Open Genomics Engine User Manual

Getting Started

Open Genomics Engine (OGE) is a lightweight workflow-management framework that allows easy creation and sharing of genome analysis pipelines. Plug-in and workflow are two key concepts in OGE. A plug-in is a wrapper around an application or a script. A workflow defines how a set of plug-ins are connected together to assemble an analysis pipeline. Plug-ins and workflows are defined in the XML format.

An example OGE repository is included in the “ogeroot” directory in the package. We refer a directory that contains an OGE repository as an OGE root. There are two important configuration files in an OGE root.

1) oge.ini
This file stores important system parameters, such as debugging configuration and scratch space for workflow execution.

2) catalog.xml
This file stores meta information of all plug-ins and workflows in the repository.

An initial set of plug-ins and workflows are included in the “plugins” and “workflows” directories. Note that the actual applications are not included in the package. Users need to download these applications from their original websites.

In addition to directly editing the XML files, users can work with OGE through three ways: a command line interface (CLI), a programmable interface and an annotated scripting interface.

The CLI provides a set of tools for managing the OGE repository and executing workflows. To launch the CLI:
$ python cli/cli.py ogeroot/

After the CLI is launched, you can list the supported commands with the “help” command. The following example shows how to execute a workflow (i.e., bwa_aln_sampe) that maps pair-end reads with BWA. Before execution, the indexes of genome.fa created by BWA needs to be placed under the “index_dir”.

CLI Example:
oge > executeWorkflow bwa_aln_sampe --input-read1 in1.fq --input-read2 in2.fq --ref_genome index_dir/genome.fa --output_sam test.sam

The programmable interface allows users to dynamically create and execute a workflow. It helps task automation and offers a powerful way to create complex workflows. An example of how to use the programmable interface to access OGE is given in examples/c++/align_workflow.cpp.

The annotated script interface is designed to help users import users to import existing scripts into OGE. An example of the annotated script is given in examples/annotated_scripts/script_example. To import from a script:
$ import_script <ogeroot> <script>

XML Definitions

OGE stores definitions of plugins and workflows in the XML format[1].

Plug-in Definition

A XML plug-in definition includes the *header information* and three sections: *inputs*, *outputs* and *commands*. The header information includes *plug-in id*, *name*, *version* and *description*. OGE uses the plug-in id as the unique identifier of a plug-in.

Inputs Section

The inputs section defines input parameters of this plug-in. Each parameter has five fields: *name*, *type*, *format*, *value* and *label*. An input parameter is referenced in the commands section by its name, and the label field is used for a short description of a parameter.

Currently, OGE supports five types: *file*, *bool*, *float*, *int* and *string*. For the file type, the format field is required. For instance, if the input parameter specifies a BAM file, use “bam” for the format.

[1] The plug-in XML definition in OGE is inspired by the Galaxy (http:http://galaxy.psu.edu/) project.

The value field specifies the default value of a parameter. This default value can be overridden by the parameter value defined at the workflow level.

Outputs Section

The outputs section specifies the output parameters of this plugin. The definition of an output parameter is the same as an input parameter.

Commands Section

The commands section defines execution commands of a plug-in. Note that multiple logically grouped commands can be specified for one plug-in. A command can reference input and output parameters, it can also reference environment variables defined in the shell. Each referenced parameter or variable starts with '$' followed by its name. The parameter name can be surrounded by a pair of curly brackets.

Example variable names: $var1, ${var1}

There are two predefined system variables that can be used in a command.
$ENGINE_DIR: The OGE root directory.
$PLUGIN_DIR: The directory where the plugin XML file is located.

For some applications, there are parameters that behave like on-off switches and do not accept any input values. There are also parameters whose values are dependent on another parameter. To handle these cases, OGE allows specifying conditionals with a ternary operation, which is expressed in the following form:

[condition? true_case: false_case]

A condition can be a binary operation comparing two strings or a boolean variable. The binary operation currently supports two operators: "==" and "!=." The value of a boolean variable can be either "true" or "false." The ":" and "false_case" can be omitted.

Example 1: [$seed_len != ""? -l $seed_len]

In this example, if variable $seed_len is given, then this ternary operation will be replaced with "-l $seed_len" during the command parsing. Otherwise, the "-l" option will not be included.

Example 2: [$use_gpu? --use-gpu]

In this example, if variable $use_gpu is equal to "true," the --use-gpu flag will be included during the command parsing. Otherwise, the "--use-gpu" option will not be included.

Note that recursive definition of conditionals as well as concatenation of conditions, e.g., using "||" and "&&" are currently not supported.

