The GriPhyN Virtual Data System Properties

Property Documentation

V 1.4.6 automatically generated at

2006-07-28 11:58

Contents

1 Introduction

This file is the reference guide to all properties regarding the Virtual Data System, 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, ${vds.home} means that the value depends on the value of the vds.home property plus any noted additions. You can use this notation to refer to other properties, though the extent of the subsitutions 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 *${vds.home.sysconfdir}/properties*, have the lowest priority. These properties are expected to be set up by the submit host's administrator.
2. The properties defined in the user property file *${user.home}/.vdsrc* 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:

```
vds.db.vdc.schema        ChunkSchema
vds.db.ptc.schema        InvocationSchema
vds.db.*.driver          Postgres
vds.db.*.driver.url      jdbc:postgresql:${user.name}
vds.db.*.driver.user     ${user.name}
vds.db.*.driver.password XXXXXXXX
vds.rc    rls
vds.rc.url               rls://sheveled.mcs.anl.gov
vds.exitcode             all
vds.transfer             multiple
vds.sc                   xml
vds.sc.file              ${vds.home}/contrib/Euryale/Grid3/pool.config
```

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

1.1 vds.home

Systems	all
Type	directory location string
Default	""$VDS_HOME""

The property vds.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 $VDS_HOME. Knowledge about this property is important for developers who want to invoke VDS classes without the shell wrappers.

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 vds.properties

Systems	all
Type	file location string
Default	${vds.home.sysconfdir}/properties

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

2.2 vds.user.properties

Systems	all
Type	file location string
Default	${user.home}/.vdsrc

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 .vdsrc 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 VDS 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-vdsrc 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 vds.home.datadir

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

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

3.2 vds.home.sysconfdir

Systems	all
Type	directory location string
Default	${vds.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 vds.home.sharedstatedir

Systems	all
Type	directory location string
Default	${vds.home}/com

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

3.4 vds.home.localstatedir

Systems	all
Type	directory location string
Default	${vds.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, file-based VDCs, and the file-based replica catalog of the shell planner.

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 vds.dir.exec

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

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

4.2 vds.dir.storage

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

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

4.3 vds.dir.create.mode

New Name	vds.dir.create, section 4.4 (page 9)
Until	1.3.9

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

4.4 vds.dir.create

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

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.

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 VDS. The schema backups in the installed file-system permit VDS operations without being online.

5.1 vds.schema.vdl

Systems	VDC (Chimera)
Type	XML schema file location string
Default	${vds.home.sysconfdir}/vds-1.21.xsd

This file is a copy of the XML schema that describes VDLx files. VDLx files and fragments are used in various places throughout the abstract planning 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.

5.2 vds.schema.dax

Systems	all
Type	XML schema file location string
Value[0]	${vds.home.sysconfdir}/dax-1.5.xsd
Value[1]	${vds.home.sysconfdir}/dax-1.6.xsd
Value[2]	${vds.home.sysconfdir}/dax-1.7.xsd
Value[3]	${vds.home.sysconfdir}/dax-1.8.xsd
Value[4]	${vds.home.sysconfdir}/dax-1.9.xsd
Value[5]	${vds.home.sysconfdir}/dax-1.10.xsd
Default	${vds.home.sysconfdir}/dax-1.8.xsd (Pegasus)
Default	${vds.home.sysconfdir}/dax-1.10.xsd (all others)

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.

5.3 vds.schema.pdax

Systems	Pegasus
Type	XML schema file location string
Value[0]	${vds.home.sysconfdir}/pdax-1.1.xsd
Default	${vds.home.sysconfdir}/pdax-1.1.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.

5.4 vds.schema.poolconfig

New Name	vds.schema.sc, section 5.5 (page 12)
Until	1.3.10

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

5.5 vds.schema.sc

Systems	Pegasus
Since	1.3.10
Type	XML schema file location string
Value[0]	${vds.home.sysconfdir}/gvds-poolcfg-1.3.xsd
Value[1]	${vds.home.sysconfdir}/gvds-poolcfg-1.4.xsd
Value[2]	${vds.home.sysconfdir}/gvds-poolcfg-1.5.xsd
Default	${vds.home.sysconfdir}/gvds-poolcfg-1.5.xsd
Old name	vds.schema.poolconfig

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.

