network path chosen by the IP routing layer. Unfortunately, the IP routing protocols notoriously produce suboptimal routes [12, 13, 29], since their choice of network paths is not guided by performance considerations as they are primarily concerned with the exchange of connectivity information. As a consequence, it is not infrequent the case that shorter transfer times might be obtained by choosing a network path different from that chosen by the IP routing algorithms. For instance, Savage *et al.* observe [13] that for 30 to 80 percent of the network paths chosen by the IP routing algorithms between pairs of Internet hosts, taken from a relatively large set of machines, it was possible to find alternative paths with better performance characteristics. Another consequence of the reliance on the IP routing algorithms is the possibly high amount of time required to recover from link failures. As a matter of fact, the fault recovery mechanisms used by typical Internet routing protocols sometimes take many minutes to converge to a consistent form [19], and there are times when path outages lead to significant disruptions in communication lasting even tens of minutes or more [10, 23, 24]. As a consequence, transfers along faulty paths may be delayed for a very long time, so the effective bandwidth obtained in these situations drops below any reasonable value.

In this paper we describe the *File Mover*, a software system that addresses the above problems by exploiting an *overlay network* architecture. An overlay network is a virtual network, layered on top of the existing Internet, whose member nodes are placed at the edges of the underlying physical network and communicate by means of a transport-level protocol (e.g, TCP or UDP). Each pair of member nodes of an overlay network (henceforth referred to as *Relays*) communicate by means of a *virtual link*, that corresponds to the network path chosen by the IP layer to transfer data from one member to the other one. The relays of an overlay network agree to forward each other's traffic along one or more virtual links, until the destination host is reached. Figure 1 schematically depicts a possible configuration of the File Mover in which an overlay network comprising four relays (A,B,C, and D) is assumed. Note that virtual links are unidirectional, as IP routing is in gen-

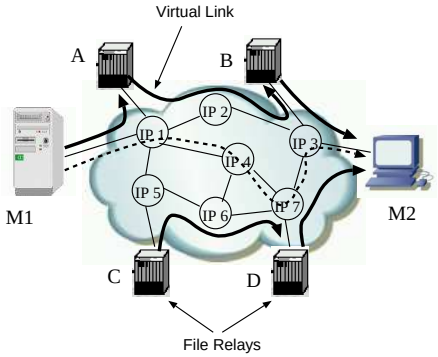

Figure 1. An example File Mover overlay network. Circles, thin lines, thick lines, and dotted line represent respectively IP routers, physical links, virtual links, and network paths.

eral asymmetric (that is the path taken by packets that host A sends to host B may be different than the one followed by packets that B sends to A).

The advantages of the File Mover over conventional IP-based solutions for file transfer are many-fold. First of all, network paths resulting in better throughput than the one chosen by the IP routing algorithms may be exploited by interposing, between the file source and the file destination, a suitable set of relays. For instance, considering the situation depicted in Figure 1, assume that the network path chosen by IP to connect machine M1 to machine M2 (denoted by a dotted arrow) traverses routers IP 1, IP 4, IP 7, and IP 3. If the path traversing routers IP1, IP2, and IP3 has a better throughput, the File Mover is able to use it by transferring the file on the virtual links connecting M1 to A, A to B, and B to M2. Conversely, a conventional IP-based solution (e.g., FTP) would be forced to use the less efficient network path.

Second, if more paths are available between a source and a destination machine, they can be used simultaneously to transfer different parts of the same file in parallel, so that download time is reduced, as for instance proposed by Rao [27] to reduce end-to-end latency. For instance, in the situation depicted in Figure 1, it is possible to transfer a portion of the file on the path IP 1, IP 2, IP 3 (by using Relays A and B) and another portion on the path IP 1, IP 5, IP 6, IP 7, IP 3 (by using C and D). Multipath transfers may also be used to increase the throughput between a source and a destination machine by transferring different files at the same time when a batch of files are requested to the same server. In other words, the File Mover is able to aggregate several network paths to increase the available bandwidth, and consequently the throughput, between the data source and the data destination. In contrast, the parallel download of a file or batch of files with a conventional file transfer system requires the usage of multiple simultaneous transfer sessions between the same pair of hosts. These sessions, however, must use the same network path, so the aggregate throughput that can be achieved is limited by the path throughput.

Third, in case of a link failure, a shorter recovery time may be obtained by using a new sequence of virtual links that does not include faulty physical links or routers. In contrast, as already mentioned, systems based on IP routing must wait until the IP routing protocols converge to a consistent configuration, and this may take several minutes.

