OpenMOLE: a Workflow Engine for Distributed Medical Image Analysis

Jonathan Passerat-Palmbach, Mathieu Leclaire, Romain Reuillon, Zehan Wang, Daniel Rueckert

OpenMOLE: a Workflow Engine for Distributed Medical Image Analysis

Jonathan Passerat-Palmbach* ,
Mathieu Leclaire † ,
Romain Reuillon†,
Zehan Wang*,
Daniel Rueckert*

Originally published in: High Performance Computing MICCAI Workshop
— September 2014 — pp n/a-n/a
unpublished conference paper

Abstract: This works demonstrates how the OpenMOLE platform can provide a straightforward way to distribute heavy workloads generated by medical imaging analysis. OpenMOLE allows its users to benefit from a large set of distributed computing infrastructures such as clusters or computing grids, no matter the kind of application they are running. Here we extend the OpenMOLE tools to two new cluster job schedulers: SLURM and Condor. We also contribute to the Yapa packaging tool to support the widely spread virtual environment package from the Python programming language. Our test case shows how our developments allow a medical imaging application to be distributed using the OpenMOLE toolkit.

Keywords: Distributed Computing; Grid Computing; Workflow; Reproducibility; Medical Image Analysis; OpenMOLE

* Imperial College London, Department of Computing, London SW7 2AZ
† Institut des Systèmes Complexes Paris, Île de France

RESEARCH CENTRE
Imperial College London
Department of Computing
Huxley Building
London SW7 2AZ
United Kingdom

1 Introduction

Medical image analysis can be very computationally expensive. General computation times have increased as data sets have grown and become more complex to process. At the same time, recent languages such as Python have spread in the community. They are easy to learn and simple to use. In the case of Python, a very thorough set of packages dedicated to scientific applications make it a first choice for many scientists. Still, as an interpreted language, it introduces an overhead at runtime.

These complex applications also come with numerous input parameters that can be tuned. In the search of the ideal parameter set among all the possible combinations, it is ideal to match all these different parameters to each of the images from the data set. If we think of the processed image as just another parameter taking different values, this leads to an overwhelming number of potential combinations. Testing all these combinations with the studied application becomes unmanageable on a regular desktop platform.

Two options can lead to performance improvements for this problem. The first strategy is to optimize and/or parallelize the base application. This is obviously a very interesting approach, but it can be difficult to obtain decent improvements without being an expert in this domain. Also, this does not reduce the number of parameter combinations to execute.

One other approach is to use distributed computing platforms to spread the workload across several machines. In an ideal world, the time required to process the whole experiment would be equivalent to the execution time of the application fed with a single instance of parameters set. Please note that this approach is not incompatible from any attempt to parallelize or optimize the base application.

This work focuses on making distributed computing available in the most straightforward way to the medical imaging community. Depending on the applications, several problems might arise when attempting to distribute the execution. Software tools can help scientists overcome these hindrances. Our study describes the input of such a tool called OpenMOLE and its software ecosystem.

OpenMOLE is a workflow distribution platform which aims at offering scientists from any field a tool to model their experiments as workflows, and then distribute their execution remotely to spread the workload. Compared to other workflow processing engines, it relies on a zero-deployment approach to a wide range of distributed computing environments. OpenMOLE also encourages the use of software components developed in heterogeneous programming languages and enable its user to easily replace the elements involved in the workflow. Workflows can be designed using either a GUI, or a Scala DSL which exposes advanced workflow design constructs. For more details regarding the core implementation and features of OpenMOLE, the interested reader can refer to [Reuillon et al., 2010, Reuillon et al., 2013] and to the public Git repository

hosting OpenMOLE's source code[1].

In this paper, we first detail the problems faced by the medical imaging community to effectively distribute an application. Then, we present how the tools from the OpenMOLE ecosystem can answer these problems. The next section addresses changes to OpenMOLE tools induced by challenges related to software configurations frequently encountered in medical imaging applications. We finally present a test case showing how an actual medical imaging application has been successfully distributed using OpenMOLE.