5.6 vds.schema.ivr

Systems	all
Type	XML schema file location string
Value[0]	${vds.home.sysconfdir}/iv-1.2.xsd
Value[1]	${vds.home.sysconfdir}/iv-1.3.xsd
Value[2]	${vds.home.sysconfdir}/iv-1.4.xsd
Value[3]	${vds.home.sysconfdir}/iv-1.6.xsd
Default	${vds.home.sysconfdir}/iv-1.6.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.

5.7 vds.db.rc

Until	1.4
System	Shell planner (Chimera)
Type	file location string
Default	${vds.home.localstatedir}/rc.data

The shell planner completely by-passes any replica management catalogs defined by the rest of VDS. This is due to the very local nature of the shell planner. The shell planner uses a simple three column textual file to store the mapping of pool to LFNs to PFNs. The pool must be string "local".

Note, this property is inoperative. The new shell planner uses the new RC API to access any of the various replica catalog implementations.

5.8 vds.db.tc

Until	1.4
System	Shell planner (Chimera)
Type	file location string
Default	${vds.home.localstatedir}/tc.data

The shell planner completely by-passes any replica management catalogs defined by the rest of VDS. This is due to the very local nature of the shell planner. The shell planner uses a simple three column textual file to store the mapping of pool to transformation names to application names. And optional fourth column may contain environment specification. The pool must be string "local". It is effectively the "OldFile" format.

Please note that the file location for the shell planner's transformation catalog clashes with the general VDS file-based transformation catalogs - for historic reasons. Hence, it is recommended that you override this property on the command-line, and provide an alternate location for the simplistic transformation catalog.

Note that the new shell planner uses the regular transformation catalog API and its properties. This property is inoperative.

6 Virtual Data And Provenance Tracking Catalog

6.1 vds.db.vdc.schema

System	VDC, PTC, TC, ...
Type	Java class name
Value[0]	SingleFileSchema
Value[1]	ChunkSchema
Value[2]	AnnotationSchema
Value[3]	NXDSchema
Default	org.griphyn.vdl.dbschema.SingleFileSchema
See also	vds.db.*.driver, section 7.1 (page 15)

This property denotes the schema that is being used to access a VDC. The VDC is a multi-layer architecture, with the dbschema sitting on top of the dbdrivers. Some schemas, like the default schema, do not access the driver level at all.

Currently available are only SingleFileSchema and ChunkSchema. These 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 VDC interface, users can provide their own classes here. Anything further properties in "vds.db.vdc.schema.*" will be copied into the schema properties, and thus made available to the schema.

SingleFileSchema This schema uses a file-based VDC. It is the default, as files are thought to be always available.

ChunkSchema This is a small VDC schema that uses an underlying rDBMS. Data outside the keys is stored in XML chunks, thus the name.

AnnotationSchema This is an extension of the ChunkSchema by providing annotation to filenames, transformations, derivations, formal arguments.

NXDSchema This is a schema storing XML into an native XML database. The database driver properties are not used.

6.2 vds.db.vdc.schema.xml.url

System	VDC (Chimera), SingleFileSchema, ChunkSchema
Type	VDLx XML Schema location
Default	${vds.schema.vdl}

Various schemas manipulate VDLx internally, and need the knowledge of the schemas location to construct their parsers. This schema property enables users to provide backward compatible parsers for VDC data independent of the new data being read from files.

6.3 vds.db.vdc.schema.file.store

System	VDC (Chimera), SingleFileSchema
Type	file location string
Default	${vds.home.localstatedir}/vdc.db

The VDC can be stored in a single file, which is the default mode of operation. This property determines the basename of the file. Extensions allow for a degree of rollbacks.

6.4 vds.db.ptc.schema

System	Provenance Tracking Catalog (PTC)
Type	Java class name
Value[0]	InvocationSchema
Value[1]	NXDInvSchema
Default	(no default)
See also	vds.db.*.driver, section 7.1 (page 15)

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. Anything further properties in "vds.db.ptc.schema.*" will be copied into the schema properties, and thus made available to the schema.

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 Database Drivers For All Relational Catalogs

7.1 vds.db.*.driver

System	VDC (Chimera)
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	vds.db.vdc.schema, section 6.1 (page 13)
See also	vds.db.ptc.schema, section 6.4 (page 14)

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 GVDS.

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 vdc, 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.

