Timely Offloading of Result-Data in HPC Centers*

Henry M. Monti, Ali R. Butt
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061 USA
{hmonti, butta}@cs.vt.edu

Sudharshan S. Vazhkudai
Oak Ridge National Laboratory
Oak Ridge, TN 37831 USA
vazhkudaiss@ornl.gov

ABSTRACT

High performance computing is facing an exponential growth in job output dataset sizes. This implies a significant commitment of supercomputing center resources—most notably, precious scratch space—in handling data staging and offloading. However, the scratch area is typically managed using simple "purge policies", without sophisticated "end-user data services" that are required to balance center's resource consumption and user serviceability. End-user data services such as offloading are performed using point-to-point transfers that are unable to reconcile center's purge and users delivery deadlines, unable to adapt to changing dynamics in the end-to-end data path and are not fault-tolerant.

We propose a robust framework for the timely, decentralized offload of result data, addressing the aforementioned significant gaps in extant direct-transfer-based offloading. The decentralized offload is achieved using an overlay of user-specified intermediate nodes and well known landmark nodes. These nodes serve as a means both to provide multiple data-flow paths, thereby maximizing bandwidth as well as provide fail-over capabilities for the offload. We have implemented our techniques within a production job scheduler (PBS) and data transfer tool (BitTorrent), and our evaluation shows that the offloading times can be significantly reduced (90.2% for a 2.1 GB file), while also meeting center-user Service Level Agreements.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Network operating systems; D.4.2 [**Storage Management**]: Storage hierarchies; D.4.5 [**Reliability**]: Fault-tolerance

*This research is sponsored in part 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, and by the U.S. National Science Foundation Faculty Early Career Development (CAREER) Program CCF-0746832.

ICS'08, June 7–12, 2008, Island of Kos, Aegean Sea, Greece.

General Terms

Design, Performance, Reliability

Keywords

High performance data management, offloading, HPC center serviceability, end-user data delivery

1. INTRODUCTION

Supercomputing centers routinely generate huge amounts of data, resulting from high-throughput computing jobs. These are often result-datasets or checkpoint snapshots from long-running simulations, which 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. 7 in the Top500 supercomputers as of this writing, is 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, 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 at 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 High Performance Computing (HPC) center *user services* are often reminiscent of early computers. Supercomputer user services involve a host of issues such as data transfers, storage, software configuration and compilation, all of which are critical to a successful application run [3]. In this paper, we confine ourselves to data services. 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 constraint and enforce purge policies to manage the precious scratch space by deleting data based on a time window (ranging from a few hours to a few days) [2, 1]. As supercomputer centers become crowded, the purge policies get more stringent to ensure space for incom-

ing jobs. The purge window is, therefore, a product of the center's load, its provisioned storage and its desire to maintain a certain level of serviceability. 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 [6], `sftp`, hsi [13], and `scp`. The inherent problem with several point-to-point transfer tools, used to offload data from supercomputers, is that they are only optimized for transfers between two well-endowed sites. For example, the TeraGrid offers several optimizations (TCP buffer tuning, parallel flows, etc.) for GridFTP transfers between the various site pairs that make up the TeraGrid, which are already well connected (10-40 Gbps links). In contrast, end-user data delivery involves providing access to the data at the user's desktop. It cannot be ignored as a "last-mile" issue.

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 offload 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 implies 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 site B (Figure 1). Large-scale user facilities such as the Spallation Neutron Source (SNS) [8] and LEAD [7] that employ distributed workflows are already facing these problems and require efficient end-user data delivery techniques.

The common thread in both of the example cases above 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.*

1.1 Designing an Offload Scheme

Current solutions for offloading large data to end-user sites are often mired by various factors. First, a direct download from the HPC center to the end-user requires that end resources be available for the entire duration of the transfer. If this is not the case, the result-data is rendered vulnerable to center purge policies. 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.

Second, current data offloading schemes from HPC centers do not exploit orthogonal (residual, unused) bandwidth that might be available between two end points. Exploiting such bandwidth 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 [9]. However, these techniques have not been applied to large, scientific data and are also not aware of application-level delivery constraints [14]. Further, p2p techniques are optimized to work for a *pull* model rather than a *push*.

What is needed is an architecture for timely end-user data delivery that is able to reconcile both the HPC center's as well as a user's constraints amidst varying bandwidth and resource availability conditions.

1.2 Our Contributions

This paper makes the following contributions.

Staged and decentralized offloading: We propose a combination of both a staged as well as a decentralized offloading scheme for job output data. This is fundamentally different way of delivering result-data in HPC centers. Compared to a direct transfer, our techniques have the added benefits of resilience in the face of end-resource failure and the exploitation of orthogonal bandwidth that might be available in the end-to-end data path.

User-specified intermediate nodes: We adopt a novel variation to the use of intermediate nodes that differs from how they are used in most decentralized systems. The nodes participating in the transfer are specified and trusted by the user, thereby eliminating the fundamental concern of data delivery through a set of unreliable nodes in a decentralized environment. We propose ways in which these nodes can be specified.

Bandwidth adaptation and on-the-fly decision making: We employ a decision making component that factors in several parameters such as a center's purge deadline, user delivery schedule and a snapshot of current network conditions between the center and the end-user to determine the most suitable approach to offload.

