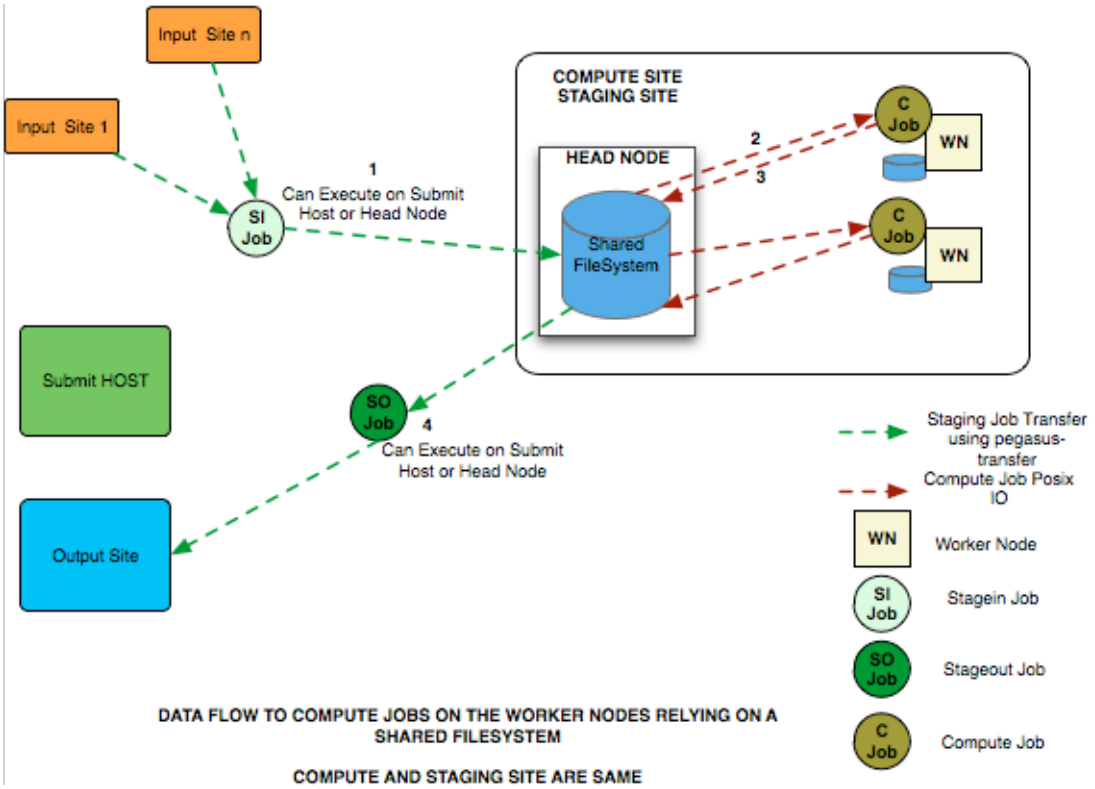
Chapter 1: Getting the Data to Worker Nodes

Last Updated: March 14th, 2011

1 Current Site Layout in Pegasus

In the current version of Pegasus, the worker nodes and the head node of the site share a filesystem. This allows Pegasus to place input and intermediate data files in a shared directory on the filesystem. All the jobs are launched from that directory and have direct POSIX access to the data as illustrated in figure 1. This model works well in the traditional super computing environment where the systems are well maintained by dedicated support staff, and scale well with carefully selected hardware and file system software. For example, TeraGrid sites have demonstrated scalability on huge super computers such as TACC Ranger.

Figure 1



2 Emerging Infrastructure Layouts

In the past few years, the use of clouds for scientific computation has become increasingly common. In the cloud model, the user is expected to setup and maintain a large part of the infrastructure by himself. For example, shared filesystems like the ones in the traditional super computing environments are not provided by the cloud provider, but can be installed and configured by the users. One solution to provide a shared fileystem is to setup a virtual cluster in the cloud. The cloud paradigm also allows for on-demand provisioning and de-provisioning of resource. Going down the virtal cluster route makes it harder to leverage the dynamic resizing of the compute resources. The reason being, it is non-trivial to add and remove nodes from a shared filesystem setup.

Similarly, the infrastructure provided by Open Science Grid is moving away from a shared filesystem model. It is common on OSG to use a provisioner to create compute resource overlays across many OSG sites, and use late binding for the jobs. The resulting overlays donot have a common shared filesystem, but allows

users to dynamically provision more resources depending on workload demands. Additionally, the OSG provides a dedicated high performance data staging infrastructure, like SRM, for the jobs to utilize at runtime. The data staging infrastructure is not colocated with the computational resources. This is in stark contrast to the current Pegasus model where the the file servers have to be part of the compute site.

The above two infrastructures are common in the sense that they do not fit well with the current Pegasus shared filesystem model.

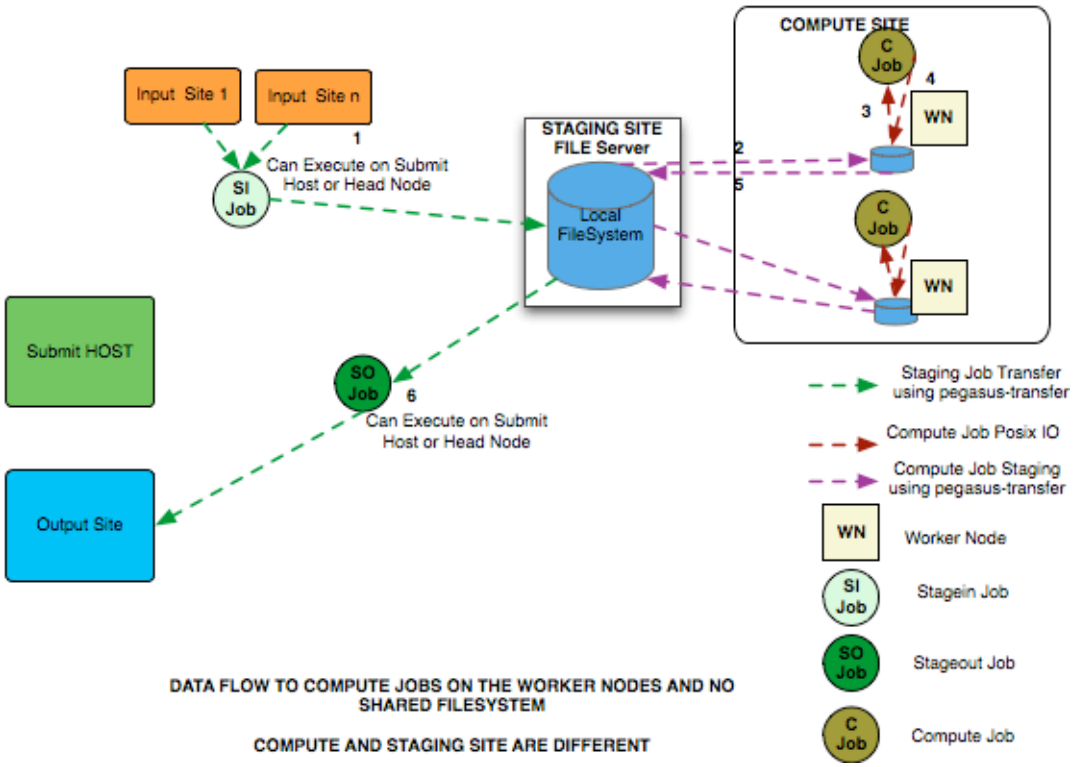
3 Design Overview

We propose to give Pegasus a more generic and flexible data model, which improves upon the current data model in the following aspects:

1. Support of sites with no shared filesystem
2. Separation of the data staging servers from the compute resources
3. Late binding and run time determination of work directory on the worker nodes

Note that the current shared file system model is not being removed. In fact, it will remain the default data model for the time being. The proposed general site layout is illustrated below:

Figure 2



Making these changes allows Pegasus to be used more easily and effectivly on the infrastructure layouts mentioned in the previous section. For example in the cloud environment users can run on a pure Condor pool consisting of Condor workers installed on the VMs. The data model in this case would be for the workers to pull and push against S3. This solution is much more native to the Cloud environment as compared to setting up a virtual cluster with a shared file system. Additionally, the users will have the option to stage data from and to the submit host using Condor file transfers.

On OSG, the users will able to use dynamic provisioning tools such as glideinWMS and use dedicated SRM servers for data staging to and from nodes spread out across OSG.

The dynamic and more generic data model provided by these changes will provide Pegasus with a better platform to target new environments in the future.

4 Glossary of Definitions

This document lists the changes planned to Pegasus to make it easier for users to run jobs on the worker nodes and pull data directly. Some terms that will be referred to in the document are explained below.

1. Submit Host

   The host from where the workflows are submitted . This is where Pegasus and Condor DAGMan are installed. This is referred to as the **"local"** site in the site catalog .

