

# **Improvements to GlideinWMS Data Handling System**

**Evan Boldt**

July 29, 2010

Northern Illinois University Research and Development Internship  
Fermi National Laboratories

## Table of Contents

1. Introduction.....	3
2. Changes Made.....	4
2.1 Slot Management.....	4
2.2 Transfer Resilience.....	5
3. Testing.....	6
3.1 Timing of Data Handling (DH) Phases.....	6
3.2 Slot Performance Testing with no Transfer Failures.....	7
3.3 Throughput Change During Connectivity Issues.....	9
4. Further Improvements.....	11
5. Conclusion.....	11
6. Acknowledgments.....	12
7. References.....	12

## 1. Introduction

During this Research and Development Internship (RDI) for Fermi National Accelerator Laboratories, (FNAL) computer programming work was done to improve the reliability and performance of distributed computing systems used by FNAL and its stakeholders. Specifically, work was done on Condor and GlideinWMS.

Condor [1] is a high throughput distributed computing system. High throughput systems are intended to fully utilize as many resources over time as possible, unlike high performance systems, which are intended to quickly produce results. Condor is one of many kinds of job distribution (batch) systems in use.

GlideinWMS [2], a management system widely used by the Open Science Grid (OSG) [3], US CMS, Fermilab, and other OSG stakeholders, is used to create a uniform distributed computing system. GlideinWMS distributes a glidein to grid resources, which receives condor jobs from the main pool. In doing so, a grid resource of any type of batch system can be used through the same Condor interface.

The enhancements also integrated with Globus.org file transfer services [4] since transfers over supported methods can be activated through a standardized interface based on user-specified transfer destinations rather than forcing the user to interface directly with the service in their job.

There are three main phases to a job: stage in, when data and executables are downloaded to the worker node; processing, when the real work of the job is being done; and stage out, when data is uploaded back to the user. These three phases must happen sequentially and do not compete for resources on the worker node. The concept of instruction set pipelining [5] in computer architecture is similar to an assembly line and can be applied to job phases. Independent operations can be performed on separate jobs simultaneously. One job can be in the processing phase while the previous job is in the stage out phase. Currently, phases are not separated and are not pipelined. So, the worker node's resources are not fully utilized while data is being transferred over the network. Another downfall to the lack of phase separation is that a transfer failure is not distinguished from a runtime failure, causing the job to be restarted from the beginning if the results are not returned. Separating the phases allows the job to be restarted directly to the transfer phase, rather than downloading and executing the job again.

## 2. Changes Made

### 2.1 Slot Management

The RDI project for the Center for Enabling Distributed Petascale Science (CEDPS) [6] aims to increase throughput on grids using GlideinWMS by allowing another job to run during data return for the previous job, and by improving data return reliability. The Data Handling (DH) Slot Management system is written in BASH, and replaces user-written, in-job upload systems. Instead of including the return mechanism in the executable, the job is submitted with extra information in the submission description file. When this extra information is present and the resource-intensive processing phase of the job is done, the Data Handling (DH) begins. First, the system looks for files that match a user-specified regular expression. If the files are large enough, a new slot is created in the background, which can take new jobs.

