Introduction to workflows with GC3Pie

GC3: Grid Computing Competence Center,
University of Zurich

Oct. 2, 2012

The GC3Pie approach to workflows, I

Workflows are Python code.

The GC3Pie approach to workflows, II

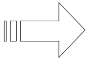
Thus, in order to run a workflow, you write a Python script using GC3Pie to orchestrate the running of applications.

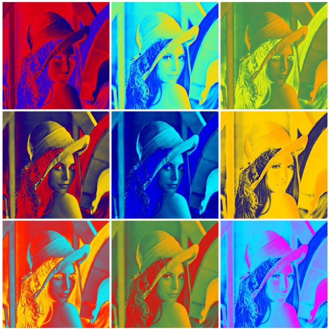
So far we've seen how to write scripts that control many instances of a single application.

Now it's time to introduce the GC3Pie classes that allow orchestrating the execution of jobs of several different types. (For instance, specify that certain applications should be executed in a sequence.)

Let's start with a colorful example.

Warholize!





How do we “warholize” an arbitrary image?

1. Convert the original image to grayscale.
2. Colorize the grayscale image using three different colors for each tile.
3. Arrange all the colorized images into an $N \times N$ frame.

Reference: `http://gc3pie.googlecode.com/svn/trunk/gc3pie/docs/html/gc3libs/tutorial/warholize.html`

The GC3Pie approach to workflows, II

The basic unit of work in a GC3Pie workflow is called a `Task`.

The `Application` class that you already know is a kind of `Task` (indeed, it's a derived class).

From now on, we'll speak of `Task`s rather than applications. Examples of `Task` instances that are not applications will arrive shortly.

Running tasks in sequence

To run tasks in an ordered sequence, one after the other, GC3Pie provides a `SequentialTaskCollection` class.

It is created with a list of tasks, and runs all of them in the order given. The sequenceis dynamic, in that you can add new tasks on the fly, re-run existing ones, or remove future tasks.

A `SequentialTaskCollection` is itself a task.

Running tasks in parallel

To run tasks in parallel (i.e., they have no inter-dependency), GC3Pie provides a `ParallelTaskCollection` class.

It is created with a (Python) list of tasks, and runs all of them in parallel (compatibly with the computational resource limits).

A `ParallelTaskCollection` is itself a task.

Putting it all together

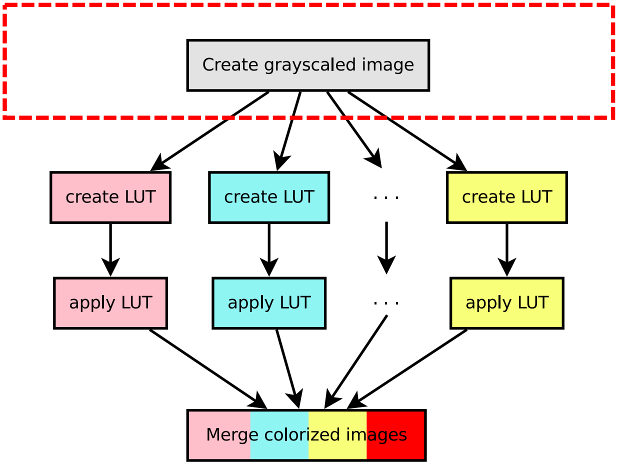
So tasks can be:

- `Application` instances,
- `SequentialTaskCollection`s,
- `ParallelTaskCollection`s.

So you can nest them, and create parallelly-running sequences, or sequences of “job explosions” (many jobs in parallel), or any combination of this.