2. Compute Site

   The site where the jobs mentioned in the DAX are executed. There needs to be an entry in the Site Catalog for every compute site. The compute site is passed to pegasus-plan using **--sites** option

3. Staging Site

   A site to which the separate transfer jobs in the executable workflow (jobs with stage_in , stage_out and stage_inter prefixes that Pegasus adds using the transfer refiners) stage the input data to and the output data from to transfer to the final output site. Currently, the staging site is always the compute site where the jobs execute.

4. Output Site

   The output site is the final storage site where the users want the output data from jobs to go to. The output site is passed to pegasus-plan using the **--output** option. The stageout jobs in the workflow stage the data from the staging site to the final storage site.

5. Input Site

   The site where the input data is stored. The locations of the input data are catalogued in the Replica Catalog, and the pool attribute of the locations gives us the site handle for the input site.

6. Workflow Execution Directory

   This is the directory created by the create dir jobs in the executable workflow on the Staging Site. This is a directory per workflow per staging site. Currently, the Staging site is always the Compute Site.

7. Worker Node Directory

   This is the directory created on the worker nodes per job usually by the job wrapper that launches the job.

5 Configurations Supported Currently

Currently in Pegasus, the Compute Site and Staging Site are co-located i.e to ship data to a compute site user needs to rely on the file server on the headnode of the Compute Site.

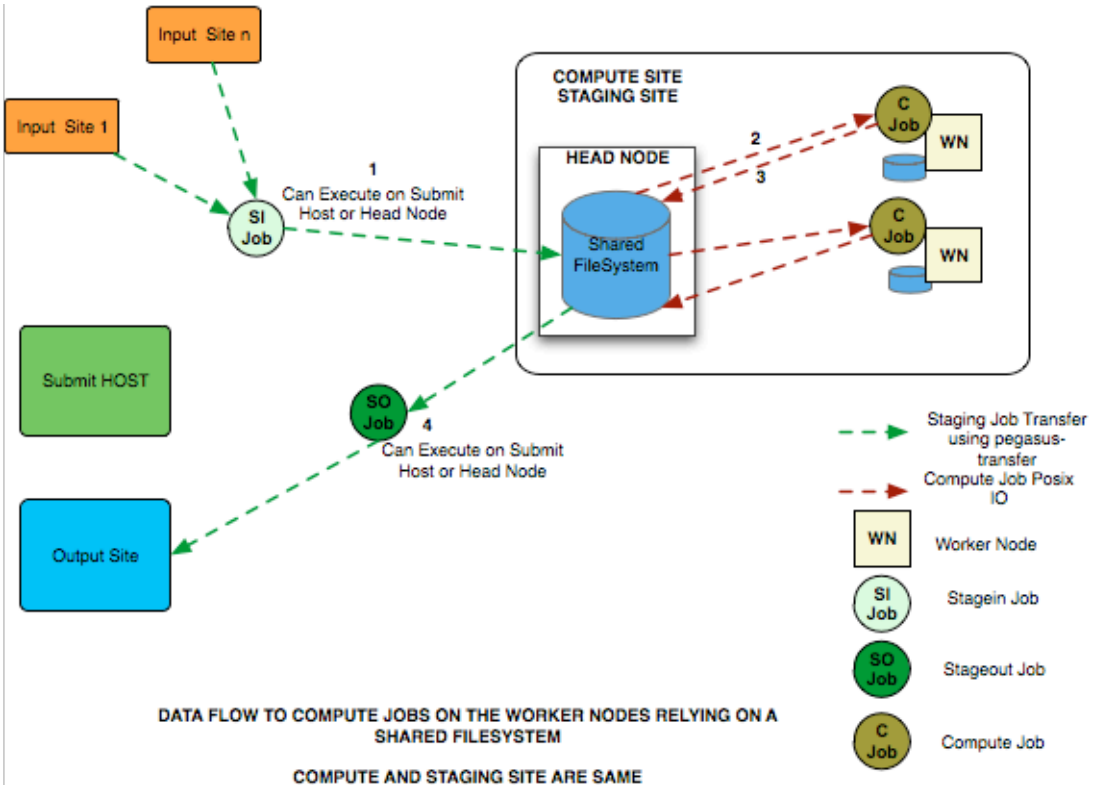
Keeping this in mind, currently Pegasus broadly supports the following three main configurations

1. On the Compute Site, Head Node and the Worker Nodes share a filesystem. (Default Case).
2. On the Compute Site, Head Node and the Worker Nodes don't share a filesystem.
3. Condor Pools Without a Shared Filesystem and Using Condor File Transfers

5.1 Compute Site with the Head Node and Worker Nodes Sharing a Filesystem.

In this case, all the data staging to the shared filesystem happens via the file server on the headnode of the compute site.

Figure 3



The data flow in the above case is as follows

1. Stagein Job executes (either on Submit Host or Head Node) to stage in input data from Input Sites (1---n) to a workflow specific execution directory on the shared filesystem.
2. Compute Job starts on a worker node in the workflow execution directory. Accesses the input data using Posix IO
3. Compute Job executes on the worker node and writes out output data to workflow execution directory using Posix IO
4. Stageout Job executes (either on Submit Host or Head Node) to stage out output data from the workflow specific execution directory to a directory on the final output site.

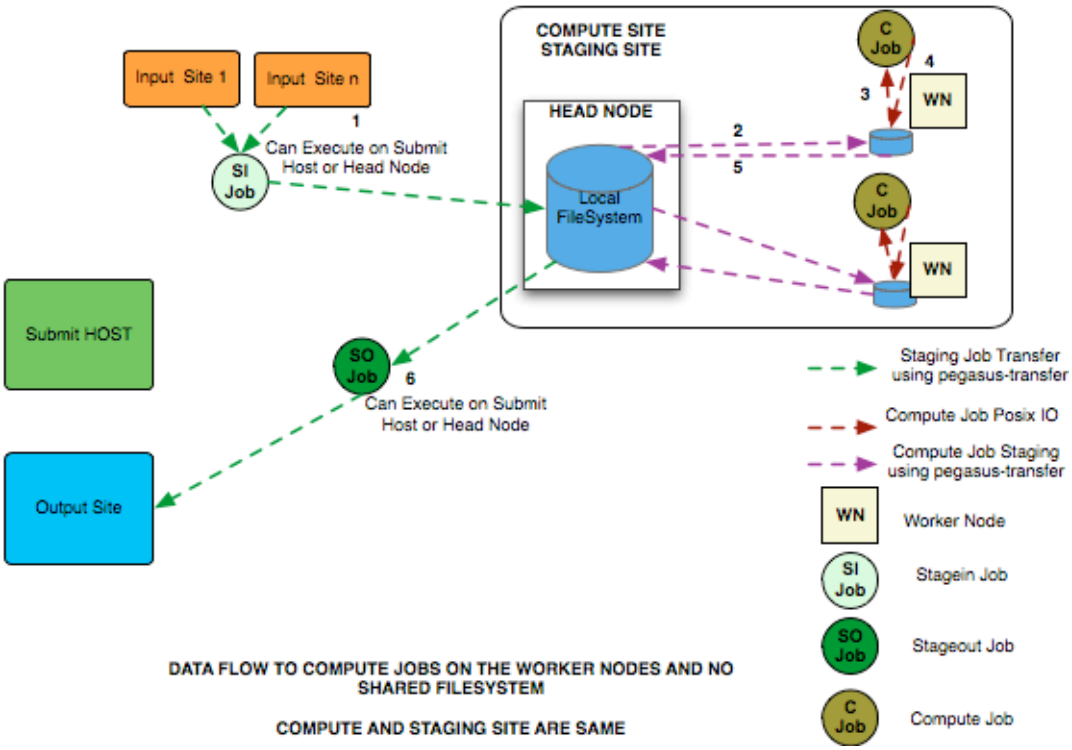
This is the default configuration and allows us to run workflows on

1. Campus Clusters with shared filesystem
2. OSG using shared filesystem on a cluster referred to by OSG_DATA
3. OSG where worker nodes mount a filesystem shared via hadoop fs. The fileserver on the head node is a SRM server
4. Teragrid Sites with shared filesystem like Ranger, NCSA
5. Cloud Setup with a Shared Filesystem

5.2 Compute Site with HeadNode and Worker Nodes NOT Sharing a filesystem

In this case, the data transfer nodes in the executable workflow stage data to the file-server on the headnode of the compute site.

Figure 4



The data flow in the above case is as follows

1. Stagein Job executes (either on Submit Host or Head Node) to stage in input data from Input Sites (1---n) to a workflow specific execution directory on the local filesystem of the headnode
2. Compute Job Wrapper starts on a worker node in the workflow execution directory. As part of it's execution it pulls the input data from the headnode using pegasus-transfer to a directory on the worker node in WN_TMP.
3. Compute Job starts on a worker node in the worker node directory. Accesses the input data using Posix IO
4. Compute Job executes on the worker node and writes out output data to the worker node directory using Posix IO
5. The Compute Job Wrapper detects the compute job has finished and invokes pegasus-transfer to stage out the data from worker node directory to the workflow execution directory on the head node.
6. Stageout Job executes (either on Submit Host or Head Node) to stage out output data from the workflow specific execution directory to a directory on the final output site.

