The Pegasus Workflow Planner Properties

Property Documentation

V 3.0.0cvs automatically generated at

2010-03-26 17:15

Contents

14 Workflow Partitioning And Job Clustering Properties   52

15 Logging Properties   56

16 Miscellaneous Properties   58

Index   64

1 Introduction

This file 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 subsitions 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:

```
pegasus.catalog.provenance InvocationSchema
pegasus.catalog.*.db.driver Postgres
pegasus.catalog.*.db.url jdbc:postgresql:${user.name}
pegasus.catalog.*.db.user ${user.name}
pegasus.catalog.*.db.password XXXXXXXX
pegasus.catalog.replica RLS
pegasus.catalog.replica.url rls://smarty.isi.edu
pegasus.catalog.transformation File
pegasus.catalog.transformation.file    ${pegasus.home}/var/sample.tc.data
pegasus.catalog.site XML
pegasus.catalog.site.file              ${pegasus.home}/etc/sample.sites.xml
```

If you are in doubt which properties are actually visible, a sample application called `show-properties` dumps all properties after reading and prioritizing them.

Property : pegasus.home Systems : all Type : directory location string Default : ""$PEGASUS_HOME""

The property pegasus.home cannot be set in the property file. Any of the shell wrapper scripts for the applications will set this property from the value of the environment variable $PEGASUS_HOME. Knowledge about this property is important for developers who want to invoke PEGASUS classes without the shell wrappers.

pegasus.home ""$PEGASUS_HOME""

2 Property Files And Locations

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

2.1 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.

2.2 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.

3 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.

3.1 pegasus.home.datadir

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

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

3.2 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.

3.3 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.

3.4 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.

3.5 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

4 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.

4.1 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.

4.2 pegasus.dir.exec

System	Pegasus
Since	2.0
Type	remote directory location string
Default	(no default)
Old Name	vds.dir.exec

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.

4.3 pegasus.dir.storage

System	Pegasus
Since	2.0
Type	remote directory location string
Default	(no default)
Old Name	vds.dir.storage

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.

4.4 pegasus.dir.storage.deep

System	Pegasus
Since	2.1
Type	Boolean
Default	false
See Also	pegasus.dir.storage, section 4.3 (page 9)
See Also	pegasus.dir.useTimestamp, section 4.1 (page 8)

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

4.5 pegasus.dir.create.strategy

System	Pegasus
Since	2.2
Type	enumeration
Value[0]	HourGlass
Value[1]	Tentacles
Default	Tentacles
Old name	pegasus.dir.create , vds.dir.create.mode, vds.dir.create

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 tentacleas 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.

4.6 pegasus.dir.create.impl

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

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.

5 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.

5.1 pegasus.schema.dax

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

5.2 pegasus.schema.pdax

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

This file is a copy of the XML schema that describes partition dag files that are the result of the partitioning process. 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.

5.3 pegasus.schema.sc

Systems	Pegasus
Since	2.0
Type	XML schema file location string
Value[0]	${pegasus.home.sysconfdir}/sc-2.0.xsd
Default	${pegasus.home.sysconfdir}/sc-2.0.xsd
Old name	vds.schema.poolconfig, vds.schema.sc

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.

5.4 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.

6 Database Drivers For All Relational Catalogs

6.1 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, section 7.4.1 (page 22)

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
```

6.2 pegasus.catalog.*.db.url

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

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
```

6.3 pegasus.catalog.*.db.user

System	PTC, TC, ...
Type	string
Default	(no default)
Example	${user.name}

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
```

6.4 pegasus.catalog.*.db.password

System	PTC, TC, ...
Type	string
Default	(no default)
Example	${user.name}

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
```

6.5 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
```

7 Catalog Properties

7.1 Replica Catalog

7.1.1 pegasus.catalog.replica

System	Pegasus
Since	2.0
Type	enumeration
Value[0]	RLS
Value[1]	LRC
Value[2]	JDBCRC
Value[3]	SimpleFile
Value[4]	File
Value[5]	MRC
Default	RLS
Old name	vds.rc, vds.replica.mode

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

SimpleFile 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!p. 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 SimpleFile, the user additionally needs to specify pegasus.catalog.replica.file property to specify the path to the file based RC.

File This mode is an alias for the SimpleFile mode above.

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.

7.1.2 pegasus.catalog.replica.url

System	Pegasus
Since	2.0
Type	URI string
Default	(no default)
Old name	vds.rc.url, vds.rls.url

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.

7.1.3 **pegasus.catalog.replica.chunk.size**

System	Pegasus, rc-client
Since	2.0
Type	Integer
Default	1000
Old name	vds.rc.chunk.size

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.

7.1.4 **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, section 7.1.5 (page 18)
Old name	vds.rc.lrc.ignore

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

7.1.5 **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, section 7.1.4 (page 18)
Old name	vds.rc.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).

7.1.6 **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.

7.1.7 **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, section 7.1.1 (page 16)
Old name	vds.cache.asrc

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 gencdag 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.

7.2 Site Catalog

7.2.1 pegasus.catalog.site

System	Site Catalog
Since	2.0
Type	enumeration
Value[0]	XML
Value[1]	Text
Default	XML
Old name	vds.pool.mode, vds.sc

The site catalog file is available in two major flavors:

1. The "XML" format is an XML-based file. It is generated using the genpoolconfig client application program that is shipped with Pegasus. The XML input file for Pegasus can be generated in various ways, that can be used exclusively or combined at your option:

   a) It can also be published by converting the new, easier to read and modify local multiline pool config file. An example is provided in sample.pool.config.new. Use this option if you have no network connectivity, or for tests.

2. 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

7.2.2 pegasus.catalog.site.file

System	Site Catalog
Since	2.0
Type	file location string
Default	${pegasus.home.sysconfdir}/sites.xml | ${pegasus.home.sysconfdir}/sites.txt
See also	pegasus.catalog.site, section 7.2.1 (page 20)
Old name	vds.pool.file, vds.sc

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.

7.3 Transformation Catalog

7.3.1 pegasus.catalog.transformation

System	Transformation Catalog
Since	2.0
Type	enumeration
Value[0]	File
Value[1]	Database
Default	File
See also	pegasus.catalog.*.driver, section **??** (page **??**)
See also	pegasus.catalog.transformation.file, section 7.3.2 (page 22)
Old name	vds.tc.mode, vds.tc

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: IN-STALLED, STATIC_BINARY, DYNAMIC_BINARY, SCRIPT. 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 seperated by a comma "," and multiple namespaces are seperated 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 = MySQL|Postgres
pegasus.catalog.transformation.db.url =
jdbc:mysql://[hostname[:port]]/database |
jdbc:postgres://[hostname[:port]]/database
pegasus.catalog.transformation.db.username = dbusername
pegasus.catalog.transformation.db.password = passoword
```