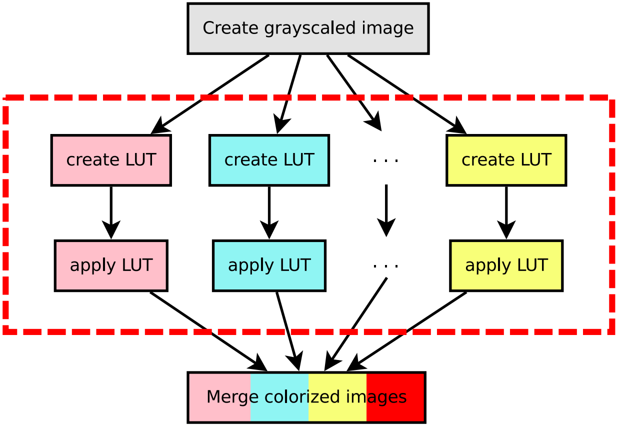
The Warholize workflow, I

1. Convert the original image to grayscale.



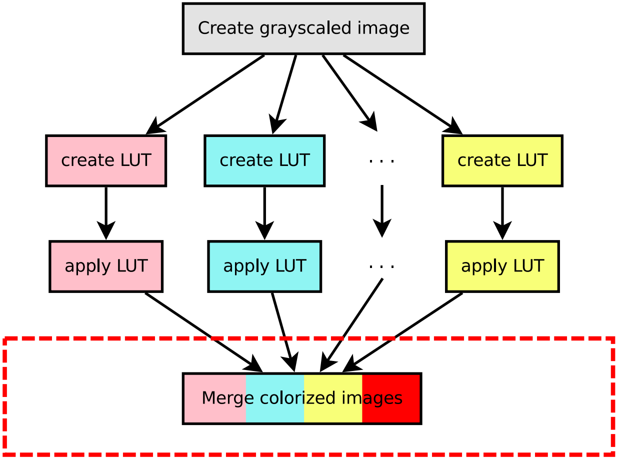
The Warholize workflow, II

2. Colorize the grayscale image using three different colors for each tile.



The Warholize workflow, III

3. Arrange all the colorized images into an $N \times N$ frame.

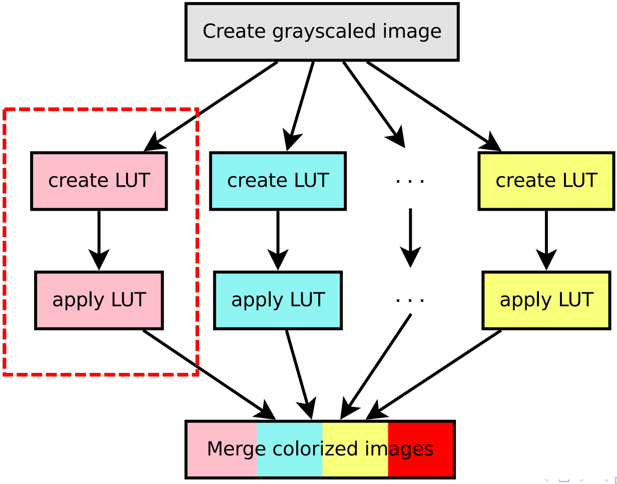


The Warholize workflow, IV

Step 2 actually entails two sub-steps:

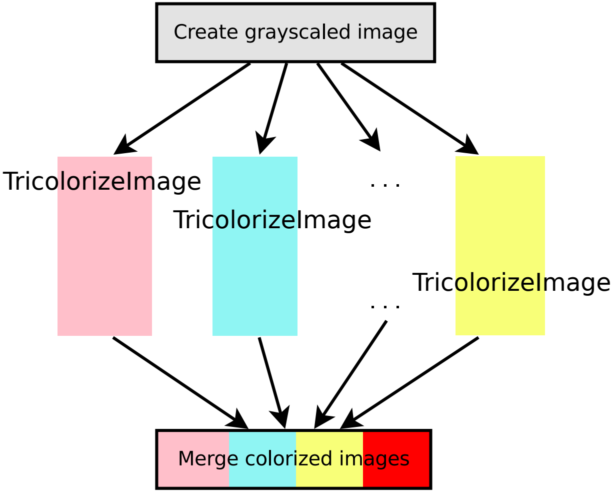
a) mapping greyscale levels to random colors,

b) applying this mapping to produce new images



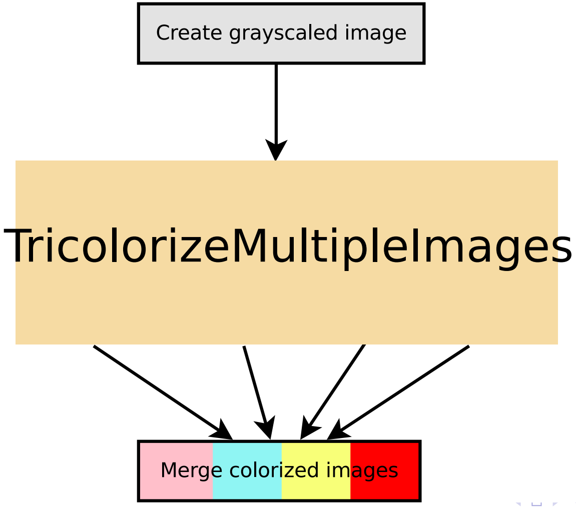
The Warholize workflow, V

So, Step 2 is a `SequentialTaskCollection`-type task. Let's call this two-pass sequence `TricolorizeImage` .



The Warholize workflow, VI

All the `TricolorizeImage` instances run in parallel. Collect them into a `ParallelTaskCollection`-type task, called `TricolorizeMultipleImages`.



The Warholize workflow, VII

Now we are left with a three-step sequence: greyscale, `TricolorizeMultipleImages`, montage. This can be defined again as a `SequentialTaskCollection`-type task, the `WarholizeWorkflow`.