2 The Challenges of Distributing Applications

Let us consider all the dependencies introduced by software bundles explicitly used by the developer. They can take the form of shared libraries for binary applications or packages for Python.

These software dependencies become a problem when distributing an application, as it is unlikely that a large number of remote hosts will be deployed in the very same configuration as a researcher's desktop computer. If a dependency is missing, the remote execution will simply fail on the remote hosts where the requested dependencies are not installed.

An application can also be prevented from running properly due to incompatibilities between versions of the deployed dependencies. This case can lead to silent errors, where a software dependency would be present in a different configuration and would generate different results for the studied application. This problem breaks Provenance, a major concern of the scientific community [Miles et al., 2007, MacKenzie-Graham et al., 2008]. Provenance criteria are satisfied when an application is documented thoroughly enough to be reproducible. This can only happen in distributed computing environments if the software dependencies are clearly described and available.

Some programming environments provide a solution to these problems. Compiled languages such as C and C++ offer to build a static binary, which packages all the software dependencies. Some applications can be very difficult to compile statically. A typical case is an application using a closed source library, for which only a shared library is available.

Another approach is to rely on an archiving format specific to a programming language. The most evident example falling into this category are Java Archives (JAR) that embed all the Java libraries an application will need.

The next section will detail how OpenMOLE tackles these problems in three steps.

[1] `https://github.com/openmole/openmole`

3 Harnessing OpenMOLE for Medical Image Analysis

OpenMOLE shows several main advantages with respect to the other workflow management tools available. The interested reader will find thorough surveys describing the scientific workflow processing tools in [Barker and Van Hemert, 2008, Mikut et al., 2013].

First, OpenMOLE distinguishes as a tool that does not target a specific scientific community, but offers generic tools to explore large parameter sets.

Second, while OpenMOLE offers a GUI to design workflows as most workflow management systems, it also features a Domain Specific Language (DSL) to describe the workflows. According to [Barker and Van Hemert, 2008], workflow platforms should not introduce new languages but rely on established ones. OpenMOLE's DSL is based on the high level Scala programming language [Odersky et al., 2004]. Section 3.2 details the role played by the DSL to establish the Provenance of the workflow.

Finally, OpenMOLE features a great range of platforms to distribute the execution workflows, thanks to the underlying GridScale library[2]. GridScale is part of the OpenMOLE ecosystem and acts as one of its foundation layers. It is responsible of accessing the different execution environments. The last release of OpenMOLE can target SSH servers, multiple cluster managers and computing grids ruled by the gLite/EMI middleware.

In this section, we describe the three tools from the OpenMOLE ecosystem that answer the problems evoked in Section 2. These three tools refer to three simple steps to explore a set of parameters and distribute the resulting workload.

3.1 Packaging the application with Yapa

The first step in order towards spreading the workload across heterogeneous computing elements is to make the studied application executable on the greatest number of environments. We have seen previously that this could be difficult with the complex software environments entanglements available nowadays. For instance, a Python script will run only in a particular version of the interpreter and make use of binary dependencies. The best solution to make sure the execution will run as seamlessly on a remote host as it does on the desktop machine of the scientist is to track all the dependencies of the application and ship them with it on the execution site.

The OpenMOLE ecosystem provides this feature through a tool called Yapa[3]. Yapa relies on the CDE (Code, Data, and Environment packaging) application [Guo, 2012] to create archives containing all the items required by an application to run on any recent Linux platform.

CDE tracks all the files that interact with the application and creates the base archive. Yapa completes the package by adding specific customizations

[2] `https://github.com/openmole/gridscale`

[3] `https://github.com/openmole/yapa`

related to the integration of the application within an OpenMOLE workflow. In OpenMOLE, workflows are mainly composed of tasks. Yapa creates the task corresponding to the packaged application, simply by studying the output of CDE. The result is a ready to run package containing the application and all its dependencies, independent from the configuration of the host executing it. As an OpenMOLE task, the generated package is ready to be added to the OpenMOLE scene to be integrated in a workflow.