7.2 vds.db.*.driver.url

System	VDC, 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 vds.db.schema.driver.* keys or vds.db.*.driver.* 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
```

7.3 vds.db.*.driver.user

System	VDC, 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.

7.4 vds.db.*.driver.password

System	VDC, 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.

7.5 vds.db.*.driver.*

System	VDC, PTC, TC, ...

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 "vds.db.driver." prefix from them.

Postgres 7.3 parses the following properties:

```
vds.db.*.driver.user
vds.db.*.driver.password
vds.db.*.driver.PGHOST
vds.db.*.driver.PGPORT
vds.db.*.driver.charSet
vds.db.*.driver.compatible
```

MySQL 4.0 parses the following properties:

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

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

```
vds.db.*.driver.User
vds.db.*.driver.Password
vds.db.*.driver.DatabaseName
vds.db.*.driver.ServerName
vds.db.*.driver.HostProcess
vds.db.*.driver.NetAddress
vds.db.*.driver.PortNumber
vds.db.*.driver.ProgramName
vds.db.*.driver.SendStringParametersAsUnicode
vds.db.*.driver.SelectMethod
```

8 Replica Catalog Properties

8.1 vds.replica.mode

New Name	vds.rc, section 8.2 (page 18)
Until	1.3.9

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

8.2 vds.rc

System	Pegasus, shell planner
Since	1.3.9
Type	enumeration
Value[0]	rls
Value[1]	lrc
Value[2]	JDBCRC
Value[3]	SimpleFile
Default	rls
Old name	vds.replica.mode

The replica mode controls which replica management mechanism the planner will use. The recommended value is "rls". All other options are experimental. Each option will require a different subsequent properties.

RLS (Replica Location Service) is a distributed replica catalog, which ships with GT4. Details about RLS can be found at http://www.globus.org/toolkit/data/rls/

8.3 vds.rls.url

New Name	vds.rc.url, section 8.4 (page 18)
Until	1.3.9

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

8.4 vds.rc.url

System	Pegasus, RLS
Since	1.4.0
Type	URI string
Default	(no default)
Old name	vds.rls.url

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

8.5 vds.rc.chunk.size

System	Pegasus, rc-client
Since	1.4.5
Type	Integer
Default	1000

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

8.6 vds.rls.exit

System	Pegasus, RLS
Type	enumeration
Value[0]	error
Value[1]	never
Default	error

The Pegasus code first queries the RLI to find the LRC urls and then queries each individual LRC. If the mode is set to error (default), the Pegasus will exit if it encounters any LRC to which it cannot authenticate or connect. If the mode is set to never the any LRC that cannot be authenticated or connected to is skipped unless if all of them fail then the Pegasus exits.

8.7 vds.rls.query

System	Pegasus, RLS
Type	enumeration
Value[0]	bulk
Value[1]	multiple
Default	bulk

The RLS server offers significant performance improvement when being bulk queried for filenames. Bulk operations are only available starting with version 2.0.5 of the RLS software. The possible values for this key are "bulk" and "multiple". With the current RLS version being 2.0.7, "bulk" is the default. For backward compatibility, you can use "multiple".

8.8 vds.rls.query.attrib

System	Pegasus, RLS
Since	1.2.0
Type	boolean
Value[0]	true
Value[1]	false
Default	false
See also	vds.transfer.links, section 12.7 (page 35)
See also	vds.rls.query, section 8.7 (page 19)

Pegasus prefers that each PFN be associated with a pool attribute that lets it associate a PFN with a gvds pool. This can be used in conjunction with vds.transfer.links and multiple mode of vds.transfer to create symbolic links to the PFNs. This is only when a PFN is found to be at the same execution pool where Pegasus has scheduled the job. This property should be used if the query mode to rls is bulk in order to enable bulk query of attributes in RLS. This bulk query of attributes in RLS is available from version 2.0.9 or later.

8.9 vds.rc.lrc.ignore

System	Pegasus, RLS
Since	1.3.6
Type	comma separated list of LRC urls
Default	(no default)
See also	vds.rls.exit, section 8.6 (page 19)
See also	vds.rc.lrc.restrict, section 8.10 (page 20)