The remainder of this paper is organized as follows. In Section 2 we describe the architecture of the File Mover. In Section 3 we present some experimental results, obtained with a proof-of-concept implementation, showing the performance benefits that the File Mover can obtains over conventional file transfer systems. In Section 4 we place our work in the context of the related research. Finally, Section 5 concludes the paper and outlines future research work.

2 The File Mover

As anticipated in the Introduction, the File Mover is a software system, whose architecture (schematically depicted in Figure 2) is based on the overlay networks paradigm. The File Mover provides a data transfer service between a *File Server* and a *Client* by using a suitable set of *File Relays* as intermediate nodes. A client that needs a file contacts one of the *File Proxies*, that requests the file to the server and notifies the client when the transfer has been completed. The overlay network is fully connected, that is each File Relay may communicate directly (via a TCP connection) with any other File Relay. In order to minimize the

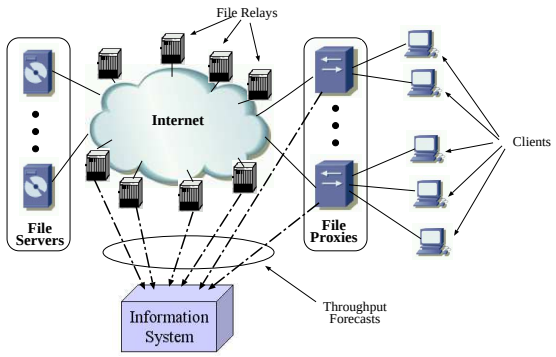

Figure 2. The File Mover Architecture

time taken by file transfers, each Server, upon receiving a file transfer request, computes – by exploiting the throughput forecasts for all the virtual links in the overlay – the *virtual path* (i.e., a sequence of one or more virtual links) that is expected to provide the highest possible throughput for the whole duration of the transfer. Throughput forecasts for

the virtual links are computed by measuring with Iperf [1] the available bandwidth on each virtual link, and providing these measurements to the statistical forecasting algorithms of the Network Weather Service [33] (NWS). More specifically, each Server and each Relay periodically (1) measures the available bandwidth of all its outgoing virtual links, (2) computes the corresponding forecasts, and (3) stores them into the *Information System*, from which they are extracted by File Servers when a transfer must be performed. In order to increase the accuracy of forecasts, it is necessary to collect a suitable number of throughput measurements for all the virtual links of the overlay network, including those connecting Relays and Servers to Clients. For these links, however, this may be problematic, since a client "appears" in the system when it requests a file, and "disappears" when the transfer is over. File Proxies have been introduced to solve this problem: a File Proxy is a permanent component of the File Relay, continuously participating to the monitoring and performance forecasting activity. An additional benefit of using a proxy is that it can serve multiple clients (possibly within the same LAN), so simultaneous file requests can be properly scheduled in such a way that they do not interfere with each other unless it is really necessary. Moreover, it can act as a cache for the clients it serves, saving valuable time if it already holds a copy of the requested file.

In order to fully exploit all the virtual links in the chosen virtual path, file transfers are pipelined, that is the file is split into a sequence of *blocks* of suitable size, that are transmitted independently from each other. That is, at any given time, different blocks of the file are simultaneously transmitted across different virtual links, so that the transfer time is reduced with respect to a "store-and-forward" technique where the whole file is stored at each File Relays before being forwarded. Moreover, pipelining lowers the memory requirements of File Relays, as only a portion of the file needs to be stored locally at any given time.

Let us now discuss in detail the behavior of the File Mover by describing each of the steps that its components perform to carry out a file transfer.

2.1 Virtual Path Computation

As already discussed, when a file must be transferred the source Server computes the virtual path that is expected to yield the highest throughput for the entire transfer duration. Recalling that the throughput of a pipeline corresponds to that of its slowest component, the highest throughput is obtained by choosing the virtual path whose slowest virtual link has the highest possible throughput. To illustrate, consider the scenario shown in Figure 3, where four File Relays are available to transfer a file between the *Server* and the *Proxy*. For the sake of readability, we have assumed