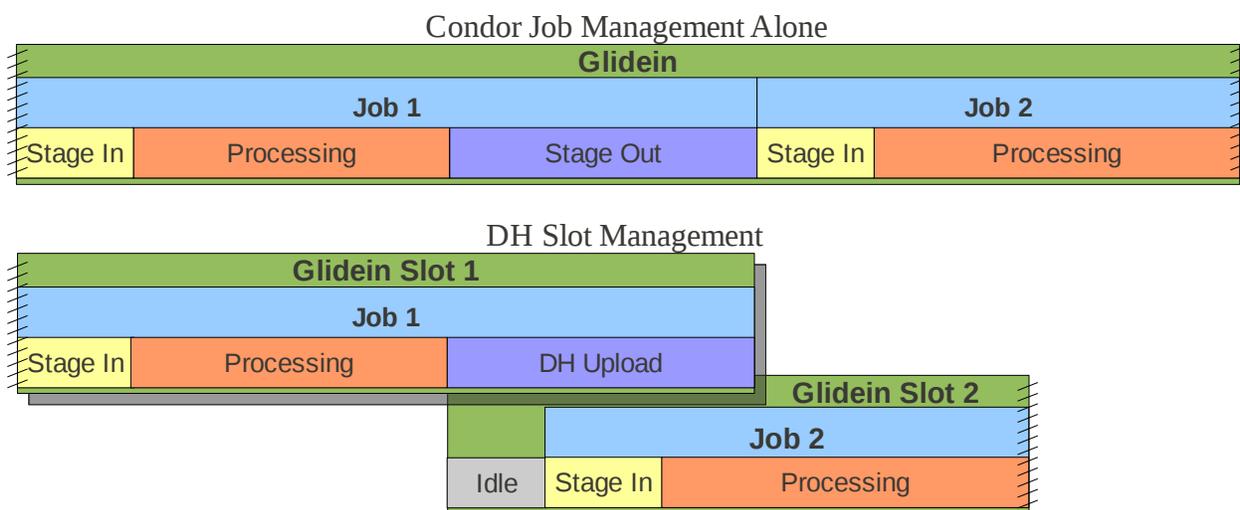


Figure 1: Timeline showing that Data Handling (DH) reduces CPU idle time by creating another glidein. (Phases not to scale)

The overlapping of jobs causes upload and download to happen simultaneously, as seen in Figure 1. Upload and download speeds do not compete locally due to network duplexing. The network usage for simultaneous upload and download should be less than the maximum throughput of the hard disk system on the worker node. Overall bandwidth usage for the entire grid will not increase more than the throughput increase due to slot management. So, as long as the network connection of the grid to the destinations is not at capacity, simultaneous network usage will improve throughput.

Ideally, resource-intensive phases should be adjacent, allowing the full resources of the system to be used continuously. Data Handling (DH) slot management does not replace the download mechanism. One reason Data Handling does not support download methods is because file selection from remote sources could be unreliable. More importantly, however, a job should not be claimed until it can be executed. The claimed job could be used by an idle worker node, and the claimed job would have to wait for the current job to finish executing, which may take a long time. Even after the waiting, the claimed job may never execute since the glidein could expire after the current job is done.

## 2.2 Transfer Resilience

While a new slot is being created and a new job is running, Data Handling transfers the data files through the path tree (Shown in Figure 2), going through each waypoint, which includes intermediate storages and destination storages, and fallback path until all data gets to a destination. [10] Fallback paths may reuse waypoints, or it may follow new ones. For each transfer, a plugin is activated, based on the protocol in the URI, for the next waypoint to be traversed. Users may download their own custom plugins to support new file transfer protocols. At runtime, plugins are selected based on a simple match from the protocol prefix in the URI to the plugin filenames.

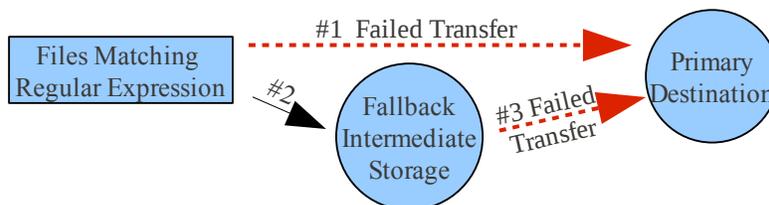


Figure 2: An example path tree that the data would travel through for this path string:  
+WMS\_OUTPUT\_LOCATION="gsiftp://primarydestination/;gsiftp://fallbackintermediate/,gsiftp://primarydestination/"

When a job ends, all files for the job are removed from the worker node by condor. In order to resume the transfer, the returning files must be moved off of the worker node so setting a local intermediate to act as a fallback, as done in Figure 2, allows Data Handling to resume the transfer to the destination when the job is restarted, only using the intermediate in the event of transfer failure.

Many data transfer protocols can be supported. Currently, there are two plugins written. One utilizes globus-url-copy, which can transfer to and from http, https, ftp, file, and gsiftp. The other plugin uses globus, which manages gsiftp transfers. Protocols can be mixed, but only to an extent. The previous waypoint must use a protocol that the current transfer waypoint supports. So, a path could transfer using ftp, then gsiftp, then globus. However, a path cannot transfer to ftp then to globus. Users may also use custom plugins to support new protocols, or to override functionality of default ones. These plugins should be downloaded using condor's "transfer\_input\_files" in the submission description file because if user-added plugins were downloaded as a part of the job, they will not be downloaded if a job is restarted to resume a transfer on another worker node since the download and CPU phase of the job are skipped. So, the custom protocol support plugins must be transferred outside of the job.