Fault-tolerant Offload: We utilize erasure coding schemes to ensure that the offload is fault-tolerant.

Integration with real-world tools: Finally, we have developed our solution in the context of real-world tools such as PBS [5] job submission system and BitTorrent [9].

2. RELATED WORK

The use of intermediate buffers to hide latency or to provide fault tolerance is a common practice in OS as well as file systems. Kangaroo [26] extends this idea to Grid computing environments, with the goal to provide reliability against transient resource availability. However, Kangaroo simply provides a staged transfer mechanism and does not concern itself with network vagaries or changing route dynamics in an end-to-end data path.

IBP [20] offers a data distribution infrastructure with a set of strategically placed resources to move data. Our approach also exploits the presence of pre-installed storage nodes for

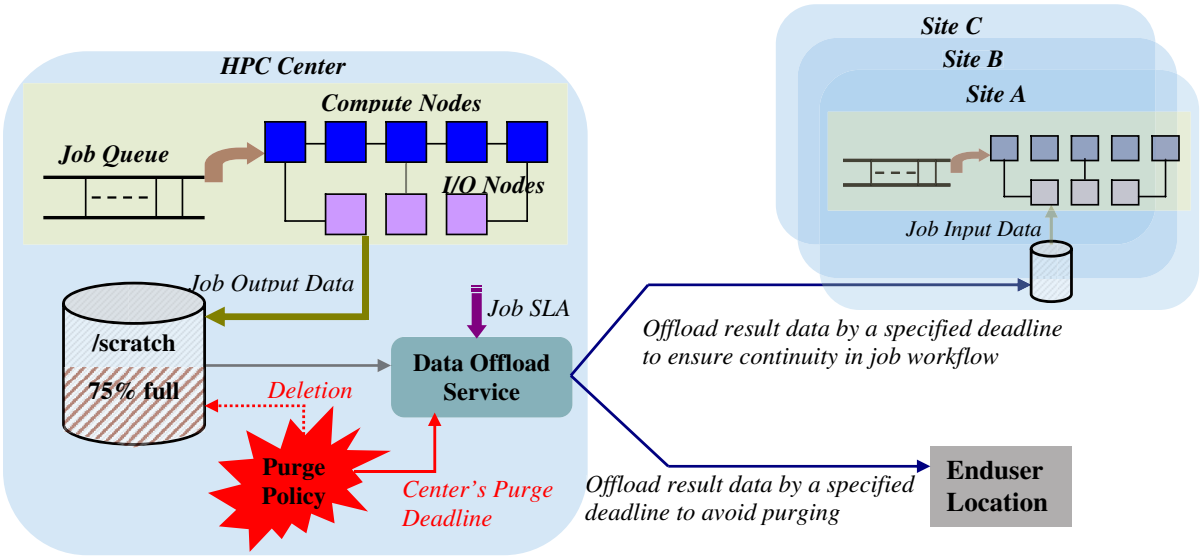


Figure 1: Depiction of usecases for a timely offload of result-data: (a) an expeditious offload to release center scratch space and to protect the data against a purge, (b) an end-user data delivery, and (c) data delivery to another part of the job workflow. The figure also shows the interplay between the various components in the HPC center.

data delivery as and when they are available. However, we differ significantly in our attempt to combine both a staged as well as a decentralized data delivery. The induction of user-specified nodes also allows the system to optimize the offload on a per user basis, which is not possible with IBP. Further, our approach is unique in comparison to the above techniques since we strive to meet a deadline in delivering as well as in timely offloading from the HPC center.

A number of systems such as Bullet[16, 15], Shark [4], CoDeeN [30], and CoBlitz [18] 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. The target offloading in this work requires factoring in center-user service agreements and dynamic resource availability, which are not considered in these systems.

The approach of downloading large files from several mirror sites has been validated by its wide-spread use in BitTorrent [9], and many protocols for parallel downloading from mirror sites have been proposed [24, 23, 10]. These works are complimentary, and we built on the principles developed in these systems, especially BitTorrent.

The Network Weather Service (NWS) [31] provides a powerful framework which allows the resources of distributed computers to be monitored. A number of resources can be monitored such as the pair-wise bandwidth between computers and each computer's CPU utilization, though there are many other options. NWS bandwidth measurements have been used in a static context to determine a Grid data site, offering optimal download rates, from among multiple replicated alternatives [29, 28, 27]. In this work, however, we use measurements to determine a path within a network of nodes and dynamically adjust it based on bandwidth degradation.

3. DESIGN

In this section, we discuss the design of our offloading system. We first present an overview of the system architecture, followed by details of techniques used for selecting the intermediate nodes. Finally, we describe the data offloading process adopted in our system.

3.1 Architecture Overview

A decentralized data offloading scheme for HPC centers that ensures timely data delivery (Figure 1) is achieved using a combination of strategies both at the center as well as the end-user to orchestrate the transfers.