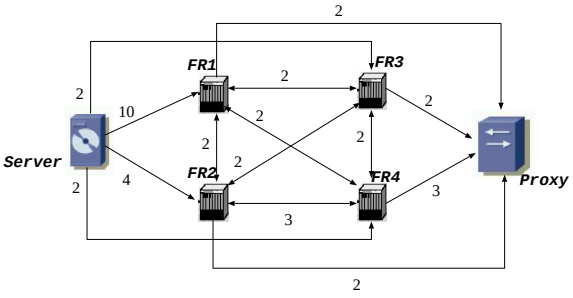

Figure 3. Computation of the best-throughput path. Arcs represent virtual links, and labels correspond to their throughputs expressed in Mbits/sec.

that the virtual links connecting the File Relays have the same throughput, so that the corresponding arcs are drawn as bidirectional. In the case shown in Figure 3, the highest throughput (3 Mbits/sec.) is obtained by choosing the virtual path *Server* $\rightarrow$ *FR2* $\rightarrow$ *FR4* $\rightarrow$ *Proxy*. As a matter of fact, all the paths beginning with the virtual links *Server* $\rightarrow$ *FR3* and *Server* $\rightarrow$ *FR4* have a smaller throughput, as the throughput of these links is 2 Mbits/sec. Moreover, the slowest link of all the virtual paths beginning with the virtual link *Server* $\rightarrow$ *FR1* is 2 Mbits/sec. (all the virtual links departing from *FR1* have a 2 Mbits/sec. throughput). The best-throughput path is computed by using a variant of the Dijkstra [5] single-source shortest path algorithm, that has been suitably modified to fit our needs.

In order to avoid that ongoing transfers are slowed down by new ones, we exclude from the computation of the virtual path for a new transfers all virtual links already in use, or that share at least a physical link with a currently used one. Note that two virtual links that share the same destination File Relay, share at least a physical link (the one connecting the File Relay to its default router). Therefore, a File Relay already involved in another transfer is not included in the graph, so it cannot be involved in more than one transfer at a time. We are aware that this is a rough simplification, as two transfers using the same virtual link would slow down each other only if the virtual link represents a bottleneck for both of them. However, the identification of shared bottleneck among network paths is an open research problem for which there are only partial solutions [18, 28], so we decided to adopt the more conservative approach outlined before. Avoiding the interference among different transfers may also have the additional benefit of leaving unaffected the accuracy of the NWS predictions, that may be useful in many situations. For instance, when multiple copies of the same file exist in different locations, better application performance may be obtained by transferring it from the replica yielding the smallest transfer time. Similarly, if the computational resources must be

reserved in advance, and users are charged also for reserved but unused time, making reservations at the right time (and not too early) may result in better resource utilization and lower resource costs.

2.2 Virtual Path Creation

Before a virtual path can be used to carry out a file transfer, it must be established, that is each virtual link must be "connected" to the next one. This is accomplished by having each File Relay in the path "connect" its input and output virtual links, so that the file blocks sent by its preceding Relay can be forwarded to the next one along the path. As shown in Figure 4, the File Relay connects these

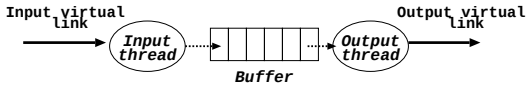

Figure 4. Connecting the input with the output virtual links of a File Relay.

virtual links by creating an *Input* thread, that removes the file blocks from the input virtual link and stores them into a buffer, and *Output* thread that extracts the file blocks from the buffer and sends them on the output virtual link.

Virtual path setup is cooperatively carried out by all the involved File Relays by means of the *virtual path allocation protocol*, that is discussed in the remainder of this section by referring to Figure 5 (showing the creation of a virtual path including three File Relays). Virtual path creation is started by the source File Server, that creates a *Transfer Manager* thread in charge of handling the transfer. This thread prepares a request message containing the addresses of all the File Relays in the virtual path and sends it to the first one (step (1)). When this File Relay receives the allocation message, it creates a local *Transfer Manager* thread (step (2)), in charge of performing control some functions before and during the transfer, that strips its address from the set of the involved File Relays, and forwards the allocation message to the second File Relay (step (3)) in the virtual path. Eventually, the last File Relay in the virtual path will forward the allocation message to the File Proxy (step (7)). Upon receipt of the allocation message, the File Proxy creates its local *Transfer Manager* thread (step (8)), that in turn creates (step (9)) a *Transfer thread* (in charge of receiving and reassembling the file blocks), and sends to the last File Relay an *OK* message (step (10)) carrying the information of the TCP port where the Transfer thread is waiting for the file blocks. When the File Relay receives this message, it creates the Input and Output threads (step (11)), and sends an *OK* message to the preceding File Relay (step (12)). The Output thread then creates a TCP connection with the Transfer thread of the Proxy (step (13)), while the Input thread waits for an incoming TCP creation request from the Output thread of its preceding File Relay. The previous File Relay performs in turn the same actions, but this time its Output thread creates a TCP connection with the Input thread of the next File Relay (step (16)). Eventually, the *OK* message gets to the Transfer Manager of the File Server (step (18)) that, after updating the information in the Information Service in order to mark all the File Relays and virtual links used for the transfer, starts the file transfer. In particular, it creates (step (20)) a *Disk* thread, that will read the file blocks from disk, and a *Network* thread, that will send these blocks to the first Input thread after a TCP connection has been created (step (21)).

