

Pegasus Advanced Properties

Pegasus Advanced Properties

Table of Contents

1. Advanced Properties	1
pegasus.home	1
Property Files And Locations	1
pegasus.properties	2
pegasus.user.properties	2
Local Directories	2
pegasus.home.datadir	2
pegasus.home.sysconfdir	2
pegasus.home.sharedstatedir	2
pegasus.home.localstatedir	3
pegasus.dir.submit.logs	3
Site Directories	3
pegasus.dir.useTimestamp	3
pegasus.dir.exec	3
pegasus.dir.storage	4
pegasus.dir.storage.deep	4
pegasus.dir.create.strategy	4
pegasus.dir.create.impl	5
Schema File Location Properties	5
pegasus.schema.dax	5
pegasus.schema.sc	5
pegasus.schema.ivr	6
Database Drivers For All Relational Catalogs	6
pegasus.catalog.*.db.driver	6
pegasus.catalog.*.db.url	7
pegasus.catalog.*.db.user	7
pegasus.catalog.*.db.password	7
pegasus.catalog.*.db.*	8
Catalog Properties	9
Replica Catalog	9
Site Catalog	12
Transformation Catalog	13
Provenance Catalog	14
Work Catalog	15
Replica Selection Properties	15
pegasus.selector.replica	15
pegasus.selector.replica.*.ignore.stagein.sites	16
pegasus.selector.replica.*.prefer.stagein.sites	17
pegasus.selector.replica.regex.rank.[value]	17
Site Selection Properties	18
pegasus.selector.site	18
pegasus.selector.site.path	19
pegasus.site.selector.env.*	19
pegasus.selector.site.timeout	20
pegasus.selector.site.keep.tmp	20
Transfer Configuration Properties	20
pegasus.transfer.*.impl	20
pegasus.transfer.refiner	21
pegasus.transfer.sls.*.impl	22
pegasus.transfer.arguments	23
pegasus.transfer.sls.arguments	23
pegasus.transfer.stage.sls.file	24
pegasus.transfer.worker.package	24
pegasus.transfer.links	24
pegasus.transfer.*.remote.sites	25
pegasus.transfer.staging.delimiter	25
pegasus.transfer.disable.chmod.sites	25

pegasus.transfer.proxy	26
pegasus.transfer.setup.source.base.url	26
Gridstart And Exitcode Properties	26
pegasus.gridstart	26
pegasus.gridstart.kickstart.set.xbit	27
pegasus.gridstart.kickstart.stat	27
pegasus.gridstart.generate.lof	28
pegasus.gridstart.invoke.always	28
pegasus.gridstart.invoke.length	28
Interface To Condor And Condor Dagman	28
pegasus.condor.logs.symlink	28
pegasus.condor.arguments.quote	29
pegasus.dagman.nofity	29
pegasus.dagman.verbose	29
pegasus.dagman.[category].maxjobs	30
Monitoring Properties	30
pegasus.monitord.events	30
pegasus.monitord.output	30
Job Clustering Properties	31
pegasus.clusterer.job.aggregator	31
pegasus.clusterer.job.aggregator.seqexec.log	31
pegasus.clusterer.job.aggregator.seqexec.log_global	32
pegasus.clusterer.job.aggregator.seqexec.firstjobfail	32
pegasus.clusterer.label.key	32
Logging Properties	32
pegasus.log.manager	33
pegasus.log.manager.formatter	33
pegasus.log.*	34
pegasus.log.metrics	34
pegasus.log.metrics.file	34
Miscellaneous Properties	34
pegasus.code.generator	34
pegasus.file.cleanup.strategy	35
pegasus.file.cleanup.impl	35
pegasus.file.cleanup.scope	36
pegasus.catalog.transformation.mapper	36
pegasus.selector.transformation	37
pegasus.execute.*.filesystem.local	37
pegasus.parser.dax.preserver.linebreaks	38

Chapter 1. Advanced Properties

This is the reference guide to all properties regarding the Pegasus Workflow Planner, and their respective default values. Please refer to the user guide for a discussion when and which properties to use to configure various components. Please note that the values rely on proper capitalization, unless explicitly noted otherwise.

Some properties rely with their default on the value of other properties. As a notation, the curly braces refer to the value of the named property. For instance, `${pegasus.home}` means that the value depends on the value of the `pegasus.home` property plus any noted additions. You can use this notation to refer to other properties, though the extent of the substitutions are limited. Usually, you want to refer to a set of the standard system properties. Nesting is not allowed. Substitutions will only be done once.

There is a priority to the order of reading and evaluating properties. Usually one does not need to worry about the priorities. However, it is good to know the details of when which property applies, and how one property is able to overwrite another.

1. Property definitions in the system property file, usually found in `${pegasus.home.sysconfdir}/properties`, have the lowest priority. These properties are expected to be set up by the submit host's administrator.
2. The properties defined in the user property file `${user.home}/.pegasusrc` have higher priority. These can overwrite settings found in the system's properties. A set of sensible property values to set on a production system is shown below.
3. Commandline properties have the highest priority. Each commandline property is introduced by a `-D` argument. Note that these arguments are parsed by the shell wrapper, and thus the `-D` arguments must be the first arguments to any command. Commandline properties are useful for debugging purposes.

The following example provides a sensible set of properties to be set by the user property file. These properties use mostly non-default settings. It is an example only, and will not work for you:

<code>pegasus.catalog.replica</code>	File
<code>pegasus.catalog.replica.file</code>	<code>\${pegasus.home}/etc/sample.rc.data</code>
<code>pegasus.catalog.transformation</code>	Text
<code>pegasus.catalog.transformation.file</code>	<code>\${pegasus.home}/etc/sample.tc.text</code>
<code>pegasus.catalog.site</code>	XML3
<code>pegasus.catalog.site.file</code>	<code>\${pegasus.home}/etc/sample.sites.xml3</code>

If you are in doubt which properties are actually visible, `pegasus` during the planning of the workflow dumps all properties after reading and prioritizing in the submit directory in a file with the suffix properties.

pegasus.home

Systems:	all
Type:	directory location string
Default:	"\$PEGASUS_HOME"

The property `pegasus.home` cannot be set in the property file. This property is automatically set up by the `pegasus` clients internally by determining the installation directory of `pegasus`. Knowledge about this property is important for developers who want to invoke PEGASUS JAVA classes without the shell wrappers.

Property Files And Locations

This section describes the property file locations. Please refer to the introduction on issues about precedence of property definitions.

pegasus.properties

Systems:	all
Type:	file location string
Default:	\${pegasus.home.sysconfdir}/properties

The system-wide properties file will be looked for in its default place. It will usually reside in \$PEGASUS_HOME/etc as file named properties.

pegasus.user.properties

Systems:	all
Type:	file location string
Default:	\${user.home}/.pegasusrc

Each user can overwrite the system-wide properties with his or her own definitions. The user properties rely on the system's notion of the user home directory, as reflected in the JRE system properties. In the user's home directory, a file .pegasusrc will be taken to contain user property definitions. Note that \${user.home} is a system property provided by the Java run-time environment (JRE).

Older version of PEGASUS used to support a dot-chimerarc file. For a while, both files are supported. However, in the presence of both files, precedence will be granted to the dot-pegasusrc file.

Local Directories

This section describes the GNU directory structure conventions. GNU distinguishes between architecture independent and thus sharable directories, and directories with data specific to a platform, and thus often local. It also distinguishes between frequently modified data and rarely changing data. These two axis form a space of four distinct directories.

pegasus.home.datadir

Systems:	all
Type:	directory location string
Default:	\${pegasus.home}/share

The datadir directory contains broadly visible and possibly exported configuration files that rarely change. This directory is currently unused.

pegasus.home.sysconfdir

Systems:	all
Type:	directory location string
Default:	\${pegasus.home}/etc

The system configuration directory contains configuration files that are specific to the machine or installation, and that rarely change. This is the directory where the XML schema definition copies are stored, and where the base pool configuration file is stored.

pegasus.home.sharedstatedir

Systems:	all
----------	-----

Type:	directory location string
Default:	\${pegasus.home}/com

Frequently changing files that are broadly visible are stored in the shared state directory. This is currently unused.

pegasus.home.localstatedir

Systems:	all
Type:	directory location string
Default:	\${pegasus.home}/var

Frequently changing files that are specific to a machine and/or installation are stored in the local state directory. This directory is being used for the textual transformation catalog, and the file-based replica catalog.

pegasus.dir.submit.logs

System:	Pegasus
Since:	2.4
Type:	directory location string
Default:	false

By default, Pegasus points the condor logs for the workflow to /tmp directory. This is done to ensure that the logs are created in a local directory even though the submit directory maybe on NFS. In the submit directory the symbolic link to the appropriate log file in the /tmp exists.

However, since /tmp is automatically purged in most cases, users may want to preserve their condor logs in a directory on the local filesystem other than /tmp

Site Directories

The site directory properties modify the behavior of remotely run jobs. In rare occasions, it may also pertain to locally run compute jobs.

pegasus.dir.useTimestamp

System:	Pegasus
Since:	2.1
Type:	Boolean
Default:	false

While creating the submit directory, Pegasus employs a run numbering scheme. Users can use this property to use a timestamp based numbering scheme instead of the runxxxx scheme.

pegasus.dir.exec

System:	Pegasus
Since:	2.0
Type:	remote directory location string
Default:	(no default)

This property modifies the remote location work directory in which all your jobs will run. If the path is relative then it is appended to the work directory (associated with the site), as specified in the site catalog. If the path is absolute then it overrides the work directory specified in the site catalog.

pegasus.dir.storage

System:	Pegasus
Since:	2.0
Type:	remote directory location string
Default:	(no default)

This property modifies the remote storage location on various pools. If the path is relative then it is appended to the storage mount point specified in the pool.config file. If the path is absolute then it overrides the storage mount point specified in the pool config file.

pegasus.dir.storage.deep

System:	Pegasus
Since:	2.1
Type:	Boolean
Default:	false
See Also:	pegasus.dir.storage
See Also:	pegasus.dir.useTimestamp

This property results in the creation of a deep directory structure on the output site, while populating the results. The base directory on the remote end is determined from the site catalog and the property pegasus.dir.storage.