We first discuss the *Data Offload Manager* at the HPC center. The *Manager* takes as input, guidelines regarding the purge deadline, D_{purge}, from the HPC center's scratch space purging system, and job specification from the job submission system. The specifications include the output data size, S, the job's data delivery schedule as par the Service Level Agreement (SLA), J_{SLA}, and other details such as any potentially available intermediate nodes, $< N_i, P_i, BW_i >$, where P_i denotes usage properties/constraints of the node, N_i, and BW_i denotes the current snapshot of the observed NWS bandwidth between the HPC center and N_i. Based on these parameters, the *Manager* decides upon either a direct or a decentralized transfer of the job's output data. The decision, which we call an offload schedule, O_s, delivers the data in time, $T_{offload}$, which satisfies the property,

$$T_{offload} \leq Min(D_{purge}, J_{SLA})$$

Even after a particular course of action, such as a decentralized transfer, is chosen, a decision-making component constantly re-evaluates the offload based on an updated $< N_i, P_i, BW_i' >$, where BW_i' is the latest snapshot of NWS

bandwidth measurements. If the re-evaluated time to offload, $T'_{offload}$, satisfies the property,

$$T'_{offload} > J_{SLA},$$

then, alternate routes are taken to meet the SLA.

For specification of input parameters, we instrument the center's job submission system so that end-users can specify delivery constraints and deadlines as part of their regular PBS [5] job scripts.

To automatically initiate data offload upon job completion, we use and extend our earlier work [32] on instrumenting the job submission system for starting user-specified direct data transfers (e.g., scp or GridFTP) upon job completion. Here, we use that work to intimate the offload manager of the availability of a job's result dataset for decentralized stage-out of the data. Having a center-wide offload manager has the advantage that the manager can perform global optimization (e.g., higher priority to a stage-out that is on a tight deadline).

A final piece in the data offload architecture is the utilization of a number of user-specified intermediate nodes to which data from the center is offloaded, and from which the submission site can then asynchronously retrieve the data. The intermediaries provide multiple data flow paths from the center to the submission site, which lead to better offload bandwidth utilization, faster retrieval speeds, as well as fault-tolerance in the face of failure.

3.2 Intermediate Nodes

3.2.1 Motivation for Collaboration

The decentralized offload put forth in this paper makes extensive use of intermediate nodes. We envision these to be nodes that are specified and trusted by the user. More specifically, consider the following collaboration scenarios that present a strong case for the participation of intermediate nodes in the data offloading process.

In today's HPC environment, supercomputing jobs are almost always collaborative in nature. In fact, a quick survey of jobs awarded compute time on the ORNL NLCF, through the DOE's INCITE [11] program, suggests that these jobs involve multiple users from multiple institutions. This collaborative property is even more true in the TeraGrid, which is one of our key drivers for end-user data delivery. Jobs in the TeraGrid are usually from a *virtual organization*, which is a set of geographically dispersed users from different sites, coming together to solve a problem of mutual interest for a certain duration. In such cases, it is clear that many users, from different sites will be interested in the resulting job output data. Thus, there is a natural need to dispatch the result data to more than a single location.

This emerging property of collaborative science can be exploited to perform a collaborative offload of job output data. Participating sites of the job can come together to form an overlay of intermediate nodes that contribute space and bandwidth for the offload. We argue that there exists a *natural incentive* for the participating sites to do so. Such a definition of intermediate nodes makes them more reliable and alleviates a key concern of precious result-data being transferred through an unreliable substrate.

3.2.2 Discovery

Although the intermediate nodes (N_i's) are selected from among the participating sites, nodes may be unavailable at any given time, necessitating discovery.

We use a p2p overlay (Pastry [25]) to arrange the N_i's. Use of the overlay provides reliable communication with other participants in the network. The participating sites use the overlay to advertise their availability to other nodes in the overlay using random broadcast. Nodes that receive these messages build local information about available nodes for offload. A given node can use its own policies and information about a remote node's capacity to make a decision regarding whether to use the remote node for the offload. Finally, before submitting a job to the HPC center, the submission site 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 intermediate nodes, and becomes an integral part of the job's workflow.

3.2.3 Landmark Nodes

The reliance of our design on intermediate nodes exposes the offload system to possible failures due to lack of sufficient N_i's. For instance, the submission site may not have access to any (or sufficient enough) intermediate nodes on the path to the HPC center. This could be either due to the lack of many participating sites in the job or due to the volatility of the intermediate nodes. To avoid such a scenario, we propose to utilize a number of geographically distributed Landmark nodes that are always available and can serve as intermediate nodes. The Landmark nodes can be other HPC centers, or nodes along national links such as, Internet2, Lambda Rail or the TeraGrid to which many end-users may be connected. The location and number of the Landmarks is determined through out-of-band agreements with the HPC center. We note that the envisioned Landmark nodes cannot provide best offload options for all submission sites, and thus are used only as a backup option when user-specified intermediaries are not available.

