Developing and Deploying Applications using Granules

This document describes the process of developing and deploying applications using Granules. In both cases Granules incorporates support for utility classes whose behavior may be extended to suit specific needs.

1 Developing applications

Granules simplifies the process of developing applications. Developers simply extend the MapReduceBase class. This class implements functionality that encompasses both the map and reduce roles of a computation. One requirement is that the derived class have exactly one constructor, which does not take any arguments. Developers of the derived class only need to implement the `execute()` method. Typical steps involved in implementing this method include initialization of the datasets and data structures, processing logic, and specification of a scheduling strategy.

1.1 Initialization

Typically, depending on the type of the dataset, initialization of the datasets involved in the processing is performed automatically. The designer simply specifies the identifiers for the dataset. Initializations of the data structures needed by the computation can be performed either in the *null* constructor or in the `execute()` method. In the latter case, care must be taken to ensure that the initializations are performed only once across successive invocations of the `execute()` method.

1.2 Processing Logic

The processing logic within the `execute()`method is domain-specific. This processing would involve either the generation of results, or the management and collation of previously produced results. In the reduce role it is also possible to check if outputs have been received from all the preceding maps in addition to discarding any duplicate results that were generated.

Generation of results is easy, and the system allows entities to attach different payloads to these results. The system currently allows for the payloads for these results to be <*key, value*> pairs where the elements of these tuples could be objects that encapsulate compound data types. The system allows instances, arrays(`[]`), and 2D arrays (`[][]`) of primitive data types such as `int`, `short`, `long`, `double`, `float`, and `char` to be attached as payloads of these results. The system handles the marshalling, and un-marshalling, of these payloads automatically.

The processing logic also needs to cope with exceptions that will be thrown as result of the processing. These exceptions could result from problems with the datasets, marshalling issues and networking problems.

1.3 Scheduling strategy

A computational task can change its scheduling strategy during execution. This change is reflected during the next iteration of the `execute()`method. The system enforces the newly created scheduling strategy as soon as the current iteration of the `execute()` method terminates. Computational tasks that have specified a scheduling strategy that constitutes either a stay-alive primitive, or implies a certain number of iterations, can assert that its termination condition has been reached. At this time, the computational task is scheduled for garbage collection as soon as control returns from the `execute()` method.

2 Deploying applications using Granules

Granules provides a helper class -- the InstanceDeployer -- to enable applications, and the computational tasks that comprise it, to be deployed on a set of resources. This class performs several operations related to initializing communications, resource discovery, and deployment of computations. It is recommended that a deployer be created for each application. This can be done by simply extending the InstanceDeployer.

2.1 Initializing Communications and Resource Discovery

The first step that an application deployer needs to perform is to initialize communications with the content distribution network (NaradaBrokering). This can be performed by invoking the constructor for the base class (InstanceDeployer) which takes a set of properties as its argument. This is typically done by invoking the `super(streamingProperties)` in the derived class's constructor. Some of the elements that are typically part of this set of properties include the hostname, port and transport type for one of the router nodes within the content dissemination network. Depending on the transport over which communications take place there would be additional elements that may need to be specified. For e.g. if the SSL communications are used, additional elements that need to be specified include the location of the *truststore* and *keystore* that would be used for secure communications.

Once communications have been established, Granules automatically discovers resources that are currently available. This list could be periodically refreshed should the need arise.

2.2 Initializing and Deploying Computational Tasks

The developer then needs to provide a method that initializes the computational tasks. This involves one or more of the following --

1. *Initializing the Processing Directives associated with an instance*: These directives are used to encode instance specific information that is accessible only to the instance in question.
2. *Specification of the datasets and collection associated with the computation*: Granules is responsible for configuring access to these datasets.
3. *Linking of the Map-Reduce roles*: Granules ensures that once linked results produced by the maps are automatically routed to the appropriate reducers.
4. *Specifying the scheduling strategy for the computational tasks*: By default, the exactly-once scheduling strategy is used.
5. *Distribution of datasets across these instances*: Granules incorporates utilities that allow this distribution to be performed in an efficient fashion.

To deploy an application, the developer only needs to invoke the `deploy()` method in the InstanceDeployer. This method deploys the computational tasks on the set of resources that were discovered during the initialization phase.

2.3 Tracking / Steering a deployed Application

The InstanceDeployer implements the JobLifecycleObserver interface which allows one to track the status of multiple jobs, and the computational tasks that comprise them. Classes that extend the InstanceDeployer have the option to override methods specific to the JobLifecycleObserver interface. Specifically, for a given Job, Granules maintains its registered JobLifecycleObserver and invokes methods on this observer whenever there is an update to the deployment or execution status of the computational tasks that comprise it.

Associated with each Job, Granules maintains a ProgressTracker which maintains information about the execution state of each of the computational tasks that comprise the application. The LifecycleMetrics associated with every computational task includes information about:

1. The arrival time for the computational task.
2. The queuing overhead for the computational task.
3. The total CPU-bound time for the computational task across multiple iterations (if there are any).
4. The processing time for the computational task
5. The current status of the computational task {Awaiting Data, Queued for Execution, Executing, Terminated, Successful, FAILED}

The status of a Job is the cumulative status of the computational tasks that comprise it.

The InstanceDeployer also incorporates methods for tracking/steering a computation. There are methods to refresh the status of a specific computational task or the entire Job. These methods result in updates to the lifecycle metrics of the relevant computational tasks. Additionally, Granules also allows computational tasks to be aborted when they are in execution. The system allows either a specific computational task to be suspended or the entire Job.

3 Instructions to Run the Demos

1. First, download NaradaBrokering and **start the broker**
   You can download the software from http://www.naradabrokering.org . Once you have downloaded the software, go to the bin directory and click on the startBroker.bat file or execute the ./startBr.sh command.

2. To **configure the Resource**, please go to the **`GRANULES_HOME/config`** directory. Edit the **`ResourceConfig.txt`** file to
   a. Reflect the broker's host, port and transport information
   b. You can also change the number of threads spawned by the resource. Please set this value carefully.

3. To **run the Resource** go to the **`GRANULES_HOME/bin`** directory and launch the appropriate script.
   a. In case of **multiple resources** on different machines that mount the same file-system, you do not need to create multiple .bat files if you are OK with the all the resources connecting to the same broker. Otherwise, you could set-up a broker network and change the broker arguments in the resource-configuration files appropriately.

4. In the **`cgl.granules.samples`** directory there are several examples that have been organized as packages. These reflect different capabilities provided by Granules. There is a **`XYZDeployer.java`**

in each of these packages. In all cases the arguments for these classes is the **`hostname`** and **`portnum`** of the broker that you need to connect to.

5. When each **`XYZDeployer`** runs it prints out the set of commands that you can enter in the SHELL.