In our case, we extended the scope of Yapa to enable it to correctly handle Python applications using the *Virtual Environment*[4] package. Virtual Environment is quite common in Python developments as it allows to install and configure a Python environment on a system, without interfering with the main installation. This way, different versions of the same Python package can cohabit on the same machine without impacting the execution of other Python scripts.

The problem is, Virtual Environment sets up the execution environment in a specific way to isolate the application. This behaviour conflicted with Yapa which could not supervise the whole execution in this case. Yapa returned prematurely, preventing CDE to capture all the dependencies. Our study has raised this problem and proposed a correction that now enables Yapa to package correctly Python applications using the Virtual Environment package. The updated release is available from Yapa's source code repository.

The only constraint regarding Yapa is to create the archive on a platform running a Linux kernel from the same generation as those running on the targeted computing elements. As a rule of thumb, a good way to ensure that the deployment will be successful is to create the Yapa package from a system running Linux 2.6.32. Many HPC environments run this version, as it is the default kernel used by science-oriented Linux distribution, such as Scientific Linux and CentOS.

3.2 Describing the application as an OpenMOLE workflow

Scientific experiments are characterized by their ability to be reproduced. For medical image analysis applications, this implies capturing all the processing stages leading to the result. Many execution platforms introduce the notion of workflow to do so [Barker and Van Hemert, 2008, Mikut et al., 2013]. Likewise, OpenMOLE manipulates workflows and distribute their execution across various computing environments.

Reproducibility becomes a concern when a scientist wants to reproduce a given workflow on an environment different from the original computing platform. With classic workflow descriptions, this process can be tedious since in spite of some efforts of standardization [Barker and Van Hemert, 2008], no standard workflow description has been adopted yet. It is then difficult to provide the right Provenance information that will enable workflows reproducibility.

[4] `https://github.com/pypa/virtualenv`

Two choices are available when it comes to describe a workflow in OpenMOLE: the Graphical User Interface (GUI) and the Domain Specific Language (DSL). Both strategies result in identical workflows. They can be shared by users as a way to reproduce their execution. In terms of Provenance information, the DSL is more relevant as its self-explicit format can be read apart from OpenMOLE.

OpenMOLE's DSL is based upon the Scala programming language, and embeds new operators to manage the construction and execution of the workflow. The advantage of this approach lies in the fact that workflows can exist even outside the OpenMOLE environment. As a high-level language, the DSL can be assimilated to an algorithm described in pseudocode, understandable by most scientists. Moreover, it denotes all the types and data used within the workflow, as well as their origin. This reinforces the capacity to reproduce workflow execution both within the OpenMOLE platform or using another tool.

3.3 Running the workflow

With the studied application packaged and the associated workflow described, there is left to determine the execution environment. OpenMOLE enables delegating the workload to a wide range of HPC environments including remote servers (through SSH), clusters (supporting the job scheduler PBS) and computing grids running the gLite/EMI middleware. For the purpose of this study, we also made available plugins for SLURM and Condor, two other job schedulers for clusters.

Submitting jobs to distributed computing environments can be complex for some users. This difficulty is hidden by the GridScale library from the OpenMOLE ecosystem. GridScale provides a high level abstraction to all the execution platforms mentioned previously.

When GridScale was originally conceived, a choice was made not to rely on a standard API (Application Programming Interface) to interface with the computing environments, but to take advantage of the command line tools available instead. As a result, GridScale can embed any job submission environment available from a command line. From a higher perspective, this allows OpenMOLE to work seamlessly with any computing environment that the user can access.

Users are only expected to select the execution environment for the tasks of the workflow. This choice can be guided by two considerations: the availability of the resources and their suitability to process a particular problem. The characteristics of each available environments must be considered and matched with the application's characteristics. Depending on the size of the input and output data, the execution time of a single instance and the number of independent executions to process, some environments might show more appropriate than others.