To this base directory, the relative submit directory structure (\$user/\$vogroup/\$label/runxxxx) is appended.

$\$storage = \$base + \$relative_submit_directory$

Depending on the number of files being staged to the remote site a Hashed File Structure is created that ensures that only 256 files reside in one directory.

To create this directory structure on the storage site, Pegasus relies on the directory creation feature of the Grid FTP server, which appeared in globus 4.0.x

pegasus.dir.create.strategy

System:	Pegasus
Since:	2.2
Type:	enumeration
Value[0]:	HourGlass
Value[1]:	Tentacles
Default:	Tentacles

If the

--randomdir

option is given to the Planner at runtime, the Pegasus planner adds nodes that create the random directories at the remote pool sites, before any jobs are actually run. The two modes determine the placement of these nodes and their dependencies to the rest of the graph.

HourGlass	It adds a make directory node at the top level of the graph, and all these concat to a single dummy job before branching out to the root nodes of the original/ concrete dag so far. So we introduce a classic X shape at the top of the graph. Hence the name HourGlass.
Tentacles	This option places the jobs creating directories at the top of the graph. However instead of constricting it to an hour glass shape, this mode links the top node to all the relevant nodes for which the create dir job is necessary. It looks as if the node spreads its tentacles all around. This puts more load on the DAGMan because of the added dependencies but removes the restriction of the plan progressing only when all the create directory jobs have progressed on the remote pools, as is the case in the HourGlass model.

pegasus.dir.create.impl

System:	Pegasus
Since:	2.2
Type:	enumeration
Value[0]:	DefaultImplementation
Value[1]:	S3
Default:	DefaultImplementation

This property is used to select the executable that is used to create the working directory on the compute sites.

DefaultImplementation	The default executable that is used to create a directory is the dirmanager executable shipped with Pegasus. It is found at \$PEGASUS_HOME/bin/dirmanager in the pegasus distribution. An entry for transformation pegasus::dirmanager needs to exist in the Transformation Catalog or the PEGASUS_HOME environment variable should be specified in the site catalog for the sites for this mode to work.
S3	This option is used to create buckets in S3 instead of a directory. This should be set when running workflows on Amazon EC2. This implementation relies on s3cmd command line client to create the bucket. An entry for transformation amazon::s3cmd needs to exist in the Transformation Catalog for this to work.

Schema File Location Properties

This section defines the location of XML schema files that are used to parse the various XML document instances in the PEGASUS. The schema backups in the installed file-system permit PEGASUS operations without being online.

pegasus.schema.dax

Systems:	Pegasus
Since:	2.0
Type:	XML schema file location string
Value[0]:	\${pegasus.home.sysconfdir}/dax-3.2.xsd
Default:	\${pegasus.home.sysconfdir}/dax-3.2.xsd

This file is a copy of the XML schema that describes abstract DAG files that are the result of the abstract planning process, and input into any concrete planning. Providing a copy of the schema enables the parser to use the local copy instead of reaching out to the internet, and obtaining the latest version from the GriPhyN website dynamically.

pegasus.schema.sc

Systems:	Pegasus
----------	---------

Since:	2.0
Type:	XML schema file location string
Value[0]:	\${pegasus.home.sysconfdir}/sc-3.0.xsd
Default:	\${pegasus.home.sysconfdir}/sc-3.0.xsd

This file is a copy of the XML schema that describes the xml description of the site catalog, that is generated as a result of using `genpoolconfig` command. Providing a copy of the schema enables the parser to use the local copy instead of reaching out to the internet, and obtaining the latest version from the GriPhyN website dynamically.

pegasus.schema.ivr

Systems:	all
Type:	XML schema file location string
Value[0]:	\${pegasus.home.sysconfdir}/iv-2.0.xsd
Default:	\${pegasus.home.sysconfdir}/iv-2.0.xsd

This file is a copy of the XML schema that describes invocation record files that are the result of the a grid launch in a remote or local site. Providing a copy of the schema enables the parser to use the local copy instead of reaching out to the internet, and obtaining the latest version from the GriPhyN website dynamically.

Database Drivers For All Relational Catalogs

pegasus.catalog.*.db.driver

System:	Pegasus
Type:	Java class name
Value[0]:	Postgres
Value[1]:	MySQL
Value[2]:	SQLServer2000 (not yet implemented!)
Value[3]:	Oracle (not yet implemented!)
Default:	(no default)
See also:	pegasus.catalog.provenance

The database driver class is dynamically loaded, as required by the schema. Currently, only PostgreSQL 7.3 and MySQL 4.0 are supported. Their respective JDBC3 driver is provided as part and parcel of the PEGASUS.

A user may provide their own implementation, derived from `org.griphyn.vdl.dbdriver.DatabaseDriver`, to talk to a database of their choice.

For each schema in `ptc` and `tc`, a driver is instantiated separately, which has the same prefix as the schema. This may result in multiple connections to the database backend. As fallback, the schema "*" driver is attempted.

The * in the property name can be replaced by a catalog name to apply the property only for that catalog. Valid catalog names are

```

replica
transformation
provenance
work

```

pegasus.catalog.*.db.url

System:	PTC, TC, ...
Type:	JDBC database URI string
Default:	(no default)
Example:	<code>jdbc:postgresql:\${user.name}</code>

Each database has its own string to contact the database on a given host, port, and database. Although most driver URLs allow to pass arbitrary arguments, please use the `pegasus.catalog.[catalog-name].db.*` keys or `pegasus.catalog.*.db.*` to preload these arguments. THE URL IS A MANDATORY PROPERTY FOR ANY DBMS BACKEND.

```
Postgres : jdbc:postgresql:[//hostname[:port]/]database
MySQL    : jdbc:mysql://hostname[:port]/database
SQLServer: jdbc:microsoft:sqlserver://hostname:port
Oracle    : jdbc:oracle:thin:[user/password]@//host[:port]/service
```

The * in the property name can be replaced by a catalog name to apply the property only for that catalog. Valid catalog names are

```
replica
transformation
provenance
work
```

pegasus.catalog.*.db.user

System:	PTC, TC, ...
Type:	string
Default:	(no default)
Example:	<code>\${user.name}</code>

In order to access a database, you must provide the name of your account on the DBMS. This property is database-independent. THIS IS A MANDATORY PROPERTY FOR MANY DBMS BACKENDS.

The * in the property name can be replaced by a catalog name to apply the property only for that catalog. Valid catalog names are

```
replica
transformation
provenance
work
```

pegasus.catalog.*.db.password

System:	PTC, TC, ...
Type:	string
Default:	(no default)
Example:	<code>\${user.name}</code>

In order to access a database, you must provide an optional password of your account on the DBMS. This property is database-independent. THIS IS A MANDATORY PROPERTY, IF YOUR DBMS BACKEND ACCOUNT REQUIRES A PASSWORD.

The * in the property name can be replaced by a catalog name to apply the property only for that catalog. Valid catalog names are

```
replica
transformation
provenance
work
```

pegasus.catalog.*.db.*

System: | PTC, TC, WORK, RC

Each database has a multitude of options to control in fine detail the further behaviour. You may want to check the JDBC3 documentation of the JDBC driver for your database for details. The keys will be passed as part of the connect properties by stripping the "pegasus.catalog.[catalog-name].db." prefix from them. The catalog-name can be replaced by the following values provenance for Provenance Catalog (PTC) transformation for Transformation Catalog (TC) replica for Replica Catalog (RC) work for Workflow Catalog

Postgres 7.3 parses the following properties:

```
pegasus.catalog.*.db.user
pegasus.catalog.*.db.password
pegasus.catalog.*.db.PGHOST
pegasus.catalog.*.db.PGPORT
pegasus.catalog.*.db.charSet
pegasus.catalog.*.db.compatible
```

MySQL 4.0 parses the following properties:

```
pegasus.catalog.*.db.user
pegasus.catalog.*.db.password
pegasus.catalog.*.db.databaseName
pegasus.catalog.*.db.serverName
pegasus.catalog.*.db.portNumber
pegasus.catalog.*.db.socketFactory
pegasus.catalog.*.db.strictUpdates
pegasus.catalog.*.db.ignoreNonTxTables
pegasus.catalog.*.db.secondsBeforeRetryMaster
pegasus.catalog.*.db.queriesBeforeRetryMaster
pegasus.catalog.*.db.allowLoadLocalInfile
pegasus.catalog.*.db.continueBatchOnError
pegasus.catalog.*.db.pedantic
pegasus.catalog.*.db.useStreamLengthsInPrepStmts
pegasus.catalog.*.db.useTimezone
pegasus.catalog.*.db.relaxAutoCommit
pegasus.catalog.*.db.paranoid
pegasus.catalog.*.db.autoReconnect
pegasus.catalog.*.db.capitalizeTypeNames
pegasus.catalog.*.db.ultraDevHack
pegasus.catalog.*.db.strictFloatingPoint
pegasus.catalog.*.db.useSSL
pegasus.catalog.*.db.useCompression
pegasus.catalog.*.db.socketTimeout
pegasus.catalog.*.db.maxReconnects
pegasus.catalog.*.db.initialTimeout
pegasus.catalog.*.db.maxRows
pegasus.catalog.*.db.useHostsInPrivileges
pegasus.catalog.*.db.interactiveClient
pegasus.catalog.*.db.useUnicode
pegasus.catalog.*.db.characterEncoding
```

MS SQL Server 2000 support the following properties (keys are case-insensitive, e.g. both "user" and "User" are valid):

```
pegasus.catalog.*.db.User
pegasus.catalog.*.db.Password
pegasus.catalog.*.db.DatabaseName
pegasus.catalog.*.db.ServerName
pegasus.catalog.*.db.HostProcess
pegasus.catalog.*.db.NetAddress
pegasus.catalog.*.db.PortNumber
pegasus.catalog.*.db.ProgramName
```

```
pegasus.catalog.*.db.SendStringParametersAsUnicode
pegasus.catalog.*.db.SelectMethod
```

The * in the property name can be replaced by a catalog name to apply the property only for that catalog. Valid catalog names are

```
replica
transformation
provenance
work
```

Catalog Properties

Replica Catalog

pegasus.catalog.replica

System:	Pegasus
Since:	2.0
Type:	enumeration
Value[0]:	RLS
Value[1]:	LRC
Value[2]:	JDBCRC
Value[3]:	File
Value[4]:	MRC
Default:	RLS

Pegasus queries a Replica Catalog to discover the physical filenames (PFN) for input files specified in the DAX. Pegasus can interface with various types of Replica Catalogs. This property specifies which type of Replica Catalog to use during the planning process.