   This configuration allows us to run on

   1. Campus Clusters without a shared filesystem using Condor Glidein's
   2. OSG using Corral WMS and not relying on shared filesystem.
   3. Amazon Cloud where the local filesystem on head node is actually the S3 block storage

5.2.1 Transfer of Proxy and SLS Input Files

In the above scenario, the proxy and SLS Input Files are staged to worker node using Condor File IO.

In a pure Condor environment (Condor Pool or a Condor Pool constructed dynamically using glidein's) the Condor File IO gets the proxy and SLS files to the worker node directory without any troubles.

However, if we are submitting directly to a GRAM server on headnode using CondorG, then even though the jobs execute on the worker nodes, we need a shared filesystem between the head node and worker nodes. This is because in this case, Condor will stage the proxy and sls files only to a filesystem on the

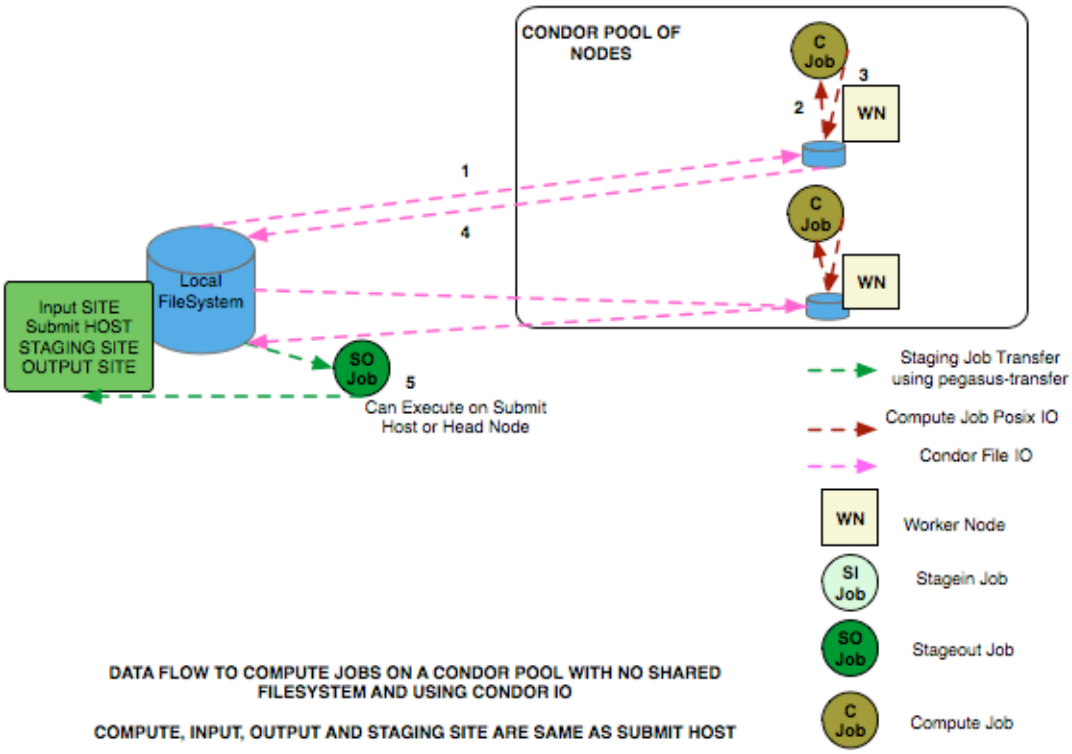
head node, and GRAM does not forward these files to the worker nodes.

5.3 Condor Pools Without a Shared Filesystem and Using Condor File Transfers

This is a special case of non shared filesystem setup where

1. The Submit Host, Input Site, Staging Site and Output Site are the same.
2. We use Condor File Transfers to Transfer the input data to the worker nodes
3. We dont have access to grid certificates

Figure 5



Note Condor File IO only works if we have the Input Site, Submit Host , Staging Site and Output Site co-located on the same machine

The data flow in the above case is as follows

1. Before a Compute Job starts, we use Condor File IO to transfer the input data directly from the Submit Host using Condor File IO
2. Compute Job starts on a worker node in the worker node directory. Accesses the input data using Posix IO
3. Compute Job executes on the worker node and writes out output data to the worker node directory using Posix IO
4. After the Compute Job finishes, we use Condor File IO to transfer the **output data to the workflow execution directory** on the submit host.
5. Stageout Job executes (on Submit Host in local universe) to stage out output data from the workflow specific execution directory to the output directory on the submit host.

   This configuration allows us to run

   1. SIPHT workflows
   2. Workflows on the Cloud
   3. Workflows on FutureGrid

6 Current Job Wrapper for Running Jobs in Non Shared FS

In order to run on the non shared filesystem setup, we need a job wrapper around the jobs when they start running on the worker node. The wrapper is required to pull input data for the job from the head node file server.

6.1 SeqExec Job Wrapper

For running in the non shared filesystem setup we rely on SeqExec to be the job wrapper for the jobs. The job wrapper is specified by setting the following property

```
pegasus.gridstart       SeqExec
```

In this case, for each compute job Pegasus launches the job using the seqexec executable. The seqexec.in file has the following commands

1. Create a directory on the Worker Node in WNTMP for the job. The value for WNTMP needs to be specified beforehand in the site catalog for the site.
2. Pull the input data and user executables from the workflow execution directory on the head node to the worker node directory.
3. Execute the job
4. Push the output data from the worker node directory on the worker node to the workflow execution directory on the head node.
5. Remove the worker node directory.

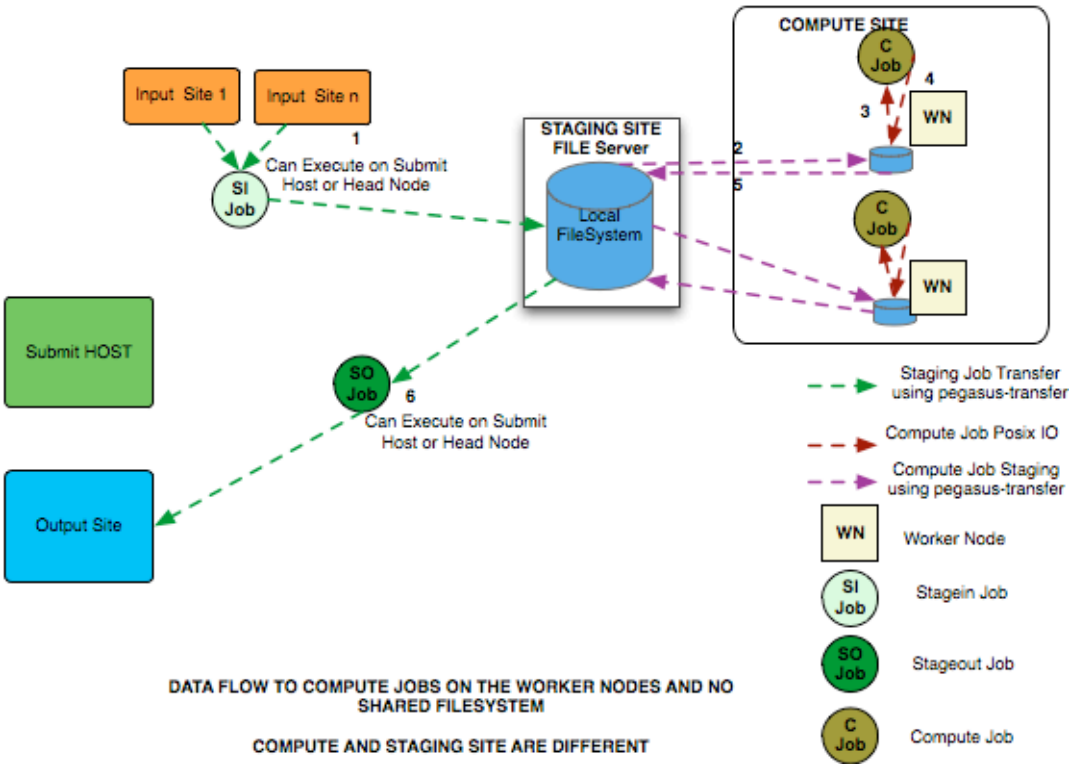
6.2 Deficiencies of Current Wrapper

1. The current SeqExec wrapper invocation is static i.e the commands that need to be invoked via seqexec need to be fully specified at planning time. This means that the directory that job executes in on the worker node needs to be decided at planning time.
2. In case of error there is no way for us to transfer any intermediate files that may be created during the failed execution.
3. We rely on the pegasus worker package to be preinstalled on the worker node or on the shared filesystem. Pegasus , currently gets the right worker package to the workflow execution directory, but not to the worker node. To start a job on worker node we need seqexec and that is part of the pegasus worker package.