Workflow Definition

The XML definition of a workflow includes header information and four sections: *inputs*, *steps, outputs* and defines. The header information includes *workflow id*, *name*, *version* and *description*. OGE uses the workflow id as the unique identifier of a workflow.

Inputs Section

The inputs section defines input parameters for running the workflow. Each parameter has five fields: *name*, *type*, *format*, *value* and *label*, similar to those defined in a plug-in. The input parameters can be referenced by plug-ins included in the workflow.

Steps Section

The steps section defines the execution flow of a workflow, where each step can be a plug-in or another workflow. Allowing constructing a workflow from other workflows makes it easy to develop large and complex workflows.

The metadata of a step includes *id*, *type* and *plugin_id*. The id field is a unique identifier of a step in the workflow. The type field specifies whether this step is executing a plug-in or a workflow, whose id is given in the plugin_id field.

The definition of each step includes an *inputs* section and an *outputs* section, which specify input and output parameters that will be passed into the execution of each step. Each input or output parameter must have been defined in the corresponding plug-in or workflow.

The value of a input parameter of a step can be a input parameter of the workflow, an output parameter of a previous step, or a literal value, e.g., an integer number. The dependencies between different steps are defined by connecting the input and output parameters between these steps. Note that an output parameter of a previous step is referenced by adding the step id and "." as the prefix.

Examples:
<param name="var1" value="$genome" />
The input value of var1 is taken from the workflow input parameter "genome."
<param name="var1" value="$s1.var2" />
The input value of var1 is taken from the output variable "var2" of step "s1."
<param name="var1" value="10" />
The input value of var1 is integer 10.

The value of an output parameter, typically a file, of a step will be automatically generated by OGE during the workflow execution, using the type of the parameter as the suffix.

Outputs Section

The outputs section defines the output parameters of the workflow. The value of a output parameter should be an output of a step in the workflow.

Example: <param name="align_output" type="file" format="sam" value="$s2.output_sam" />

In the above example, the workflow output parameter "align_output" is connected to the "output_sam" output parameter of step "s2." During the workflow execution, the output file generated in step "s2" will be moved to an output file specified by the "align_output" parameter.

Workflow Execution

When executing a workflow, OGE processes each step one by one and generates a series of executable commands. The input and output parameters defined in a step will be passed into a plug-in or a sub-workflow. If this step is a plug-in, OGE parses the commands section and resolve various variables defined in command lines.

OGE provides a testing mode for executing a workflow. In this mode, OGE only generates all commands but without actually executing them. The testing mode is useful for pinpointing errors in the plug-in and workflow definition during the development of plug-ins and workflows.

OGE creates a temporary directory to store intermediate data for each workflow. By default, the temporary directory is located under the "workspace" directory under the OGE root. Users can change the location of temporary directories by modifying the "work_dir" entry in oge.ini. The temporary directory and the data in the directory will be kept after each execution. Users can manually clean up all temporary directories or use the "clearWorkspace" tool in the CLI.

By default, OGE will print out the execution log of a workflow. Users can use this log information to trace a problem if something goes wrong during the workflow execution. The log information can also be directed to a file by configuring the "log_file" entry in oge.ini.

Command Line Interface

The Command Line Interface (CLI) offers a set of tools for managing plug-ins and workflows. Currently, it is the primary way of working with OGE.

To launch CLI:
$ python cli/cli.py ogeroot/

Plugin Management Tools

listPlugins

List all plug-ins in the repository.

createPluginTemplate <plugin_id>

Create a template for a new plug-in. The suite id is used to specify where to place the plugin in the catalog. After the template is created, users can edit the template XML file to configure the detail information, e.g., input and output parameters, of a plug-in.

removePlugin <plugin_id>

Remove a plug-in from the catalog. The corresponding XML file will also be removed.

queryPlugin <plugin_id>

Query the details of a plug-in.

testPlugin <plugin_id> [--para1 val1] [--para2 val2] ...

Simulate the execution of a plug-in by parsing and generating execution commands. These commands will not be executed. Parameters are supplied as a list after the plug-in id. Each parameter name is identified by the "--" prefix.

executePlugin <plugin_id> [--para1 val1] [--para2 val2] ...

Execute a plug-in.