RLS RLS (Replica Location Service) is a distributed replica catalog, which ships with GT4. There is an index service called Replica Location Index (RLI) to which 1 or more Local Replica Catalog (LRC) report. Each LRC can contain all or a subset of mappings. In this mode, Pegasus queries the central RLI to discover in which LRC's the mappings for a LFN reside. It then queries the individual LRC's for the PFN's. To use RLS, the user additionally needs to set the property `pegasus.catalog.replica.url` to specify the URL for the RLI to query. Details about RLS can be found at <http://www.globus.org/toolkit/data/rls/>

LRC If the user does not want to query the RLI, but directly a single Local Replica Catalog. To use LRC, the user additionally needs to set the property `pegasus.catalog.replica.url` to specify the URL for the LRC to query. Details about RLS can be found at <http://www.globus.org/toolkit/data/rls/>

JDBCRC In this mode, Pegasus queries a SQL based replica catalog that is accessed via JDBC. The sql schema's for this catalog can be found at `$PEGASUS_HOME/sql` directory. To use JDBCRC, the user additionally needs to set the following properties

1. `pegasus.catalog.replica.db.url`
2. `pegasus.catalog.replica.db.user`
3. `pegasus.catalog.replica.db.password`

File In this mode, Pegasus queries a file based replica catalog. It is neither transactionally safe, nor advised to use for production purposes in any way. Multiple concurrent instances *will clobber* each other!. The site attribute should be specified whenever possible. The attribute key for the site attribute is "pool".

The LFN may or may not be quoted. If it contains linear whitespace, quotes, backslash or an equality sign, it must be quoted and escaped. Ditto for the PFN. The attribute key-value pairs are separated by an equality sign without any whitespaces. The value may be in quoted. The LFN sentiments about quoting apply.

```
LFN PFN
LFN PFN a=b [...]
LFN PFN a="b" [...]
"LFN w/LWS" "PFN w/LWS" [...]
```

To use File, the user additionally needs to specify `pegasus.catalog.replica.file` property to specify the path to the file based RC.

MRC In this mode, Pegasus queries multiple replica catalogs to discover the file locations on the grid. To use it set

```
pegasus.catalog.replica MRC
```

Each associated replica catalog can be configured via properties as follows.

The user associates a variable name referred to as [value] for each of the catalogs, where [value] is any legal identifier (concretely [A-Za-z][_A-Za-z0-9]*) For each associated replica catalogs the user specifies the following properties.

```
pegasus.catalog.replica.mrc.[value]      specifies the type of replica catalog.
pegasus.catalog.replica.mrc.[value].key  specifies a property name key for a
particular catalog
```

For example, if a user wants to query two lrc's at the same time he/she can specify as follows

```
pegasus.catalog.replica.mrc.lrc1 LRC
pegasus.catalog.replica.mrc.lrc2.url rls://sukhna
pegasus.catalog.replica.mrc.lrc2 LRC
pegasus.catalog.replica.mrc.lrc2.url rls://smarty
```

In the above example, lrc1, lrc2 are any valid identifier names and url is the property key that needed to be specified.

pegasus.catalog.replica.url

System:	Pegasus
Since:	2.0
Type:	URI string
Default:	(no default)

When using the modern RLS replica catalog, the URI to the Replica catalog must be provided to Pegasus to enable it to look up filenames. There is no default.

pegasus.catalog.replica.chunk.size

System:	Pegasus, rc-client
Since:	2.0
Type:	Integer
Default:	1000

The rc-client takes in an input file containing the mappings upon which to work. This property determines, the number of lines that are read in at a time, and worked upon at together. This allows the various operations like insert, delete happen in bulk if the underlying replica implementation supports it.

pegasus.catalog.replica.lrc.ignore

System:	Replica Catalog - RLS
Since:	2.0
Type:	comma separated list of LRC urls
Default:	(no default)
See also:	pegasus.catalog.replica.lrc.restrict

Certain users may like to skip some LRCs while querying for the physical locations of a file. If some LRCs need to be skipped from those found in the rli then use this property. You can define either the full URL or partial domain names that need to be skipped. E.g. If a user wants `rls://smarty.isi.edu` and all LRCs on `usc.edu` to be skipped then the property will be set as `pegasus.rls.lrc.ignore=rls://smarty.isi.edu,usc.edu`

pegasus.catalog.replica.lrc.restrict

System:	Replica Catalog - RLS
Since:	1.3.9
Type:	comma separated list of LRC urls
Default:	(no default)
See also:	pegasus.catalog.replica.lrc.ignore

This property applies a tighter restriction on the results returned from the LRCs specified. Only those PFNs are returned that have a pool attribute associated with them. The property "pegasus.rc.lrc.ignore" has a higher priority than "pegasus.rc.lrc.restrict". For example, in case a LRC is specified in both properties, the LRC would be ignored (i.e. not queried at all instead of applying a tighter restriction on the results returned).

pegasus.catalog.replica.lrc.site.[site-name]

System:	Replica Catalog - RLS
Since:	2.3.0
Type:	LRC url
Default:	(no default)

This property allows for the LRC url to be associated with site handles. Usually, a pool attribute is required to be associated with the PFN for Pegasus to figure out the site on which PFN resides. However, in the case where an LRC is responsible for only a single site's mappings, Pegasus can safely associate LRC url with the site. This association can be used to determine the pool attribute for all mappings returned from the LRC, if the mapping does not have a pool attribute associated with it.

The `site_name` in the property should be replaced by the name of the site. For example

```
pegasus.catalog.replica.lrc.site.isi rls://lrc.isi.edu
```

tells Pegasus that all PFNs returned from LRC `rls://lrc.isi.edu` are associated with site `isi`.

The `[site_name]` should be the same as the site handle specified in the site catalog.

pegasus.catalog.replica.cache.asrc

System:	Pegasus
Since:	2.0

Type:	Boolean
Value[0]:	false
Value[1]:	true
Default:	false
See also:	pegasus.catalog.replica

This property determines whether to treat the cache file specified as a supplemental replica catalog or not. User can specify on the command line to pegasus-plan a comma separated list of cache files using the --cache option. By default, the LFN->PFN mappings contained in the cache file are treated as cache, i.e if an entry is found in a cache file the replica catalog is not queried. This results in only the entry specified in the cache file to be available for replica selection.

Setting this property to true, results in the cache files to be treated as supplemental replica catalogs. This results in the mappings found in the replica catalog (as specified by pegasus.catalog.replica) to be merged with the ones found in the cache files. Thus, mappings for a particular LFN found in both the cache and the replica catalog are available for replica selection.

Site Catalog

pegasus.catalog.site

System:	Site Catalog
Since:	2.0
Type:	enumeration
Value[0]:	XML3
Value[1]:	XML
Value[2]:	Text
Default:	XML3

The site catalog file is available in three major flavors: The Text and and XML formats for the site catalog are deprecated. Users can use pegasus-sc-converter client to convert their site catalog to the newer XML3 format.

1. The "XML" format is an XML-based file. The XML format reads site catalog conforming to the old site catalog schema available at <http://pegasus.isi.edu/wms/docs/schemas/sc-2.0/sc-2.0.xsd>
2. The "XML3" format is an XML-based file. The XML format reads site catalog conforming to the old site catalog schema available at <http://pegasus.isi.edu/wms/docs/schemas/sc-3.0/sc-3.0.xsd>
3. The "Text" format is a multiline site catalog format. It is described in the site catalog guide. It can be directly given to Pegasus starting with PEGASUS-1.4

pegasus.catalog.site.file

System:	Site Catalog
Since:	2.0
Type:	file location string
Default:	<code>\${pegasus.home.sysconfdir}/sites.xml </code> <code>\${pegasus.home.sysconfdir}/sites.txt</code>
See also:	pegasus.catalog.site

Running things on the grid requires an extensive description of the capabilities of each compute cluster, commonly termed "site". This property describes the location of the file that contains such a site description. As the format is currently in flow, please refer to the userguide and Pegasus for details which format is expected. The default value is dependant on the value specified for the property `pegasus.sc`. `pegasus.sc` denotes the type of site catalog being used.

Transformation Catalog

pegasus.catalog.transformation

System:	Transformation Catalog
Since:	2.0
Type:	enumeration
Value[0]:	Text
Value[1]:	File
Value[2]:	Database
Default:	Text
See also:	pegasus.catalog.*.driver
See also:	pegasus.catalog.transformation.file

Text In this mode, a multiline file based format is understood. The file is read and cached in memory. Any modifications, as adding or deleting, causes an update of the memory and hence to the file underneath. All queries are done against the memory representation.

The file `sample.tc.text` in the `etc` directory contains an example

Here is a sample textual format for transformation catalog containing one transformation on two sites

```
tr example::keg:1.0 {
#specify profiles that apply for all the sites for the transformation
#in each site entry the profile can be overridden
profile env "APP_HOME" "/tmp/karan"
profile env "JAVA_HOME" "/bin/app"
site isi {
profile env "me" "with"
profile condor "more" "test"
profile env "JAVA_HOME" "/bin/java.1.6"
pfn "/path/to/keg"
arch "x86"
os "linux"
osrelease "fc"
osversion "4"
type "INSTALLED"
site wind {
profile env "me" "with"
profile condor "more" "test"
pfn "/path/to/keg"
arch "x86"
os "linux"
osrelease "fc"
osversion "4"
type "STAGEABLE"
```

File In this mode, a file format is understood. The file is read and cached in memory. Any modifications, as adding or deleting, causes an update of the memory and hence to the file underneath. All queries are done against the memory representation. The new TC file format uses 6 columns:

1. The resource ID is represented in the first column.
2. The logical transformation uses the colonized format `ns::name:vs`.