In the particular case of medical imaging, the European Grid Initiative (EGI) gathers a pool of resources under the name Biomed. Biomed is a Virtual Organization (VO), i.e.: a subset of a computing grid gathering users and resources

sharing the same interests. This infrastructure connects computing clusters from universities and research laboratories across Europe to offer more than 300,000 CPU cores to its members. This large scale computing environment is supported by OpenMOLE.

At this stage, OpenMOLE's simple workflow description is extremely convenient to determine the computing environment best suited for a workflow. Switching from one environment to another is achieved either by a single click (if the workflow was designed with the GUI) or by modifying a single line (for workflows described using the DSL).

4 Test Case: Distribution of a Patch-Based Segmentation Application

Our test case involves a patch-based segmentation application that implements a label propagation approach. To do so, it uses relative geodesic distances to define patient-specific coordinate systems as spatial context [Wang et al., 2014]. For the sake of our tests, we focus on running this application to perform multi-organ segmentation of abdominal CT images.

The workflow we describe in OpenMOLE studies a set of 150 different input images. The resulting workflow was distributed using a cluster managed using the SLURM job scheduler, and on the European Grid Infrastructure's Biomed VO, which is managed by the gLite/EMI middleware.

Depending on the input image, the processing time for a single image ranges from approximately 10 to 24 hours. The application is written in Python and uses common packages such as SciPy and NumPy. The Python configuration is handled through the Virtual Environment package.

Packaging the application using Yapa is straightforward and as explained earlier, does not impact the original command line, as shown in Listing 1. In this example, the application's parameters are set to arbitrary values selected from the set of existing combinations.

```
1 java -jar yapa.jar -o /tmp/package.yapa -c "python ./scripts/
    abdominalwithreg.py --spatialWeight 7 --queryDilation 4 --
    preDtErosion 2 --atlasBoundaryDilation 6 -k 40 --
    numAtlases 50 --patchSize 5 --spatialInfoType coordinates
    --resLevel 3 --numProcessors 1 --dtLabels 0 3 4 7 8 --
    savePath /vol/bitbucket/jpassera/PatchBasedStandAlone/
    AbdominalCT/Results --transformedImagesPath /vol/bitbucket
    /jpassera/AbdominalCT/TransformedImages --
    transformedLabelsPath /vol/bitbucket/jpassera/AbdominalCT/
    TransformedLabels --transformedGdtImagesPath /vol/
    bitbucket/jpassera/AbdominalCT/TransformedGdtImagesPath --
    transformedDtLabelsPath /vol/bitbucket/jpassera/
    AbdominalCT/TransformedDtLabels --fileName nusurgery001
    .512.nii.gz"
```