If all transfer paths fail to reach a destination, each path will be retried for up to 6 hours with exponential backoff [7], exponentially increasing delay between retry attempts. If the glidein is killed or the transfers were retried for 6 hours, the local data will be deleted by condor and the job will be restarted on another worker node where transfers can be resumed from intermediate storages, if the path specified in the job description travels through intermediates in primary or fallback paths.

One potentially beneficial side effect is that the URI's for any storage that had a transfer failure are advertised in job classads [8], in order to allow the job to be restarted on another node. These classads are all gathered by condor in a central collector. So, administrators can easily monitor these classad attributes to be quickly notified of network or storage problems.

### 3. Testing

All testing was done on a single virtual machine, which ran the factory and frontend, which are components of GlideinWMS. The factory submits glideins to the local machine, so jobs are executed locally as well. The factory submits only one glidein to the local machine.

#### 3.1 Timing of Data Handling (DH) Phases

A Data Handling job starts the same way that a regular job would. It is submitted, negotiations determine if and where the job shall be executed, then the necessary files are downloaded and execution begins.

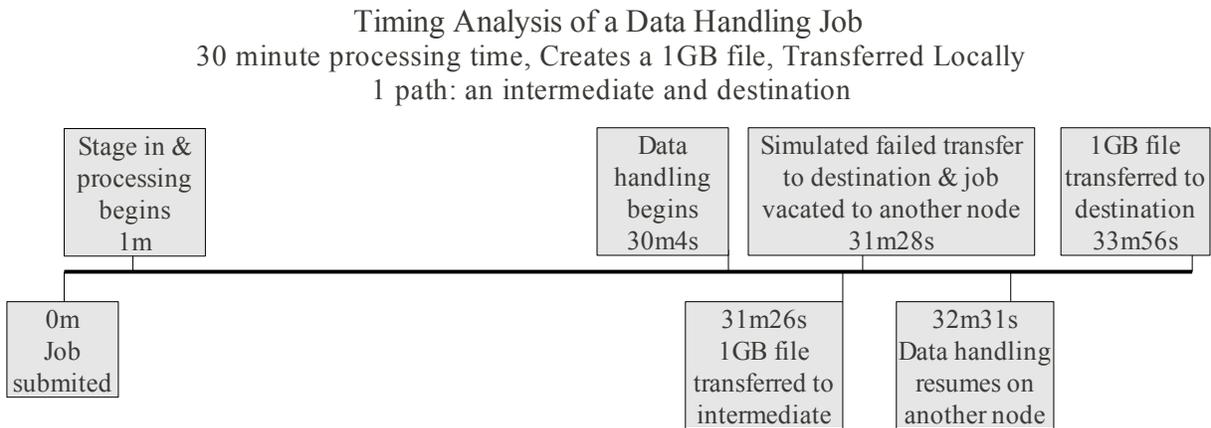
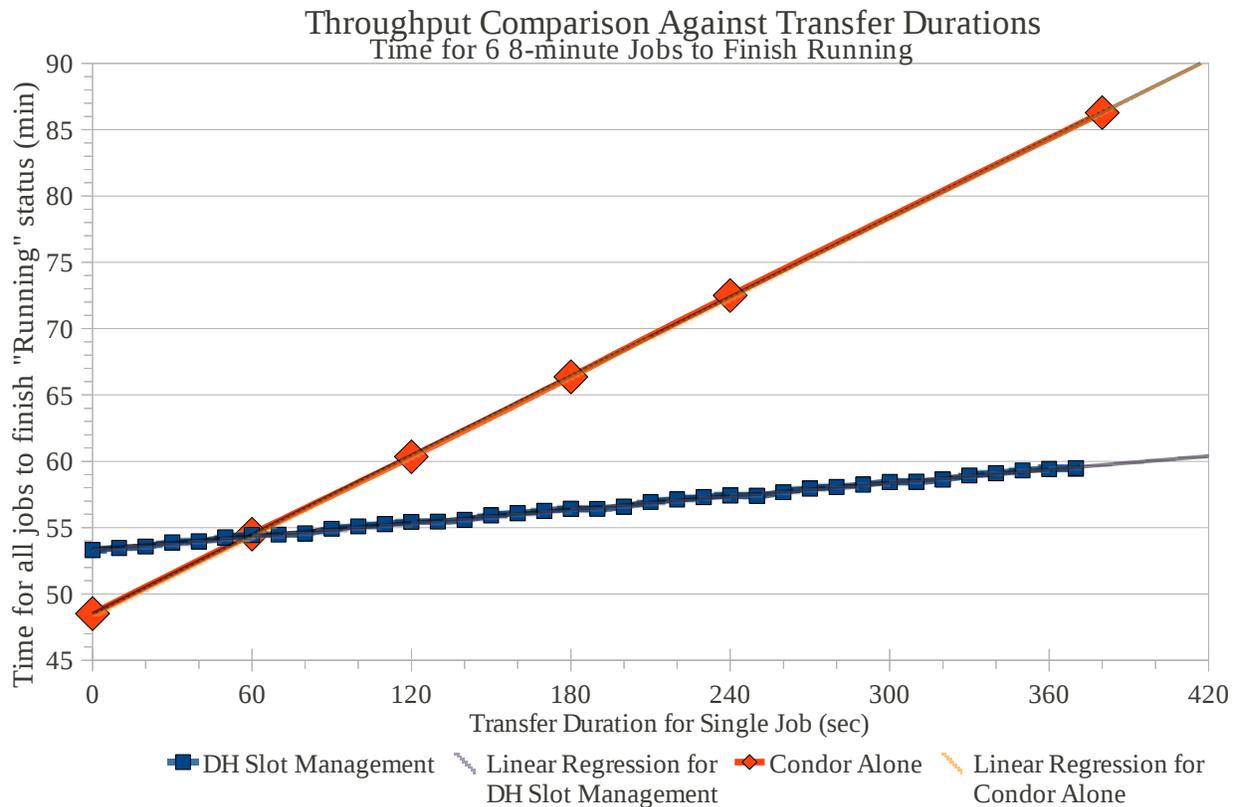