Conversely, if one of the File Relays does not respond withing a given amount of time, or if it cannot participate to the file transfer for some reasons, it does not create the corresponding Transfer Manager but replies with a *FAIL* message to the sending Transfer Manager, that will forward this message to the previous one until the File Server is informed of the failure, so that it can use a different virtual path.

2.3 File Transfer

After the virtual path connecting the File Server to the destination File Proxy has been set up as discussed in the previous section, the transfer is started. In order to achieve the maximum possible performance, two issues have to be tackled, namely the minimization of the forwarding delay that each File Relay introduces when moving the file block from the input to the output virtual link, and the maximization of the throughput on the various virtual links used to transmit data among File Relays.

The minimization of the forwarding delay is achieved by, as already discussed, pipelining the data transfers. Pipelined data transfers are implemented as follows. In order to maximize the rate at which data blocks are injected in the network, the Disk thread of the File Server continuously read blocks from disk and stores them into a buffer shared with the Network thread. The Network thread, instead, repeatedly reads a block from the buffer and sends it on the virtual link connecting it to the first File Relay. On each File Relay in the virtual path, the Input thread receives file blocks on the input virtual links and stores them into a buffer (see Figure 4), while the Output thread reads blocks from the buffer and sends them to the next File Relay. If the output link is slower than the input link, the blocks waiting to be transmitted remain in the buffer until they are transmitted on the next virtual link.

The maximization of the throughput on each virtual link is achieved by properly setting the congestion window size for the endpoints of the corresponding TCP connection [21, 31]. To compute this value, it is necessary to measure both the Round Trip Time (RTT) and the available

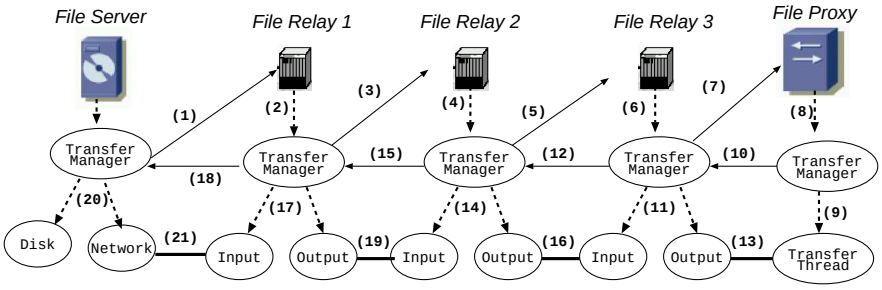

Figure 5. The Virtual Path Allocation Protocol. Continuous arrows, dotted arrows, and thick lines correspond respectively to messages, thread creation operations, and TCP connections. The numbers between parenthesis denote the step of the protocol in which the various actions are carried out.

bandwidth of the network path over which the TCP connection is mapped, that in turn corresponds to measure the bandwidth of the slowest (physical) link of the path. While RTT estimation does not require special access rights, bandwidth estimation tools typically require superuser permissions [14, 17, 20] (although tools that run in user mode are starting to appear [3]). Superuser permissions are also required to modify the system-wide value of the maximum congestion window size. Unfortunately, assuming the possibility of running the File Mover processes with superuser permissions is unrealistic, as the machines acting as File Relays will typically be placed in different administrative domains. Therefore, we have adopted the pragmatic approach of setting the congestion window size to the maximum possible value, specified in the local operating systems settings, for each machine.