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.

7.3.2 pegasus.catalog.transformation.file

Systems	Transformation Catalog
Type	file location string
Default	${pegasus.home.localstatedir}/tc.data
See also	pegasus.catalog.transformation, section 7.3.1 (page 21)
Old name	vds.tc.file

The transformation catalog is a 6 column textual file that describes in a simple column based format the
mapping from a logical transformation for each pool to the physical application, and optional environment
settings. All concrete planners (Pegasus and Euryale) use this repository to map the lTR from the abstract
DAX into an application invocation. Refer to the user guide for details.

7.4 Provenance Catalog

7.4.1 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, section 6.1 (page 12)

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.

7.4.2 **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, section 6.1 (page 12)

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

7.5 **Work Catalog**

7.5.1 **pegasus.catalog.work**

System	Work Catalog
Since	2.0
Type	Java class name
Value[0]	Database
Default	Database
See also	pegasus.catalog.*.db.driver, section 6.1 (page 12)

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.

8 **Replica Selection Properties**

8.1 **pegasus.selector.replica**

System	Replica Selection
Since	2.0
Type	URI string
Default	default
See also	pegasus.replica.*.ignore.stagein.sites, section **??** (page **??**)
See also	pegasus.replica.*.prefer.stagein.sites, section **??** (page **??**)
Old name	vds.rc.selector, vds.replica.selector

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 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.

Restricted 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.

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.

The good | bad sites are specified by the properties

```
pegasus.replica.*.prefer.stagein.sites
pegasus.replica.*.ignore.stagein.sites
```

where the * in the property name denotes the name of the compute site. A * in the property key is taken to mean all sites.

The pegasus.replica.*.prefer.stagein.sites property takes precedence over pegasus.replica.*.ignore.stagein.sites 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.

Regex 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.

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.

```
pegasus.selector.replica.regex.rank.[value]   regex-expression
```

The value is an integer value that denotes the rank of an expression with a rank value of 1 being the highest rank.

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.

Local 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.

8.2 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, section 8.1 (page 23)
See also	pegasus.selector.replica.*.prefer.stagein.sites, section 8.3 (page 25)
Old name	vds.rc.restricted.sites, vds.replica.*.ignore.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.

8.3 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, section 8.1 (page 23)
See also	pegasus.selector.replica.*.ignore.stagein.sites, section 8.2 (page 25)
Old name	vds.replica.*.prefer.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.

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

System	Replica Selection
Type	Regex Expression
Since	2.3.0
Default	no default
See also	pegasus.selector.replica, section 8.1 (page 23)