3.3 The Data-Offloading Process

Once the job execution completes, the data-offloading process is initiated. First, the center chooses a number of nodes from the set of N_i's ordered by available bandwidth. The exact number of nodes used for this purpose, i.e., the fan-out, is chosen to achieve maximum (pre-specified) outbound center bandwidth utilization, or to meet previously agreed-upon offload deadlines. These chosen N_i's serve as the Level-1 intermediate nodes. Note that the selected fan-out is not static, and can vary depending on the transfer speeds achieved. Second, the result-data is split into chunks and parallel transfer of the chunks to Level-1 nodes is initiated. Since the Level-1 nodes are much closer to the center than the submission site, the offload time is expected to be much smaller than a direct transfer to the submission site. Third, Level-1 intermediate nodes may also further transfer data to the Level-2 intermediate nodes (once again chosen from N_i's), and so on. Consequently, data flows towards N_s, though it is not pushed to N_s. Finally, N_s can asynchronously retrieve the data from the Level-N nodes. 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)

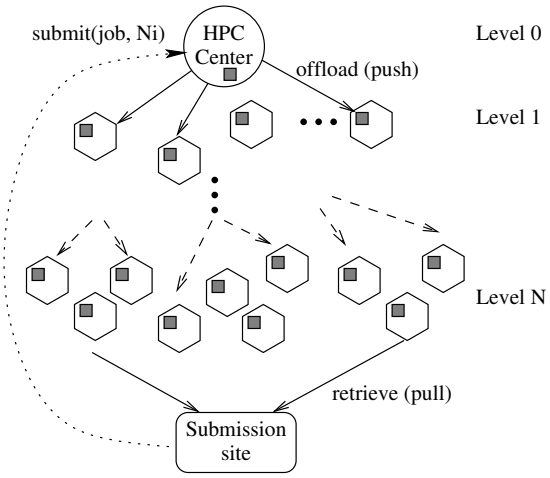


Figure 2: The data flow path from the HPC center to the submission site. The intermediate nodes are represented by hexagons. The participants also run an instance of the NWS (gray square) for bandwidth monitoring.

of N_s, while enabling N_s to pull (retrieve) data from N_i's as necessary. The various steps in the offload process are illustrated in Figure 2.

The use of intermediate nodes in our system provides multiple data-flow paths from the center to the submission site N_s, leading to several alternative options for data delivery. For instance, 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.

3.3.1 *Providing Service Guarantees*

The submission site and the HPC center have SLAs regarding how quickly data can be offloaded from the center. Similar to the intermediate node specification, the SLAs are also specified in a job submission script.

Given the dynamically changing bandwidths between participants, a fixed or statically chosen fan-out is insufficient. Therefore, we propose a bandwidth monitoring-based scheme to dynamically adjust the fan-out and ensure meeting the SLA. We employ the Network Weather Service (NWS) [31] to monitor and estimate the available bandwidth between participating nodes. Each participating node joins a "clique", which is a group of sensors that measure bandwidth. The clique gives the center an estimate of the bandwidth available from it to different nodes. The center uses this information to decide whether a chosen fan-out is sufficient to meet a particular SLA, or needs to be increased. If needed, additional nodes from the set of N_i's can be chosen to increase the fan-out and meet the SLA. At each level, a decision making component re-evaluates the time to offload as mentioned earlier. 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 intermediate nodes or accept the best effort from the HPC center.

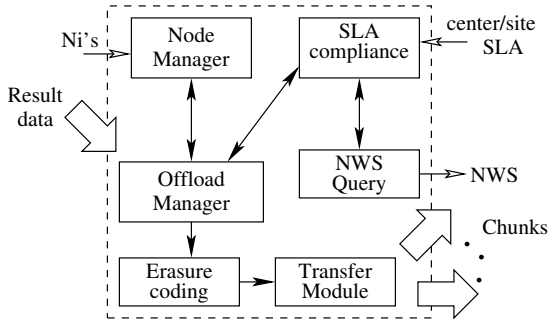


Figure 3: System components and their interactions.

3.3.2 *Fault Tolerance through Erasure Coding*

As stated earlier, pieces of the result-data can be replicated across many participating intermediate nodes, facilitating retrieval from any subset of the nodes. In addition to this, we apply erasure code [17, 22] to the data to improve the reliability of the transfer, while minimizing the amount of transferred data. The computational cost of erasure coding can be paid by the Level-1 intermediate nodes if coding at the HPC center (which will be part of the job's time allocation) is an issue.

In summary, by way of eagerly offloading result-data from the center, our system avoids data loss due to center's purge policies. This in turn 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 from the intermediate nodes as and when the end-user resource becomes available.

4. IMPLEMENTATION

We have implemented the system as described in Section 3 using about 3000 lines of C code. The p2p substrate of our system is built using FreePastry [12]. Figure 3 shows the architecture of the software that runs on all the participating nodes, and the interactions between key components. The list of N_i's and the SLA are provided through the job submission script, and the bandwidth measurements are obtained via NWS queries.

The erasure code that we have used is Reed-Solomon [21], in 4:5 coding configuration, i.e., four input chunks are coded to produce five output chunks, with a redundancy of 25%. The chunk-size is a tunable parameter which can be set based on the size of the data-sets being offloaded.

In the following we discuss how we have leveraged and instrumented several widely-used tools for the specification and utilization of intermediate nodes.