7 Changes Planned for 3.1

At a very high level, the main change we want for 3.1 is the ability for the users to specify a staging site for the various compute sites.

Figure 6



7.1 Changes to How Clustered Jobs Are Rendered

Currently, the invocations through kickstart for the constituent jobs happen in the clustering module when the jobs making up a cluster are identified. The clustered job itself is GridStarted (that is assigned a job wrapper) later in the planning process during the Code Generation.

The above separation has created problems when we want a clustered job to run in non shared filesystem setup etc. For 3.1, I think it will help if we move the rendering of the constituent jobs also in the Code Generation Module.

7.1.1 Decided Approach

For 3.1 we have decided to retire the SeqExec executable. Even when we execute clustered jobs in the default case, we will generate a Shell Script that will be transferred to the remote side for the clustered job. The Shell Script will invoke the jobs in the cluster.

7.2 Addition of --staging-sites option to pegasus-plan

For 3.1 we want want to have a command line option (--staging-sites) to pegasus-plan that tells the planner what staging site to use for a particular compute site. The value will be a comma separated list of key=value pairs, where key is the name of the compute site, and value is the staging site to use. An example invocation is listed below

```
pegasus-plan .... --staging-sites nebraska=firefly_srm,osg=firefly_srm,*=local
```

The above means all of the following

1. for nebraska site use the file server on firefly_srm to stage the data .
2. for osg site use the file server of firefly_srm to stage the data
3. for all other compute sites use the file server on the local site to do the transfer.

> **Note** There needs to be an entry in the site catalog for a staging site.

7.2.1 Changes to create dir jobs

The move to a staging site different than a compute site has ramifications on the create dir jobs. The create dir jobs that create the workflow execution directory must now create these directories on the staging site instead of the compute site. However, there is no guarantee that we can actually run a job on a staging site. For example, it is not possible to run jobs on machines that host the SRM servers on OSG. We would need the create dir jobs on the local site and try to create the directory remotely. The best way we can think of is to get pegasus-transfer transfer a file to the remote file server and in the process letting the transfer create directory. This would work well against gridftp/SRM servers. For s3 etc, s3cmd maybe used to create a bucket.

Changes are summarized below

- The create dir job runs locally whenever possible. In cases, where the compute site acts as the staging site, and there is only a file server available, we would need to schedule create dir jobs on the compute site instead locally.
- There will be a new python based **pegasus-dirmanager** that will share code with pegasus-transfer e.g looking at the URL to determine what underlying tool to invoke. It will be passed an argument that specifies the destination directory or bucket that needs to be created.
  - To create directories against a GridFTP Server, **pegasus-dirmanager** will rely on a new java client **pegasus-gridftp-client** that uses the COG api to create a directory.
  - For S3 it will use pegasus-s3.
  - For SRM it will use srm-make-dir
  - In cases, where there is only a file server at a compute site, it will use mkdir to create a directory and the create dir job will be scheduled remotely

Advantage of above approach

Running the create dir job locally has the obvious advantage of not running it remotely. The create dir job runs very fast (barely for a second) and is not a good candidate as is for doing a remote submission. On top of it we know from experience that CondorG and GRAM have trouble tracking very short running jobs.

Open Questions?

1. Is there any case where we would need to create the workflow execution directory both on the compute site and the staging site?

7.2.2 Changes to cleanup jobs

The separate cleanup jobs that are created by Pegasus now will have to delete files from the staging site. This can create problems when the staging site does not support a way to submit jobs to it.

Changes are summarized below

- There will be a new python based client **pegasus-cleanup** that cleanup jobs will use. The cleanup jobs will be scheduled by the planned to execute locally whenever possible. For example, if the compute site acts as a staging site also, and only has a file server the cleanup jobs would need to be executed remotely. pegasus-cleanup will be passed an input file that contains the list of URL's to be deleted.
  - The only client that can be used to delete files remotely from a GridFTP server is uberftp client. However, uberftp client may not be installed on the submit host. In the case, where uberftp client is not available pegasus-cleanup will transfer zero byte files to the workflow execution directory on the staging site. This will result in the existing files being overwritten.
  - For deleting file from S3 it will use pegasus-s3
  - For deleting files from SRM it will use srm-rm
  - In cases, where there is only a file server at a compute site, the cleanup job will use rm to remove the files and the cleanup job will be scheduled remotely.

- Pegasus will also add leaf remove directory jobs that remove the workflow execution directory to the executable workflow. Currently, a separate cleanup workflow is created by Pegasus.

Open Questions?

1. Addition of leaf remove directory jobs cannot be added in case of hierarchal workflows. If the sub workflows are sharing files, then the child sub workflows grab the input files from the workflow execution directories of the parent workflows.

7.3 New Shell Job Wrapper for executing jobs on worker nodes

For 3.1 instead of relying on seqexec to launch the jobs on the worker nodes, we want Pegasus to create a shell script. The shell script when launched on the worker node will do the following Generating a shell script allows us to do discover at runtime what directory to run the compute job in.

1. Figure out and create the directory on the worker node in which to execute the job. It will do it on the existence of certain environment variables. The environment variables will be checked in the following order
    1. $PEGASUS_WN_TMP (if a user specifies the local filesystem for the worker nodes in the site catalog for the site, pegasus will set this env variable in the submit file or if an env profile with KEY PEGASUS_WN_TMP is specified)
    2. $OSG_WN_TMP - osg specific
    3. $OSG_DATA - osg specific
    4. $TG_NODE_SCRATCH - teragrid specific
    5. $TG_CLUSTER_SCRATCH - teragrid specific
    6. $_CONDOR_SCRATCH_DIR - pure condor pool
    7. $TMPDIR
    8. $TMP
    9. .

    Also the script will create a directory only if there is a predefined amount of space available in the remote directory, else would move to next directory in the above list.
2. Pull the worker package from a predefined location determined at planning time, unless PEGASUS_HOME is defined for the remote site in site catalog. Or we could try determining at runtime from PATH env variable.
3. Construct the input URL's (source and destination pairs) for pegasus-transfer to stage in for the job. The source URL's will be specified in the shell script
4. Execute the job
5. Construct the URL's for the output files to transfer back to staging server. The destination URL's will be specified in the shell script. Irrespective of whether job succeeds or fails, the shell script will attempt to transfer the output files back to the workflow execution directory on staging site. This will rely on new option in pegasus-transfer **--continue-on-error** .
6. Cleanup the directory created.

The shell script for the jobs will be transferred using Condor File IO from the submit directory.

7.3.1 Open Questions?

1. How to pull the worker package in case pegasus worker package is not installed/accessible on the worker nodes.
    1. Condor File Transfer - Easy to do. Probably not scalable/efficient
    2. Wget - Pull the worker package from the workflow execution directory on the staging site. The worker package will be placed on the staging sites via pegasus stagein jobs.

However, this relies on wget/curl for remote pulls.

3. In the case there is a shared filesystem amongst worker nodes, we could stage the worker package there and let the shell script copy the worker package to worker node. **In this we would want the create dir job to create workflow execution directory on the compute site also!!! .**

Decided Approach

We have decided that we will transfer worker package using Condor IO only. Transfer of Worker Package is always on, unless user specifies PEGASUS_HOME environment variable in the site catalog for the site. When the pegasus worker package is pre-installed (usually the case in Cloud VM's), then the shell script for the jobs can determine the location on the basis of PEGASUS_HOME environment variable. Relying on PATH could create problems on OSG where an old version of kickstart etc maybe installed by sysadmins and be in the PATH.

2. Do we need to expose a way for the user to change the order in which the environment variables are picked up ?

7.4 Changes to pegasus-transfer

Some changes are required to pegasus-transfer

1. We need pegasus-transfer to continue on error in case of failures. This is to ensure that in case of error, we get as many output files as possible back to the workflow execution directory. This is to make it easier to debug.In the SIPHT environment we have a hard time debugging because of lack of similar support in Condor IO

   This feature will be implemented as transfer queue with retries. First pass, pegasus-transfer will do optimizations such as grouping of transfers and using pipe lining, but in case of failure, the transfers will be split up and executed one by one, maybe using safer but slower options. The default number of attempts will be 2 and it can be changed with the --max-attempts flag.

2. pegasus-transfer will also have to be smart about creating destinations directories. In order for pipelining and s3 to work correctly and not have to many unnecessary create dir command roundtrips, we will have to sort the transfers, and send the create dir once for the group before starting the transfers.
3. Integration with pegasus-s3

   - Currently pegasus-transfer does not support transfer to s3. It needs to interface with the s3 transfer tool (pegasus-s3) that Gideon has checked into the svn.
   - Additionally, there is the S3 token that needs to be transferred . For that Pegasus will transfer the S3 token similar to how we transfer proxies using Condor File I/O. The planner will look at the URL's to determine if the S3 token needs to be transferred. **The rule for determining the S3 token location is as follows**

     1. The S3CFG environment variable
     2. ~/.s3cfg

   pegasus-plan will look for these two during the planning time, and tag the file for transfer via condor IO for the transfer jobs that are doing the transfers against s3.

