A Result-Data Offloading Service for HPC Centers*

Henry Monti†, Ali R. Butt†, and Sudharshan S. Vazhkudai‡

†Virginia Polytechnic Institute and State University
Blacksburg, VA 24061
{hmonti,butta}@cs.vt.edu

‡Oak Ridge National Laboratory
Oak Ridge, TN 37831
vazhkudaiss@ornl.gov

ABSTRACT

Modern High-Performance Computing applications are consuming and producing an exponentially increasing amount of data. This increase has lead to a significant number of resources being dedicated to data staging in and out of Supercomputing Centers. The typical approach to staging is a direct transfer of application data between the center and the application submission site. Such a direct data transfer approach becomes problematic, especially for staging-out, as (i) the data transfer time increases with the size of data, and may exceed the time allowed by the center's purge policies; and (ii) the submission site may not be online to receive the data, thus further increasing the chances for output data to be purged. In this paper, we argue for a systematic data staging-out approach that utilizes intermediary data-holding nodes to quickly offload data from the center to the intermediaries, thus avoiding the peril of a purge and addressing the two issues mentioned above. The intermediary nodes provide temporary data storage for the staged-out data and maximize the offload bandwidth by providing multiple data-flow paths from the center to the submission site. Our initial investigation shows such a technique to be effective in addressing the above two issues and providing better QOS guarantees for data retrieval.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design—*Distributed Systems*; D.4.2 [**Operating Systems**]: Storage Management—*Storage hierarchies*

General Terms

Algorithms, Design, Reliability

*This research is sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. DE-AC05-00OR22725.

Supercomputing'07, Nov. 10–16, 2007, Reno, NV.

Keywords

Data offloading, peer-to-peer, high-performance computing

1. PROBLEM STATEMENT

Supercomputing centers routinely generate huge amounts of data, resulting from high-throughput compute jobs. These are often result-datasets or checkpoint snapshots from long-running simulations that are required to be offloaded to end-user locations, where they can be visualized for further scientific insights. For example, the Department of Energy's (DOE) National Leadership Class Facility (NLCF) at Oak Ridge National Laboratory (ORNL), which is No. 2 in the Top500 supercomputers as of this writing, is already generating terabytes of data from user jobs from a wide-spectrum of science applications in Fusion, Astrophysics, Climate and Combustion. As we approach the petascale realm—with machines such as the NLCF—we might soon be faced with offloading a petabyte of data from a single application run! Another example is the TeraGrid where result-data—from computations in any of the nine sites nation-wide—is required to be delivered to the end-user. Accessing these national user facilities, is a geographically distributed user-base with varied end-user connectivity, resource availability and application requirements.

It is quoted that modern HPC center user services are often reminiscent of early computers. Traditionally, centers have operated under the premise that users come to them with all of their storage and computing needs. The legacy of this approach still weighs heavily when it comes to provisioning a center as significant portions of the operational budget is spent on large data stores and archives. End-user data services are marginalized!

With the explosive growth in application data production, it is impractical to store all user data indefinitely. HPC centers are aware of this and enforce purge policies to manage the scratch space by deleting data based on a time window (ranging from a few hours to a few days) [1]. However, there is no corresponding end-user service for a timely offload of data, to avoid purging. This is largely left to the user and is a manual process, wherein users stage out result-data using point-to-point transfer tools such as GridFTP [3], ftp, HSI [5], and scp.

The lack of a sophisticated solution for result-data offloading affects not only end-user service, but also center operations. The output data of a supercomputing job is the result of a

multi-hour—even several days'—run. Result output data is usually stored in the supercomputer center's scratch space, which is a valuable commodity. The scratch space is served through a high-speed parallel file system and is used to store the input and output data of currently running or soon to run jobs. A delayed transfer of a job's output data results in sub-optimal use of scratch space in that the space is held up by a job that is currently not running. Further, a delayed offload also renders output-data vulnerable to center purge policies. The loss of output-data leads to wasted user time allocation, which is very precious and obtained through rigorous peer-review. Thus, a timely offload can help optimize both center as well as user resources.

The need for timely data offloading is also fueled by the, often, distributed nature of computing services and users' job workflow, which means that data needs to be shipped to where it is needed. For example, several HPC applications analyze intermediate results of a running job, through visualizations, to study the validity of initial parameters and change them if need be. This process requires the expeditious delivery of the result-data to the end-user visualization application for online feedback. A slightly offline version of this scenario is a pipelined execution, where the output from one computation at supercomputer site A is the input to the next stage in the pipeline, at supercomputer site B.