4.1 Integration with Job Submission

We have instrumented the PBS [5] job submission system that is prevalent in HPC centers to enable specification of user-defined intermediate nodes and deadlines. To this end, we have devised a way for specifying intermediate nodes and delivery deadlines as annotations within a standard PBS script. These annotations are specified as directives, much like other PBS directives (e.g., `#PBS`). The intermediate nodes can be further qualified with policy spec-

```
#PBS -N myjob
#PBS -l nodes=128, walltime=12:00
mpirun -np 128 ~/MyComputation
#Stageout Output DestinationSite
#InterNode node1.Site1:49665:50GB
...
#InterNode nodeN.SiteN:49665:30GB
#Deadline 1/14/2007:12:00
```

Figure 4: An example instrumented PBS script.

ification that captures usage constraints. These constraints include the amount of space available for offload on a node, and the node's availability. More fine grained policies can be easily added.

Figure 4 shows an instrumented PBS script, wherein a user specifies the stage out to a destination, the use of intermediate nodes with their space constraints, a port number where our transfer protocol is listening, and a delivery deadline.

The annotated PBS script is submitted for execution to the job scheduler at the HPC center. It is intercepted by our parser that filters out directives specific to data offloading, and passes those details to an offloading service for data delivery. The remaining PBS Script is then handed over to the PBS queue for standard processing. The offloading service is aware of the center's purge deadline and attempts to reconcile that with user delivery deadline and intermediate/landmark nodes to achieve a desired data transfer schedule.

4.2 Integration with BitTorrent and NWS

We have designed our offloading mechanism to exploit the data dissemination abilities of BitTorrent [9] and network monitoring facilities of the Network Weather Service (NWS) [31].

Each participating node in our system runs an NWS daemon. We have configured NWS sensors that keep track of the vital statistics of each node, as well as record bandwidth measurements between nodes. These measurements are retrieved by our *Offload Manager* via periodic queries and used in determining appropriate offload paths that can sustain sufficient bandwidth to meet specified SLAs. The *Offload Manager* also employs the data from NWS to select additional peer nodes in case an SLA cannot be met.

The decision to add additional nodes to the offload path is driven by several factors: user-center delivery and purge deadlines, storage capacity of nodes (specified via the PBS script), and the available bandwidth.

Once a set of intermediate nodes is selected using NWS, we use BitTorrent's scatter-gather protocol to transfer the file from the center to the selected intermediate nodes. We have instrumented and extended the original BitTorrent protocol in the following novel ways: the entire result-data file is not transmitted to all the intermediate nodes, rather different (overlapping) subsets arrive at different nodes; the number of receiving nodes change dynamically based on our offload predictions and global utilization information; the receiving nodes are not necessarily the end-recipients of data; there are several levels of intermediate nodes that provide better control over SLA enforcement; and most importantly, the offload from the center is decoupled from the download at the end-host which occurs separately depending on end-host availability.

The *Offload Manager* creates a "torrent" file for the subset of data to be transmitted to a set of chosen intermediate nodes. The torrent file contains meta-data information about the data. The Manager also provides BitTorrent *tracking* services for providing the intermediate nodes information about what data has been transmitted to which node. Once the nodes receive the torrent file, they use the metadata information along with the tracker to "download" the data subset to their local storage. The process is repeated at all the intermediate node levels. Finally, the end-host can also use appropriate torrent files to download the result-data from the intermediaries, thus completing the offloading process.

5. EVALUATION

In this section, we present an evaluation of our result-data offloading approach using the implementation of Section 4. In the following, we discuss the effectiveness of our design in achieving faster HPC center offloads.

5.1 Experimental Setup

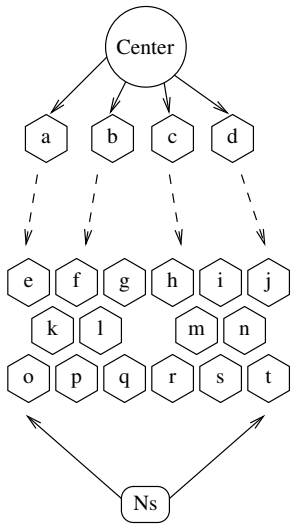
We emulated the dynamic behavior of the proposed data offload model using the distributed testbed facilities of PlanetLab [19]. For our experiments, we chose 22 PlanetLab sites such that the HPC center and the submission site were on opposites coasts of the US, while the rest of the nodes were geographically scattered in between. All the nodes were arranged in a tree with the HPC center as root, the 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. Figure 5 shows the experimental setup, as well as the observed pair-wise bandwidths between various nodes on the data flow path. In the following experiments the chunk size was set to 256 KB. Moreover, to account for the dynamic behavior of our testbed, the reported numbers represent averages over a set of three runs.

5.2 Approach Feasibility

In the first set of experiments, we determined the feasibility of our approach compared to a simple point-to-point direct transfer using scp (Direct). For this purpose, we used a range of file sizes from 100 MB to 2.1 GB and measured the time of a direct transfer between the center and the submission site. For our offloading approach, we used a combination of BitTorrent along with NWS as outlined earlier.

In Table 1, we compare Direct transfers with the times to offload data from the source (HPC center) to Level-1 nodes (Offload), time to forward the data from Level-1 to Level-2 (Push), and the time it takes the submission site to pull the data (Pull). Compared to a direct transfer, the Offload is able to release the HPC center scratch space by up to 86.7% to 90.2% sooner for the data sizes we considered. This has a significant impact on the HPC center serviceability since the free space can now be used for new incoming jobs.