3. The path to the application on the system
4. The installation type is identified by one of the following keywords - all upper case: INSTALLED, STAGEABLE. If not specified, or **NULL** is used, the type defaults to INSTALLED.
5. The system is of the format ARCH::OS[:VER:GLIBC]. The following arch types are understood: "INTEL32", "INTEL64", "SPARCV7", "SPARCV9". The following os types are understood: "LINUX", "SUNOS", "AIX". If unset or **NULL**, defaults to INTEL32::LINUX.
6. Profiles are written in the format NS::KEY=VALUE,KEY2=VALUE;NS2::KEY3=VALUE3. Multiple key-values for same namespace are separated by a comma "," and multiple namespaces are separated by a semicolon ";". If any of your profile values contains a comma you must not use the namespace abbreviator.

Database In this mode, the transformation catalog is kept in a relational database. Currently mysql DB and Postgre are supported. To set up the the database, use the schema in \$PEGASUS_HOME/sql/create-my-init.sql followed by \$PEGASUS_HOME/sql/create-my-tc.sql .

```
The following properties need to be set
pegasus.catalog.transformation.db.driver = MySQL|Postgres
pegasus.catalog.transformation.db.url =
jdbc:mysql://[hostname[:port]]/database |
jdbc:postgres://[hostname[:port]]/database
pegasus.catalog.transformation.db.user = dbusername
pegasus.catalog.transformation.db.password = password
```

If the pegasus.catalog.transformation.db.* properties are not defined, the database implementation picks up the properties specified by pegasus.catalog.*.db.* .

Future modifications to the TC may extend the enumeration. To implement your own TC implementation see org.girphyn.cPlanner.tc.TCMechanism. To load the class set pegasus.catalog.transformation to the TC implementation class.

pegasus.catalog.transformation.file

Systems:	Transformation Catalog
Type:	file location string
Default:	\${pegasus.home.sysconfdir}/tc.data
See also:	pegasus.catalog.transformation

This property is used to set the path to the textual transformation catalogs of type File or Text.

Provenance Catalog

pegasus.catalog.provenance

System:	Provenance Tracking Catalog (PTC)
Since:	2.0
Type:	Java class name
Value[0]:	InvocationSchema
Value[1]:	NXDInvSchema
Default:	(no default)
See also:	pegasus.catalog.*.db.driver

This property denotes the schema that is being used to access a PTC. The PTC is usually not a standard installation. If you use a database backend, you most likely have a schema that supports PTCs. By default, no PTC will be used.

Currently only the InvocationSchema is available for storing the provenance tracking records. Beware, this can become a lot of data. The values are names of Java classes. If no absolute Java classname is given, "org.griphyn.vdl.dbschema." is prepended. Thus, by deriving from the DatabaseSchema API, and implementing the PTC interface, users can provide their own classes here.

Alternatively, if you use a native XML database like eXist, you can store data using the NXDInvSchema. This will avoid using any of the other database driver properties.

pegasus.catalog.provenance.refinement

System:	PASOA Provenance Store
Since:	2.0.1
Type:	Java class name
Value[0]:	Pasoa
Value[1]:	InMemory
Default:	InMemory
See also:	pegasus.catalog.*.db.driver

This property turns on the logging of the refinement process that happens inside Pegasus to the PASOA store. Not all actions are currently captured. It is still an experimental feature.

The PASOA store needs to run on localhost on port 8080 <https://localhost:8080/prserv-1.0>

Work Catalog

pegasus.catalog.work

System:	Work Catalog
Since:	2.0
Type:	Java class name
Value[0]:	Database
Default:	Database
See also:	pegasus.catalog.*.db.driver

This property denotes the schema that is being used to store workflow monitoring entries. This catalog is populated at the end of the planning process by pegasus-plan and at workflow execution by the tailstatd daemon.

Replica Selection Properties

pegasus.selector.replica

System:	Replica Selection
Since:	2.0
Type:	URI string

Default:	default
See also:	pegasus.replica.*.ignore.stagein.sites
See also:	pegasus.replica.*.prefer.stagein.sites

Each job in the DAX maybe associated with input LFN's denoting the files that are required for the job to run. To determine the physical replica (PFN) for a LFN, Pegasus queries the replica catalog to get all the PFN's (replicas) associated with a LFN. Pegasus then calls out to a replica selector to select a replica amongst the various replicas returned. This property determines the replica selector to use for selecting the replicas.

Default	<p>If a PFN that is a file URL (starting with file:///) and has a pool attribute matching to the site handle of the site where the compute is to be run is found, then that is returned. Else, a random PFN is selected amongst all the PFN's that have a pool attribute matching to the site handle of the site where a compute job is to be run. Else, a random pfn is selected amongst all the PFN's.</p>
Restricted	<p>This replica selector, allows the user to specify good sites and bad sites for staging in data to a particular compute site. A good site for a compute site X, is a preferred site from which replicas should be staged to site X. If there are more than one good sites having a particular replica, then a random site is selected amongst these preferred sites.</p> <p>A bad site for a compute site X, is a site from which replica's should not be staged. The reason of not accessing replica from a bad site can vary from the link being down, to the user not having permissions on that site's data.</p> <p>The good bad sites are specified by the properties</p> <pre>pegasus.replica.*.prefer.stagein.sites pegasus.replica.*.ignore.stagein.sites</pre> <p>where the * in the property name denotes the name of the compute site. A * in the property key is taken to mean all sites.</p> <p>The <code>pegasus.replica.*.prefer.stagein.sites</code> property takes precedence over <code>pegasus.replica.*.ignore.stagein.sites</code> property i.e. if for a site X, a site Y is specified both in the ignored and the preferred set, then site Y is taken to mean as only a preferred site for a site X.</p>
Regex	<p>This replica selector allows the user allows the user to specific regex expressions that can be used to rank various PFN's returned from the Replica Catalog for a particular LFN. This replica selector selects the highest ranked PFN i.e the replica with the lowest rank value.</p> <p>The regular expressions are assigned different rank, that determine the order in which the expressions are employed. The rank values for the regex can expressed in user properties using the property.</p> <pre>pegasus.selector.replica.regex.rank.[value] regex-expression</pre> <p>The value is an integer value that denotes the rank of an expression with a rank value of 1 being the highest rank.</p> <p>Please note that before applying any regular expressions on the PFN's, the file URL's that dont match the preferred site are explicitly filtered out.</p>
Local	<p>This replica selector prefers replicas from the local host and that start with a file: URL scheme. It is useful, when users want to stagin files to a remote site from your submit host using the Condor file transfer mechanism.</p>

pegasus.selector.replica.*.ignore.stagein.sites

System:	Replica Selection
Type:	comma separated list of sites

Since:	2.0
Default:	no default
See also:	pegasus.selector.replica
See also:	pegasus.selector.replica.*.prefer.stagein.sites

A comma separated list of storage sites from which to never stage in data to a compute site. The property can apply to all or a single compute site, depending on how the * in the property name is expanded.

The * in the property name means all compute sites unless replaced by a site name.

For e.g setting `pegasus.selector.replica.*.ignore.stagein.sites` to `usc` means that ignore all replicas from site `usc` for staging in to any compute site. Setting `pegasus.replica.isi.ignore.stagein.sites` to `usc` means that ignore all replicas from site `usc` for staging in data to site `isi`.

pegasus.selector.replica.*.prefer.stagein.sites

System:	Replica Selection
Type:	comma separated list of sites
Since:	2.0
Default:	no default
See also:	pegasus.selector.replica
See also:	pegasus.selector.replica.*.ignore.stagein.sites

A comma separated list of preferred storage sites from which to stage in data to a compute site. The property can apply to all or a single compute site, depending on how the * in the property name is expanded.

The * in the property name means all compute sites unless replaced by a site name.

For e.g setting `pegasus.selector.replica.*.prefer.stagein.sites` to `usc` means that prefer all replicas from site `usc` for staging in to any compute site. Setting `pegasus.replica.isi.prefer.stagein.sites` to `usc` means that prefer all replicas from site `usc` for staging in data to site `isi`.

pegasus.selector.replica.regex.rank.[value]

System:	Replica Selection
Type:	Regex Expression
Since:	2.3.0
Default:	no default
See also:	pegasus.selector.replica

Specifies the regex expressions to be applied on the PFNs returned for a particular LFN. Refer to

<http://java.sun.com/javase/6/docs/api/java/util/regex/Pattern.html>

on information of how to construct a regex expression.

The [value] in the property key is to be replaced by an int value that designates the rank value for the regex expression to be applied in the Regex replica selector.

The example below indicates preference for file URL's over URL's referring to gridftp server at `example.isi.edu`

```
pegasus.selector.replica.regex.rank.1 file:///.*
```

```
pegasus.selector.replica.regex.rank.2 gsiftp://example\.\isi\.\edu.*
```

Site Selection Properties

pegasus.selector.site

System:	Pegasus
Since:	2.0
Type:	enumeration
Value[0]:	Random
Value[1]:	RoundRobin
Value[2]:	NonJavaCallout
Value[3]:	Group
Value[4]:	Heft
Default:	Random
See also:	pegasus.selector.site.path
See also:	pegasus.selector.site.timeout
See also:	pegasus.selector.site.keep.tmp
See also:	pegasus.selector.site.env.*

The site selection in Pegasus can be on basis of any of the following strategies.