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.*
```

9 Site Selection Properties

9.1 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, section 9.2 (page 28)
See also	pegasus.selector.site.timeout, section 9.4 (page 28)
See also	pegasus.selector.site.keep.tmp, section 9.5 (page 29)
See also	pegasus.selector.site.env.*, section ?? (page ??)
Old name	vds.site.selector

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 pegasus.site.selector.path. The environment variables that need to be set to run the site selector can be specified using the properties with a pegasus.site.selector.env. 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 =.

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 teh 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 occurances of the same site. There can be multiple occurances of this key.
input.lfn	is an input LFN, optionally followed by a whitespace and file size. There can be multiple occurances 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.

9.2 pegasus.selector.site.path

System	Site Selector
Since	2.0
Type	String
Old name	vds.site.selector.path

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.

9.3 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.

9.4 pegasus.selector.site.timeout

System	Site Selector
Since	2.0
Type	non negative integer
Default	60
Old name	vds.site.selector.timeout

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

9.5 pegasus.selector.site.keep.tmp

System	Pegasus
Since	2.0
Type	enumeration
Value[0]	onerror
Value[1]	always
Value[2]	never
Default	onerror
Old name	vds.site.selector.keep.tmp

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.

10 Transfer Configuration Properties

10.1 pegasus.transfer.*.impl

System	Pegasus
Type	enumeration
Value[0]	OldGUC
Value[1]	Transfer
Value[2]	T2
Value[3]	Stork
Value[4]	RFT
Value[5]	CRFT
Value[6]	SRM
Value[7]	S3
Default	Transfer
Old name	vds.transfer.mode, vds.transfer, vds.transfer.*.impl
See also	pegasus.transfer.refiner, section 10.2 (page 32)
See also	pegasus.transfer.single.quote, section 10.8 (page 35)
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
pegasus.transfer.stagein.impl	the stage in transfer jobs
pegasus.transfer.stageout.impl	the stage out transfer jobs
pegasus.transfer.inter.impl	the inter pool transfer jobs
pegasus.transfer.setup.impl	the setup transfer job
pegasus.transfer.symlink.impl	the symlink transfer job
pegasus.transfer.*.impl	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

OldGUC This refers to the old guc client that can manage only one file transfer per invocation. This should only be used if you have a very old globus installation.

There should be an entry in the transformation catalog with the fully qualified name as globus-url-copy for all the sites where workflow is run, or on the local site in case of third party transfers.

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.

Transfer This is a wrapper around the globus-url-copy client that is distributed with the PEGASUS. It is able to manage multiple file transfers at the same time. For more details refer to the manpage of transfer. Transfer binary can be found at $PEGASUS_HOME/bin/transfer.

There should be an entry in the transformation catalog with the fully qualified name as transfer for all the sites where workflow is run, or on the local site in case of third party transfers.

T2 This is a successor to the Transfer tool. It is also distributed with the PEGASUS and the binary can be found at $PEGASUS_HOME/bin/T2. In addition, T2 can handle conditional transfers, whereby it signals a success in case of failures during transfers. The input file for the transfer job that is constructed contains multiple source and destination urls for the same transfer. The transfer fails only if all pair candidates are attempted without success.

There should be an entry in the transformation catalog with the fully qualified name as pegasus::T2 for all the sites where workflow is run, or on the local site in case of third party transfers.

Stork This results in the transfers being scheduled using Stork, a data placement scheduler out of Condor. It leads to the transfer submit files generated in stork format. At present it can handle only one file transfer per invocation like OldGUC.

RFT RFT is a GT4 based webservice, that does reliable file transfer.This refers to using the java based rft-client distributed with GLOBUS to do the file transfers.

The RFT transfer implementation parses the following properties.

Name	Default Value
pegasus.transfer.rft.host	localhost
pegasus.transfer.rft.port	8080
pegasus.transfer.rft.binary	true
pegasus.transfer.rft.bs	16000
pegasus.transfer.rft.tcpbs	16000
pegasus.transfer.rft.notpt	false
pegasus.transfer.rft.streams	1
pegasus.transfer.rft.DCAU	true
pegasus.transfer.rft.processes	1
pegasus.transfer.rft.retry	3

There should be an entry in the transformation catalog with the fully qualified name as globus::rft on the local site.

CRFT RFT is a GT4 based webservice, that does reliable file transfer.This refers to using the C based rft-client distributed with GLOBUS to do the file transfers.

There should be an entry in the transformation catalog with the fully qualified name as globus::crft for all the sites where workflow is run, or on the local site in case of third party transfers.

SRM This refers to srmcp client that can talk to a SRM server. The srmcp client against which it was tested is version 1.2.0 out of FNAL. At present it can handle only one file transfer per invocation. Hence, pegasus.transfer.refiner needs to be set to either SDefault or SChain.

There should be an entry in the transformation catalog with the fully qualified name as SRM::srmcp for all the sites where workflow is run, or on the local site in case of third party transfers.

S3 This implementation refers to the s3cmd transfer client that is used for first level staging of data. The data can only be staged between the filesystem on the submit node to the workflow specific bucket on S3. The s3cmd is invoked on the submit node during workflow execution.

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

Symlink This implementation refers to the symlink perl executable shipped with the Pegasus worker package since the 2.4.0 release. This executable only creates symlinks , and does not do any copies.

It is to be used only for the symlink jobs.

10.2 pegasus.transfer.refiner

System	Pegasus
Type	enumeration
Value[0]	SDefault
Value[1]	Default
Value[2]	Bundle
Value[3]	Chain
Value[4]	Condor
Value[5]	Cluster
Default	Default
Since	2.0
Old name	vds.transfer, vds.transfer.*.impl
See also	pegasus.transfer.*.impl, section 10.1 (page 29)
See also	pegasus.transfer.single.quote, section 10.8 (page 35)

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.

SDefault This is the default refinement strategy for the transfer tools that can handle only one file transfer per invocation like OldGUC and Stork. Thus, each file that needs to be transferred will generate one transfer job. This 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.

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_Refine
```

10.3 pegasus.transfer.sls.*.impl

System	Pegasus
Type	enumeration
Value[0]	Transfer
Value[1]	S3
Default	Transfer
Since	2.2.0
See also	pegasus.execute.*.filesystem.local, section 16.8 (page 62)

This implementation 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

Transfer This is a wrapper around the globus-url-copy client that is distributed with the PEGASUS. It is able to manage multiple file transfers at the same time. For more details refer to the manpage of transfer. Transfer binary can be found at $PEGASUS_HOME/bin/transfer.

There should be an entry in the transformation catalog with the fully qualified name as transfer for all the sites where workflow is run, or on the local site in case of third party transfers.

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.

10.4 pegasus.transfer.arguments

System	Pegasus
Since	2.0
Type	String
Default	(no default)
See also	pegasus.transfer.sls.arguments, section 10.5 (page 34)
Old name	vds.transfer.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.

