The Pegasus Workflow Planner Properties

Pegasus Workflow Planner Basic Properties Property File

V 3.0.0cvs automatically generated on

2010-11-18 14:13

Contents

1 Introduction

This file is the reference guide to the basic properties regarding the Pegasus Workflow Planner, and their respective default values. Please refer to the advanced properties guide to know about all the properties that a user can use to configure the Pegasus Workflow Planner. 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 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 *${pegasus.home.sysconfdir}/properties*, have the lowest priority. These properties are expected to be set up by the submit host's administrator.
2. The properties defined in the user property file *${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.replica SimpleFile
pegasus.catalog.replica.file ${pegasus.home}/etc/sample.rc.data
pegasus.catalog.transformation Text
pegasus.catalog.transformation.file   ${pegasus.home}/etc/sample.tc.text
pegasus.catalog.site XML3
pegasus.catalog.site.file             ${pegasus.home}/etc/sample.sites.xml3
```

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 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.

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 Catalog Properties

3.1 Replica Catalog

3.1.1 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

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.

3.1.2 pegasus.catalog.replica.url

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

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

3.2 Site Catalog

3.2.1 pegasus.catalog.site

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

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

3.2.2 pegasus.catalog.site.file

System	Site Catalog
Since	2.0
Type	file location string
Default	${pegasus.home.sysconfdir}/sites.xml3 | ${pegasus.home.sysconfdir}/sites.xml
See also	pegasus.catalog.site, section 3.2.1 (page 6)

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.

3.3 Transformation Catalog

3.3.1 pegasus.catalog.transformation

System	Transformation Catalog
Since	2.0
Type	enumeration
Value[0]	File
Value[1]	Text
Value[2]	Database
Default	Text
See also	pegasus.catalog.transformation.file, section 3.3.2 (page 8)

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

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

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

```
tr example::keg:1.0 {

#specify profiles that apply for all the sites for the transformation
#in each site entry the profile can be overriden
profile env "APP_HOME" "/tmp/karan"
profile env "JAVA_HOME" "/bin/app"

site isi {
profile env "me" "with"
profile condor "more" "test"
profile env "JAVA_HOME" "/bin/java.1.6"
pfn "/path/to/keg"
arch  "x86"
os    "linux"
osrelease "fc"
osversion "4"
type "INSTALLED"
}

site wind {
profile env "me" "with"
profile condor "more" "test"
pfn "/path/to/keg"
arch  "x86"
os    "linux"
osrelease "fc"
osversion "4"
type "STAGEABLE"
}
}
```

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

1. The resource ID is represented in the first column.
2. The logical transformation uses the colonized format ns::name:vs.
3. The path to the application on the system
4. The installation type is identified by one of the following keywords - all upper case: INSTALLED, STAGEABLE. If not specified, or `NULL` is used, the type defaults to INSTALLED.
5. The system is of the format ARCH::OS[:VER:GLIBC]. The following arch types are understood: "INTEL32", "INTEL64", "SPARCV7", "SPARCV9". The following os types are understood: "LINUX", "SUNOS", "AIX". If unset or `NULL`, defaults to INTEL32::LINUX.
6. Profiles are written in the format NS::KEY=VALUE,KEY2=VALUE;NS2::KEY3=VALUE3 Multiple key-values for same namespace are 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 edu.isi.pegasus.catalog.TransformationCatalog. To load the class set pegasus.catalog.transformation to the TC implementation class.

3.3.2 pegasus.catalog.transformation.file

Systems	Transformation Catalog
Type	file location string
Default	${pegasus.home.sysconfdir}/tc.data
See also	pegasus.catalog.transformation, section 3.3.1 (page 7)

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) use this repository to map the lTR from the abstract DAX into an application invocation. Refer to the user guide for details.

Index