Figure 3: Measured timing of a the actions a Data Handling job takes  
(timeline spacing not to scale)

Once a Data Handling job enters the DH phase, it searches for files, and attempts to transfer them along the first path, falling back to alternate paths. When a failure happens, typically it will retry the transfer for up to 6 hours, but in the above timeline that was disabled. So, it vacated the job to another node, where the transfer resumes. In the above test, what had caused the transfer failure was quickly fixed, and the transfer succeeded on restart.

### 3.2 Slot Performance Testing with no Transfer Failures

In this test, run time is the time for the job to finish, and includes the time to download data to the worker node and the CPU intensive phase, but does not include upload time. New slot idle time is the time for a newly created glidein to be recognized by the collector and assigned a job, which should take about 60 seconds. There are six jobs with eight minute runtimes are submitted in each pass, each job having an eight-minute run time. Each pass increases simulated upload durations for each of the six jobs.



Graph 1: Shows the difference in performance with respect to transfer duration

A pass does not finish until all jobs have finished reporting “Running” status in the condor queue. As a result, the run time for Data Handling (DH) slot management increased by a time of one transfer duration. This is because of the transfer for the last job. Even though another glidein was created, and could accept another job, there were no jobs remaining to accept. So, there is no throughput change for Data Handling jobs if there are no jobs waiting in the queue.

While the run time for DH slot management increases by one upload duration for every pass, the run time for condor alone increases by six upload durations for every pass. However, since the DH slot management system is able to start a new job during the last transfer, the difference in run time per pass is six upload durations, assuming that more jobs are always available and neglecting new glidein idle time.

Since a glidein is ready to receive a job at the start of each pass, there are 5 new slot idle times for each DH slot management pass, instead of six. According to the pass with zero seconds of simulated upload duration, new slot idle time is approximately 60 seconds per slot or five minutes per pass. In cases where transfer time is less than new slot idle time, decreased performance is seen. If transfer durations are constant and are 60 seconds per job, and the transfer speed is 10MBps, then the total size of the transfers per job must be greater than 600MB for a gain in throughput by creating a new slot. DH Slot Management could attempt to guard against these cases, but it is difficult to reliably predict transfer duration because transfer speeds can change. Theoretically, new slot idle time could be reduced by decreasing the communication interval between condor components, though it is unclear that this would be practical due to the extra network traffic and strain on the main collector.