10.5 pegasus.transfer.sls.arguments

System	Pegasus
Since	2.4
Type	String
Default	(no default)
See also	pegasus.transfer.arguments, section 10.4 (page 34)
See also	pegasus.transfer.sls.*.impl, section 10.3 (page 33)

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.

10.6 pegasus.transfer.links

System	Pegasus
Type	boolean
Default	false
Since	2.0
Old name	vds.transfer.mode.links, vds.transfer.links
See also	pegasus.transfer, section **??** (page **??**)
See also	pegasus.transfer.force, section 10.7 (page 35)

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.

10.7 pegasus.transfer.force

System	Pegasus
Since	2.0
Type	boolean
Default	false
Old name	vds.transfer.force, vds.transfer.mode.force

This determines if the force option is to be passed to the underlying transfer mechanism that is being invoked. The force option is set only if the underlying grid transfer executable exposes the force option. The force option generally is -f and is supported by globus-url-copy (Single Transfer Mode), transfer (Mutliple Transfer Mode), T2 (T2 transfer mode).

10.8 pegasus.transfer.single.quote

System	Pegasus
Type	boolean
Default	true
Since	2.0
Old name	vds.transfer.single.quote

This affects only the single transfer mode and all its extensions. It determines whether to introduce quoting around the URLs or not while writing out them in the condor submit files. In certain setups this alleviates the condor quoting problem.

10.9 pegasus.transfer.throttle.processes

System	Pegasus
Since	2.0
Type	integer
Default	4
See also	pegasus.transfer, section **??** (page **??**)
See also	pegasus.transfer.throttle.streams, section 10.10 (page 36)
Old name	vds.transfer.throttle.processes

This property is picked up when transfer mode (pegasus.transfer) is multiple. In this mode, multiple files are transferred using a transfer executable that comes with the PEGASUS system. This transfer executable attempts to transfer multiple files by spawning multiple g-u-c processes. By default a maximum of 4 processes are spawned to transfer the files. Using this one can change the number of processes that are spawned by the transfer executable.

10.10 pegasus.transfer.throttle.streams

System	Pegasus
Since	2.0
Type	integer
Default	1
See also	pegasus.transfer, section **??** (page **??**)
See also	pegasus.transfer.throttle.processes, section 10.9 (page 36)
Old name	vds.transfer.throttle.streams

Whatever the transfer mode specified, at present each uses globus-url-copy (g-u-c) as the underlying transfer mechanism. g-u-c can open multiple streams to do the ftp data transfer. This property can be used to set the number of streams that are used to transfer one file by underlying g-u-c. It directly maps to the -p option of g-u-c.

10.11 pegasus.transfer.*.thirdparty.sites

System	Pegasus
Type	comma separated list of sites
Default	no default
Old name	vds.transfer.thirdparty.pools, vds.transfer.*thirdparty.sites
Since	2.0

By default Pegasus employs the push/pull model of transfer for transferring files in and out of a site. It does use the third party transfer option that grid ftp servers provide. This list specifies the list of sites on which the user wants third party transfers instead of the normal mode. Normally for a push transfer the

source URL is file://blah and destination URL is gsiftp://blah. However in case of a third party site both the urls would be gsiftp://blah. For all these sites, the transfers are actually scheduled on the submit host (site local) in the scheduler universe. The * in the property name can be replaced to achieve finer grained control to dictate what type of transfer jobs need to be third party enabled for a particular site.

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

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

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

10.12 pegasus.transfer.*.thirdparty.remote

System	Pegasus
Type	comma separated list of sites
Default	no default
Since	2.0
Old name	vds.transfer.*.thirdparty.remote

By default Pegasus schedules all the third party transfers on the submit host i.e. site 'local' in the site catalog. This property, allows the user to run third party transfers on the remote sites, instead of site 'local'. This is useful in case of transfer tools like RFT and CRFT clients that can only do third party transfers. Hence, the default push and pull model of transfer does not work for them where one of the URL's is a file:/// URL.

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

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

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

10.13 pegasus.transfer.staging.delimiter

System	Pegasus
Since	2.0
Type	String
Default	:
See also	pegasus.transformation.selector, section **??** (page **??**)
Old name	vds.transfer.staging.delimiter

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.

10.14 pegasus.transfer.disable.chmod.sites

System	Pegasus
Since	2.0
Type	comma separated list of sites
Default	no default
Old name	vds.transfer.disable.chmod.sites

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 maynot be required. The permissions might be preserved, or there maybe 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.

10.15 pegasus.transfer.proxy

System	Pegasus
Since	2.0
Type	Boolean
Default	false
Old name	vds.transfer.proxy

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.

10.16 pegasus.transfer.*.priority

System	Pegasus
Type	Integer
Default	no default
Since	2.0
See also	pegasus.job.priority, section 16.4 (page 60)
Old name	vds.transfer.*.priority

This property sets the priority of the transfer jobs. It results in the creation of a priority classad in the submit files for the transfer jobs. To give priorities to different types of transfer jobs you can specify the following more specific properties.

Property Name	Applies to
pegasus.transfer.stagein.priority	the stage in transfer jobs
pegasus.transfer.stageout.priority	the stage out transfer jobs
pegasus.transfer.inter.priority	the inter pool transfer jobs

10.17 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 maybe required if the remote computes sites don't allows files transfers from a http server.

11 Gridstart And Exitcode Properties

11.1 pegasus.gridstart

System	Pegasus
Since	2.0
Type	enumeration
Value[0]	Kickstart
Value[1]	NoGridStart
Value[2]	DCLauncher
Default	Kickstart
Old Name	vds.gridstart

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 ${pegasus.home.bin}/kickstart.