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 vds.rls.lrc.ignore=rls://smarty.isi.edu,usc.edu

8.10 vds.rc.lrc.restrict

System	Pegasus, RLS
Since	1.3.9
Type	comma separated list of LRC urls
Default	(no default)
See also	vds.rc.lrc.ignore, section 8.9 (page 20)

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 "vds.rc.lrc.ignore" has

a higher priority than "vds.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).

8.11 vds.cache.asrc

System	Pegasus
Since	1.3.10
Type	Boolean
Value[0]	false
Value[1]	true
Default	false
See also	vds.rc, section 8.2 (page 18)

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 vds.rc) 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.

9 Site Catalog And Transformation Catalog Properties

9.1 vds.pool.mode

New Name	vds.sc, section 9.2 (page 22)
Until	1.3.9

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

9.2 vds.sc

System	Pegasus, MDS, PoolConfig
Since	1.3.9
Type	enumeration
Value[0]	XML
Value[1]	Text
Default	XML
Old name	vds.pool.mode

The site catalog file is available in two major flavors:

1. The old textual format employs multiple columns in a textual file, see sample.pool.config.old. This format is NO LONGER SUPPORTED and has been removed. The users should now convert the textual format to the xml format before using Pegasus. This can be done using the client genpool-config found in $VDS_HOME/bin directory.

2. 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) The site catalog file can be generated using information that is published in MDS (see VDS user guide document). This is the recommended way.

   b) 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.

3. The "Text" format is a multiline site catalog format. It is described in the site catalog guide. It can be directly given to Pegasus starting with VDS-1.4

9.3 vds.pool.file

New Name	vds.sc.file, section 9.4 (page 23)
Until	1.3.9

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

9.4 vds.sc.file

System	Pegasus
Since	1.3.9
Type	file location string
Default	${vds.home.sysconfdir}/sites.xml | ${vds.home.sysconfdir}/sites.txt
Old name	vds.pool.file
See also	vds.sc, section 9.2 (page 22)

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 vds.sc . vds.sc denotes the type of site catalog being used.

9.5 vds.giis.host

System	Pegasus
Type	string
Default	(no default)
See also	vds.sc, section 9.2 (page 22)

This property needs to be set if you set the vds.pool to xml and you wish to obtain the pool config information dynamically using MDS along with or without local pool config support files. The property needs to be set to the hostname and optional port of the GIIS service that is aggregating pool information, e.g. smarty.isi.edu:2135

9.6 vds.giis.dn

System	Pegasus
Type	X.400 string
Default	(no default)
See also	vds.sc, section 9.2 (page 22)

If the vds.giis.host is set, and MDS is used to obtain the pool configuration, the ’distinguished name’ to contact the GIIS needs to be specified with this property, e.g. "MDS-vo-name=GRIPHYN,o=Grid"

9.7 vds.tc.mode

New Name	vds.tc, section 9.8 (page 24)
Until	1.3.9

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

9.8 vds.tc

System	Pegasus, TC
Since	1.3.9
Type	enumeration
Value[0]	single
Value[1]	multiple
Value[2]	OldFile
Value[3]	File
Value[4]	Database
Default	File
Old name	vds.tc.mode
See also	vds.db.*.driver, section 7.1 (page 15)
See also	vds.tc.file, section 9.9 (page 25)

The Transformation Catalog (TC) is currently a three-column file. Similar to the pool configuration, for speed and memory consumption, two parse modes are available:

single, multiple,OldFile In these modes, the old TC file is read once and cached in main memory. This yields a good performance while consuming some memory. We recommend to use this methods for files with less than 10000 entries.

The old file format consists of for tabulator-separated columns: The resource identifier, the logical transformation name, the physical installation path to the application, and optional environment variables or the string `null`. The old format understands the logical transformation specification in both, the ancient format using underscores ns__id_vs and the modern colonized version ns::id:vs.

The old format has been DEPRECATED. Please use the conversion tool