Random	In this mode, the jobs will be randomly distributed among the sites that can execute them.
RoundRobin	In this mode, the jobs will be assigned in a round robin manner amongst the sites that can execute them. Since each site cannot execute everytype of job, the round robin scheduling is done per level on a sorted list. The sorting is on the basis of the number of jobs a particular site has been assigned in that level so far. If a job cannot be run on the first site in the queue (due to no matching entry in the transformation catalog for the transformation referred to by the job), it goes to the next one and so on. This implementation defaults to classic round robin in the case where all the jobs in the workflow can run on all the sites.
NonJavaCallout	In this mode, Pegasus will callout to an external site selector. In this mode a temporary file is prepared containing the job information that is passed to the site selector as an argument while invoking it. The path to the site selector is specified by setting the property <code>pegasus.site.selector.path</code> . The environment variables that need to be set to run the site selector can be specified using the properties with a <code>pegasus.site.selector.env.</code> prefix. The temporary file contains information about the job that needs to be scheduled. It contains key value pairs with each key value pair being on a new line and separated by a <code>=</code> .

The following pairs are currently generated for the site selector temporary file that is generated in the NonJavaCallout.

version	is the version of the site selector api, currently 2.0.
transformation	is the fully-qualified definition identifier for the transformation (TR) namespace::name:version.
derivation	is the fully qualified definition identifier for the derivation (DV), namespace::name:version.

job.level	is the job's depth in the tree of the workflow DAG.
job.id	is the job's ID, as used in the DAX file.
resource.id	is a pool handle, followed by whitespace, followed by a gridftp server. Typically, each gridftp server is enumerated once, so you may have multiple occurrences of the same site. There can be multiple occurrences of this key.
input.lfn	is an input LFN, optionally followed by a whitespace and file size. There can be multiple occurrences of this key, one for each input LFN required by the job.
wf.name	label of the dax, as found in the DAX's root element. wf.index is the DAX index, that is incremented for each partition in case of deferred planning.
wf.time	is the mtime of the workflow.
wf.manager	is the name of the workflow manager being used .e.g condor
vo.name	is the name of the virtual organization that is running this workflow. It is currently set to NONE
vo.group	unused at present and is set to NONE.

Group In this mode, a group of jobs will be assigned to the same site that can execute them. The use of the PEGASUS profile key group in the dax, associates a job with a particular group. The jobs that do not have the profile key associated with them, will be put in the default group. The jobs in the default group are handed over to the "Random" Site Selector for scheduling.

Heft In this mode, a version of the HEFT processor scheduling algorithm is used to schedule jobs in the workflow to multiple grid sites. The implementation assumes default data communication costs when jobs are not scheduled on to the same site. Later on this may be made more configurable.

The runtime for the jobs is specified in the transformation catalog by associating the pegasus profile key runtime with the entries.

The number of processors in a site is picked up from the attribute idle-nodes associated with the vanilla jobmanager of the site in the site catalog.

pegasus.selector.site.path

System:	Site Selector
Since:	2.0
Type:	String

If one calls out to an external site selector using the NonJavaCallout mode, this refers to the path where the site selector is installed. In case other strategies are used it does not need to be set.

pegasus.site.selector.env.*

System:	Pegasus
---------	---------

Since:	1.2.3
Type:	String

The environment variables that need to be set while callout to the site selector. These are the variables that the user would set if running the site selector on the command line. The name of the environment variable is got by stripping the keys of the prefix "pegasus.site.selector.env." prefix from them. The value of the environment variable is the value of the property.

e.g pegasus.site.selector.path.LD_LIBRARY_PATH /globus/lib would lead to the site selector being called with the LD_LIBRARY_PATH set to /globus/lib.

pegasus.selector.site.timeout

System:	Site Selector
Since:	2.0
Type:	non negative integer
Default:	60

It sets the number of seconds Pegasus waits to hear back from an external site selector using the NonJavaCallout interface before timing out.

pegasus.selector.site.keep.tmp

System:	Pegasus
Since:	2.0
Type:	enumeration
Value[0]:	onerror
Value[1]:	always
Value[2]:	never
Default:	onerror

It determines whether Pegasus deletes the temporary input files that are generated in the temp directory or not. These temporary input files are passed as input to the external site selectors.

A temporary input file is created for each that needs to be scheduled.

Transfer Configuration Properties

pegasus.transfer.*.impl

System:	Pegasus
Type:	enumeration
Value[0]:	Transfer3
Value[1]:	GUC
Default:	Transfer3
See also:	pegasus.transfer.refiner
Since:	2.0

Each compute job usually has data products that are required to be staged in to the execution site, materialized data products staged out to a final resting place, or staged to another job running at a different site. This property determines the underlying grid transfer tool that is used to manage the transfers.

The * in the property name can be replaced to achieve finer grained control to dictate what type of transfer jobs need to be managed with which grid transfer tool.

Usually, the arguments with which the client is invoked can be specified by

- the property `pegasus.transfer.arguments`
- associating the PEGASUS profile key `transfer.arguments`

The table below illustrates all the possible variations of the property.

Property Name	Applies to
<code>pegasus.transfer.stagein.impl</code>	the stage in transfer jobs
<code>pegasus.transfer.stageout.impl</code>	the stage out transfer jobs
<code>pegasus.transfer.inter.impl</code>	the inter pool transfer jobs
<code>pegasus.transfer.setup.impl</code>	the setup transfer job
<code>pegasus.transfer.*.impl</code>	apply to types of transfer jobs

Note: Since version 2.2.0 the worker package is staged automatically during staging of executables to the remote site. This is achieved by adding a setup transfer job to the workflow. The setup transfer job by default uses GUC to stage the data. The implementation to use can be configured by setting the property

```
pegasus.transfer.setup.impl
```

property. However, if you have `pegasus.transfer.*.impl` set in your properties file, then you need to set `pegasus.transfer.setup.impl` to GUC

The various grid transfer tools that can be used to manage data transfers are explained below

Transfer3 This results in `pegasus-transfer` to be used for transferring of files. It is a python based wrapper around various transfer clients like `globus-url-copy`, `lcg-copy`, `wget`, `cp`, `ln`. `pegasus-transfer` looks at source and destination url and figures out automatically which underlying client to use. `pegasus-transfer` is distributed with the PEGASUS and can be found at `$PEGASUS_HOME/bin/pegasus-transfer`.

For remote sites, Pegasus constructs the default path to `pegasus-transfer` on the basis of `PEGASUS_HOME` env profile specified in the site catalog. To specify a different path to the `pegasus-transfer` client, users can add an entry into the transformation catalog with fully qualified logical name as `pegasus::pegasus-transfer`

GUC This refers to the new `guc` client that does multiple file transfers per invocation. The `globus-url-copy` client distributed with Globus 4.x is compatible with this mode.

pegasus.transfer.refiner

System:	Pegasus
Type:	enumeration
Value[0]:	Default
Value[1]:	Bundle
Value[2]:	Chain
Value[3]:	Condor
Value[4]:	Cluster
Default:	Bundle

Since:	2.0
See also:	pegasus.transfer.*.impl

This property determines how the transfer nodes are added to the workflow. The various refiners differ in the how they link the various transfer jobs, and the number of transfer jobs that are created per compute jobs.

Default This is the default refinement strategy for the transfer tools that can handle multiple file transfers per invocation. In this, all files required by a particular transfer job will be attempted to be transferred with just one transfer job. This also takes care of file clobbering while staging in data to a remote grid site. File Clobbering can occur when two jobs scheduled on the same site require the same input file to be staged.

Bundle In this refinement strategy, the number of stage in transfer nodes that are constructed per execution site can vary. The number of transfer nodes can be specified, by associating the pegasus profile "bundle.stagein". The profile can either be associated with the execution site in the site catalog or with the "transfer" executable in the transformation catalog. The value in the transformation catalog overrides the one in the site catalog. This refinement strategy extends from the Default refiner, and thus takes care of file clobbering while staging in data.

Chain In this refinement strategy, chains of stagein transfer nodes are constructed. A chain means that the jobs are sequentially dependant upon each other i.e. at any moment, only one stage in transfer job will run per chain. The number of chains can be specified by associating the pegasus profile "chain.stagein". The profile can either be associated with the execution site in the site catalog or with the "transfer" executable in the transformation catalog. The value in the transformation catalog overrides the one in the site catalog. This refinement strategy extends from the Default refiner, and thus takes care of file clobbering while staging in data.

Condor In this refinement strategy, no additional staging transfer jobs are added to the workflow. Instead the compute jobs are modified to have the transfer_input_files and transfer_output_files set to pull the input data. To stage-out the data a separate stage-out is added. The stage-out job is a /bin/true job that uses the transfer_input_file and transfer_output_files to stage the data back to the submit host. This refinement strategy is used workflows are being executed on a Condor pool, and the submit node itself is a part of the Condor pool.

Cluster In this refinement strategy, clusters of stage-in and stageout jobs are created per level of the workflow. It builds upon the Bundle refiner. The differences between the Bundle and Cluster refiner are as follows.

- stagein is also clustered/bundled per level. In Bundle it was for the whole workflow.
- keys that control the clustering (old name bundling are) cluster.stagein and cluster.stageout

This refinement strategy also adds dependencies between the stagein transfer jobs on different levels of the workflow to ensure that stagein for the top level happens first and so on.

An image of the workflow with this refinement strategy can be found at

http://vtcpc.isi.edu/pegasus/index.php/ChangeLog#Added_a_Cluster_Transfer_Refiner

pegasus.transfer.sls.*.impl

System:	Pegasus
Type:	enumeration
Value[0]:	Transfer3
Value[1]:	S3
Default:	Transfer3
Since:	2.2.0
See also:	pegasus.execute.*.filesystem.local

This property specifies the transfer tool to be used for Second Level Staging (SLS) of input and output data between the head node and worker node filesystems.

Currently, the * in the property name CANNOT be replaced to achieve finer grained control to dictate what type of SLS transfers need to be managed with which grid transfer tool.

The various grid transfer tools that can be used to manage SLS data transfers are explained below

Transfer3 This results in pegasus-transfer to be used for transferring of files. It is a python based wrapper around various transfer clients like globus-url-copy, lcg-copy, wget, cp, ln . pegasus-transfer looks at source and destination url and figures out automatically which underlying client to use. pegasus-transfer is distributed with the PEGASUS and can be found at \$PEGASUS_HOME/bin/pegasus-transfer.