NoGridStart 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.

DCLauncher In this mode, all the jobs are launched on the remote site via DataCutter launcher script. This is primarliy used to launch DataCutter jobs that talks to runs in Condor parallel universe. Users can use this mode, to launch jobs via their own script. The script invocation looks as follows

```
script  path/to/user/executable   user executable arguments in DAX
```

To specify the path to the script launching the jobs, the user needs to associate pegasus profile key named gridstart.path.

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

11.2 pegasus.gridstart.kickstart.set.xbit

System	Pegasus
Since	2.4
Type	Boolean
Default	false
See also	pegasus.transfer.disable.chmod.sites, section 10.14 (page 38)

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.

11.3 pegasus.gridstart.kickstart.stat

System	Pegasus
Since	2.1
Type	Boolean
Default	false
See also	pegasus.gridstart.generate.lof, section 11.4 (page 41)

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.

11.4 pegasus.gridstart.generate.lof

System	Pegasus
Since	2.1
Type	Boolean
Default	false
See also	pegasus.gridstart.kickstart.stat, section 11.3 (page 41)

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.

11.5 pegasus.gridstart.invoke.always

System	Pegasus
Since	2.0
Type	Boolean
Default	false
See also	pegasus.gridstart.invoke.length, section 11.6 (page 42)
Old Name	vds.gridstart.invoke.always

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.

11.6 pegasus.gridstart.invoke.length

System	Pegasus
Since	2.0
Type	Long
Default	4000
See also	pegasus.gridstart.invoke.always, section 11.5 (page 42)
Old Name	vds.gridstart.invoke.length

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 overriden by specifying this property.

11.7 pegasus.exitcode.scope

System	Pegasus
Since	2.0
Type	enumeration
Value[0]	all
Value[1]	none
Value[2]	essential
Default	none
Old name	vds.exitcode, vds.exitcode.mode
See also	pegasus.exitcode.arguments, section 11.10 (page 45)

Jobs that are "kickstarted" by a grid launcher report back information about the execution, resource consumption, and - most importantly - the exit code of the remote application. Armed with this knowledge, it is possible to have DAGMan stop the workflow and create a rescue workflow on remote execution errors.

1. In "all" mode, each kickstarted job's invocation record will be parsed by a DAGMan postscript.

2. In "none" mode, the default, remote failures will not abort the DAG.

3. In "essential" mode, only certain classes of remote jobs get the ability to abort the workflow, while other, non-essential, classesof jobs (at present the replica registration jobs) will not have their invocation record abort the workflow.

11.8 pegasus.exitcode.impl

System	Pegasus
Since	2.0
Type	enumeration
Value[0]	exitpost
Value[1]	exitcode
Value[2]	none
Value[3]	any legal identifier (concretely [_A-Za-z][_A-Za-z0-9]*)
Default	no default
See also	pegasus.exitcode.path.[value], section 11.9 (page 44)
See also	pegasus.exitcode.arguments, section 11.10 (page 45)
Old Name	vds.exitcode.impl

Each of the kickstarted jobs can have a postscript associated with them. This postscript is executed on the submit host, after a job has finished execution. By default, each of the GridStart implementations have their own postscripts, that are able to parse their output and determine the exit status. For example, for kickstart PEGASUS provides an exitpost utility that parses the kickstart record and can populate the data into a provenance tracking catalog. However, you can specify your own postscript to be executed if desired.

This property applies to all the jobs in the workflow. The value can be overriden selectively for jobs by associating Dagman profile key POST with the jobs either in DAX, site in the Site Catalog or the transformation in the transformation catalog.

The path to the postscript can be specified by the property pegasus.exitcode.path.[value] and arguments by pegasus.exitcode.arguments, where [value] is the value of this property (pegasus.exitcode.impl).

The following types of postscript are supported.

exitpost This is the postscript that can be found at $PEGASUS_HOME/bin/exitpost, and is by default invoked on all jobs that are launched via kickstart. It is a wrapper around exitcode, that makes backups of the processed files. It assumes that the error file ends in ".err" (to be fixed), and looks into it for certain PBS failures like wall-time exceeded, which may fail a job without producing

good kickstart output. This thin wrapper is the preferred way now to invoke exitcode from within a DAG.

exitcode This is the postscript that can be found at $PEGASUS_HOME/bin/exitcode. It can be invoked on jobs that are launched via kickstart. This is a legacy postscript, and has now been replaced by exitpost.

none This means that no postscript is generated for the jobs. This is useful for MPI jobs that are not launched via kickstart currently.

any legal identifier Any other identifier of the form ([_A-Za-z][_A-Za-z0-9]*), than one of the 3 reserved keywords above, signifies a user postscript. This allows you to specify your own postscript for the jobs in the workflow. The path to the postscript can be specified by the property

```
pegasus.exitcode.path.[value]
```

where [value] is this legal identifier that you have specified. The arguments to be passed to your postscript can be set by the property pegasus.exitcode.arguments. If the path is not set for the user postscript, then the default path $PEGASUS_HOME/bin/exitcode is picked up.