```
$VDS_HOME/contrib/tc-converter/tc-old2new
```

to convert to the new format or run

```
tc-client -Dvds.tc=single -q B > tc.data.net
```

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

1. The resource ID is represented in the first column.
2. The logical transformation uses the colonized format ns::name:vs.
3. The path to the application on the system

4. The installation type is identified by one of the following keywords - all upper case: INSTALLED, 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 $VDS_HOME/sql/create-my-init.sql followed by $VDS_HOME/sql/create-my-tc.sql .

```
The following properties need to be set
vds.db.tc.driver = MySQL|Postgres
vds.db.tc.driver.url =
jdbc:mysql://[hostname[:port]]/database |
jdbc:postgres://[hostname[:port]]/database
vds.db.tc.driver.username = dbusername
vds.db.tc.driver.password = passoword
```

If the vds.db.tc.driver.* properties are not defined, the database implementation picks up the properties specified by vds.db.*.driver.* .

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 vds.tc to the TC implementation class.

9.9 vds.tc.file

Systems	all
Type	file location string
Default	${vds.home.localstatedir}/tc.data
See also	vds.tc, section 9.8 (page 24)

The transformation catalog is a 3+ 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.

10 Replica Selection Properties

10.1 vds.rc.selector

New Name	vds.replica.selector, section 10.2 (page 26)
Since	1.4.0
Until	1.4.5

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

10.2 vds.replica.selector

System	Pegasus, RLS
Since	1.4.5
Type	URI string
Default	default
See also	vds.replica.*.ignore.stagein.sites, section 10.4 (page 27)
See also	vds.replica.*.prefer.stagein.sites, section 10.5 (page 27)

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

```
vds.replica.*.prefer.stagein.sites
vds.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 vds.replica.*.prefer.stagein.sites property takes precedence over vds.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.

10.3 vds.rc.restricted.sites

New Name	vds.replica.*.ignore.stagein.sites, section 10.4 (page 27)
Since	1.4.0
Until	1.4.5
See also	vds.replica.*.prefer.stagein.sites, section 10.5 (page 27)

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

10.4 vds.replica.*.ignore.stagein.sites

System	Pegasus
Type	comma separated list of sites
Since	1.4.5
Default	no default
See also	vds.rc.selector, section 10.1 (page 26)
See also	vds.replica.*.prefer.stagein.sites, section 10.5 (page 27)

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 vds.replica.*.ignore.stagein.sites to usc means that ignore all replicas from site usc for staging in to any compute site. Setting vds.replica.isi.ignore.stagein.sites to usc means that ignore all replicas from site usc for staging in data to site isi.

10.5 vds.replica.*.prefer.stagein.sites

System	Pegasus
Type	comma separated list of sites
Since	1.4.5
Default	no default
See also	vds.rc.selector, section 10.1 (page 26)
See also	vds.replica.*.ignore.stagein.sites, section 10.4 (page 27)

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 vds.replica.*.prefer.stagein.sites to usc means that prefer all replicas from site usc for staging in to any compute site. Setting vds.replica.isi.prefer.stagein.sites to usc means that prefer all replicas from site usc for staging in data to site isi.

11 Site Selection Properties

11.1 vds.site.selector

System	Pegasus
Since	1.2.8
Type	enumeration
Value[0]	Random
Value[1]	RoundRobin
Value[2]	NonJavaCallout
Value[3]	Group
Default	Random
See also	vds.site.selector.path, section 11.2 (page 29)
See also	vds.site.selector.timeout, section 11.4 (page 30)
See also	vds.site.selector.keep.tmp, section 11.5 (page 30)
See also	vds.site.selector.env.*, section 11.3 (page 30)

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 vds.site.selector.path. The environment variables that need to be set to run the site selector can be specified using the properties with a vds.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 VDS 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 defaultMANAGEMENT group. The jobs in the default group are handed over to the "Random" Site Selector for scheduling.

11.2 vds.site.selector.path

System	Pegasus
Since	1.2.3
Type	String

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

11.3 vds.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 "vds.site.selector.env." prefix from them. The value of the environment variable is the value of the property.

e.g vds.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.

11.4 vds.site.selector.timeout

System	Pegasus
Since	1.3.0
Type	non negative integer
Default	60

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

11.5 vds.site.selector.keep.tmp

System	Pegasus
Since	1.3.2
Type	enumeration
Value[0]	onerror
Value[1]	always
Value[2]	never
Default	onerror

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

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

12 Transfer Configuration Properties

12.1 vds.transfer.mode

New Name	vds.transfer, section 12.2 (page 31)
Until	1.3.9

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

12.2 vds.transfer

New Name	vds.transfer.*.impl, section 12.3 (page 31)
New Name	vds.transfer.refiner, section 12.4 (page 33)
Since	1.3.9
Until	1.4.4

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

12.3 vds.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
Default	Transfer
Old name	vds.transfer
See also	vds.transfer.refiner, section 12.4 (page 33)
See also	vds.transfer.single.quote, section 12.10 (page 36)
Since	1.4.4