2.4 Fault Management

The File Mover, as many other Grid infrastructures, relies on the availability of resources that may become unavailable (because the respective owners reclaim them back) during a transfer. These unavailabilities may be particularly harmful given the size of the transferred files: the retransmission of a potentially very large file (either from the beginning or from the last received byte) from the source File Server may require too much time to be feasible. Moreover, given the potentially very long time required to carry out the transfer, it is not possible to assume that the user requesting the transfer is always ready to restart it. Consequently, the File Mover has been equipped with proper mechanisms that eliminate the need of any user intervention and enable it to avoid, to the maximum possible extent, the need to retransmit the file from the source File Server. More specifically, during a transfer, each File Relay in the virtual path temporarily buffers some file blocks. Periodically, the Transfer Manager thread of the Proxy acknowledges the blocks that have been successfully stored in the local file system to the Transfer Thread of the last File Relay, that in turn removes them from its local buffer and propagates backward in the path this information, so that eventually all File Relays remove the same blocks from their local buffers. In order to deal with hardware/software faults, each Transfer Threads continuously probes its successor in the virtual path. If a File Relay becomes unresponsive, the Transfer Thread that discovers it (its predecessor) suspends the transmission of file blocks, sends a *PAUSE* control message to its predecessor in the virtual path (that in turn propagates it backwards), computes a new partial virtual path connecting it with the destination Proxy, and starts its setup process. Upon receiving a *PAUSE* message, a Relay (or a Server) suspends the transmission of file blocks, and waits for a forthcoming *GO* message. During the allocation of the new (partial) virtual path, the destination Proxy sends to the Transfer Thread that discovered the failure the sequence number of the last file block received via the previous (faulty) virtual path, so that this thread may resume the transmission from the first unacknowledged piece of data. This thread sends also to its predecessor a *GO* message, that in turn resumes the transmission from the first unsent block, and propagates it backwards until it reaches the first machine that has not propagated backwards the *PAUSE* message (possibly the source Server).

3 Experimental Evaluation

In order to show the feasibility of the File Mover, we have developed a proof-of-concept implementation, written in C++ and running on Linux and FreeBSD, that has been deployed on a subset of the wide-area nodes of the Emulab Testbed [11]. We performed a set of experiments, aimed at comparing the performance of the File Mover with those achieved by other data transfer systems, namely GridFTP [30], bbftp [2], bbcp [16], SABUL [15], FOBS [7], and Tsunami [32]. However, we were unable to run most of these systems on Emulab, either because they do not run on FreeBSD or because of specific constraints posed by Emulab's security policies. Therefore, we had to

limit our comparison to FOBS, the only data transfer systems that we could run on Emulab. A more thorough comparison with all the tools mentioned above is subject of future work.

Our testbed comprised 19 nodes from the RON [6] testbed accessible via Emulab, and two external nodes. In particular, we placed 12 Relays and 7 Servers on the RON nodes, and 2 Proxies on the external nodes. External Proxies have been used since we were unable to setup FOBS in such a way that files could be transferred among Emulab nodes, while file transfers toward external machines were possible without any extra configuration step. Table 1, where fully-qualified domain names have been used only for machines not belonging to the 'ron.lcs.mit.edu' domain, lists all the machines belonging to our testbed. Note that, despite their common domain name, these machines are geographically distributed and belong to different organizations participating to the RON testbed. The comparison between the File Mover and FOBS has been performed by running a set of experiments, each one consisting in the transfer of a file using the File Mover followed by the transfer of the same file using FOBS, in order to keep the probability of sudden changes in the network performance as low as possible. Two file sizes were used for each experiment (50 MBytes and 100 MBytes), and each experiment was repeated five times. Table 2 shows the results we obtained for the 50 MBytes file (the results for the 100 MBytes file are practically identical, so are not shown here). In this table we show, for both the File Mover and FOBS, the estimated transfer throughput (Est. Thr.), the average transfer time, and the average throughput actually achieved (Eff. Thr.) (both computed as the arithmetic average of the values measured in the 5 runs). From Table 2 we observe that in some cases (corresponding to rows 1–4) the File Mover achieves a throughput higher, and hence a shorter transfer time, than FOBS. In three cases (rows 1–3), the File Mover chooses a network path different from the one chosen by the IP routing algorithms (used by FOBS) that is characterized by a higher expected throughput, while in the fourth one both systems used the "direct", IP-chosen network path (note that this transfer is a repetition of the one performed in Experiment 2, performed at a different time with different network conditions, as evidenced by the choice of a different network path). [1]. Conversely, in Experiments 5–9, FOBS achieved a throughput higher than the File Mover, in spite of the predictions for the throughput achievable on the direct and the indirect path. This can be explained by observing that FOBS always achieved a throughput higher than the predicted value (as can be observed from the values of Est. Thr. and Eff. Thr. in the corresponding columns) and, while in Experiments 1–4 the throughput on the alternative network paths was anyways higher than on the direct path, for Experiments 5–9 the opposite was true. This is not surprising, as we measure virtual link performance by using Iperf in "TCP mode", while FOBS employs UDP that, not using any flow control technique, can achieve a throughput higher than a TCP connection. Moreover, Iperf measured the throughput that was possible to achieve with the maximum window size value imposed by the operating system configuration of Emulab machines that we could not alter, and this value was too low to achieve a bandwidth close to the available one. Finally, in two cases (Experiments 10 and 11) FOBS failed to carry out the transfer because of system errors due to the lack of buffer space (as reported by the tool).

