A Service-Oriented Architecture enabling dynamic services grouping for optimizing distributed workflows execution

Tristan Glatard, Johan Montagnat, David Emsellem, Diane Lingrand

A Service-Oriented Architecture enabling dynamic service grouping for optimizing distributed workflow execution

Tristan Glatard [a,b,c,*] Johan Montagnat [a,c] David Emsellem [c]
Diane Lingrand [c,a]

[a] *I3S, CNRS, 2000 route des Lucioles, 06903 Sophia Antipolis, France*

[b] *Asclepios, INRIA Sophia, 2004 route des Lucioles, 06902 Sophia Antipolis, France*

[c] *University of Nice-Sophia Antipolis, 930 route des Colles, 06903 Sophia Antipolis, France*

Abstract

In this paper, we describe a Service-Oriented Architecture allowing the optimization of the execution of service workflows. We discuss the advantages of the service-oriented approach with regards to the enactment of scientific applications on a grid infrastructure. Based on the development of a generic Web-Services wrapper, we show how the flexibility of our architecture enables dynamic service grouping for optimizing the application execution time. We demonstrate performance results on a real medical imaging application. On a production grid infrastructure, the optimization proposed introduces a significant speed-up (from 1.2 to 2.9) when compared to a traditional execution.

Key words: Grid workflows, Service Oriented Architecture, Legacy code wrapper, Service grouping

* Corresponding author
Email addresses: `glatard@i3s.unice.fr` (Tristan Glatard), `johan@i3s.unice.fr` (Johan Montagnat), `emsellem@polytech.unice.fr` (David Emsellem), `lingrand@i3s.unice.fr` (Diane Lingrand).
URLs: `http://www.i3s.unice.fr/~glatard` (Tristan Glatard), `http://www.i3s.unice.fr/~johan` (Johan Montagnat), `http://www.i3s.unice.fr/~lingrand` (Diane Lingrand).

1 Introduction

Grid technologies are very promising for addressing the computing and storage needs arising from many scientific and industrial application areas. Grids have a potential for massive parallelism that can drastically improve applications execution time but, except in a few simple cases, it is often not straightforward to exploit for application developers. A tremendous amount of work has been put in the development of various sequential data processing algorithms without taking into account properties of distributed systems nor specific middlewares. Even considering new codes development, instrumenting applications with middleware specific interfaces or designing applications to explicitly take advantage of distributed grid resources is a significant burden for the developers, who are often reluctant to allocate sufficient effort on non application specific problems. Grid middlewares are therefore expected to ease as much as possible the migration of both legacy and new codes to a grid infrastructure by:

- proposing a non-intrusive interface to existing application code; and
- optimizing the execution of applications on grid resources.

The first point can be addressed by generic code wrappers which do not require code instrumentation for executing non-specific codes on a grid. In particular, they ease the reuse of legacy codes.

For a large range of scientific applications, the second point is addressed by workflow managers. Scientific data processing procedures often require to apply many data filtering, modeling, quantification and analysis procedures. Furthermore, large data sets often have to be processed. A workflow manager can describe the processing dependencies independently from the actual scientific codes involved. The associated workflow enactor can optimize the execution on a grid infrastructure by exploiting the data and code parallelisms intrinsically expressed in the workflow.

This paper deals with code migration on grid infrastructures. Our ultimate goal is to propose a generic system, able to gridify any legacy code efficiently. The two aspects highlighted above will be studied and solutions will be proposed.

Service-Oriented Architectures (SOA) have encountered a large success both in the Grid and in the Web communities. Most recent middlewares have adopted it in order to address interoperability and extensibility problems. Although SOAs have been widely adopted for middlewares design, and despite their known advantages, there are less frequently encountered in the design of scientific applications.

The advantages and drawbacks caused by service-based design are first discussed in section 2. A code wrapper that allows to benefit from a service-based approach at a very low development cost, even when considering legacy codes, is then described in section 3. An SOA workflow engine design is introduced in section 4 and optimization results measured through workflow executions on a grid by grouping sequential computing tasks, thus reducing overheads due to grid jobs submission, are shown in section 5. Experiments on a real application to medical images analysis are finally presented in section 6. They demonstrate that significant speeds-up can be achieved thanks to our grouping optimization.

2 Task-based and service-based applications

Two main paradigms are used in grid middlewares for describing and controlling application processings. The task-based approach is the most widely adopted and it has been exploited for the longest time. It consists in an exhaustive description of the command-line and the remote execution of the application code. The service-based approach has more recently emerged. It consists in using a standard invocation protocol for calling application code embedded in the service. It is usually completed by a service discovery and an interface description mechanism.

2.1 Task-based job submission

In the task-based job submission approach, each processing is related to an executable code and described through an individual computation *task*. A task description encompasses at least the executable code name and a command-line to be used for code invocation. It may be completed by additional parameters such as input and output files to be transferred prior or next to the execution, and additional task scheduling information such as minimum system requirements. Tasks may be described either directly on the job submission tool command-line, or indirectly through a task description file. Unless considering very simple code invocation use cases, description files are often needed to specify the task in details. Many file description formats have been proposed and the OGF [1] unified different formats in the Job Submission Description Language (JSDL) [1]. The task-based approach is also often referred to as *global computing*.