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 vds.transfer.arguments
- associating the VDS profile key transfer.arguments
```

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

Property Name	Applies to
vds.transfer.stagein.impl	the stage in transfer jobs
vds.transfer.stageout.impl	the stage out transfer jobs
vds.transfer.inter.impl	the inter pool transfer jobs
vds.transfer.*.impl	apply to types of transfer jobs

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.

Transfer This is a wrapper around the globus-url-copy client that is distributed with the VDS. 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 $VDS_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 VDS and the binary can be found at $VDS_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 vds::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
vds.transfer.rft.host	localhost
vds.transfer.rft.port	8080
vds.transfer.rft.binary	true
vds.transfer.rft.bs	16000
vds.transfer.rft.tcpbs	16000
vds.transfer.rft.notpt	false
vds.transfer.rft.streams	1
vds.transfer.rft.DCAU	true
vds.transfer.rft.processes	1
vds.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, vds.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.

12.4 vds.transfer.refiner

System	Pegasus
Type	enumeration
Value[0]	SDefault
Value[1]	Default
Value[2]	Bundle
Value[3]	Chain
Default	Default
Old name	vds.transfer
See also	vds.transfer.*.impl, section 12.3 (page 31)
See also	vds.transfer.single.quote, section 12.10 (page 36)

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 stageing 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 stageing 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 vds 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 vds 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.

12.5 vds.transfer.arguments

System	Pegasus
Since	1.4.2
Type	String
Default	(no default)
Old name	vds.transfer

This determines the arguments with which the transfer executable 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 vds profile key transfer.arguments either with the site in the site catalog or the corresponding transfer executable in the transformation catalog.

12.6 vds.transfer.mode.links

New Name	vds.transfer.links, section 12.7 (page 35)
Until	1.3.9

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

12.7 vds.transfer.links

System	Pegasus
Type	boolean
Default	false
Old name	vds.transfer.mode.links
See also	vds.transfer, section 12.2 (page 31)
See also	vds.transfer.force, section 12.9 (page 35)
See also	vds.rls.query.attrib, section 8.8 (page 20)

If this is set, and the transfer mode is set to multiple i.e. using the transfer executable distributed with the VDS. 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.

12.8 vds.transfer.mode.force

New Name	vds.transfer.force, section 12.9 (page 35)
Until	1.3.9

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

12.9 vds.transfer.force

System	Pegasus
Since	1.3.9
Type	boolean
Default	false
Old name	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).

12.10 vds.transfer.single.quote

System	Pegasus
Type	boolean
Default	true
Since	1.3.0

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.

12.11 vds.transfer.throttle.processes

System	Pegasus
Since	1.2.0
Type	integer
Default	4
See also	vds.transfer, section 12.2 (page 31)
See also	vds.transfer.throttle.streams, section 12.12 (page 36)

This property is picked up when transfer mode (vds.transfer) is multiple. In this mode, multiple files are transferred using a transfer executable that comes with the VDS 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.

12.12 vds.transfer.throttle.streams

System	Pegasus
Since	1.2.0
Type	integer
Default	1
See also	vds.transfer, section 12.2 (page 31)
See also	vds.transfer.throttle.processes, section 12.11 (page 36)

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.

12.13 vds.transfer.thirdparty.pools

New Name	vds.transfer.thirdparty.sites, section 12.14 (page 37)
Since	1.2.0
Until	1.3.10

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

12.14 vds.transfer.thirdparty.sites

New Name	vds.transfer.*.thirdparty.sites, section 12.15 (page 37)
Since	1.3.10
Until	1.4.4

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

12.15 vds.transfer.*.thirdparty.sites

System	Pegasus
Type	comma separated list of sites
Default	no default
Old name	vds.transfer.thirdparty.pools
Since	1.4.4

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
vds.transfer.stagein.thirdparty.sites	the stage in transfer jobs
vds.transfer.stageout.thirdparty.sites	the stage out transfer jobs
vds.transfer.inter.thirdparty.sites	the inter pool transfer jobs
vds.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.

12.16 vds.transfer.*.thirdparty.remote

System	Pegasus
Type	comma separated list of sites
Default	no default
Since	1.4.4

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
vds.transfer.stagein.thirdparty.remote	the stage in transfer jobs
vds.transfer.stageout.thirdparty.remote	the stage out transfer jobs
vds.transfer.inter.thirdparty.remote	the inter pool transfer jobs
vds.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.