Compared to Direct, the time to pull the data on the submission site is reduced by 86.0% to 90.4%. The reported pull time represents the time to transfer the file from Level-1 and Level-2 nodes to the submission site, and does not include the transfer time from the source. However, the submission site pull is asynchronous, and can start as soon as chunks begin to arrive at Level-2 nodes. We note that the overall transfer time, i.e., the time from when the source starts



(i) Relationship between nodes.

Node	PlanetLab site	Observed Bandwidth (Mbps)				
		N_s	a	b	c	d
Center	jerry.cc.vt.edu	2.05	45.7	12.6	13.5	11.1
N_s	planet1.scs.stanford.edu	-	2.13	2.02	2.37	2.28
a	bob.cc.vt.edu	2.13	-	12.6	13.4	8.36
b	pepper.planetlab.cs.umd.edu	2.02	12.6	-	30.4	12.5
c	salt.planetlab.cs.umd.edu	2.37	13.4	30.4	-	14.3
d	plgmu2.ite.gmu.edu	2.28	8.36	12.5	14.3	-
e	planet2.scs.stanford.edu	42.6	2.28	2.19	2.15	2.25
f	planetlab-1.cs.princeton.edu	2.08	7.93	7.12	9.32	10.9
g	planetlab-2.cs.princeton.edu	1.84	8.84	12.5	12.5	13.1
h	planetlab-3.cs.princeton.edu	1.96	8.26	7.63	9.69	11.3
i	planetlab-4.cs.princeton.edu	1.82	7.77	10.7	10.9	11.9
j	planetlab1.cs.purdue.edu	2.36	4.38	5.18	5.21	4.98
k	planetlab2.cs.purdue.edu	2.28	4.46	5.79	5.66	5.17
l	planetlab1.cs.wisc.edu	1.82	3.31	3.67	3.90	3.83
m	planetlab2.flux.utah.edu	3.48	2.50	2.73	2.83	2.65
n	planetlab4.cs.duke.edu	2.14	8.56	8.18	8.36	1.72
o	pl1.unm.edu	3.57	2.53	2.29	2.31	1.01
p	pl2.unm.edu	3.39	2.55	2.21	2.27	1.18
q	planetlab2.cis.upenn.edu	2.05	1.66	10.6	11.3	11.1
r	ricepl-2.cs.rice.edu	4.13	3.58	4.03	4.10	4.22
s	planetlab8.millennium.berkeley.edu	28.7	1.92	1.99	1.79	1.94
t	planetlab9.millennium.berkeley.edu	29.9	1.88	2.03	1.93	1.99

(ii) Chosen PlanetLab sites, and observed bandwidths between nodes. The dash represents node pairs that do not interact.

Figure 5: The experimental setup used for evaluation.

Table 1: The time, in seconds, to transfer varying file sizes, using a direct transfer (scp), and our decentralized approach.

File Size	100 MB	240 MB	500 MB	2.1 GB
Direct	286	727	1443	5834
Offload	38	95	169	570
Push	82	179	349	1123
Pull	29	93	202	562

Table 2: The time to transfer a 2.1 GB file using standard BitTorrent. The equivalent phases for our scheme are shown in brackets.

Phase	Time(s)
Send one copy from center (Offload)	1172
Send to all intermediate nodes (Push)	1593
Submission site download (Pull)	571

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.

5.3 Dynamic Data Scheduling

In this section, we compare our approach with a regular BitTorrent-based data transfer. In this case, we use NWS bandwidth measurements to greedily provision Level-1 nodes to increase the fan-out until a maximum (predetermined) center outbound bandwidth is utilized.

Table 1, discussed in the previous section, shows data offloading using the bandwidth measurement-based approach. Table 2 shows the time taken to deliver a 2.1 GB dataset using the regular, unmodified BitTorrent protocol. Our results indicate that all three steps in our approach: Offload, Push and Pull out-perform the corresponding steps in regular BitTorrent transfer. The Offload from the HPC center to Level-1 nodes is 50.4% faster, while the Push from Level-1 nodes to Level-2 nodes is 29.5% faster. Use of bandwidth measurements, therefore, results in reduced intermediate forwarding time. The time to pull the file to the submission site is slightly improved by 1.5%. This is expected, as the flow paths do not affect the time it would take for the submission site to pull the file. These results show that bandwidth measurement provides a good tool for improving offload times.

Table 3: Relative improvement in file transfer times using BitTorrent under varying chunk sizes, compared to the default chunk size of 256 KB.

Chunk Size	128 KB	256 KB	512 KB	1024 KB
Time saved (%)	-2.14	0	5.46	6.58

5.3.1 *Effect of Chunk Size on Offload times*

In our next experiment we varied the chunk size used by BitTorrent, and observed the effects on file transfer time. The results are in Table 3. As the chunk size increases the transfer time decreases. A chunk size of 1024 KB improves transfer speed by 6.58% when compared with the default chunk size of 256 KB. These results indicate transfers can benefit from larger chunk sizes.