	Hosts
Servers	mit, utah,cornell,cmu, aros,nortel, mit-main
Relays	ccicom, mazu1, nyu, pdi, ana1-gblx, roncluster5, roncluster8, intel, coloco jfk1-gblx, digitalwest, ch1-gblx
Proxies	seal.cs.ucsb.edu, syrah.usc.edu

Table 1. Machines used for the experiments.

Another consideration that can be made by looking at Table 2 is that the throughput achieved by the File Mover is reasonably close to the values measured with Iperf and predicted by the NWS. This in turn means that our implementation, although not optimized, introduces a reasonable forwarding overhead, that in most cases is mainly due to the fact that data blocks cross the same network link twice when passing through each Relay. We are therefore confident that there is room for increasing performance by developing a more optimized implementation. Moreover, we note also that the difference between predicted and achieved throughput for the File Mover is roughly constant, and this means that we can rely on the NWS predictions to estimate the file transfer times.

4 Related Work

As already mentioned, in the recent past many projects have investigated new mechanisms for efficient data transfers across Grid infrastructures, and a set of new data transfer systems have been designed and implemented as result. Some of these systems, like GridFTP [30], BBFTP [2], and bbcp [16], rely on TCP connections to move data, and try to optimize transfer performance by performing some tuning actions [21, 31] (e.g., proper setting of the TCP window size and usage of parallel TCP streams). Other systems, like FOBS [7], Tsunami [32], and SABUL [15], use UDP to move data, and employ rate-based control algorithms to avoid to saturate the network path used to move

[1] Note that in this experiment the throughput achieved by both FOBS and the File Mover was higher than expected, since throughput forecasts were generated before the transfers.

Experiment	Source	Destination	File Mover				FOBS		
			Path	Est. Thr.	Transfer time	Eff. Thr.	Est. Thr.	Transfer time	Eff. Thr.
1	**utah**	**seal**	**utah → digitalwest→seal**	**11.5**	**47.24**	**8.48**	**5.46**	**61.67**	**6.49**
2	**utah**	**syrah**	**utah → digitalwest→syrah**	**32.9**	**17.12**	**24.32**	**5.73**	**42.9**	**9.36**
3	**mit-main**	**seal**	**mit-main→nyu→intel→seal**	**3.65**	**125.9**	**3.2**	**2.58**	**330.9**	**1.21**
4	**utah**	**syrah**	**utah →syrah**	**5.73**	**40.5**	**9.88**	**5.73**	**41.15**	**9.49**
5	mit-main	syrah	mit-main→nyu→intel→seal	3.77	123.3	3.26	3.08	41.42	9.69
6	mit	seal	mit→nyu→intel→seal	3.76	125.2	3.21	2.73	68.9	5.85
7	nortel	seal	nortel→nyu→intel→seal	3.64	118.6	3.37	2.6	83.28	4.8
8	nortel	syrah	nortel→nyu→intel→syrah	3.77	116.9	3.42	3.1	69.3	5.77
9	aros	syrah	aros →syrah	7.1	69.3	5.77	7.1	42.1	9.5
10	cornell	seal	cornell→nyu→intel→seal	3.75	118.2	3.38	2.91	unavail.	unavail.
11	cmu	seal	cmu→nyu→intel→seal	3.63	117.5	3.4	2.95	unavail.	unavail.