The user postscript is passed the name of the .out file of the job as the last argument on the command line. For e.g. with the following property settings

```
pegasus.exitcode.impl  user_script
pegasus.exitcode.path.user_script  /bin/user_postscript
pegasus.exitcode.arguments  -verbose
```

for a job with submit file a.sub a user postscript invocation will appear in the .dag file as

```
/bin/user_postscript -verbose a.out
```

11.9 pegasus.exitcode.path.[value]

System	Pegasus
Type	string
Default	no default
Since	2.0
See also	pegasus.exitcode.impl, section 11.8 (page 43)
Old Name	vds.exitcode.path.[value]

This property allows you to override the default path to various postscripts. The [value] part of the property name needs to be replaced by the any of the values of the pegasus.exitcode.default property.

For example, if you want to override the path to the user postscript and point to /tmp/user_postscript, you need to set the property pegasus.exitcode.user.path to /tmp/user_postscript . You can set paths to multiple postscripts. A single workflow of yours can use multiple postscripts for different types of jobs, by associating different Dagman profile key POST with the jobs.

For example you can have properties set as follows

```
pegasus.exitcode.path.exitcode = /my/development/pegasus/bin/exitcode
pegasus.exitcode.path.exitmode = /my/standard/pegasus/bin/exitpost
pegasus.exitcode.path.user1 = /a/b/c/d
pegasus.exitcode.path.user2 = /e/f/g/h
```

For some jobs you can choose to have exitcode invoked by associating Dagman profile key POST with value exitcode. For others, you can choose user1 invoked by associated Dagman profile key POST with value user1.

11.10 pegasus.exitcode.arguments

System	Pegasus
Type	string
Default	no default
Since	2.0
See also	pegasus.exitcode.impl, section 11.8 (page 43)
Old Name	vds.exitcode.arguments

This specifies the arguments by which the exitcode is invoked on the kickstart output of a job. It applies to all the jobs, for which exitcode is invoked as determined by pegasus.exitcode property.

11.11 pegasus.exitcode.debug

System	Pegasus
Type	Boolean
Default	no default
Since	1.4.7
See also	pegasus.exitcode.impl, section 11.8 (page 43)
Old Name	vds.exitcode.debug

This property turns on the debug for the exitcode and exitpost postscripts. By default the debug is turned off. On setting the property to true, an exitcode log file being created for each job in the submit directory. The log file created is of the format [JOBNAME].exit.log .

12 Remote Scheduler Properties

12.1 pegasus.remote.scheduler.projects

System	Remote Schedulers
Type	list of kv pairs
Default	(no default)
Example	jazz=PDQ,pnnl=emsl12491
Old Name	vds.scheduler.remote.projects

This property allows to set the *project* name that is to be used for each pool. Usually, such project names are specific to a small set of users, and can not be well set in the pool configuration file. Please note that the key is the pool handle, and the value is the project name. Setting the project will result in the generation of an RSL term (project=xxxx) for the matching pool handle.

The value specified by this property can be overrided by Globus profile key named project, that can be specified either in the transformation catalog, the site catalog or the DAX.

12.2 pegasus.remote.scheduler.queues

System	Remote Schedulers
Type	list of kv pairs
Default	(no default)
Example	jazz=PDQ,pnnl=emsl12491
Old Name	vds.scheduler.remote.queues

This property allows to set the *queue* name that is to be used for each pool. Usually, such queue names are specific to single users, and can thus not be well set in the pool configuration file. Please note that the key is the pool handle, and the value is the queue name. The property is applicable to any remote scheduling system that employs named queues, e.g. PBS or LSF. Setting the queue will result in the generation of an RSL clause (queue=xxxx) for the matching pool handle.

The value specified by this property can be overrided by Globus profile key named queue, that can be specified either in the transformation catalog, the site catalog or the DAX.

12.3 pegasus.remote.scheduler.min.[time]

System	Remote Schedulers
Type	list of kv pairs
Since	2.0
Default	(no default)
Example	jazz=10,pnnl=20
Old Name	vds.scheduler.remote.maxwalltimes, vds.scheduler.remote.min.[time]

This property allows to set a minumum max time for your jobs on each pool. This property is applicable to any remote scheduling system that employs walltimes like PBS. Setting the walltime will result in the generation of an RSL (Resource Specification Language) clause (maxwalltime=xxxx) for the matching pool handle. Please note that most if all scheduling systems that use this kill the job if the jobs running time exceed the advertised walltime in the job description.

The [time] part of the property name needs to be replaced by the any of the following time RSL keys supported by Globus GRAM.

maxwalltime Explicitly set the maximum walltime for a single execution of the executable. The units is in minutes.

maxtime The maximum walltime or cputime for a single execution of the executable. Walltime or cputime is selected by the GRAM scheduler being interfaced. The units is in minutes.

maxcputime Explicitly set the maximum cputime for a single execution of the executable. The units is in minutes.

More information on Globus RSL can be found at http://www.globus.org/toolkit/docs/2.4/gram/gram_rsl_parameters.html

The value specified by this property can be overrided by Globus profile key named maxwalltime, that can be specified either in the transformation catalog, the site catalog or the DAX.

13 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.

13.1 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.

13.2 pegasus.condor.release

System	Condor
Type	non-negative integer
Default	3
Since	2.0
See also	pegasus.condor.remove, section 13.3 (page 48)
See also	pegasus.condor.retry, section **??** (page **??**)
Old Name	vds.scheduler.condor.release

This property determines the number of release attempts that are written into the submit file. Condor will hold jobs on certain kinds of failures. Many known failing conditions are transient, thus, if the job is automatically release after a short time, it usually progresses.