5.3.2 *When to Employ Staged Offload?*

In the experimental setup we have adopted, the bandwidth available between the center and Level-1 nodes is greater than that between the center and N_s. Thus, in this setup, the center will always decide to perform staged offloading. In the next experiment, we modified the setup to use *Node a* as the end user site, and did not use N_s. Then, we repeated the above experiment to offload a 2.1 GB file, first, without considering direct transfer and always using the staged offload mechanisms, and second, with the ability to choose between direct and staged offload depending on the ability to meet a SLA deadline. We observed that for the first case, the time to offload and pull the data was 610

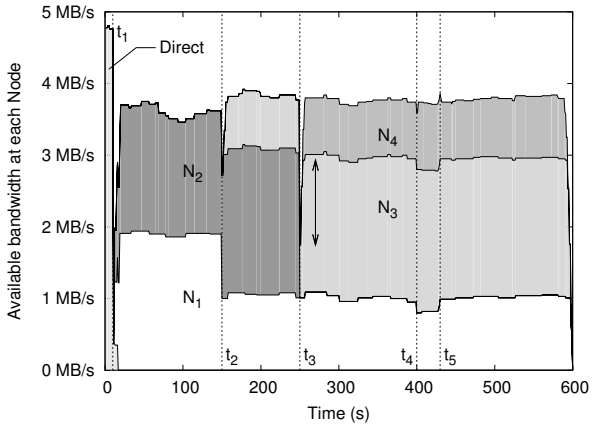


Figure 6: Utilized out-bound bandwidth at the center, as the system adjusts to failures and meets the 600s deadline for offloading. The labeled regions represent utilized bandwidth to individual nodes.

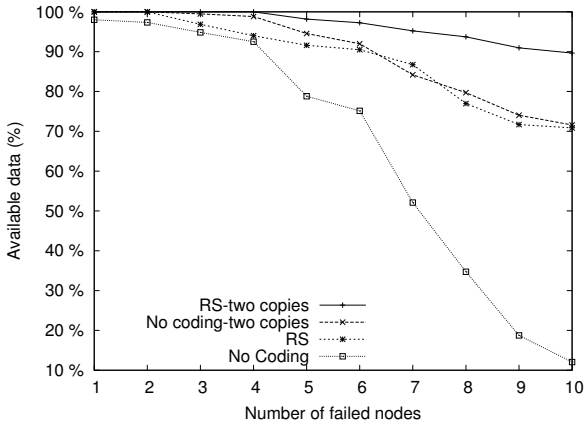


Figure 7: Available data under different error coding schemes, as intermediate nodes fail.

seconds and 400 seconds, respectively. In contrast, for the second case the direct transfer completed in 380 seconds, an improvement of 37.7% in offload times. This result coupled with the earlier experiments stress the need for the offload mechanisms to dynamically adjust to the variations in the system behavior and to not be hard-wired to simply always do a staged offload or a direct transfer.

5.4 Enforcing SLA

In the next experiment, we study the effectiveness of the proposed approach in enforcing SLAs. We assume that the submission site and the HPC center have agreed on an SLA to offload the 2.1 GB file to four Level-1 nodes (N_1 to N_4) or a direct transfer in 600s. Initially, we choose a site that supports a large bandwidth between the center and the site. Thus, our algorithm starts off by doing a direct transfer. However, at time $t_1 = 10s$, we limit the inbound bandwidth of the site to 1/10 of its value. Soon after this happens, our system realizes that the SLA cannot be met with a direct transfer and switches to a staged offload. Once an offload schedule is chosen, we utilize bandwidth provided by the NWS to estimate the time E_t it would take to offload the remaining chunks of the file. If E_t turns out to be longer than necessary to meet the SLA, the fan-out is increased. The process is repeated every time the available bandwidth predictions change. To force dynamic scheduling to come into play, we artificially introduced two bandwidth-changing events during the offload: at time $t_2 = 150s$, we limited the available bandwidth to N_1 to about 1 MB/s; and at time $t_3 = 250s$, we failed N_2. Figure 6 shows the sum of the utilized bandwidths between the center and each of the four Level-1 nodes reported every second. Initially, only N_1 and N_2 are used. Soon after t_2, the drop in N_1's bandwidth is detected causing an increase in E_t. The system reacts by increasing the fan-out to use N_3, so that E_t remains under the 600s deadline. Note that between t_2 and t_3, the maximum available bandwidth of N_3 was not needed to meet the SLA and was not utilized. However, when N_2 failed at t_3, the system first uses N_3's maximum bandwidth as observed as a spike (indicated by the arrow) in N_3's curve following t_3. However, this increase is not sufficient to compensate for the loss of N_2, hence, the fan-out is adjusted to also use N_4. Also note that between t_4 and t_5, the available bandwidth for N_1 is reduced significantly enough to cause the system to utilize a higher bandwidth to N_4 so that the overall total bandwidth is maintained to meet the SLA. Once N_1's bandwidth returns to normal, our greedy algorithm once again increases the use of N_1's bandwidth and reduces the use of N_4's bandwidth. The two spikes at t_4 and t_5 capture the system response time to these events. Finally, as observed from the figure, the system is able to transfer the file within the specified SLA by dynamically adjusting the fan-out.