For remote sites, Pegasus constructs the default path to pegasus-transfer on the basis of PEGASUS_HOME env profile specified in the site catalog. To specify a different path to the pegasus-transfer client , users can add an entry into the transformation catalog with fully qualified logical name as pegasus::pegasus-transfer

Condor This results in Condor file transfer mechanism to be used to transfer the input data files from the submit host directly to the worker node directories. This is used when running in pure Condor mode or in a Condor pool that does not have a shared filesystem between the nodes.

When setting the SLS transfers to Condor make sure that the following properties are also set

```
pegasus.transfer.refiner    Condor
pegasus.selector.replica   Local
pegasus.execute.*.filesystem.local true
```

Also make sure that pegasus.gridstart is not set.

Please refer to the section on "Condor Pool Without a Shared Filesystem" in the chapter on Planning and Submitting.

S3 This implementation refers to the s3cmd transfer client that is used for second level staging of data in the cloud. The data can be staged between the filesystem on the worker nodes and the workflow specific bucket on S3.

There should be an entry in the transformation catalog with the fully qualified name as amazon::s3cmd for the site corresponding to the cloud.

pegasus.transfer.arguments

System:	Pegasus
Since:	2.0
Type:	String
Default:	(no default)
See also:	pegasus.transfer.sls.arguments

This determines the extra arguments with which the transfer implementation is invoked. The transfer executable that is invoked is dependant upon the transfer mode that has been selected. The property can be overloaded by associated the pegasus profile key transfer.arguments either with the site in the site catalog or the corresponding transfer executable in the transformation catalog.

pegasus.transfer.sls.arguments

System:	Pegasus
Since:	2.4

Type:	String
Default:	(no default)
See also:	pegasus.transfer.arguments
See also:	pegasus.transfer.sls.*.impl

This determines the extra arguments with which the SLS transfer implementation is invoked. The transfer executable that is invoked is dependant upon the SLS transfer implementation that has been selected.

pegasus.transfer.stage.sls.file

System:	Pegasus
Since:	3.0
Type:	Boolean
Default:	(no default)
See also:	pegasus.gridstart
See also:	pegasus.execute.*.filesystem.local

For executing jobs on the local filesystem, Pegasus creates sls files for each compute jobs. These sls files list the files that need to be staged to the worker node and the output files that need to be pushed out from the worker node after completion of the job. By default, pegasus will stage these SLS files to the shared filesystem on the head node as part of first level data stagein jobs. However, in the case where there is no shared filesystem between head nodes and the worker nodes, the user can set this property to false. This will result in the sls files to be transferred using the Condor File Transfer from the submit host.

pegasus.transfer.worker.package

System:	Pegasus
Type:	boolean
Default:	false
Since:	3.0

By default, Pegasus relies on the worker package to be installed on the shared filesystem of the remote sites . Pegasus uses the value of PEGASUS_HOME environment profile in the site catalog for the remote sites, to then construct paths to pegasus auxillary executables like kickstart, pegasus-transfer, seqexec etc.

If the Pegasus worker package is not installed on the remote sites users can set this property to true to get Pegasus to deploy worker package shared scratch directory for the workflow on the remote sites.

pegasus.transfer.links

System:	Pegasus
Type:	boolean
Default:	false
Since:	2.0
See also:	pegasus.transfer
See also:	pegasus.transfer.force

If this is set, and the transfer implementation is set to Transfer i.e. using the transfer executable distributed with the PEGASUS. On setting this property, if Pegasus while fetching data from the RLS sees a pool attribute associated

with the PFN that matches the execution pool on which the data has to be transferred to, Pegasus instead of the URL returned by the RLS replaces it with a file based URL. This supposes that the if the pools match the filesystems are visible to the remote execution directory where input data resides. On seeing both the source and destination urls as file based URLs the transfer executable spawns a job that creates a symbolic link by calling `ln -s` on the remote pool. This ends up bypassing the GridFTP server and reduces the load on it, and is much faster.

pegasus.transfer.*.remote.sites

System:	Pegasus
Type:	comma separated list of sites
Default:	no default
Since:	2.0

By default Pegasus looks at the source and destination URL's for to determine whether the associated transfer job runs on the submit host or the head node of a remote site, with preference set to run a transfer job to run on submit host.

Pegasus will run transfer jobs on the remote sites

- if the file server for the compute site is a file server i.e url prefix `file://`
- symlink jobs need to be added that require the symlink transfer jobs to be run remotely.

This property can be used to change the default behaviour of Pegasus and force pegasus to run different types of transfer jobs for the sites specified on the remote site.

The table below illustrates all the possible variations of the property.

Property Name	Applies to
pegasus.transfer.stagein.remote.sites	the stage in transfer jobs
pegasus.transfer.stageout.remote.sites	the stage out transfer jobs
pegasus.transfer.inter.remote.sites	the inter pool transfer jobs
pegasus.transfer.*.remote.sites	apply to types of transfer jobs

In addition `*` can be specified as a property value, to designate that it applies to all sites.

pegasus.transfer.staging.delimiter

System:	Pegasus
Since:	2.0
Type:	String
Default:	:
See also:	pegasus.transformation.selector

Pegasus supports executable staging as part of the workflow. Currently staging of statically linked executables is supported only. An executable is normally staged to the work directory for the workflow/partition on the remote site. The basename of the staged executable is derived from the namespace,name and version of the transformation in the transformation catalog. This property sets the delimiter that is used for the construction of the name of the staged executable.

pegasus.transfer.disable.chmod.sites

System:	Pegasus
---------	---------

Since:	2.0
Type:	comma separated list of sites
Default:	no default

During staging of executables to remote sites, chmod jobs are added to the workflow. These jobs run on the remote sites and do a chmod on the staged executable. For some sites, this may not be required. The permissions might be preserved, or there may be an automatic mechanism that does it.

This property allows you to specify the list of sites, where you do not want the chmod jobs to be executed. For those sites, the chmod jobs are replaced by NoOP jobs. The NoOP jobs are executed by Condor, and instead will immediately have a terminate event written to the job log file and removed from the queue.

pegasus.transfer.proxy

System:	Pegasus
Since:	2.0
Type:	Boolean
Default:	false

By default, CondorG transfers a limited proxy to the remote site, while running jobs in the grid universe. However, certain grid ftp servers (like those in front of SRB) require a fully user proxy. In this case, the planners need to transfer the proxy along with the transfer job using Condor file transfer mechanisms. This property triggers Pegasus into creating the appropriate condor commands, that transfer the proxy from the submit host to the remote host. The source location is determined from the value of the X509_USER_KEY env profile key, that is associated with site local in the site catalog.

pegasus.transfer.setup.source.base.url

System:	Pegasus
Type:	URL
Default:	no default
Since:	2.3

This property specifies the base URL to the directory containing the Pegasus worker package builds. During Staging of Executable, the Pegasus Worker Package is also staged to the remote site. The worker packages are by default pulled from the http server at pegasus.isi.edu. This property can be used to override the location from where the worker package are staged. This may be required if the remote computes sites don't allow file transfers from a http server.

Gridstart And Exitcode Properties

pegasus.gridstart

System:	Pegasus
Since:	2.0
Type:	enumeration
Value[0]:	Kickstart
Value[1]:	None
Value[2]:	SeqExec
Default:	Kickstart

See also: | `pegasus.execute.*.filesystem.local`

Jobs that are launched on the grid maybe wrapped in a wrapper executable/script that enables information about about the execution, resource consumption, and - most importantly - the exitcode of the remote application. At present, a job scheduled on a remote site is launched with a gridstart if site catalog has the corresponding gridlaunch attribute set and the job being launched is not MPI.

Users can explicitly decide what gridstart to use for a job, by associating the pegasus profile key named gridstart with the job.

Kickstart	In this mode, all the jobs are lauched via kickstart. The kickstart executable is a light-weight program which connects the stdin,stdout and stderr filehandles for PEGASUS jobs on the remote site. Kickstart is an executable distributed with PEGASUS that can generally be found at <code>\${pegasus.home.bin}/kickstart</code> .
None	In this mode, all the jobs are launched directly on the remote site. Each job's stdin,stdout and stderr are connected to condor commands in a manner to ensure that they are sent back to the submit host.
SeqExec	In this mode, all the jobs are launched on the remote site via SeqExec clustering executable. This is useful when user wants to run on the local filesystem of the worker nodes. The generated input file to seqexec contains commands to create the directory on the worker node, pull data from the head node, execute the job , push data to the head node, and remove the directory on the worker node. This is still an experimental mode and will be refined more for 3.1.

Support for a new gridstart (K2) is expected to be added soon.

pegasus.gridstart.kickstart.set.xbit

System:	Pegasus
Since:	2.4
Type:	Boolean
Default:	false
See also:	<code>pegasus.transfer.disable.chmod.sites</code>

Kickstart has an option to set the X bit on an executable before it launches it on the remote site. In case of staging of executables, by default chmod jobs are launched that set the x bit of the user executables staged to a remote site.

On setting this property to true, kickstart gridstart module adds a -X option to kickstart arguments. The -X arguments tells kickstart to set the x bit of the executable before launching it.