7.5 Bypassing the transfer of input files to staging site.

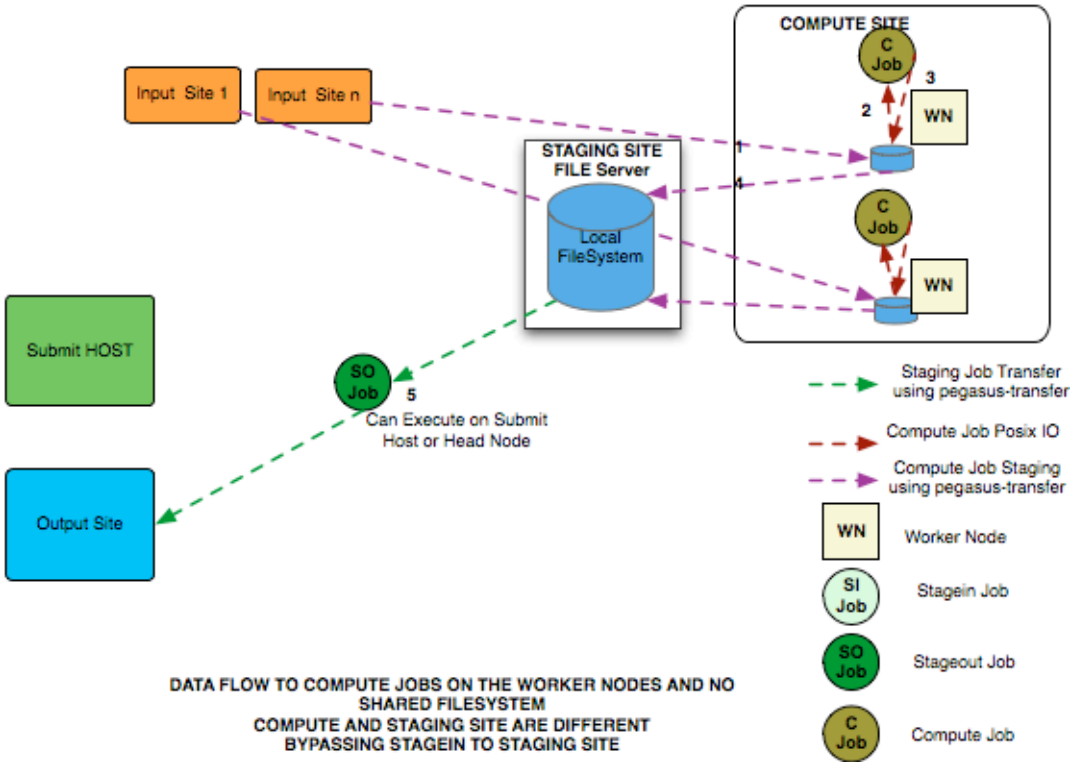
In certain cases, when the jobs are running locally on the worker nodes, the users may want to pull directly from the INPUT SITES, rather than first staging the input data to the STAGING site by separate data movement jobs. This makes sense when

1. The worker nodes have connectivity to the input sites
2. The know that the file servers on the input sites will be able to handle the load. When we stage the

input data to the staging site using data movement jobs we are able to limit the number of transfer jobs accessing the file servers on the input sites using the transfer refiner.

The scenario is illustrated below

Figure 7



The data flow in the above case is as follows

1. Compute Job Wrapper starts on a worker node in the workflow execution directory. As part of it's execution it pulls the input data directly from the input site using pegasus-transfer to a directory on the worker node in WN_TMP.
2. Compute Job starts on a worker node in the worker node directory. Accesses the input data using Posix IO
3. Compute Job executes on the worker node and writes out output data to the worker node directory using Posix IO
4. The Compute Job Wrapper detects the compute job has finished and invokes pegasus-transfer to stage out the data from worker node directory to the workflow execution directory on the staging site.
5. Stageout Job executes (either on Submit Host or Head Node) to stage out output data from the workflow specific execution directory to a directory on the final output site.

7.5.1 Current Support

Currently, there is a hacky way in Pegasus to get Pegasus to bypass the creation of stagein jobs. Pegasus reasons that if an input file has the same pool attribute as the compute site, then in the case of no shared filesystem case it will opt not to create the stagein jobs. To bypass the creation of stagein jobs when retrieving data from a input site different than a compute site, the users currents set the pool attributes for the locations in the replica catalog to the compute site. This is conceptually wrong , as the URL's clearly indicate that the input files are at some other site and can create confusion amongst new users.

As an example, lets say the user is executing jobs on site isi. That input data file f.input is on site usc at the location gsiftp://server.usc.edu/input/f.input.

If the user catalogs the f.input in the Replica Catalog as

```
f.input gsiftp://server.usc.edu/input/f.input pool="usc"
```

, then Pegasus will create a separate data stagein job to stage the data to the head node to site isi.

If the user catalogs the f.input in the Replica Catalog as

```
f.input gsiftp://server.usc.edu/input/f.input pool="isi"
```

, then Pegasus will not create a separate data stagein job to stage the data to the head node to site isi, and the job wrapper will pull the input data directly from server at USC.

7.5.2 Proposed Changes

For 3.1, we will have a simple boolean property to turn on this behavior.

pegasus.transfer.firstlevel.stagein.bypass

7.6 Bypassing transfer of output files to staging site

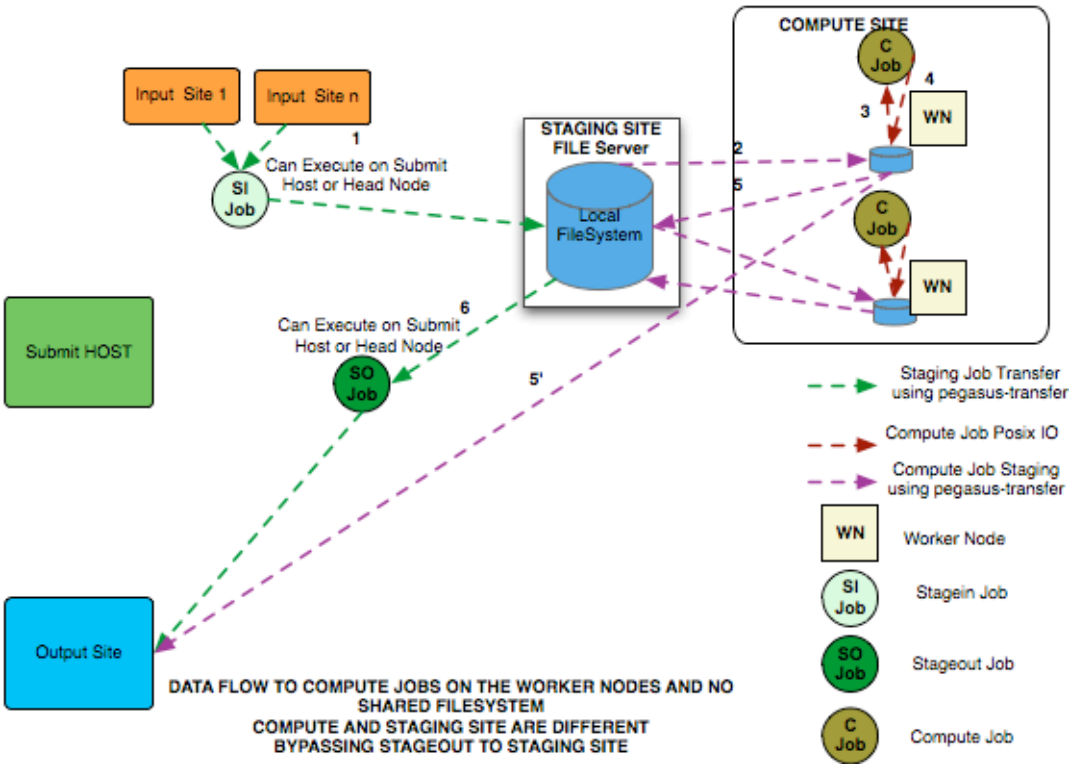
The same way we want to bypass the transfer of input files we may want to bypass the staging of output files to a staging site. Instead, staging them directly to an output site.

However, for this case there are additional things to take care of.

1. Pure output file with transfer flag set to false. This is a file that is generated by a job and is not used as a input file for any job. This file will be transferred directly to the staging site. May help in debugging.
2. Pure output file with transfer flag set to true. This is a file that is generated by a job and is not used as a input file for any job. This file will be transferred directly to the output site.
3. Intermediate Files with transfer flag set to false. This is a file created during workflow execution and is required by other jobs as input. This file will be transferred only to the staging site.
4. Intermediate Files with transfer flag set to true. This is a file created during workflow execution and is required by other jobs as input. This file will be transferred only to the staging site and also directly by the job from the worker node to the output site.