5.5 Data Availability

In this experiment, we measured the effect of Error Coding in achieving fault tolerance. For this purpose we randomly failed several intermediate nodes 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 7 shows the average results over three runs for four scenarios: with no error coding, using 4:5 Reed-Solomon [21] 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 87.9% 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 have only 25% redundancy to that of 100% of 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 intermediate nodes have failed, 89.7% availability with the extreme case of 50% of failed intermediate 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.

6. 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 intermediate nodes to construct a p2p network and transferring data based on bandwidth-adaptation.

Our results indicate that our offloading approach improves the rate at which the data is staged-out of the center (90.2% for a 2 GB file), while allowing the submission site to pull the data as and when the site becomes available, at a much higher transfer rate because the result-data has already been staged closer. Further, offloading enables us to deliver data based on a previously agreed upon SLA, dynamically varying the fan-out as necessary. Such a scheme can be extremely useful to both HPC centers and users.

Our evaluation shows that the presented offloading scheme reacts well to system variations in meeting user-center SLA's and deciding when a staged offload is preferable to a direct transfer, and achieves good fault-tolerance via its use of erasure coding and replication.

In summary, the offloading 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 is highly competitive, 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.

7. REFERENCES

[1] UC/ANL Teragrid Guide, 2004. `http://www.uc.teragrid.org/tg-docs/user-guide.html\#disk`.

[2] NCCS.GOV File Systems, 2007. `http://info.nccs.gov/computing-resources/jaguar/file-systems`.

[3] User Support and Assistance, 2007. `http://www.nccs.gov/user-support/`.

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

[5] A. Bayucan, R. L. Henderson, C. Lesiak, B. Mann, T. Proett, and D. Tweten. Portable Batch System: External reference specification, November 1999. `http://www-unix.mcs.anl.gov/openpbs/docs/v2_2_ers.pdf`.

[6] 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.

[7] M. Christie and S. Marru. The lead portal: a teragrid gateway and application service architecture: Research articles. *Concurrency and Computation : Practice and Experience*, 19(6):767–781, 2007.

[8] J. W. Cobb, A. Geist, J. A. Kohl, S. D. Miller, P. F. Peterson, G. G. Pike, M. A. Reuter, T. Swain, S. S. Vazhkudai, and N. N. Vijayakumar. The neutron science teragrid gateway: a teragrid science gateway to support the spallation neutron source: Research articles. *Concurrency and Computation : Practice and Experience.*, 19(6):809–826, 2007.

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

[10] R. L. Collins and J. S. Plank. Downloading replicated, wide-area files – a framework and empirical evaluation. In *Proc. 3rd IEEE International Symposium on Network Computing*, 2004.

[11] Department of Energy, Office of Science. Innovative and Novel Computational Impact on Theory and Experiment (INCITE), January 2008. `http://www.er.doe.gov/ascr/incite/`.

[12] Druschel et. al. Freepastry, 2004 `http://freepastry.rice.edu/`.

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

[14] 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 International Euro-Par Conference: European Conference on Parallel and Distributed Computing*, 2007.

[15] 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.

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

[17] P. Maymounkov. Online Codes. Technical Report TR2003-883, New York University, New York, Nov. 2002.

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

[19] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proc. 1st ACM Workshop on Hot Topics in Networks (HotNets-I)*, 2002.

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

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

[22] J. S. Plank. Erasure codes for storage applications, 2005. Tutorial Slides, presented at *USENIX FAST*, `http://www.cs.utk.edu/~plank/plank/papers/FAST-2005.html`.

[23] J. S. Plank, S. Atchley, Y. Ding, and M. Beck. Algorithms for high performance, wide-area distributed file downloads. *Parallel Processing Letters*, 13(2):207–224, 2003.

[24] P. Rodriguez, A. Kirpal, and E. W. Biersack. Parallel-access for mirror sites in the internet. In *Proc. IEEE Infocom*, 2000.

[25] 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.

[26] D. Thain, S. S. J. Basney, and M. Livny. The kangaroo approach to data movement on the grid. In

Proc. 10th IEEE Symposium on High Performance Distributed Computing (HPDC10), 2001.

[27] S. Vazhkudai and J. Schopf. Predicting sporadic grid data transfers. In *Proc. 11th IEEE Int'l Symposium on High Performance Distributed Computing (HPDC-11)*, 2002.

[28] S. Vazhkudai, J. Schopf, and I. Foster. Predicting the performance of wide-area data transfers. In *Proc. 16th Int'l Parallel and Distributed Processing Symposium (IPDPS 2002)*, 2002.

[29] S. Vazhkudai, S. Tuecke, and I. Foster. Replica selection in the globus data grid. In *Proc. IEEE International Conference on Cluster Computing and the Grid (CCGRID 2001)*, 2001.

[30] L. Wang, K. Park, R. Pang, V. Pai, and L. Peterson. Reliability and security in the CoDeeN content distribution network. In *Proc. USENIX ATC*, 2004.

[31] 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.

[32] Z. Zhang, C. Wang, S. S. Vazhkudai, X. Ma, G. Pike, J. Cobb, and F. Mueller. Optimizing center performance through coordinated data staging, scheduling and recovery. In *Proc. Supercomputing 2007 (SC07): Int'l Conference on High Performance Computing, Networking, Storage and Analysis*, 2007.