The common thread in both these cases is the timely offload or delivery of output data. In the former usecase, it can be stated as: *Offload by a specified deadline to avoid being purged*; In the latter, the twist is to: *Deliver by a specified deadline to ensure continuity in the job workflow.*

2. SOLUTION SPACE

Offloading large data to end-user sites is often mired by various factors. First, a direct download from the HPC center to the end-user requires both end resources to be available for the entire duration of the transfer. This can be a significant space and bandwidth commitment from both the HPC center and the end-user. For instance, the end-user resource might be unavailable when the data needs to be offloaded from the supercomputer scratch space. This renders the result-data vulnerable to center purge policies or delays eventual delivery. A desirable alternative, however, is to quickly move the data from center scratch space—to an intermediate storage location—so that the high-end, expensive resource can be relieved. Better yet, the intermediate storage location can be on the data path to the end-user so the data can be delivered at the destination when the end-resource becomes available again. IBP [11] and Kangaroo [16] explore this space to some extent. However, these techniques offer merely a staged offloading mechanism and do not factor in bandwidth variations, orthogonal bandwidth or delivery constraints.

Second, current data offloading schemes from HPC centers do not exploit orthogonal bandwidth that might be available between two end points. Exploiting residual, unused bandwidth in the data path between the center and the end-user can help alleviate several problems endemic to data downloading, such as bandwidth volatility. Peer-to-peer (p2p) data delivery schemes have explored this space with much success [4]. However, these techniques have not been applied to large, scientific data and are also not aware of application-level delivery constraints [6].

What is needed is an architecture for timely end-user data delivery/offloading that encompasses the data path between the supercomputing center and the end-user. We propose a combination of the above schemes, coupled with a monitoring component to react to bandwidth degradation so a specified delivery constraint can be met.

3. APPROACH

We now discuss how systematic staging of data is achieved to retrieve result-data from HPC centers. In order to support high-bandwidth data transfer from the center, our system utilizes a number of pre-specified intermediary nodes. The objective is to choose resources that are closer (in terms of bandwidth) to the center, so that data can be quickly offloaded to such resources.

We propose to utilize structured p2p networks [14, 15, 13] to locate and select intermediary nodes (N_i's) in a decentralized manner. Resources that intend to participate in the staging system join a p2p network, and through it are able to reliably communicate with other participants in the network. Before submitting a job to the HPC Center, the submission site (N_s) utilizes the overlay to discover appropriate N_i's between itself and the center as follows. N_s sends out a number of discovery messages on the p2p network with random destination addresses. By virtue of the p2p routing property [14], the messages are received at some N_i's. On receiving such a discovery message, an N_i replies with its IP address. In this way, N_s *discovers* a number of available N_i's. N_s then interacts with the center to sort the N_i's with increasing latency from the center, while at the same time with decreasing latency from N_s. This set of nodes is provided to the center to utilize as the intermediary nodes.

Upon completion of the job, the data-offload is done as follows. The center splits the result-data into chunks and starts transferring the chunks to a number of nearby nodes from the set of N_i's. The number of nodes used for this purpose, i.e., the fan-out, is chosen to achieve maximum out-bound center bandwidth utilization. Note that these Level-1 intermediary nodes may also further transfer data to the Level-2 intermediary nodes (once again chosen from N_i's), and so on. This in essence results in data to flow towards N_s, though it is not pushed to N_s. Decoupling N_s from the data push path allows the center to offload the data at peak (pre-specified) out-bound bandwidth without worrying about the availability (and connection speed) of N_s, while enabling N_s to pull (retrieve) data from N_i's as necessary. Figure 1 illustrates the overall data-flow path between the center and the submission site.

Landmark Nodes. Since we rely on p2p-participants to serve as intermediary nodes, a scenario of insufficient intermediaries is possible. For instance, the submission site may not have access to any intermediate nodes on the path to the HPC center. To avoid such a scenario, we propose to utilize a number of geographically distributed Landmark nodes that are always available and can serve as intermediary nodes in case enough p2p-nodes are not available. The Landmark

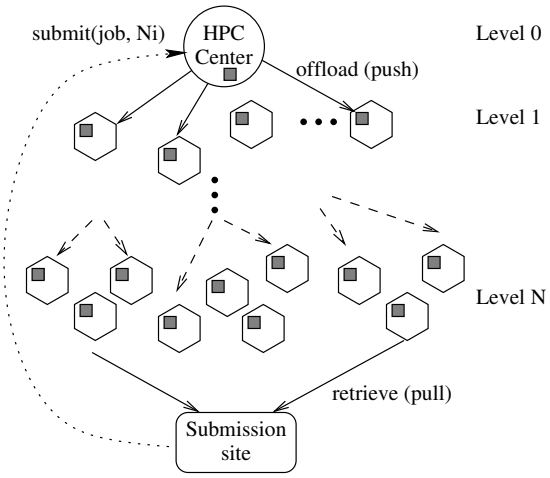

Figure 1: The data flow path from the HPC center to the submission site. The intermediary nodes are represented by hexagons. The participants also run an instance of the NWS (gray dot) for bandwidth monitoring.