This test shows the importance of Data Handling correctly deciding whether or not to create a new slot. If transfers are typically short, then new slot creation will hurt throughput the majority of the time. Throughput changes should follow the percentages shown in Table 1.

With Data Handling, a new job runs at the same time as the output data of an older job is being transferred. The idle time for the new job to start in another slot does not slow down the data transfer slot, but may take longer than the transfer alone. This slot overlapping model results in an overall higher throughput as compared with the standard serial model, as shown by the following formula:

$$\text{Projected throughput change as a percentage:} \\ \left( \text{Upload Duration} - \text{New Slot idle Time} \right) / \left( \text{Upload Duration} + \text{Run Time} \right)$$

Upload Duration (minutes)	Run Time (hours)	Idle Time (seconds)	Throughput Change (%)
0	1	60	-1.67%
5	1	60	6.15%
10	1	60	12.86%
30	1	60	32.22%
0	3	60	-0.56%
5	3	60	2.16%
10	3	60	4.74%
30	3	60	13.81%
0	6	60	-0.28%
5	6	60	1.10%
10	6	60	2.43%
30	6	60	7.44%
		Maximum	32.22%
		Minimum	-1.67%
		Average	8.93%

Table 1: Examples of projected throughput changes with estimated real-world values

While the maximum number of concurrent upload slots has not been reached, throughput should follow the formula. When the glidein hits the maximum number of data slots allowed, the benefit of overlapping transfers ceases because new slots are not created until the data is transferred. The maximum should not be reached as long as upload duration does not exceed run time, which should not happen unless there is network failure. In the event of network failure,

condor currently deletes the output data and gets a new job. Slot management allows new jobs to run while jobs only transferring data may remain in the worker node until the output is uploaded or the glidein is killed, reducing waste and improving performance during temporary storage downtime. Furthermore, in the event a glidein is killed and the job is moved to another node, Data Handling allows transfers to be resumed by using intermediate storages.

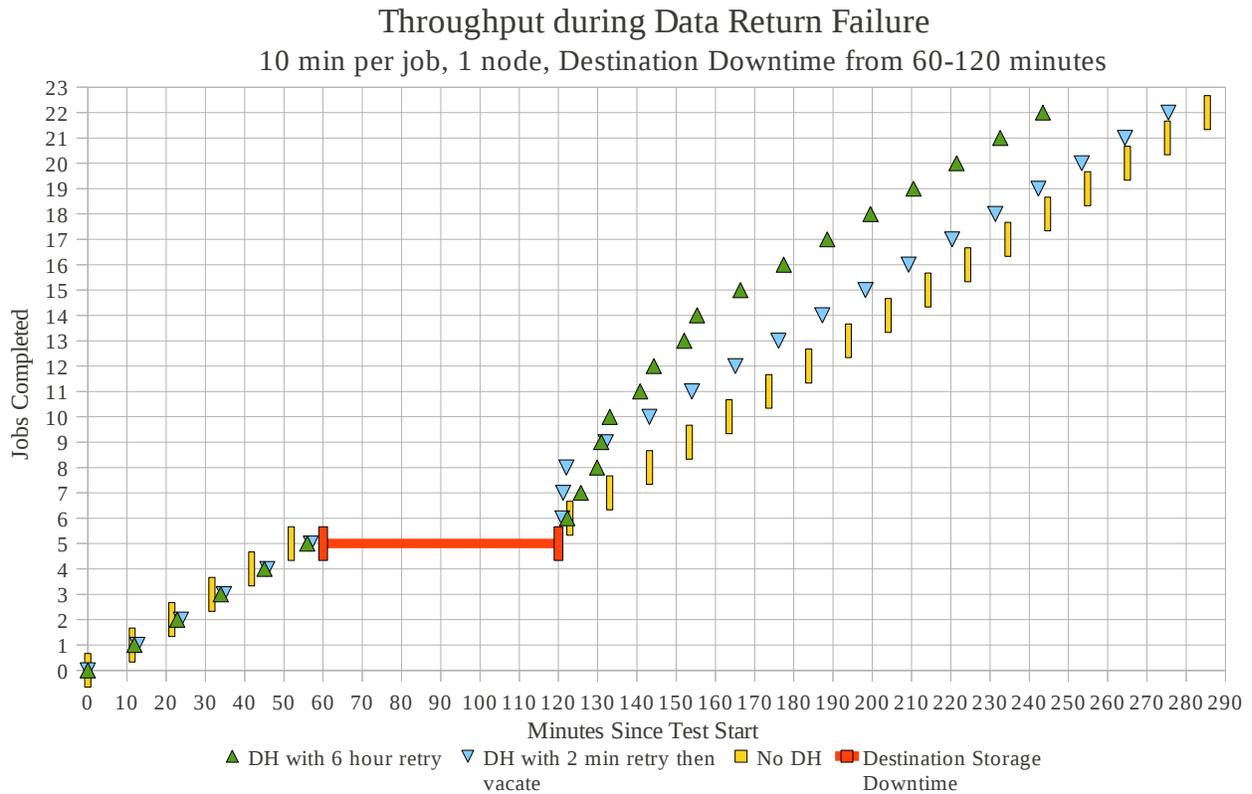
### **3.3 Throughput Change During Connectivity Issues**