Listing 1: Packaging an application with Yapa

The resulting package contains all the software dependencies and input files required for the application to work correctly. It forms an OpenMOLE task that is ready to be part of a workflow. We can now describe this workflow to explore the parameter set of 150 images, and define the inputs and outputs of the application task. The resulting workflow is presented in Listing 2.

```
1  import org.openmole.plugin.task.systemexec._
2  import org.openmole.plugin.hook.file._
3  import org.openmole.plugin.domain.collection._
4  import org.openmole.plugin.domain.file._
5  import org.openmole.plugin.environment.glite._
6
7  val imageID = Prototype[String]("imageID")
8  val idsList = List( 1 to 200 by 2, 301 to 519 by 2 ).flatten.map ("%03
       d" format _)
9
10 // construct the parameter set
11 val exploIDsTask = ExplorationTask (
12   "exploIDs",
13   Factor(
14     imageID,
15     idsList toDomain
16   )
17 )
18
19 // the SystemExecTask mimics the command line, but runs from the
       Yapa archive
20 val savePath = "/vol/bitbucket/jpassera/PatchBasedStandAlone/
       AbdominalCT/Results"
21 val cTask = SystemExecTask(
22   "myRunTask",
23   "./python.cde ./scripts/abdominalwithreg.py --spatialWeight
         7 --queryDilation 4 --preDtErosion 2 --
         atlasBoundaryDilation 6 -k 40 --numAtlases 50 --patchSize
          5 --spatialInfoType coordinates --resLevel 3 --
         numProcessors 1 --dtLabels 0 3 4 7 8 --savePath " +
         savePath + " --transformedImagesPath /vol/bitbucket/
         jpassera/AbdominalCT/TransformedImages --
         transformedLabelsPath /vol/bitbucket/jpassera/AbdominalCT
         /TransformedLabels --transformedGdtImagesPath /vol/
         bitbucket/jpassera/AbdominalCT/TransformedGdtImagesPath
         --transformedDtLabelsPath /vol/bitbucket/jpassera/
         AbdominalCT/TransformedDtLabels --fileName nusurgery${
         imageID}.512.nii.gz",
```

```
24   "cde11_zehan/cde-root/vol/bitbucket/jpassera/
         PatchBasedStandAlone"
25   )
26
27 // the packaged application is a resource of the task
28 cTask addResource "/data/cde11_zehan"
29 cTask addInput imageID
30
31 // imageID must be defined as output to be reused in Hook
32 val resultArchiveFile = Prototype[File] ("resultArchiveOut")
33 cTask addOutput ("AbdominalCT/Results", resultArchiveFile)
34 cTask addOutput imageID
35
36 // hooks allow result files retrieval
37 val resultHook = CopyFileHook(resultArchiveFile, "results_biomed
       /${imageID}")
38
39 // definition of the Biomed VO environment
40 GliteAuthentication() = P12Certificate(encrypted, "/homes/
       jpassera/.globus/grid_certificate_uk_2014.p12")
41 val biomed = GliteEnvironment("biomed")
42
43 // connect the tasks with transitions and run the workflow
44 val ex = exploIDsTask -< (cTask on biomed hook resultHook)
       toExecution
45 ex.start
```

Listing 2: Workflow describing the delegation of the studied application to the Biomed VO of the EGI computing grid

The first element introduced is an *ExplorationTask*. This is an OpenMOLE predefined task that establishes the list of different parameters combinations that can be created from the set of parameters. As we limit this example to the exploration of the whole set of images, the ExplorationTask will generate a list of the input images file names.

A *SystemExecTask* embeds the main component of the workflow: the segmentation application packaged with Yapa. Several notions appear when describing this task as it takes inputs, outputs and requires resources. The OpenMOLE DSL makes attaching all these elements to the task crystal clear to the external reader, as the associated operations are respectively *addInput*, *addOutput* and *addResource*.

At this stage, we have two tasks on the scene: the ExplorationTask generating the different combinations of input parameters and the SystemExecTask, embedding the actual application. The final part of the workflow connects these tasks and instantiates them on a computing environment.

Links between two tasks are called transitions in the OpenMOLE terminology. In our case, an *exploration transition* connects the tasks. The OpenMOLE

DSL describes this transition using the unfaltering fork symbol (-<). This symbol represents the multiple instances of the SystemExecTask created to process each parameter combination generated.

Last but not least, the SystemExecTask benefits from a specific care as its execution is delegated to a remote execution environment. This is achieved using the *on* keyword, recalling that a task is running *on* a particular environment. Brackets around this statement ensure that only this task is subject to remote execution, and the ExplorationTask remains in the local environment. This workflow delegates the execution to the EGI computing grid and more particularly to its Biomed VO. As shown in comments of Listing 2, a single word in the execution statement would delegate the task to another previously defined execution environment.

In this test case, we have chosen to study the exploration of a single parameter: the input image. Now that the application is packaged, and the workflow described, it is easy to extend this exploration to the whole set of parameters. The heavy computational workload resulting from such a workflow would only suit a large distributed computing environment such as the European Grid Infrastructure.