The scenario is illustrated below

Figure 8



The data flow in the above case is as follows

1. Stagein Job executes (either on Submit Host or Head Node) to stage in input data from Input Sites (1---n) to a workflow specific execution directory on the local filesystem of the staging site
2. Compute Job Wrapper starts on a worker node in the workflow execution directory. As part of it's execution it pulls the input data from the headnode using pegasus-transfer to a directory on the worker node in WN_TMP.
3. Compute Job starts on a worker node in the worker node directory. Accesses the input data using Posix IO
4. Compute Job executes on the worker node and writes out output data to the worker node directory using Posix IO
5. The Compute Job Wrapper detects the compute job has finished and invokes pegasus-transfer to stage out the data from worker node directory to
   - to the workflow execution directory on the staging site (step 5 in the diagram). This is for all intermediate output files (so that subsequent jobs can grab it) and pure output file for debugging purposes.
   - directly to the output site for pure output files (step 5' in the diagram). No need to go to staging site, as not required by any subsequent jobs.
6. Stageout Job executes (either on Submit Host or Head Node) to stage out output data from the workflow specific execution directory to a directory on the final output site. This is for intermediate files that have transfer flag set to true.

7.6.1 Proposed Changes

For 3.1, we will have a simple boolean property to turn on this behavior. The above behaviour is not implemented in any form as yet. Will require some work.

pegasus.transfer.firstlevel.stageout.bypass

7.7 Changes to Condor File IO Support

1. Currently to enable Condor IO, a user has to set a specific Condor Transfer Refiner . This means that Bundle Refiner cannot be used and we cannot cluster our stageout jobs.
2. Also the current setup does not create stagein jobs. What we want is that in the current case , Pegasus creates stage-in jobs that execute on the local site and from this directory the jobs pull all the input data and store the intermediate data in.

7.8 Transfer of Braindump file to Workflow Execution Directory

Starting with 3.1, we will always stage-in the braindump file from the submit directory the workflow execution directory. The braindump file has useful metadata in it that allows for us to determine what the submit directory is.

The braindump file will be staged as part of the stage-in jobs not as part of the create dir job as earlier planned.

8 Layout of various scenarios to support in 3.1 and various environments

This section attempts to draw out all the various setups we want to support in 3.1 with the changes discussed above.

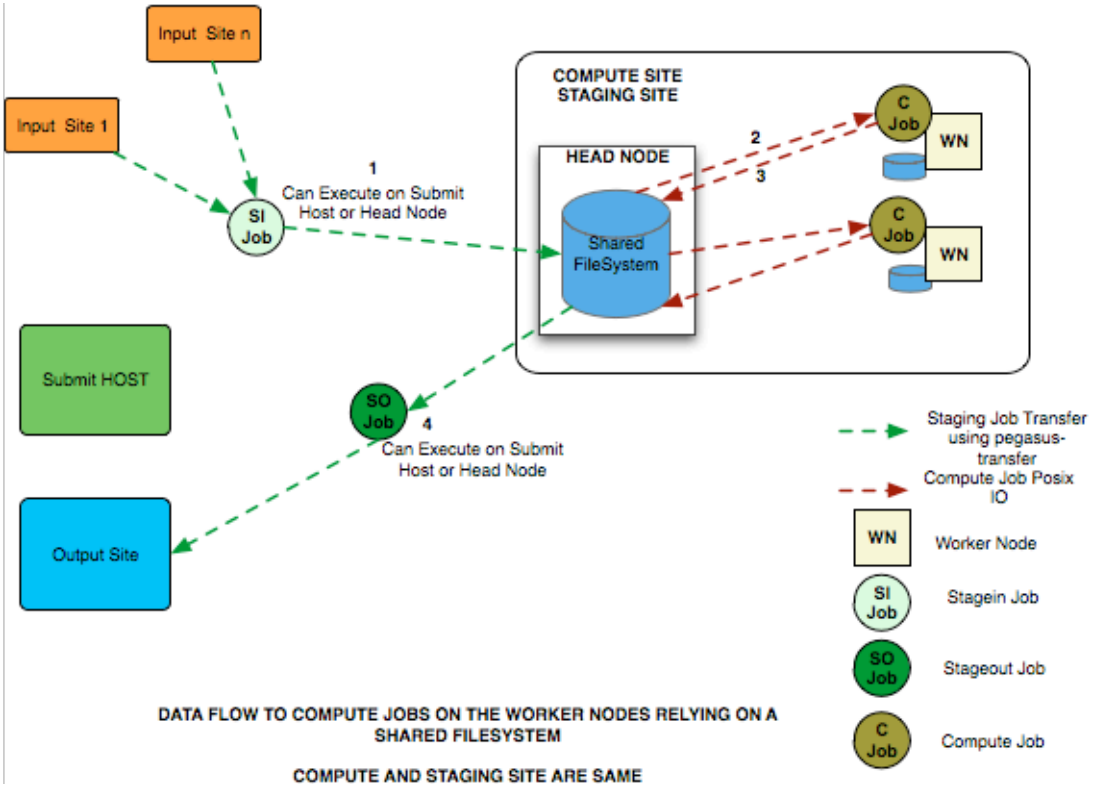
8.1 Compute Site with the Head Node and Worker Nodes Sharing a Filesystem (Default Case)

8.1.1 Data is brought in to the compute site from a remote input site.

Setup

- compute and staging site are the same
- head node and worker nodes share a filesystem
- Input Data is staged from remote sites.
- Remote Output Site i.e site other than compute site. Can be submit host.

Figure 9

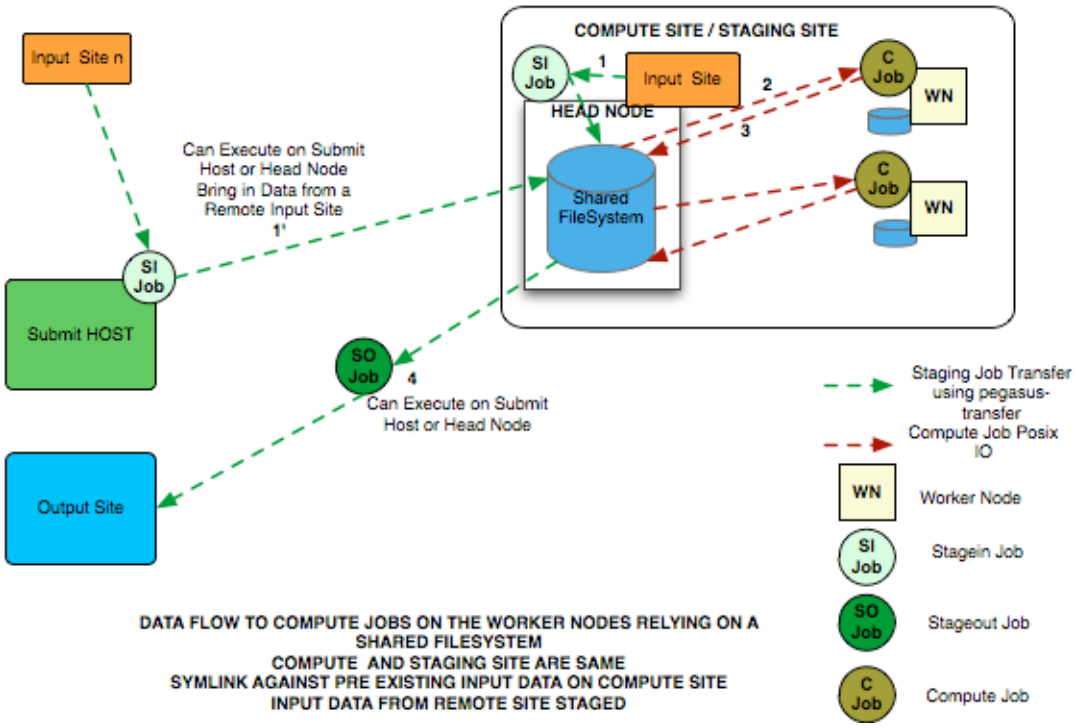


8.1.2 Symlink Against Input Data on Shared Filesystem

Setup

- compute and staging site are the same
- head node and worker nodes share a filesystem
- Some Input Data is already on the Compute Site.
- Remote Output Site i.e site other than compute site. Can be submit host.