13.3 pegasus.condor.remove

System	Condor
Type	non-negative integer
Default	3
See also	pegasus.condor.release, section 13.2 (page 48)
See also	pegasus.condor.retry, section **??** (page **??**)
Since	2.0
Old Name	vds.scheduler.condor.remove

This property determines the number of hold attemts that happen before Condor removes the job from the queue. This value is tied to the number of release attempts. Hence, Pegasus enforces release > remove >= 0. A value of zero indicates, that the user does not want the job to be removed from the queue. In that case, no periodic_remove statement would be constructed. If the property is not set, the value of pegasus.condor.release is used. If both the properties are unset, then the default value of 3 is used.

13.4 pegasus.condor.output.stream

System	Condor
Type	Boolean
Value[0]	false
Value[1]	true
Default	true
Since	2.0
See also	pegasus.condor.error.stream, section 13.5 (page 49)
Old Name	vds.scheduler.condor.output.stream

By default, GASS streams back stdout continuously. In this default mode, it will require additional filedescriptors on the gatekeeper. Recent versions of Globus and Condor allow the content of stdout

to be streamed back after the job is done. While no content is visible during the execution of the job, it saves precious gatekeeper resources.

13.5 pegasus.condor.error.stream

System	Condor
Type	Boolean
Value[0]	false
Value[1]	true
Default	true
Since	2.0
See also	pegasus.condor.output.stream, section 13.4 (page 48)
Old Name	vds.scheduler.condor.error.stream

By default, GASS streams back stderr continuously. In this default mode, it will require additional filedescriptors on the gatekeeper. Recent versions of Globus and Condor allow the content of stderr to be streamed back after the job is done. While no content is visible during the execution of the job, it saves precious gatekeeper resources.

13.6 pegasus.dagman.retry

System	Condor DAGMan
Type	non-negative integer
Since	2.0
Default	3
See also	pegasus.condor.release, section 13.2 (page 48)
See also	pegasus.condor.remove, section 13.3 (page 48)
Old Name	vds.scheduler.condor.retry

This property determines the number of attempts DAGMAN makes to execute a job in case of failure. In case of deferred planning this can be used to do replanning. A failure during the execution of a partition can trigger the execution of Pegasus via a prescript, that may result in a different plan for the partition being generated and executed. Note, this is different from pegasus.scheduler.condor.release which results in condor retrying the same job (submit file) when a job goes in HOLD state.

13.7 pegasus.dagman.maxpre

System	DAGman wrapper
Type	integer
Default	20
Document	http://www.cs.wisc.edu/condor/manual/v6.6/condor_submit_dag.html

The pegasus-submit-dag wrapper processes properties to set DAGMan commandline arguments. The argument sets the maximum number of PRE scripts within the DAG that may be running at one time. If this option is set to 0, the default number of PRE scripts is unlimited. The PEGASUS system throttles artificially to a maximum of 20 PRE scripts.

13.8 pegasus.dagman.maxpost

System	DAGman wrapper
Type	integer
Default	20
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. The argument sets the maximum number of POST scripts within the DAG that may be running at one time. If this option is set to 0, the default number of POST scripts is unlimited. The PEGASUS system throttles artificially to a maximum of 20 POST scripts.

13.9 pegasus.dagman.maxjobs

System	DAGMan wrapper
Type	integer
Default	200
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. The argument sets the maximum number of jobs within the DAG that will be submitted to Condor at one time. If the option is omitted, the default number of jobs is unlimited. The PEGASUS system throttles artificially to a maximum of 200 simultaneous jobs that are visible to the Condor system. You may want to raise this bar.

13.10 pegasus.dagman.maxidle

System	DAGMan wrapper
Type	integer
Default	None
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. The argument sets the maximum number of idle jobs within the DAG that will be submitted to Condor at one time. If the option is omitted, the default number of idle jobs is unlimited.

13.11 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.

13.12 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.

13.13 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

14 Workflow Partitioning And Job Clustering Properties

14.1 pegasus.partitioner.parser.dax.callback

System	Partitioner
Type	String
Default	DAX2Graph
Since	2.4.0

While partitioning the workflow, different partitioners may require different DAX callbacks to be loaded. These callbacks determine how the graph structure that is used by the partitioner is loaded.

Users will rarely need to specify this property , unless plugging in their own partitioner that requires implementation of a new DAX callback.

In case of label based partitioning DAX2LabelGraph callback is automatically loaded by the DAXCallbackFactory class.

14.2 pegasus.partitioner.label.key

System	Partitioner
Type	String
Default	label
Since	2.0
Old name	vds.label.key, vds.partitioner.label.key
See also	pegasus.clusterer.label.key, section 14.9 (page 55)

While partitioning the workflow into partitions, you can optionally label your graph to control how partitioning happens using a label based partitioning scheme. This is done by associating a profile key in the PEGASUS namespace with the jobs in the DAX. Each job that has the same value/label for this profile key, is put in the same partition.

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

14.3 pegasus.partitioner.horziontal.collapse.[txname]

System	Partitioner
Type	Integer
Default	no default
Since	2.0
See also	pegasus.partitioner.horziontal.bundle.[txname], section 14.4 (page 53)
Old Name	vds.partitioner.horizontal.bundle.[txname]

In case of horizontal partitioning, you can specify a "collapse" factor for transformations. The collapse factor for a transformation specifies the number of jobs referring to that transformation at a level that go into a single partition.

This property allows you to specify the collapse factors transformations where the [txname] in the property has to be replaced by the logical name of the transformation.

For example, if in a workflow you have 10 jobs referring to a transformation with a logical name txA at a particular level(l) of the workflow, and you have properties set as follows

```
pegasus.partitioner.horizontal.collapse.txA 4
```

The above would result 3 partitions being created, 2 having 4 jobs each and one having 2 jobs for that level (l).