Workflow Management Tools

listWorkflows

List all workflows in the repository.

createWorkflow <workflow_id>

A menu-driven tool for creating a workflow.

removeWorkflow <workflow_id>

Remove a workflow and its XML definition file.

queryWorkflow <workflow_id>

Query details of a workflow.

testWorkflow <workflow_id> [--para1 val1] [--para2 val2] …

Simulate the execution of a workflow by parsing and generating execution commands. These commands are not actually executed. Parameters are supplied as a list after the workflow id. Each parameter name is identified by the "--" prefix.

executeWorkflow <workflow_id> [--para1 val1] [--para2 val2] ...

Execute the workflow.

Other Tools

listSuites

List all suites and their belonging plug-ins and workflows in the repository.

removeSuite <suite_id>

Remove a suite from the repository.

clearWorkspace

Remove all temporary working directories. By default, those directories are placed under the "workspace" directory of the OGE root.

Programmable Interface

The programmable interface provides a flexible way of composing and executing complex workflows. It can also be used for automated analysis. OGE allows creating and executing a workflow on-the-fly without storing it to the repository.

Execute A Workflow

Below is an example of executing a workflow. The "Engine" class is the core component of OGE. It needs to be first initialized by supplying the directory containing an OGE repository. The core method of executing a workflow is "executeWorkflow()." To invoke this method, the parameters of a workflow need to be prepared as a C++ STL map, with the parameter name as the map key and the parameter value as the map value.

Example:

```
Engine engine(engine_dir);
String workflow_id = “bwa_aln_sampe”;

// Configure incoming parameters
map <string, string> paras;
paras["in_read1"] = "/path/to/in1.fq1";
paras["in_read2"] = "/path/to/in1.fq2";
paras["ref_genome"] = "/path/to/genome.fa";
paras["output_sam"] = "/path/to/results.sam";

engine.executeWorkflow(workflow_id, paras);
```

Create A Workflow

OGE provides a set of APIs for composing a workflow corresponding to the XML sections in a workflow definition file. The runtime instance of a workflow is implemented with class “Workflow”.

Add Workflow Inputs

To add an input, first creates an instance of an input parameter with the “Parameter” class. The “Parameter” class allows specifying the type, value, format and label of an parameter. A parameter can then be added to a workflow.

Example:

```
Parameter p1(DATA_FILE, "", "fastq", "");
workflow.addInput("in_read1", p1);
```

Add Workflow Steps

Each step to be included in the workflow is defined with the “WorkflowStep” class. The “WorkflowStep” class provides “addInput” and “addOutput” methods for defining input and output parameters. To refer to an output parameter defined in an previous step, add the step id and “.” as the prefix to the output parameter. For instance, “$aln1.output_sai” refers to the “output_sai” parameter of defined in step “aln1.”

Example:

```
WorkflowStep s_tosam(PLUGIN, "tosam", "bwa_sampe");
s_tosam.addInput("input_read1", "$in_read1");
s_tosam.addInput("input_read2", "$in_read2");
s_tosam.addInput("ref_genome", "$in_genome");
s_tosam.addInput("input_sai1", "$aln1.output_sai");
s_tosam.addInput("input_sai2", "$aln2.output_sai");
s_tosam.addOutput("output_sam");
workflow.addStep(s_tosam);
```

Add Workflow Outputs

Output parameters of a workflow are defined similar to input parameters. One difference is that an output parameter needs to specify the value, which usually refers to an output parameter defined in one of the steps.

Example:

```
Parameter p4(DATA_FILE, "$tobam.output_bam", "bam", "");
workflow.addOutput("output", p4);
```

Annotated Scripting Interface

OGE allows users to import their existing scripts with annotations[2]. An annotated script includes three sections: *header*, *defined variables* and *plug-ins*.

Header

The header section defined metadata for a workflow. This metadata includes name, id, version and suite id.

Example:

```
WORKFLOW_ID=imported_variant_calling
WORKFLOW_NAME="Call variants with samtools"
WORKFLOW_VERSION=1.0
SUITE_ID=imported
```

Defined Variables

This section defines a set of common variables that are referenced by multiple commands in the script.

Plug-ins

Users can define one or multiple commands in the script as a plug-in by including those command(s) into a pair of curly brackets. The grouped commands can then be assigned to a plug-in id with the ":=" operator.

Example:

```
align := {
   bwa aln -l -t 8 $REFERENCE $input > ${input}.sai
   bwa samse $REFERENCE ${input}.sai $input > $output
}
```

Note that each plug-in can only specify one input, i.e., "$input," and one output parameter, i.e., "$output". The $input parameter of a plug-in will be automatically connected to the last defined $output parameter in the previously defined plug-ins.

[2] The script annotations are inspired by the bpipe tool (http://code.google.com/p/bpipe/).