Figure 10



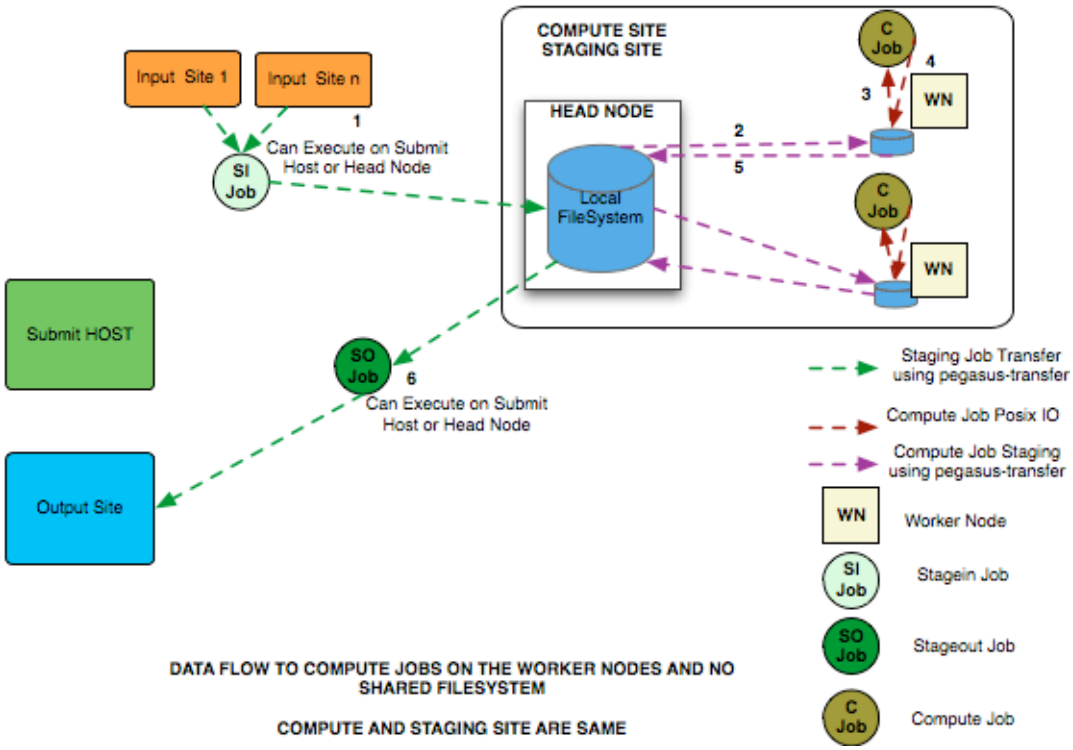
8.2 Compute Site with the Head Node and Worker Nodes Not Sharing a Filesystem

8.2.1 Data is Brought in via the File Server on Head Node

Setup

- compute and staging site are the same
- head node and worker nodes don't share a filesystem
- Input Data is staged from remote sites.
- Remote Output Site i.e site other than compute site. Can be submit host.

Figure 11

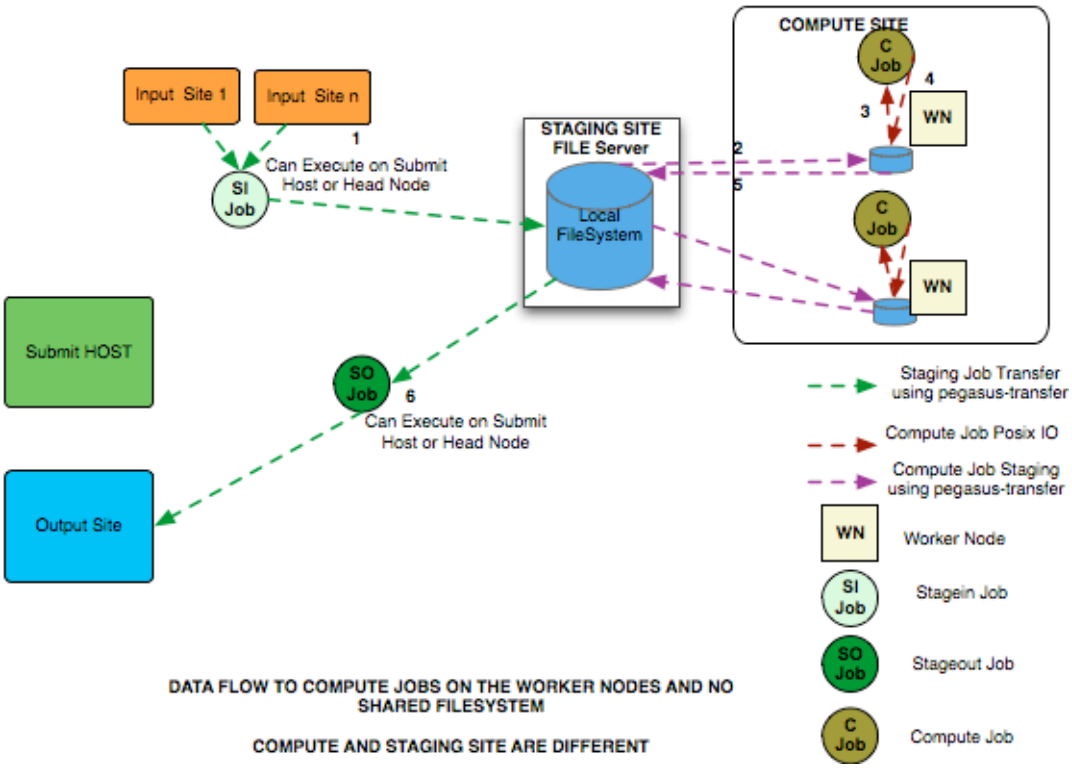


8.2.2 Data is Brought in via the File Server on Staging Site

Setup

- compute and staging site are the different
- head node and worker nodes of compute site don't share a filesystem
- Input Data is staged from remote sites.
- Remote Output Site i.e site other than compute site. Can be submit host.

Figure 12

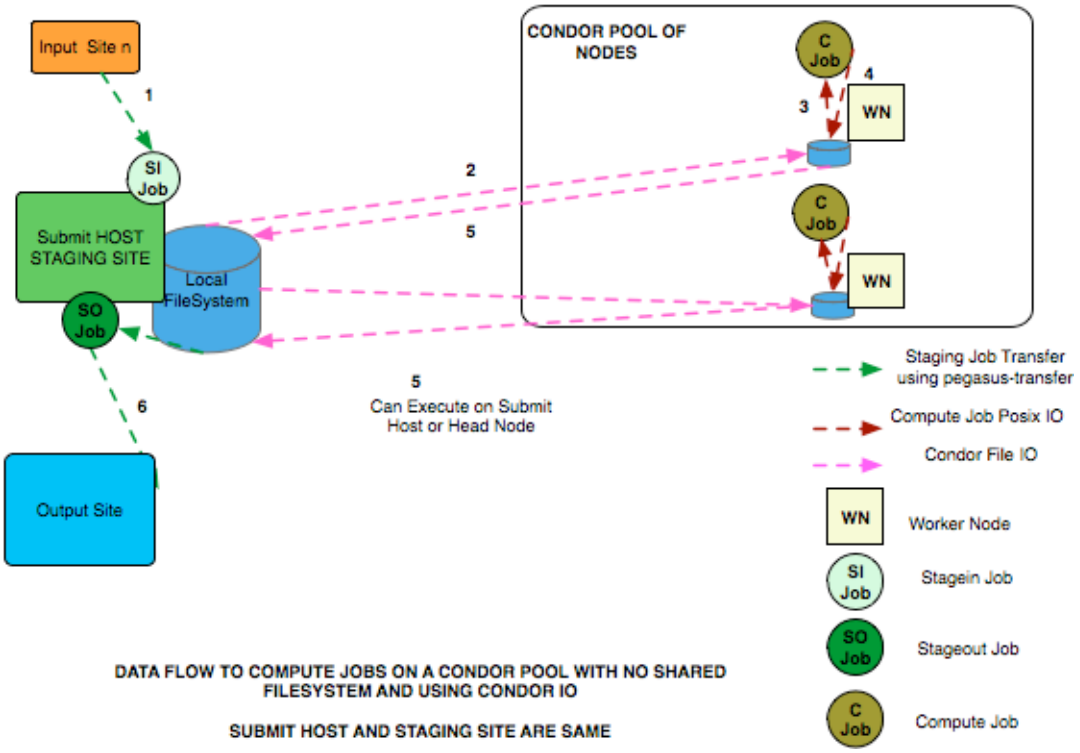


8.2.3 Data is Brought in via the File Server on Submit Host using Condor File IO

Setup

- Submit Host and staging site are same
- head node and worker nodes of compute site don't share a filesystem
- Input Data is staged from remote sites.
- Remote Output Site i.e site other than compute site. Can be submit host.