12.17 vds.transfer.staging.delimiter

System	Pegasus
Since	1.3.10
Type	String
Default	:
See also	vds.transformation.selector, section 14.10 (page 48)

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.

12.18 vds.transfer.proxy

System	Pegasus
Since	1.4.2
Type	Boolean
Default	false

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

12.19 vds.transfer.*.priority

System	Pegasus
Type	Integer
Default	no default
Since	1.4.2
See also	vds.job.priority, section 14.2 (page 44)

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
vds.transfer.stagein.priority	the stage in transfer jobs
vds.transfer.stageout.priority	the stage out transfer jobs
vds.transfer.inter.priority	the inter pool transfer jobs

13 Remote Scheduler Properties

13.1 vds.scheduler.remote.projects

System	Pegasus
Type	list of kv pairs
Default	(no default)
Example	jazz=PDQ,pnnl=emsl12491

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.

13.2 vds.scheduler.remote.queues

System	Pegasus
Type	list of kv pairs
Default	(no default)
Example	jazz=PDQ,pnnl=emsl12491

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.

13.3 vds.scheduler.remote.maxwalltimes

System	Pegasus
Type	list of kv pairs
Default	(no default)
Example	jazz=10,pnnl=20

This property allows to set the *walltime* for your jobs on each pool. Max Walltime means the maximum amount of time a job would run for on a 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.

13.4 vds.scheduler.condor.start

System	Pegasus, DAG auto-submit
Type	local file location string
Default	(no default)
See also	vds.scheduler.condor.bin, section 13.5 (page 41)
See also	vds.scheduler.condor.config, section 13.6 (page 41)

This property needs to be set if you intend to submit jobs to Condor from Pegasus itself instead of stopping after all Condor files are generated. Starting a generated workflow requires to activate condor_submit_dag in the DAG directory. An implementation for the auto-submittor for this option is provided in ${vds.home}/bin/kickstart-condor

13.5 vds.scheduler.condor.bin

System	Pegasus, DAG auto-submit
Type	local directory location string
Default	(no default)
See also	vds.scheduler.condor.start, section 13.4 (page 40)
See also	vds.scheduler.condor.config, section 13.6 (page 41)

This property needs to be set if you intend to submit jobs to Condor from Pegasus itself instead of stopping after all Condor files are generated. This property points to the location of the "bin" directory of your local Condor installation.

13.6 vds.scheduler.condor.config

System	Pegasus
Type	local file location string
Default	(no default)
See also	vds.scheduler.condor.start, section 13.4 (page 40)
See also	vds.scheduler.condor.bin, section 13.5 (page 41)

This property needs to be set if you intend to submit jobs to Condor from Pegasus itself instead of stopping after all Condor files are generated. This property points to the location where the condor_config file is stored. Refer to the Condor manual for the default locations of the Condor system.

13.7 vds.scheduler.condor.arguments.quote

System	Pegasus
Type	Boolean
Default	true
Since	1.4.5

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.

13.8 vds.scheduler.condor.release

System	Pegasus
Type	non-negative integer
Default	3
See also	vds.scheduler.condor.remove, section 13.9 (page 42)
See also	vds.scheduler.condor.retry, section 13.10 (page 42)

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.

13.9 vds.scheduler.condor.remove

System	Pegasus
Type	non-negative integer
Default	3
See also	vds.scheduler.condor.release, section 13.8 (page 41)
See also	vds.scheduler.condor.retry, section 13.10 (page 42)
Since	1.2.12

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 vds.scheduler.condor.release is used. If both the properties are unset, then the default value of 3 is used.

13.10 vds.scheduler.condor.retry

System	Pegasus
Type	non-negative integer
Default	3
See also	vds.scheduler.condor.release, section 13.8 (page 41)
See also	vds.scheduler.condor.remove, section 13.9 (page 42)
Since	1.3.0

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 vds.scheduler.condor.release which results in condor retrying the same job (submit file) when a job goes in HOLD state.

13.11 vds.scheduler.condor.output.stream

System	Pegasus
Type	Boolean
Value[0]	false
Value[1]	true
Default	true
See also	vds.scheduler.condor.error.stream, section 13.12 (page 43)

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.

13.12 vds.scheduler.condor.error.stream

System	Pegasus
Type	Boolean
Value[0]	false
Value[1]	true
Default	true
See also	vds.scheduler.condor.output.stream, section 13.11 (page 42)

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.

14 Miscellaneous Properties

14.1 vds.job.aggregator

System	Pegasus
Since	1.3.9
Type	String
Value[0]	seqexec
Value[1]	mpiexec
Default	seqexec

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