Table 2. Results for the 50 MBytes transfer. Time is expressed in seconds, throughput (Thr.) in Megabit/sec.

the data. Because of their reliance on IP routing, however, these systems cannot achieve a throughput higher than the available path bandwidth. Conversely, the File Mover looks for better alternative network paths to transfer data, and can achieve better performance is such paths do indeed exist. Furthermore, the File Mover can take advantage of many of the transmission optimizations mentioned above to speed-up transmissions over individual virtual links.

To our knowledge, the File Mover is the first file transfer system based on the overlay network paradigm. However, overlay networks are not a new concept, and the File Mover architecture has been inspired by existing overlay networks systems, like RON [6] and Detour [12], although it differs from these systems in two important aspects. First, RON and Detour route on the same network paths *all the traffic flows*, while the File Mover adopts a selfish approach in which network spots used by other traffic flows are avoided (if necessary). Second, RON and Detour route data on the paths that provide the best *current* performance, thus in case of changes in network performance, already-established TCP connections are forced to traverse a possibly suboptimal network path. Conversely, the File Mover chooses the network paths that are expected to yield the best performance in the medium-term future, so that performance degradations due to changes in network conditions are less likely to occur during a file transfer.

5 Conclusions and Future Work

In this paper we have described the File Mover, a file transfer system based on the overlay network paradigm that is able to exploit network paths that cannot be used by classical "end-to-end" file transfer approaches. We have developed a proof-of-concept implementation of the File Mover that, in spite of several limitations and of the lack of aggressive optimizations, has shown promising results. However, a more thorough experimental evaluation, comparing the File Mover to other existing file transfer tools, is required in order to gain a better confidence about the suitability of the approach.

Future work includes both the adoption of techniques aimed at increasing system scalability, such as distributed databases for the Information Service and measurement techniques requiring less than N^2 probes for N hosts (e.g., [4]), and the investigation of issues aimed at increasing file transfer performance. First of all, rate-based UDP transfer protocols seem to provide better performance than TCP, so an obvious extension for the File Mover is to adopt a protocol of this type for transferring data across virtual links. Second, better performance should be obtained if individual Relays create local caches of the files they transfer, so that subsequent transfers are started from the replica yielding best performance. The availability of multiple copies (complete or, even, partial) of the same file would enable us to exploit striped transfers, in which different portions of the same file are simultaneous transferred from different Proxies, as for instance done in [25]. Third, we plan to study dynamic reconfiguration strategies that enable a File Relay to change a portion of the virtual path while engaged into a transfer when changes in network performance occur. Finally, we plan to study better resource management policies in which Servers, when compute the best virtual path, take into considerations also Relays already engaged into ongoing transfers, since sometimes it might be more profitable to wait until a busy Relay becomes available, rather than using only the available ones.

Acknowledgments

We thank Rich Wolski and Graziano Obertelli of the University of California, Santa Barbara, for their help in setting up NWS for our experiments, and for letting us use some of their machines in our testbed. We also thank the University of Utah's Emulab team for letting us transfer large volumes of data on their infrastructure.

References

[1] Iperf: The TCP/UDP Bandwidth Measurement Tool. http://dast.nlanr.net/Projects/Iperf.

[2] The bbFTP – Large Files Transfer Protocols Web Site. http://doc.in2p3.fr/bbftp.

[3] A. Akella and S. Seshan and A. Shaikh. An Empirical Evaluation of Wide-Area Internet Bottleneck. In *Proc. of ACM SIGMETRICS '03*, San Diego, California, USA, June 2003.

[4] Y. Chen, D. Bindel, and R.H. Katz. Tomography-based Overlay Network Monitoring. In *Proc. of ACM IMC 2003*, 2003.

[5] T.H. Cormen, C.L. Leiserson, and R.L. Rivest. *Introduction to Algorithms, 2nd Edition*. MIT Press, 2001.

[6] D.G. Andersen and H. Balakrishnan and M.F. Kaashoek and R.Morris. Resilient Overlay Networks. In *Proc. of 18th ACM Symp. on Operating Systems Principles*, Banff, Canada, Oct. 2001.

[7] P.M. Dickens. FOBS: A Lightweight Communication Protocol for Grid Computing. In *Proc. of Europar 2003*, Klagenfurt, Austria, August 2003.

[8] A. Chervenak et al. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientifi c Datasets. *Journal of Network and Computer Applications*, 23:187–200, 2001.

[9] B. Allcock et al. Data Management and Transfer in High-Performance Computational Grid Environments. *Parallel Computing*, 28(5):749–771, May 2002.