Transfers can temporarily fail due to several causes: the network connection for the grid is down, the network connection for the destination is down, the destination storage is full, or the destination server is overloaded. Data Handling can wait for these issues to be resolved and resume the transfer later.

Transfers can permanently fail due to authentication issues, decommissioned servers, improperly configured transfer settings, firewalls, and invalid addresses. Data Handling can recover from these failures by using fallback paths.

In both cases, traditional transfer methods would fail to transfer the data, and the work would be destroyed. Depending on the configuration for checkpoints, when a snapshot of the job is taken as a backup to allow execution to resume on hardware failure, then the job may have to be completely run again, which makes the job run twice as long as it should, and the computing resources that were used have been wasted.

Exactly one hour after the simulation starts, the destination storage cannot receive transfers for another hour, but the intermediate storage is available. The path tree has only one path, which describes that transfers go to an intermediate then to the destination, but the destination may have downtime. Each job has an execution phase of ten minutes, and transfers one 100MB file. Transfers are done locally, but using gridftp [9]. Only the destination storage is failing during the downtime, so worker nodes can still receive more jobs.



Graph 2: Shows the difference in performance in the event of network/storage outage

Since transfers are to the same machine, they are transferred at an unrealistic speed of 100MBps, which means that the file is transferred in a second. So, the new slot idle time, typically 60 seconds, is not exceeded by transfer time in this test, making extra slot creation detrimental to performance on Data Handling jobs, as shown in Graph 1, has an impact in Graph 2, because the algorithm for determining whether a slot needs to be created did not expect the file to be transferred so quickly. In this test, each DH jobs take longer to finish because of new slot idle time, yet the reliability increase by backgrounding transfer reattempts outweigh the problem.

Jobs without Data Handling restart from the beginning if data could not be transferred to a destination. Data Handling retries transfers while allowing new jobs to run, and could vacate the job to another worker node. Depending on the job, the user added data return methods may also have a transfer retry system, ensuring data gets to the destination when it comes back online. However, while these jobs are retrying the transfers, no other jobs can run, since no extra slot is created. So, without slot management, transfer retries decrease performance, sometimes worse than if the job was restarted and the next job does not use the same destination storage.

During the downtime when Data Handling is enabled and jobs are restarted after a brief two-minute retry, only three jobs were completed, while the test with only retries finished six. This is because when a job vacates the slot, the same job goes back into the slot. After the third job is finished, new slots only receive completed jobs. This problem could be fixed by allowing a longer retry period, or by modifying the glidein's preferences to avoid jobs that entered the DH phase and were recently vacated, and to prefer DH jobs that have not been started yet. This can be done by modifying the classad for the glidein.

This reliability test proves that transfer retries, in conjunction with Data Handling slot management, improves throughput for temporary destination transfer failures, but refinement is needed for the slot creation decision algorithm and job negotiation.

#### **4. Further Improvements**

In the future, Data Handling could be enhanced in several ways. The most important improvement is intelligently deciding to create new slots. New slots should only be created when transfer time is expected to take longer than the time for a new job to be received by the new slot. There should be many ways to decide this. Currently, a user-specified data size threshold is used, but this is unreliable because transfer rates can vary drastically. Also, new slots are always created when a transfer fails, but transfers succeed more often than they fail, so it is extremely important that durations are properly predicted at the start of Data Handling. Another potential improvement, as shown in the reliability test, is that glideins should avoid receiving recently vacated jobs that entered DH phase.