nodes can be other HPC centers, or locations which are also interested in receiving the data. The location and number of the Landmark nodes is determined through out-of-band agreements. Depending on the utilization agreements, the Landmark nodes for different submission sites/applications may be different. In our implementation, we assume that the a site is aware of the necessary Landmark nodes a-priori and does not need to discover them as is the case for p2p-nodes.

Offloading Schedules. The submission site N_s specifies the N_i's and the possible data-flow paths from the center to N_s in the job submission script (e.g., a PBS script). A number of alternate approaches for data flow are possible. For instance, the data may be replicated across different N_i's during the transfer from one level to the other. This will allow N_s to pull data from a number of locations, thus providing fault tolerance against node failure, as well as better utilization of the available in-bandwidth at N_s. The schedule can also be used to simultaneously deliver data to multiple interested sites in the network.

Providing Service Guarantees. The submission site and the HPC center have Service Level Agreements (SLA's) regarding how quickly data can be offloaded from the center. The system should provide guarantees to the submission site to ensure that the SLA's are not violated. Two example scenarios of such SLA's are offloading data from the center as quickly as possible, and offloading all data from the center before a prespecified deadline. The static approach to selecting fan-out is likely to be insufficient for this purpose as available bandwidth between the HPC center and Level-1 intermediary nodes, as well the bandwidth between various intermediary nodes may change over the duration of the data offloading.

To accommodate this dynamic behavior, we propose to monitor the available bandwidth and adjust the fan-out to meet the SLA. We employ the Network Weather Service (NWS) [17] to monitor the available bandwidth between participating nodes. Each participating node joins a "clique", which is a group of sensors that measure bandwidth. The clique enables the center to determine the available bandwidth and select a fan-out based on these measurements. If the available bandwidth is not sufficient to meet a particular SLA (either fastest offload or deadline based), the fan-out is adjusted. In case the number of available N_i's are insufficient for meeting the SLA, the submission site is informed, which in turn can either provide more intermediary nodes or accept the best effort from the HPC center.

Note on Fault Tolerance. As stated earlier, pieces of the result-data can be replicated across many participating intermediate nodes, facilitating retrieval from any subset of the nodes. Yet another approach we adopt is to erasure code the data to improve the reliability of the transfer, while minimizing the amount of data transferred. The computational cost of erasure coding can be paid by the Level-1 intermediary nodes if coding at the HPC center (which will be part of the job's time allocation) is an issue.

By way of eagerly offloading result-data from the center, we avoid data loss due to accidental purging. This also allows the center to free-up precious scratch space for in-coming jobs and their data, thereby improving its serviceability. By staging it on an intermediate network of nodes, en-route to the destination, we ensure that the offload will not fail due to end-user resource unavailability. The result-data can be pulled as and when the end-user resource becomes available. A point-to-point, manual offload, on the other hand, relies on end resources being available during the transfer. Moreover, these result-data transfers are usually coded within job scripts and are simply not tolerant to failure or end-resource unavailability.

Discussion. A number of systems such as Bullet [8, 7], Shark [2], and CoBlitz [9] have explored the use of multicast and p2p-techniques for transferring large amounts of data between multiple Internet nodes. The focus of these systems is on downloading of user data, or receiving multimedia streams where meeting strict QOS guarantees is not a major issue. In contrast, the target HPC environments of this work impose much more stringent performance requirements, e.g., important application result-data may be purged if not offloaded before a deadline, which entail innovation to support an efficient data transfer model. The novelty of our approach lies in the use of submission site specified intermediary nodes that allow the center to dynamically adjust data flow paths even in the presence of failures. By placing the responsibility of selecting the intermediary nodes on the submission site, the system allows the site to spend as much resources as it deem necessary for a particular result-data. This also frees the center from spending crucial resources that can otherwise be spent on processing applications. We believe such an approach will have a profound impact on the staging-out of data, and will improve the overall center performance.

Table 1: The time to transfer a 95 MB file under different schemes.

	Transfer Time (s)			
	Direct	Random	Measurement based	Forecast based
Offload	739	245	214	210
Forward	N/A	431	393	370
Pull	739	665	663	663

4. EVALUATION

In this section, we present an evaluation of our result-data offload approach. We have implemented the system as described in Section 3 using about 3000 lines of C code. Our current implementation runs on Linux 2.6 kernel, but is readily portable to other platforms. In the following, we discuss the effectiveness of our design in achieving faster HPC center offloads.