5 Conclusion

In this paper, we have shown how to delegate the execution of medical imaging applications running in a Linux platform to a distributed computing infrastructure. To do so, we have added new features to the OpenMOLE ecosystem, and shown its relevancy to manage this kind of experiments.

The Yapa packaging application can now handle complex Python setups using the Virtual Environment package. Regarding the available environments, the SLURM and Condor job schedulers are now supported by OpenMOLE, via the development of two new plugins for the GridScale library.

We have designed a workflow to explore the parameters of a segmentation method implemented in Python. This workflow can now be reused by any scientist through OpenMOLE, in order to reproduce the exploration of the same parameter set. This solution is extremely portable since the application distributed through OpenMOLE has been packaged with Yapa and can thus be executed on any platform running 2.6.32 Linux kernel or a more recent version.

This work shows that the genericity of the OpenMOLE platform platform does not prevent it tackle efficiently the distribution of medical imaging applications. Also, the free and open source licence of the tools belonging to the OpenMOLE ecosystem allows anyone to contribute to the project and extend the support to any missing computing environment.

Future works regarding the OpenMOLE platform will target the delegation of workflows to federated heterogeneous distributed computing environments.

Acknowledgments

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 319456.

References

[Barker and Van Hemert, 2008] Barker, A. and Van Hemert, J. (2008). Scientific workflow: a survey and research directions. In *Parallel Processing and Applied Mathematics*, page 746753. Springer.

[Guo, 2012] Guo, P. (2012). CDE: a tool for creating portable experimental software packages. *Computing in Science & Engineering*, 14(4):3235.

[MacKenzie-Graham et al., 2008] MacKenzie-Graham, A. J., Van Horn, J. D., Woods, R. P., Crawford, K. L., and Toga, A. W. (2008). Provenance in neuroimaging. *NeuroImage*, 42(1):178195.

[Mikut et al., 2013] Mikut, R., Dickmeis, T., Driever, W., Geurts, P., Hamprecht, F. A., Kausler, B. X., Ledesma-Carbayo, M. J., Mare, R., Mikula, K., Pantazis, P., Ronneberger, O., Santos, A., Stotzka, Rainer, Strhle, Uwe, and Peyriras, Nadine (2013). Automated processing of zebrafish imaging data: A survey. *Zebrafish*, 10(3):401–421.

[Miles et al., 2007] Miles, S., Groth, P., Branco, M., and Moreau, L. (2007). The requirements of using provenance in e-science experiments. *Journal of Grid Computing*, 5(1):125.

[Odersky et al., 2004] Odersky, M., Altherr, P., Cremet, V., Emir, B., Maneth, S., Micheloud, S., Mihaylov, N., Schinz, M., Stenman, E., and Zenger, M. (2004). An overview of the scala programming language. Technical report, Technical Report IC/2004/64, EPFL Lausanne, Switzerland.

[Reuillon et al., 2010] Reuillon, R., Chuffart, F., Leclaire, M., Faure, T., Dumoulin, N., and Hill, D. (2010). Declarative task delegation in OpenMOLE. In *High performance computing and simulation (hpcs), 2010 international conference on*, page 5562. IEEE.

[Reuillon et al., 2013] Reuillon, R., Leclaire, M., and Rey-Coyrehourcq, S. (2013). OpenMOLE, a workflow engine specifically tailored for the distributed exploration of simulation models. *Future Generation Computer Systems*, 29(8):19811990.

[Wang et al., 2014] Wang, Z., Bhatia, K. K., Glocker, B., de Marvao, A., Dawes, T., Kazunari, M., Mori, K., and Rueckert, D. (2014). Geodesic patch-based segmentation. In *Proceedings of MICCAI 2014*, Boston. to be published.

RESEARCH CENTRE
Imperial College London
Department of Computing
Huxley Building
London SW7 2AZ
United Kingdom

Publisher
Imperial College London
Department of Computing
Huxley Building
London SW7 2AZ
United Kingdom
http://www3.imperial.ac.uk/