User should usually disable the chmod jobs by setting the property `pegasus.transfer.disable.chmod.sites` , if they set this property to true.

pegasus.gridstart.kickstart.stat

System:	Pegasus
Since:	2.1
Type:	Boolean
Default:	false
See also:	<code>pegasus.gridstart.generate.lof</code>

Kickstart has an option to stat the input files and the output files. The stat information is collected in the XML record generated by kickstart. Since stat is an expensive operation, it is not turned on by on. Set this property to true if you want to see stat information for the input files and output files of a job in it's kickstart output.

pegasus.gridstart.generate.lof

System:	Pegasus
Since:	2.1
Type:	Boolean
Default:	false
See also:	pegasus.gridstart.kickstart.stat

For the stat option for kickstart, we generate 2 lof (list of filenames) files for each job. One lof file containing the input lfn's for the job, and the other containing output lfn's for the job. In some cases, it maybe beneficial to have these lof files generated but not do the actual stat. This property allows you to generate the lof files without triggering the stat in kickstart invocations.

pegasus.gridstart.invoke.always

System:	Pegasus
Since:	2.0
Type:	Boolean
Default:	false
See also:	pegasus.gridstart.invoke.length

Condor has a limit in it, that restricts the length of arguments to an executable to 4K. To get around this limit, you can trigger Kickstart to be invoked with the -I option. In this case, an arguments file is prepared per job that is transferred to the remote end via the Condor file transfer mechanism. This way the arguments to the executable are not specified in the condor submit file for the job. This property specifies whether you want to use the invoke option always for all jobs, or want it to be triggered only when the argument string is determined to be greater than a certain limit.

pegasus.gridstart.invoke.length

System:	Pegasus
Since:	2.0
Type:	Long
Default:	4000
See also:	pegasus.gridstart.invoke.always

Gridstart is automatically invoked with the -I option, if it is determined that the length of the arguments to be specified is going to be greater than a certain limit. By default this limit is set to 4K. However, it can be overridden by specifying this property.

Interface To Condor And Condor Dagman

The Condor DAGMan facility is usually activate using the condor_submit_dag command. However, many shapes of workflows have the ability to either overburden the submit host, or overflow remote gatekeeper hosts. While DAGMan provides throttles, unfortunately these can only be supplied on the command-line. Thus, PEGASUS provides a versatile wrapper to invoke DAGMan, called pegasus-submit-dag. It can be configured from the command-line, from user- and system properties, and by defaults.

pegasus.condor.logs.symlink

System:	Condor
---------	--------

Type:	Boolean
Default:	true
Since:	3.0

By default pegasus has the Condor common log [dagname]-0.log in the submit file as a symlink to a location in /tmp . This is to ensure that condor common log does not get written to a shared filesystem. If the user knows for sure that the workflow submit directory is not on the shared filesystem, then they can opt to turn of the symlinking of condor common log file by setting this property to false.

pegasus.condor.arguments.quote

System:	Condor
Type:	Boolean
Default:	true
Since:	2.0
Old Name:	pegasus.condor.arguments.quote

This property determines whether to apply the new Condor quoting rules for quoting the argument string. The new argument quoting rules appeared in Condor 6.7.xx series. We have verified it for 6.7.19 version. If you are using an old condor at the submit host, set this property to false.

pegasus.dagman.nofity

System:	DAGman wrapper
Type:	Case-insensitive enumeration
Value[0]:	Complete
Value[1]:	Error
Value[2]:	Never
Default:	Error
Document:	http://www.cs.wisc.edu/condor/manual/v6.9/condor_submit_dag.html
Document:	http://www.cs.wisc.edu/condor/manual/v6.9/condor_submit.html

The pegasus-submit-dag wrapper processes properties to set DAGMan commandline arguments. The argument sets the e-mail notification for DAGMan itself. This information will be used within the Condor submit description file for DAGMan. This file is produced by the the condor_submit_dag. See notification within the section of submit description file commands in the condor_submit manual page for specification of value. Many users prefer the value NEVER.

pegasus.dagman.verbose

System:	DAGman wrapper
Type:	Boolean
Value[0]:	false
Value[1]:	true
Default:	false
Document:	http://www.cs.wisc.edu/condor/manual/v6.9/condor_submit_dag.html

The pegasus-submit-dag wrapper processes properties to set DAGMan commandline arguments. If set and true, the argument activates verbose output in case of DAGMan errors.

pegasus.dagman.[category].maxjobs

System:	DAGman wrapper
Type:	Integer
Since:	2.2
Default:	no default
Document:	http://vtcpc.isi.edu/pegasus/index.php/ChangeLog \\#Support_for_DAGMan_node_categories

DAGMan now allows for the nodes in the DAG to be grouped in category. The tuning parameters like maxjobs then can be applied per category instead of being applied to the whole workflow. To use this facility users need to associate the dagman profile key named category with their jobs. The value of the key is the category to which the job belongs to.

You can then use this property to specify the value for a category. For the above example you will set pegasus.dagman.short-running.maxjobs

Monitoring Properties

pegasus.monitord.events

System:	Pegasus-monitord
Type:	Boolean
Default:	true
Since:	3.0.2
See Also:	pegasus.monitord.output

This property determines whether pegasus-monitord generates log events. If log events are disabled using this property, no bp file, or database will be created, even if the pegasus.monitord.output property is specified.

pegasus.monitord.output

System:	Pegasus-monitord
Type:	String
Since:	3.0.2
See Also:	pegasus.monitord.events

This property specifies the destination for generated log events in pegasus-monitord. By default, events are stored in a sqlite database in the workflow directory, which will be created with the workflow's name, and a ".stampede.db" extension. Users can specify an alternative database by using a SQLAlchemy connection string. Details are available at:

<http://www.sqlalchemy.org/docs/05/reference/dialects/index.html>

It is important to note that users will need to have the appropriate db interface library installed. Which is to say, SQLAlchemy is a wrapper around the mysql interface library (for instance), it does not provide a MySQL driver itself.

The Pegasus distribution includes both SQLAlchemy and the SQLite Python driver. As a final note, it is important to mention that unlike when using SQLite databases, using SQLAlchemy with other database servers, e.g. MySQL or Postgres, the target database needs to exist. Users can also specify a file name using this property in order to create a file with the log events.

Example values for the SQLAlchemy connection string for various end points are listed below

SQLAlchemy End Point	Example Value
Netlogger BP File	file:///submit/dir/myworkflow.bp
SQL Lite Database	sqlite:///submit/dir/myworkflow.db
MySQL Database	mysql://user:password@host:port/databasename

Job Clustering Properties

pegasus.clusterer.job.aggregator

System:	Job Clustering
Since:	2.0
Type:	String
Value[0]:	seqexec
Value[1]:	mpiexec
Default:	seqexec

A large number of workflows executed through the Virtual Data System, are composed of several jobs that run for only a few seconds or so. The overhead of running any job on the grid is usually 60 seconds or more. Hence, it makes sense to collapse small independent jobs into a larger job. This property determines, the executable that will be used for running the larger job on the remote site.

seqexec	In this mode, the executable used to run the merged job is seqexec that runs each of the smaller jobs sequentially on the same node. The executable "seqexec" is a PEGASUS tool distributed in the PEGASUS worker package, and can be usually found at {pegasus.home}/bin/seqexec.
mpiexec	In this mode, the executable used to run the merged job is mpiexec that runs the smaller jobs via mpi on n nodes where n is the nodecount associated with the merged job. The executable "mpiexec" is a PEGASUS tool distributed in the PEGASUS worker package, and can be usually found at {pegasus.home}/bin/mpiexec.

pegasus.clusterer.job.aggregator.seqexec.log

System:	Job Clustering
Type:	Boolean
Default:	false
Since:	2.3
See also:	pegasus.clusterer.job.aggregator
See also:	pegasus.clusterer.job.aggregator.seqexec.log.global

Seqexec logs the progress of the jobs that are being run by it in a progress file on the remote cluster where it is executed.

This property sets the Boolean flag, that indicates whether to turn on the logging or not.

pegasus.clusterer.job.aggregator.seqexec.log.global

System:	Job Clustering
Type:	Boolean
Default:	true
Since:	2.3
See also:	pegasus.clusterer.job.aggregator
See also:	pegasus.clusterer.job.aggregator.seqexec.log
Old Name:	pegasus.clusterer.job.aggregator.seqexec.hasgloballog

Seqexec logs the progress of the jobs that are being run by it in a progress file on the remote cluster where it is executed. The progress log is useful for you to track the progress of your computations and remote grid debugging. The progress log file can be shared by multiple seqexec jobs that are running on a particular cluster as part of the same workflow. Or it can be per job.

This property sets the Boolean flag, that indicates whether to have a single global log for all the seqexec jobs on a particular cluster or progress log per job.

pegasus.clusterer.job.aggregator.seqexec.firstjobfail

System:	Job Clustering
Type:	Boolean
Default:	true
Since:	2.2
See also:	pegasus.clusterer.job.aggregator

By default seqexec does not stop execution even if one of the clustered jobs it is executing fails. This is because seqexec tries to get as much work done as possible.

This property sets the Boolean flag, that indicates whether to make seqexec stop on the first job failure it detects.

pegasus.clusterer.label.key

System:	Job Clustering
Type:	String
Default:	label
Since:	2.0
See also:	pegasus.partitionner.label.key

While clustering jobs in the workflow into larger jobs, you can optionally label your graph to control which jobs are clustered and to which clustered job they belong. This done using a label based clustering scheme and is done by associating a profile/label key in the PEGASUS namespace with the jobs in the DAX. Each job that has the same value/label value for this profile key, is put in the same clustered job.

This property allows you to specify the PEGASUS profile key that you want to use for label based clustering.

Logging Properties

pegasus.log.manager

System:	Pegasus
Since:	2.2.0
Type:	Enumeration
Value[0]:	Default
Value[1]:	Log4j
Default:	Default
See also:	pegasus.log.manager.formatter

This property sets the logging implementation to use for logging.

- Default** This implementation refers to the legacy Pegasus logger, that logs directly to stdout and stderr. It however, does have the concept of levels similar to log4j or syslog.
- Log4j** This implementation, uses Log4j to log messages. The log4j properties can be specified in a properties file, the location of which is specified by the property

```
pegasus.log.manager.log4j.conf
```

pegasus.log.manager.formatter

System:	Pegasus
Since:	2.2.0
Type:	Enumeration
Value[0]:	Simple
Value[1]:	Netlogger
Default:	Simple
See also:	pegasus.log.manager.formatter

This property sets the formatter to use for formatting the log messages while logging.