Experimental Setup. We emulated the dynamic behavior of the proposed data offload model using the distributed testbed facilities of PlanetLab [10]. For our experiments, we choose 22 PlanetLab sites such that the HPC center and the submission site were on opposites coasts of the US, while the rest of the nodes were geographically scattered in between. All the nodes were arranged in a tree with the center as root, as discussed in Section 3, with number of children ranging from zero to four, and two levels of intermediaries. Such a tree offers multiple data flow paths from the center to the submission site and allows for testing the approach under different scenarios. Finally, the numbers reported in the following represent averages over a set of three runs.

Approach Feasibility. In the first set of experiments, we determined the feasibility of our proposed approach in achieving improved offloading times compared to a simple point-to-point direct transfer (Direct). For this purpose, we used a 95 MB ISO file and measured the time of a direct transfer between the center and the submission site. To allow for a fair comparison, the direct transfer was done using a simple TCP connection, instead of advanced transfer tools such as GridFTP [3] and HSI [5]. Next, we used the intermediary nodes for offloading. For this experiment, we used a static data flow path, where each node in the flow tree has four children. For each chunk of the file, we utilized a random selection of intermediary nodes (Random) at each Level of the tree – a technique inspired by the RanSub [7] approach in Bullet [8].

We measured the times to offload data from the source, time to forward the data from Level-1 to Level-2, and the time it would take the submission site to pull the data. Table 1 shows the results. The direct transfer time was observed to be 739s. Note that under Direct, the time to send the file (offload) is the same as the time to receive the file (pull). In contrast, Random reduced the transfer time by 66.9%. Moreover, the forward time gives some insight into the time it would take for the entire file to be forwarded from Level-1 to Level-2, i.e., when it becomes available at the Level-2 intermediary nodes.

Compared to Direct, the time to pull the data on the submission site is reduced by 10.0%. Note that our implementation used a simple serial approach to pulling data from Level-2 nodes, and the pull time can be further reduced by more advanced techniques, e.g., parallel retrievals from Level-2 nodes similarly as in Shark [2]. Also note that the pull time represents the time to transfer the file from Level-2 nodes to the submission site, and does not include the transfer time from the source to Level-2 nodes. However, the submission site pull is asynchronous, and can start as soon as chunks begin to arrive at Level-2 nodes, which was observed to be a fraction ($< 1\%$) of the forwarding time. We note that the overall transfer time, i.e., the time from when the source starts sending the data to when the submission site has received all the data is not a suitable metric, as our approach allows the site to be offline during the offloading process and delay starting the pull as necessary.

Dynamic Data Scheduling. In this experiment, we utilize bandwidth measurements (determined using NWS) to dynamically choose appropriate data flow paths. Two techniques are tested: one based on the measured bandwidth between nodes, while the other based on the bandwidth predictions provided by NWS. In either case, the bandwidth is used to greedily provision Level-1 nodes with the aim to increase the fan-out till maximum (predetermined) center outbound bandwidth is utilized.

As seen in Table 1, both the bandwidth measurement-based and prediction-based approaches out-perform Random. Simple bandwidth measurement takes 214 seconds to offload the file from the source. This reduces the offload time by 12.7% and 71.0% compared to Random and Direct, respectively. Use of bandwidth measurements also results in reduced intermediary forwarding time, which compared to Random is reduced by 8.9%. The time to pull the file to the submission site remains fairly unchanged at about 663s. This is expected, as the flow paths do not affect the time it would take for the submission site to pull the file.

Next, we expected NWS bandwidth predictions to do significantly better, however, they only marginally improved (1.9%) the offload time compared to measurement-based. The 5.9% improvement in the forwarding time, compared to measurement-based, shows that prediction-based provisioning is promising. We believe that with larger transfers the predictions will become more accurate and the benefit will increase. As before, the pull time remains almost the same.

Finally, the results show that bandwidth measurement provides a good tool for improving the offload times. Bandwidth measurement can even be used at the beginning of the offload, between the HPC center and the end-user destination, to measure if a direct transfer will result in significantly higher throughput. Also, it can serve as a resource availability check before the offload. This way, we can decide if and when to do the distributed offload.