seqexec In this mode, the executable used to run the merged job is seqexec that runs each of the smaller jobs sequentially on the same node. The executable "seqexec" is a VDS tool distributed in the VDS worker package, and can be usually found at {vds.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 VDS tool distributed in the VDS worker package, and can be usually found at {vds.home}/bin/mpiexec.

14.2 vds.job.priority

System	Pegasus
Type	Integer
Default	no default
Since	1.4.2
See also	vds.transfer.*.priority, section 12.19 (page 39)

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 vds.transfer.*.priority.

14.3 vds.gridstart

System	Pegasus
Since	1.3.9
Type	enumeration
Value[0]	Kickstart
Value[1]	NoGridStart
Default	Kickstart

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.

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 VDS jobs on the remote site. Kickstart is an executable distributed with VDS that can generally be found at ${vds.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.

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

14.4 vds.gridstart.invoke.always

System	Pegasus
Since	1.4.2
Type	Boolean
Default	false
See also	vds.gridstart.invoke.length, section 14.5 (page 45)

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.

14.5 vds.gridstart.invoke.length

System	Pegasus
Since	1.4.2
Type	Long
Default	4000
See also	vds.gridstart.invoke.always, section 14.4 (page 45)

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.

14.6 vds.exitcode.mode

New Name	vds.exitcode, section 14.7 (page 46)
Until	1.3.9

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

14.7 vds.exitcode

System	Pegasus
Since	1.3.9
Type	enumeration
Value[0]	all
Value[1]	none
Value[2]	essential
Default	none
Old name	vds.exitcode.mode
See also	vds.exitcode.arguments, section 14.8 (page 46)

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.

14.8 vds.exitcode.arguments

System	Pegasus
Type	string
Default	no default
Since	1.2.14
See also	vds.exitcode, section 14.7 (page 46)

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 vds.exitcode property.

14.9 vds.tc.mapper

System	Pegasus
Since	1.3.6
Type	enumeration
Value[0]	All
Value[1]	Installed
Value[2]	Staged
Value[3]	Submit
Default	All
See also	vds.transformation.selector, section 14.10 (page 48)

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.

14.10 vds.transformation.selector

System	Pegasus
Since	1.3.6
Type	enumeration
Value[0]	Random
Value[1]	Installed
Value[2]	Staged
Value[3]	Submit
Default	Random
See also	vds.tc, section 9.8 (page 24)

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 vds.tc has been applied. For e.g specifying vds.tc as Staged and then vds.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.

14.11 vds.auth.gridftp.timeout

System	Pegasus
Since	1.3.10
Type	non negative integer
Default	120

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

14.12 vds.timestamp.format

System	VDC (Chimera)
Type	String
Default	yyyyMMdd'T'HHmmss.SSS:
Example	yyyy-MM-dd HH:mm:ss.SSSZZZZZ:

This property, if set, permits the user to change the output of the time stamps in the log messages.

14.13 vds.log.*

System	VDC (Chimera)
Type	filename or stdio handle
Default	(no default)
Example	vds.log.chunk=stderr

The VDC operations have (currently, about to change) a logging system that works with queues and levels. Each logging Q can be addressed separately, and given either a filename to append log information onto, "stdout" for standard output, or "stderr" for standard error. You may use the same filename for different queues.

14.14 vds.verbose

System	VDC (Chimera)
Type	flag
Default	(not specified)

If the verbose option is specified, the VDC operations will provide maximum logging on all queues. This is very verbose, but sometimes the easiest way to track an error.

15 Interface To 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, the GVDS provides a versatile wrapper to invoke DAGMan, called vds-submit-dag. It can be configured from the command-line, from user- and system properties, and by defaults.

15.1 vds.dagman.maxpre

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

The vds-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 GVDS system throttles artificially to a maximum of 20 PRE scripts.

15.2 vds.dagman.maxpost

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

The vds-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 GVDS system throttles artificially to a maximum of 20 POST scripts.

15.3 vds.dagman.maxjobs

System	DAGman wrapper
Type	integer
Default	200
Document	http://www.cs.wisc.edu/condor/manual/v6.6/condor_submit_dag.html

The vds-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 GVDS system throttles artificially to a maximum of 200 simultaneous jobs that are visible to the Condor system. You may want to raise this bar.

15.4 vds.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/v6.6/condor_submit_dag.html
Document	http://www.cs.wisc.edu/condor/manual/v6.6/condor_submit.html

The vds-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.

15.5 vds.dagman.verbose

System	DAGman wrapper
Type	Boolean
Value[0]	false
Value[1]	true
Default	false
Document	http://www.cs.wisc.edu/condor/manual/v6.6/condor_submit_dag.html

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

Index