14.4 pegasus.partitioner.horziontal.bundle.[txname]

System	Partitioner
Type	Integer
Default	no default
Since	2.0
See also	pegasus.partitioner.horziontal.collapse.[txname], section 14.3 (page 53)
Old Name	vds.partitioner.horizontal.bundle.[txname]

In case of horizontal partitioning, you can specify a "bundle" factor for transformations. The bundle factor for a transformation specifies the number of partitions that are created for each level of the workflow containing jobs that refer to that transformation.

This property allows you to specify the bundle factors transformations where the [txname] in the property has to be replaced by the logical name of the transformation.

For example, if in a workflow you have 10 jobs referring to a transformation with a logical name txA at a particular level(l) of the workflow, and you have properties set as follows

```
pegasus.partitioner.horizontal.bundle.txA 4
```

The above would result 4 partitions being created, 2 having 2 jobs each and 2 having 3 jobs for that level (l).

14.5 pegasus.clusterer.job.aggregator

System	Job Clustering
Since	2.0
Type	String
Value[0]	seqexec
Value[1]	mpiexec
Default	seqexec
Old Name	vds.job.aggregator

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.

14.6 pegasus.clusterer.job.aggregator.seqexec.log

System	Job Clustering
Type	Boolean
Default	false
Since	2.3
See also	pegasus.clusterer.job.aggregator, section 14.5 (page 54)
See also	pegasus.clusterer.job.aggregator.seqexec.log.global, section 14.7 (page 55)

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.

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

System	Job Clustering
Type	Boolean
Default	true
Since	2.3
See also	pegasus.clusterer.job.aggregator, section 14.5 (page 54)
See also	pegasus.clusterer.job.aggregator.seqexec.log, section 14.6 (page 54)
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.

14.8 pegasus.clusterer.job.aggregator.seqexec.firstjobfail

System	Job Clustering
Type	Boolean
Default	true
Since	2.2
See also	pegasus.clusterer.job.aggregator, section 14.5 (page 54)

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.

14.9 pegasus.clusterer.label.key

System	Job Clustering
Type	String
Default	label
Since	2.0
See also	pegasus.partitioner.label.key, section 14.2 (page 52)
Old Name	vds.clusterer.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.

15 Logging Properties

15.1 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, section 15.2 (page 56)

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
```

15.2 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, section 15.2 (page 56)

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 se18-gda.dax  - STARTED
event.pegasus.parsing.dax dax.id se18-gda-nested.dax  - STARTED
event.pegasus.parsing.dax dax.id se18-gda-nested.dax  - FINISHED
job.id jobGDA
job.id jobGDA query.name getpredicted performace time 10.00
event.pegasus.ranking dax.id se18-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=se18-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=se18-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-ea1cee88727d \
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-ea1cee88727d \
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
```

15.3 pegasus.log.*

System	Pegasus
Since	2.0
Type	String
Default	No default
Old Name	vds.log.*

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.

15.4 pegasus.log.metrics

System	Pegasus
Since	2.1.0
Type	Boolean
Default	true
See also	pegasus.log.metrics.file, section 15.5 (page 58)

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.

15.5 pegasus.log.metrics.file

System	Pegasus
Since	2.1.0
Type	Boolean
Default	${pegasus.home}/var/pegasus.log
See also	pegasus.log.metrics, section 15.4 (page 58)

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

16 Miscellaneous Properties

16.1 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 .

16.2 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.

16.3 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.

16.4 pegasus.job.priority

System	Pegasus
Type	Integer
Default	no default
Since	2.0
See also	pegasus.transfer.*.priority, section 10.16 (page 39)
Old Name	vds.job.priority

This property sets the priority for all the condor jobs. The priority appears as a classad in the condor submit files. Optionally, user can also set the priorities for the transfer jobs separately by specifying the property pegasus.transfer.*.priority.

16.5 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, section ?? (page ??)
Old Name	vds.tc.mapper

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 STATIC_BINARY 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 INSTALLED 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 STATIC_BINARY 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.

16.6 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, section 7.3.1 (page 21)
Old Name	vds.transformation.selector

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.

16.7 pegasus.auth.gridftp.timeout

System	Pegasus
Since	2.0
Type	non negative integer
Default	120
Old Name	vds.auth.gridftp.timeout

This property sets the socket timeout on the socket that is opened to the remote gridftp server for authentication purposes.

16.8 pegasus.execute.*.filesystem.local

System	Pegasus
Type	Boolean
Default	false
Since	2.1.0

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.

In the case, where the worker nodes cannot see the filesystem of the head node there needs to be a Second Level Staging (SLS) process that transfers the data from the head node to a directory on the worker node tmp. To achieve this, Pegasus uses the pre-job and post-job feature of kickstart to pull the input data from the head node and push back the output data of a job to the head node.

Even though we do SLS, Pegasus still relies on the existence of a shared file system due to the following two reasons

```
a) for the transfer executable to pick up the proxy, that we
transfer from the submit host to the head node.

b) to access sls input and output files that contain the file
transfer urls to manage the transfer of data to worker node and
back to headnode.
```

Additionally, if you are running your workflows on a Condor pool, one can bypass the use of kickstart to do the SLS. Please contact pegasus@isi.edu for more details of this scenario. In this case, the workflows generated by Pegasus have been shown to run in total non shared filesystem environment.

16.9 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.

Index