Error Coding Overhead. In this experiment, we measured the effect of Error Coding in achieving fault tolerance. For this purpose, we randomly failed several intermediary nodes

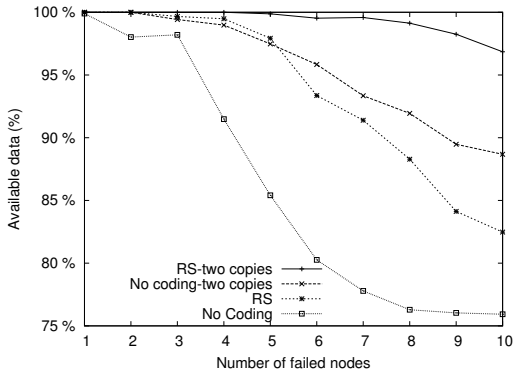

Figure 2: Available data under different error coding schemes, as intermediary nodes fail.

during the course of the transfer and determined what portions of the file have become unavailable. The experiment was repeated with increasing number of failed nodes, up to 10 (50%). Figure 2 shows the average results over three runs for four scenarios: with no error coding, using 4:5 Reed-Solomon [12] coding (RS), and using replication to create two copies under both no error coding and RS. As expected, using neither error coding nor replication causes data to become unavailable even with a single failure, with up to 24.1% data being unavailable with 10 failed nodes. Use of error coding or replication allows the file to be transferred successfully even when multiple nodes on the path from the center to the client fail. Note that both RS-single copy and replication are able to provide 100% availability with up to two (10%) node failures. This is promising as our RS code has only 25% redundancy to that of 100% with replication. However, with additional node failures, simple replication is able to provide better availability than RS. Creating two copies of data under RS further improves data availability: 100% availability when 25% of the intermediary nodes have failed, 96.9% availability with the extreme case of 50% of failed intermediary nodes. Hence, error coding at the center along with replication through multiple data-flow paths can provide excellent fault-tolerance behavior for the offloading process.

In summary, we have shown that the proposed approach has the potential to reduce data offload times from HPC centers and the availability of multiple data flow paths provides protection against data loss due to failure of intermediaries or the submission site.

5. CONCLUSION

In this paper, we have presented the design and implementation of a result-data staging-out service for HPC centers. Staging-out large data to end-user locations in a timely manner is critical to center operations, its availability and serviceability. Our approach presents a fresh look at offloading by using a set of user-specified intermediary nodes to construct a p2p network and transferring data based on bandwidth-adaptation. Our results indicate that this approach effectively utilizes orthogonal, residual bandwidth and can serve as an alternative to direct transfers, which may not always be feasible, optimal, or fault-tolerant. While a distributed stage-out can be very viable, it also throws open future research questions in terms of the strategic placement, and selection, of intermediate nodes between an HPC center and end-user destinations.

6. REFERENCES

[1] LCF Data Management, 2007. `http://info.nccs.gov/resources/jaguar/filesystems`

[2] S. Annapureddy, M. J. Freedman, and D. Mazires. Shark: Scaling file servers via cooperative caching. In *Proc. 2nd USENIX NSDI*, 2005.

[3] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke. GASS: A data movement and access service for wide area computing systems. In *Proc. 6th Workshop on I/O in Parallel and Distributed Systems*, 1999.

[4] B. Cohen. BitTorrent Protocol Specification, 2007. `http://www.bittorrent.org/protocol.html`.

[5] M. Gleicher. HSI: Hierarchical storage interface for HPSS, 2007. `http://www.hpss-collaboration.org/hpss/HSI/`.

[6] S. Kiswany, M. Ripeanu, A. Iamnitchi, and S. Vazhkudai. Are peer-to-peer data dissemination techniques viable in todays data intensive scientific collaborations? In *Proc. 13th Euro-Par Conference*, 2007.

[7] D. Kostic, A. Rodriguez, J. Albrecht, A. Bhirud, and A. M. Vahdat. Using random subsets to build scalable network services. In *Proc. 4th USENIX USITS*, 2003.

[8] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proc. 19th ACM SOSP*, 2003.

[9] K. Park and V. S. Pai. Scale and performance in the CoBlitz large-file distribution service. In *Proc. 3rd USENIX NSDI*, 2006.

[10] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proc. 1st ACM HotNets*, 2002.

[11] J. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolski. The Internet Backplane Protocol: Storage in the network. In *Proc. NSS*, 1999.

[12] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice & Experience*, 27(9):995–1012, 1997.

[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A Scalable Content-Addressable Network. In *Proc. SIGCOMM*, 2001.

[14] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM Middleware*, 2001.

[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. SIGCOMM*, 2001.

[16] D. Thain, S. S. J. Basney, and M. Livny. The kangaroo approach to data movement on the grid. In *Proc. 10th IEEE HPDC*, 2001.

[17] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computing Systems*, 15(5):757–768, 1999.