Currently, intermediate storage is specified by the user. While a user may choose which site the job should be run on, it may be desirable to allow it to run on more than one site, making it impossible to specify a local storage. Ideally, intermediate storage should be local to the grid site, and accessible over the LAN so transfers are high speed, and do not rely on a wide area internet connection. For end user convenience, more default plugins could be developed to support more transfer methods and protocols like SRM and LCG-CP. Many intermediate storage systems will delete older files, but it may be preferable to have the job delete the files from the intermediate storage when the files have been transferred to a destination, since the intermediate storage is no longer needed. Some distributed computing applications have extremely large input data, and relatively small output. In these situations, the input data may be a directory with many files, and the regular expression that chooses upload files may match unwanted files. To protect storage from being unnecessarily filled, the file selection system could be enhanced to limit the number of files matched, and the overall size.

#### **5. Conclusion**

Current condor data returns impact throughput on longer transfers, sacrifice throughput for reliability, and discards results on transfer failure. Data Handling is designed to improve throughput by allowing more jobs to run during data return and transfer retries, and can save results from being discarded by transferring data to local storage. Data Handling does improve throughput in two situations: when a new slot is created, the transfer duration is longer than the time for the new slot to receive a new job; or, when transfers cannot be completed to a destination temporarily. So, in order to maximize throughput with Data Handling, properly determining when to start a new slot will be vital. The usage of intermediate and fallback storage will improve transfer reliability and grid throughput in the event that the failure will take longer to fix than a glidein's remaining lifespan.

## 6. Acknowledgments

This work was made possible due to a Northern Illinois University (NIU) Research and Development Internship (RDI) at Fermi National Accelerator Laboratories for the Center for Enabling Distributed Petascale Science (CEDPS). Thanks to the GlideinWMS team, Globus.org team, Fermilab Computing Division, and the NIU Computer Science Department.

## 7. References

- [1] Condor  
<http://www.cs.wisc.edu/condor/htc.html>
- [2] GlideinWMS -  
[http://www.uscms.org/SoftwareComputing/Grid/WMS/glideinWMS/tutorials/structural\\_overview/structural\\_overview.pdf](http://www.uscms.org/SoftwareComputing/Grid/WMS/glideinWMS/tutorials/structural_overview/structural_overview.pdf)
- [3] Open Science Grid  
<https://twiki.grid.iu.edu/bin/view/Documentation/UsingTheGrid>
- [4] Globus File Transfer Services  
<http://www.mcs.anl.gov/~childers/quickstart/>
- [5] Instruction Set Pipleining  
[http://en.wikipedia.org/wiki/Instruction\\_pipeline](http://en.wikipedia.org/wiki/Instruction_pipeline)  
[www-csag.ucsd.edu/teaching/cse141-w00/lectures/PipeliningI.pdf](http://www-csag.ucsd.edu/teaching/cse141-w00/lectures/PipeliningI.pdf)
- [6] Center for Enabling Distributed Petascale Science  
<http://www.cedps.net/index.php/Nutshell>
- [7] Exponential Backoff  
[http://en.wikipedia.org/wiki/Exponential\\_backoff](http://en.wikipedia.org/wiki/Exponential_backoff)
- [8] Condor Job Classads  
[http://www.cs.wisc.edu/condor/manual/v6.4/4\\_1Condor\\_s\\_ClassAd.html](http://www.cs.wisc.edu/condor/manual/v6.4/4_1Condor_s_ClassAd.html)  
[http://www.cs.wisc.edu/condor/manual/v6.3/2\\_3Condor\\_Matchmaking.html](http://www.cs.wisc.edu/condor/manual/v6.3/2_3Condor_Matchmaking.html)
- [9] GridFTP (GSIFTP)  
<http://en.wikipedia.org/wiki/GridFTP>  
<http://www.globus.org/alliance/publications/papers/IFIP-2006.pdf>  
<http://www.globus.org/toolkit/docs/4.0/data/gridftp/rn01re01.html>
- [10] Waypoints and End-User Data Handling Functionality  
<http://cd-docdb.fnal.gov/cgi-bin/RetrieveFile?docid=4027&version=1&filename=dh%20usage.pdf>  
<http://cd-docdb.fnal.gov/cgi-bin/RetrieveFile?docid=4027&version=1&filename=dhComparison.pdf>