Figure 13



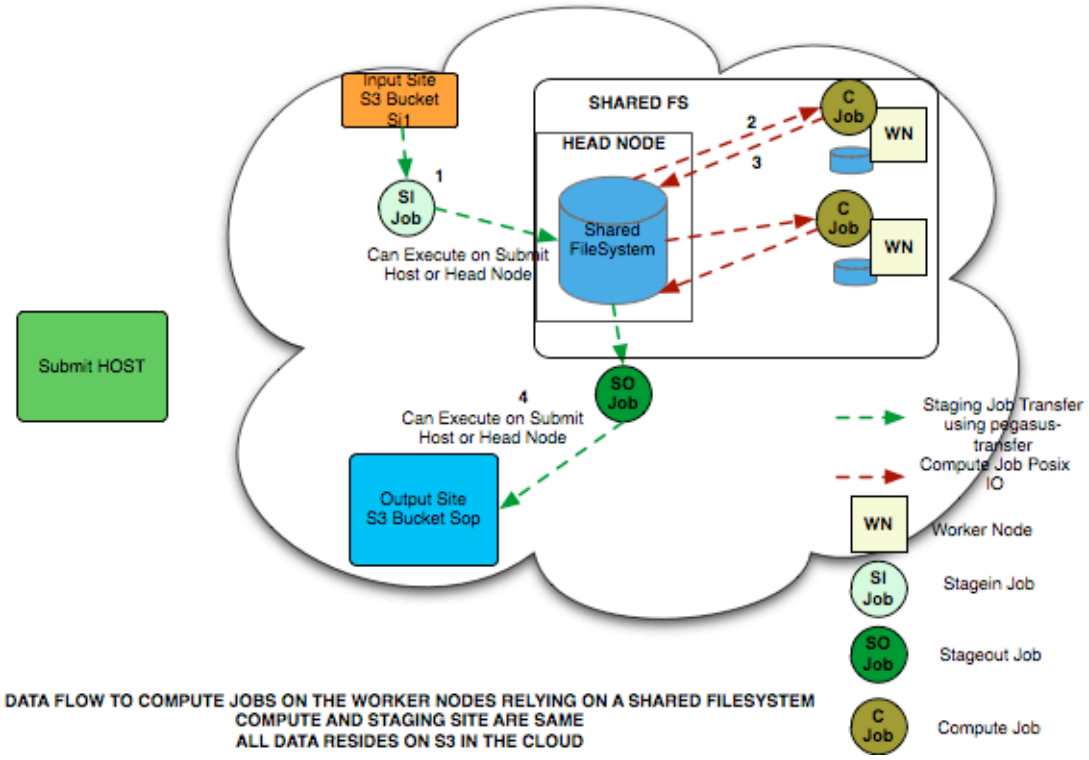
8.3 Use of S3 in the cloud

8.3.1 Shared FileSystem in the Cloud using S3 as Input and Output Sites

Setup

- compute and staging site are the same
- head node and worker nodes share a filesystem
- Input Data is staged from S3 input buckets
- Output is staged back to an output S3 bucket

Figure 14

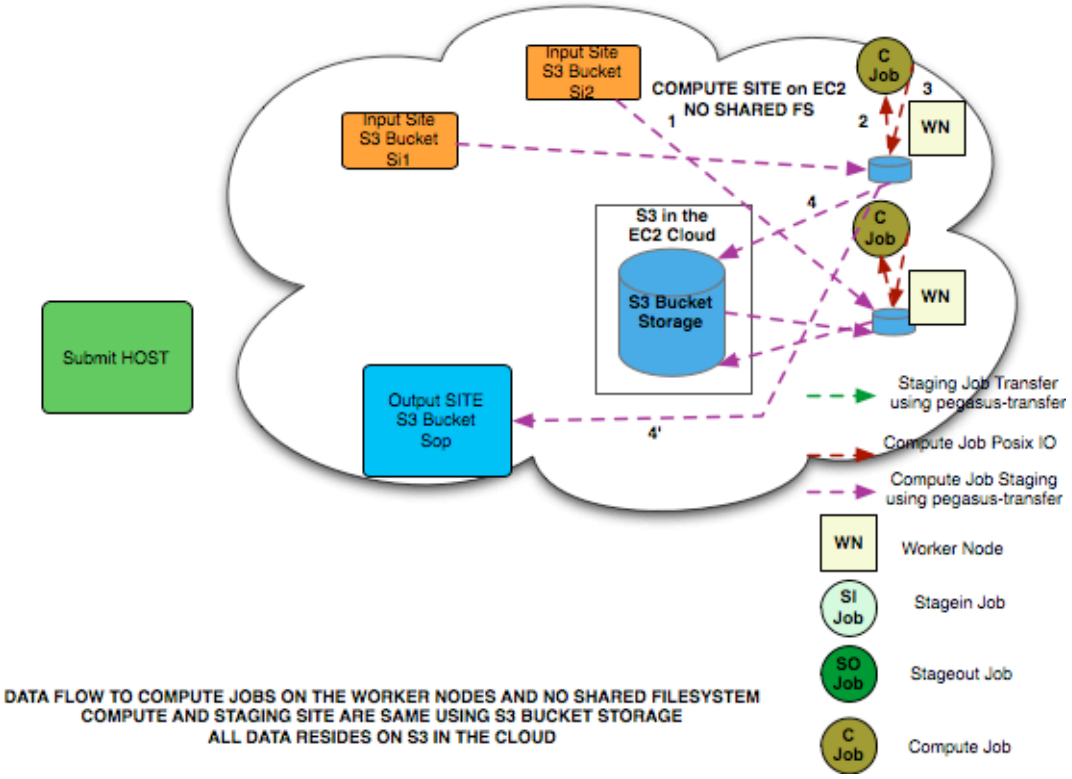


8.3.2 No Shared FileSystem in the Cloud using S3 as Input and Output Sites

Setup (This is used by Brian of UNC)

- compute and staging site are the same
- head node and worker nodes DONT share a filesystem
- Input Data is staged from S3 input buckets (1)
- There is an intermediate S3 bucket in lieu of Workflow Execution Directory
- Intermediate Output Files go to S3 bucket (4) while final output files go directly to output bucket (4')

Figure 15

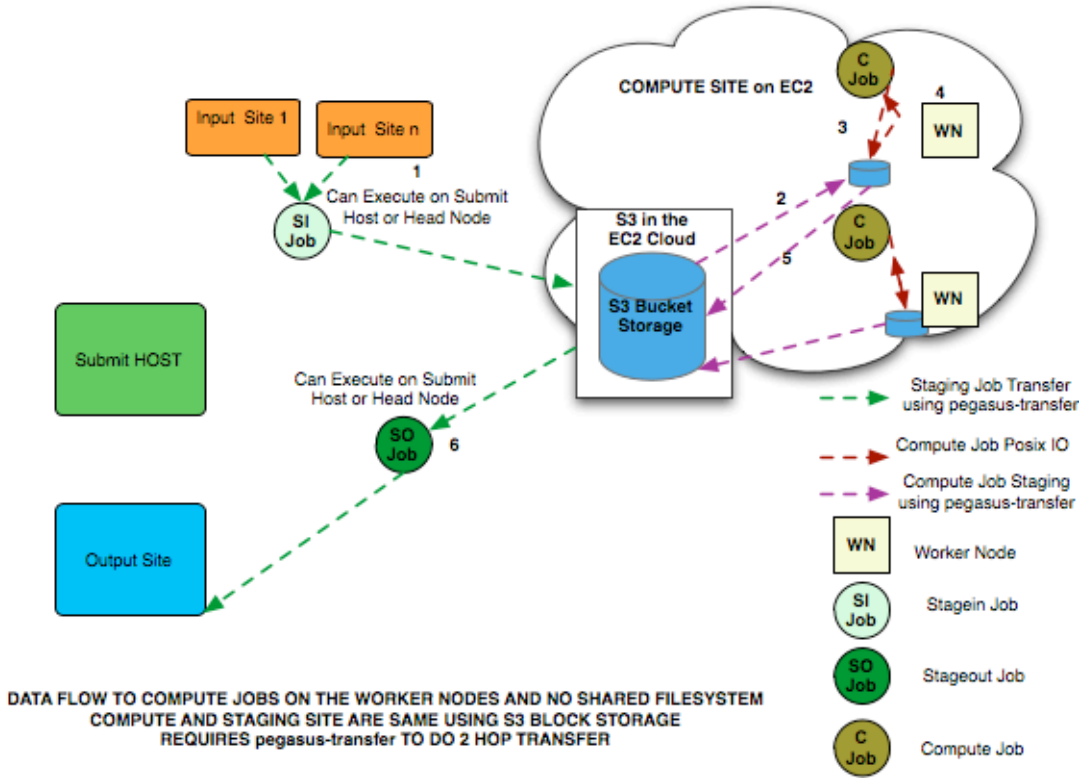


8.3.3 No Shared FileSystem in the Cloud with S3 only used on Staging Site

Setup

- compute and staging site are the same
- head node and worker nodes DONT share a filesystem
- Input Data is staged from sites outside of the cloud
- There is an intermediate S3 bucket in lieu of Workflow Execution Directory

Figure 16



DATA FLOW TO COMPUTE JOBS ON THE WORKER NODES AND NO SHARED FILESYSTEM
COMPUTE AND STAGING SITE ARE SAME USING S3 BLOCK STORAGE
REQUIRES pegasus-transfer TO DO 2 HOP TRANSFER