[10] B. Chandra et al. End-to-End WAN Service Availability. In *Proc. of 3rd Usenix Symp. on Internet Technologies and Systems (USITS)*, pages 97–108, San Francisco, CA, February 2001.

[11] B. White et al. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proc. of 5th Symp. on Operating Systems Design and Implementation*, Boston, MA, USA, Dec. 2002.

[12] S. Savage et al. Detour: A Case for Informed Internet Routing and Transport. *IEEE Micro*, 19(1):50–59, Jan. 1999.

[13] S. Savage et al. The End-to-End Effects of Internet Path Selection. In *Proc. of ACM SIGCOMM*, pages 289–299, Boston, MA, 1999.

[14] G. Jin and B. Tierney. Netest: A Tool to Measure the Maximum Burst Size, Available Bandwidth and Achievable Throughput. In *Proc. of Int. Conf. on Information Technology Research and Education*, Newark, NJ, USA, Aug. 2003.

[15] Y. Gu, X. Hong, M. Mazzucco, and R.L. Grossman. SABUL: A High Performance Data Transfer Protocol. http://www.dataspaceweb.net/papers.htm, 2003. Submitted for publication.

[16] A. Hanushevsky, A. Trunov, and L. Cottrell. Peer-to-Peer Computing for Secure High Performance Data Copying. In *Proc. of the 2001 Int. Conf. on Computing in High Energy and Nuclear Physics (CHEP 2001)*, Beijng, China, September 2001.

[17] J. Strauss and D. Katabi and F. Kasashoek. A Measurement Study of Available Bandwidth Estimation Tools. In *Proc. of the ACM Conf. on Internet Measurement*, Miami Beach, Florida, USA, October 2003.

[18] D. Katabi, I. Bazzi, and X. Yang. A Passive Approach for Detecting Shared Bottlenecks. In *Proc. of IEEE Int. Conf. on Computer Communications and Networks*, Arizona,USA, 2001.

[19] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet Routing Convergence. In *Proc. of ACM SIGCOMM*, pages 175–187, Stockolm, Sweden, September 2000.

[20] M. Jain and C. Dovrolis. End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. In *Proc. of ACM SIGCOMM '02*, Pittsburgh, Pennsylvania, USA, August 2002.

[21] M. Mathis and R. Reddy. Enabling High Performance Data Transfers. http://www.psc.edu/networking/perf_tune.html.

[22] H.B. Newman, M.H. Ellisman, and J.H. Orcutt. Data-Intensive E-Science Frontier Research. *Communications of the ACM*, 46(11), Nov. 2003.

[23] V. Paxson. End-to-End Routing Behavior in the Internet. In *Proc. of ACM SIGCOMM*, pages 25–38, Stanford, CA, August 1996.

[24] V. Paxson. End-to-End Internet Packet Dynamics. In *Proc. ACM SIGCOMM*, pages 139–152, Cannes, France, September 1997.

[25] J. Plank, S. Atcheley, Y. Ding, and M. Beck. Algorithms for High Performance, Wide-Area, Distributed File Downloads. *Parallel Processing Letters*, 13(2), June 2003.

[26] K. Ranganathan and I. Foster. Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids. *Journal of Grid Computing*, 1(1):53–62, 2003.

[27] N.S.V. Rao. Multiple Paths for End-to-End Delay Minimization in Distributed Computing Over Internet. In *Proc. of Supercomputing 2001*, 2001.

[28] D. Rubenstein, J. Kurose, and D. Towsley. Detecting Shared Congestion of Flows Via End-to-end Measurement. *IEEE/ACM Transactions on Networking*, 10(3), June 2002.

[29] H. Tangmunarunkit, R. Govindan, S. Shenker, and D. Estrin. The Impact of Routing Policy on Internet Paths. In *Proc. IEEE Infocom '01*, Anchorage, Alaska, April 2001.

[30] The Globus Team. GridFTP: Universal Data Transfer for the Grid. http://www.globus.org. White Paper.

[31] B. Tierney. TCP tuning Guide for Distributed Applications on Wide Area Networks. *Usenix ;login Journal*, page 33, Feb. 2001.

[32] S. Wallace. Tsunami File Transfer Protocol. In *Proc. of First Int. Workshop on Protocols for Fast Long-Distance Networks*, CERN, Geneva, Switzerland, February 2003.

[33] R. Wolski. Dynamically Forecasting Network Performance Using the Network Weather Service. *Cluster Computing*, 1(1):119–132, Jan. 1998.