- Simple** This formats the messages in a simple format. The messages are logged as is with minimal formatting. Below are sample log messages in this format while ranking a dax according to performance.

```
event.pegasus.ranking dax.id sel8-gda.dax - STARTED
event.pegasus.parsing.dax dax.id sel8-gda-nested.dax - STARTED
event.pegasus.parsing.dax dax.id sel8-gda-nested.dax - FINISHED
job.id jobGDA
job.id jobGDA query.name getpredicted performace time 10.00
event.pegasus.ranking dax.id sel8-gda.dax - FINISHED
```

- Netlogger** This formats the messages in the Netlogger format , that is based on key value pairs. The netlogger format is useful for loading the logs into a database to do some meaningful analysis. Below are sample log messages in this format while ranking a dax according to performance.

```
ts=2008-09-06T12:26:20.100502Z event=event.pegasus.ranking.start \
msgid=6bc49c1f-112e-4cdb-af54-3e0afb5d593c \
eventId=event.pegasus.ranking_8d7c0a3c-9271-4c9c-a0f2-1fb57c6394d5 \
dax.id=sel8-gda.dax prog=Pegasus
ts=2008-09-06T12:26:20.100750Z event=event.pegasus.parsing.dax.start \
msgid=fed3ebdf-68e6-4711-8224-a16bb1ad2969 \
eventId=event.pegasus.parsing.dax_887134a8-39cb-40f1-b11c-b49def0c5232 \
dax.id=sel8-gda-nested.dax prog=Pegasus
ts=2008-09-06T12:26:20.100894Z event=event.pegasus.parsing.dax.end \
msgid=a81e92ba-27df-451f-bb2b-b60d232ed1ad \
```

```
eventId=event.pegasus.parsing.dax_887134a8-39cb-40f1-b11c-b49def0c5232
ts=2008-09-06T12:26:20.100395Z event=event.pegasus.ranking \
msgid=4dcecb68-74fe-4fd5-aa9e-ealcee88727d \
eventId=event.pegasus.ranking_8d7c0a3c-9271-4c9c-a0f2-1fb57c6394d5 \
job.id="jobGDA"
ts=2008-09-06T12:26:20.100395Z event=event.pegasus.ranking \
msgid=4dcecb68-74fe-4fd5-aa9e-ealcee88727d \
eventId=event.pegasus.ranking_8d7c0a3c-9271-4c9c-a0f2-1fb57c6394d5 \
job.id="jobGDA" query.name="getpredicted performace" time="10.00"
ts=2008-09-06T12:26:20.102003Z event=event.pegasus.ranking.end \
msgid=31f50f39-efe2-47fc-9f4c-07121280cd64 \
eventId=event.pegasus.ranking_8d7c0a3c-9271-4c9c-a0f2-1fb57c6394d5
```

pegasus.log.*

System:	Pegasus
Since:	2.0
Type:	String
Default:	No default

This property sets the path to the file where all the logging for Pegasus can be redirected to. Both stdout and stderr are logged to the file specified.

pegasus.log.metrics

System:	Pegasus
Since:	2.1.0
Type:	Boolean
Default:	true
See also:	pegasus.log.metrics.file

This property enables the logging of certain planning and workflow metrics to a global log file. By default the file to which the metrics are logged is `${pegasus.home}/var/pegasus.log`.

pegasus.log.metrics.file

System:	Pegasus
Since:	2.1.0
Type:	Boolean
Default:	<code>\${pegasus.home}/var/pegasus.log</code>
See also:	pegasus.log.metrics

This property determines the file to which the workflow and planning metrics are logged if enabled.

Miscellaneous Properties

pegasus.code.generator

System:	Pegasus
Since:	3.0

Type:	enumeration
Value[0]:	Condor
Value[1]:	Shell
Default:	Condor

This property is used to load the appropriate Code Generator to use for writing out the executable workflow.

Condor This is the default code generator for Pegasus . This generator generates the executable workflow as a Condor DAG file and associated job submit files. The Condor DAG file is passed as input to Condor DAGMan for job execution.

Shell This Code Generator generates the executable workflow as a shell script that can be executed on the submit host. While using this code generator, all the jobs should be mapped to site local i.e specify --sites local to pegasus-plan.

pegasus.file.cleanup.strategy

System:	Pegasus
Since:	2.2
Type:	enumeration
Value[0]:	InPlace
Default:	InPlace

This property is used to select the strategy of how the the cleanup nodes are added to the executable workflow.

InPlace This is the only mode available .

pegasus.file.cleanup.impl

System:	Pegasus
Since:	2.2
Type:	enumeration
Value[0]:	Cleanup
Value[1]:	RM
Value[2]:	S3
Default:	Cleanup

This property is used to select the executable that is used to create the working directory on the compute sites.

Cleanup The default executable that is used to delete files is the dirmanager executable shipped with Pegasus. It is found at \$PEGASUS_HOME/bin/dirmanager in the pegasus distribution. An entry for transformation pegasus::dirmanager needs to exist in the Transformation Catalog or the PEGASUS_HOME environment variable should be specified in the site catalog for the sites for this mode to work.

RM This mode results in the rm executable to be used to delete files from remote directories. The rm executable is standard on *nix systems and is usually found at /bin/rm location.

S3 This mode is used to delete files/objects from the buckets in S3 instead of a directory. This should be set when running workflows on Amazon EC2. This implementation relies on s3cmd command line client to create the bucket. An entry for transformation amazon::s3cmd needs to exist in the Transformation Catalog for this to work.

pegasus.file.cleanup.scope

System:	Pegasus
Since:	2.3.0
Type:	enumeration
Value[0]:	fullahead
Value[1]:	deferred
Default:	fullahead

By default in case of deferred planning InPlace file cleanup is turned OFF. This is because the cleanup algorithm does not work across partitions. This property can be used to turn on the cleanup in case of deferred planning.

fullahead	This is the default scope. The pegasus cleanup algorithm does not work across partitions in deferred planning. Hence the cleanup is always turned OFF , when deferred planning occurs and cleanup scope is set to full ahead.
deferred	If the scope is set to deferred, then Pegasus will not disable file cleanup in case of deferred planning. This is useful for scenarios where the partitions themselves are independant (i.e. dont share files). Even if the scope is set to deferred, users can turn off cleanup by specifying --nocleanup option to pegasus-plan.

pegasus.catalog.transformation.mapper

System:	Staging of Executables
Since:	2.0
Type:	enumeration
Value[0]:	All
Value[1]:	Installed
Value[2]:	Staged
Value[3]:	Submit
Default:	All
See also:	pegasus.transformation.selector

Pegasus now supports transfer of statically linked executables as part of the concrete workflow. At present, there is only support for staging of executables referred to by the compute jobs specified in the DAX file. Pegasus determines the source locations of the binaries from the transformation catalog, where it searches for entries of type `STATIC_BINARY` for a particular architecture type. The PFN for these entries should refer to a globus-url-copy valid and accessible remote URL. For transfer of executables, Pegasus constructs a soft state map that resides on top of the transformation catalog, that helps in determining the locations from where an executable can be staged to the remote site.

This property determines, how that map is created.

All	In this mode, all sources with entries of type <code>STATIC_BINARY</code> for a particular transformation are considered valid sources for the transfer of executables. This the most general mode, and results in the constructing the map as a result of the cartesian product of the matches.
Installed	In this mode, only entries that are of type <code>INSTALLED</code> are used while constructing the soft state map. This results in Pegasus never doing any transfer of executables as part of the workflow. It always prefers the installed executables at the remote sites.
Staged	In this mode, only entries that are of type <code>STATIC_BINARY</code> are used while constructing the soft state map. This results in the concrete workflow referring only to the staged executables, irrespective of the fact that the executables are already installed at the remote end.

Submit In this mode, only entries that are of type `STATIC_BINARY` and reside at the submit host (pool local), are used while constructing the soft state map. This is especially helpful, when the user wants to use the latest compute code for his computations on the grid and that relies on his submit host.

pegasus.selector.transformation

System:	Staging of Executables
Since:	2.0
Type:	enumeration
Value[0]:	Random
Value[1]:	Installed
Value[2]:	Staged
Value[3]:	Submit
Default:	Random
See also:	pegasus.catalog.transformation

In case of transfer of executables, Pegasus could have various transformations to select from when it schedules to run a particular compute job at a remote site. For e.g it can have the choice of staging an executable from a particular remote pool, from the local (submit host) only, use the one that is installed on the remote site only.

This property determines, how a transformation amongst the various candidate transformations is selected, and is applied after the property `pegasus.tc` has been applied. For e.g specifying `pegasus.tc` as `Staged` and then `pegasus.transformation.selector` as `INSTALLED` does not work, as by the time this property is applied, the soft state map only has entries of type `STAGED`.

Random In this mode, a random matching candidate transformation is selected to be staged to the remote execution pool.

Installed In this mode, only entries that are of type `INSTALLED` are selected. This means that the concrete workflow only refers to the transformations already pre installed on the remote pools.

Staged In this mode, only entries that are of type `STATIC_BINARY` are selected, ignoring the ones that are installed at the remote site.

Submit In this mode, only entries that are of type `STATIC_BINARY` and reside at the submit host (pool local), are selected as sources for staging the executables to the remote execution pools.

pegasus.execute.*.filesystem.local

System:	Pegasus
Type:	Boolean
Default:	false
Since:	2.1.0
See also:	pegasus.gridstart

Normally, Pegasus transfers the data to and from a directory on the shared filesystem on the head node of a compute site. The directory needs to be visible to both the head node and the worker nodes for the compute jobs to execute correctly.

By setting this property to true, you can get Pegasus to execute jobs on the worker node filesystem. In this case, when the jobs are launched on the worker nodes, the jobs grab the input data from the workflow specific execution directory on the compute site and push the output data to the same directory after completion. The transfer of data to and from the worker node directory is referred to as Second Level Staging (SLS).

It is recommended that a user when setting this property to true, also sets the following property, unless the user is running in a Condor Pool without a shared filesystem and is relying on Condor to transfer the files.

`pegasus.gridstart` `SeqExec`

pegasus.parser.dax.preserver.linebreaks

System:	Pegasus
Type:	Boolean
Default:	false
Since:	2.2.0

The DAX Parser normally does not preserve line breaks while parsing the CDATA section that appears in the arguments section of the job element in the DAX. On setting this to true, the DAX Parser preserves any line line breaks that appear in the CDATA section.