In the task-based paradigm, code invocation is straight-forward, and does

[1] OGF, Open Grid Forum, http://www.gridforum.org/

not require any adaptation of the user code and for this reason it has been implemented in most existing batch systems for decades (e.g. PBS [2], NQS [3], OAR [4]). Many grid middlewares, such as Globus Toolkit 2 [5], CONDOR [6] and gLite [7] are also task-based from the application code perspective. Indeed, even if those middlewares (in particular gLite) may themselves be designed as a set of interoperating services, the computing resources of the grid are accessed through task submissions.

2.2 Service-based code execution

The service-based approach was widely adopted for dealing with heterogeneous and distributed systems. In particular, for middleware development, the OGSA framework [8] and the subsequent WSRF standard encountered a wide adoption from the international community. In the service-based approach, the code is embedded in a standard service shell. The standard defines an interface and an invocation procedure. The Web-Services standard [9], supported by the W3C is the most widely available although many existing implementations do not conform to the whole standard yet. It has been criticized for the low efficiency resulting from using text messages in XML format and alternatives such as GridRPC [10] have been designed to speed-up message exchanges. The service-based approach is also often referred to as *meta computing*. Middlewares such as DIET [11], Ninf [12], Netsolve [13] or Globus Toolkit 4 [14] adopted this approach.

The main advantage of the service based approach is the flexibility that it offers. Clients can discover and invoke any service through standard interfaces without any prior knowledge on the code to be executed. The service-based approach delegates to the server side the actual code execution procedure. However, all application codes need to be instrumented with the service interface to become available. In the case of legacy code applications, it is often not the case and an intermediate code invocation layer or some code reworking is needed to exploit this paradigm. Users are often reluctant to invest efforts in writing specific code for services on the application side for different reasons:

- The complexity of standards makes service conformity a matter of specialists. Some tooling are available for helping in generating service interfaces but they cannot be fully automated and often require a developer intervention.
- Standards tend to evolve quickly, especially in the grid area, obsoleting earlier efforts in a too short time scale.
- Multiple standards exist and a same application code may need to be executed through different service interfaces.
- In the case of legacy code, recompilation for instrumenting the code may be

very difficult or even impossible (in case of non availability of source code, compilers, dependencies, etc).

Therefore, a user-friendly way to deal with legacy code is to propose a generic service-compliant code execution interface.

2.3 Discussion

Apart from the invocation procedures and the implementation difficulties mentioned above, the task-based and the service-based approaches differ by several fundamental points which impact their usage:

- To submit a task-based job, a user needs to precisely know the command-line format of the executable, taking into account all of its parameters. In the scientific community, it is not always the case when the user is not one of the developers. Conversely, in the service-based approach, the actual code invocation is delegated to the service which is responsible for the correct handling of the invocation parameters. The service is a black box from the user side and to some extent, it can deal with the correct parametrization of the code to be executed.
- The handling of input/output data is very different in both cases. In the task-based approach, input/output data have to be explicitly specified in the task description. Executing the same code on different data items requires the rewriting of a new task description. Services better decouple the computation and data handling parts. A service dynamically receives inputs as parameters. This decoupling between treatments and data is particularly important when considering the processing of complete data sets rather than single data, as it is commonly targeted on grid infrastructures.
- The service-based approach enables discovery mechanisms and dynamic invocation even for *a priori* unknown services. This provides a lot of flexibility both for the user (discovery of available data processing tools and their interface) and the middleware (automatic selection of services, alternative service discovery, fault tolerance, etc).
- In the service-based framework, the code reusability is also improved by the availability of a standard invocation interface. In particular, services are naturally well adapted to describe applications as complex workflows, chaining different processings whose outputs are piped to the inputs of each other.
- Services are adding an extra layer between the code invocation and the grid infrastructure on which jobs are submitted. The caller does not need to know anything about the underlying middleware that will be directly invoked internally by the service. Different services might even communicate with different middlewares and/or different grid infrastructures.

- Yet, service deployment introduces an extra effort with regards to the task-based approach. Indeed, to enable the invocation, services first have to be installed on all the targeted resources, which becomes a challenging problem when their number increases.

The flexibility and dynamic nature of services described above is usually very appreciated from the user point of view. Given that application services can be deployed at a very low development cost, there are number of advantages in favor of this approach.

From the middleware developers point of view, the efficient execution of application services is more difficult though. As mentioned above, a service is an intermediate layer between the user and the grid middleware. Thus, the user does not know anything of the underlying infrastructure. Tuning the jobs submission for a specific application is more difficult. Services are completely independent from each other and global optimization strategies are thus hardly usable. Therefore, some precautions need to be taken when considering service based applications to ensure good performance.

2.4 Workflow of services

Building applications by assembling legacy codes for processing and analyzing data is very common. It allows code reusability without introducing a too high load on the application developers. The logic of such a composed application, referred to as the *application workflow*, is described through a set of computation tasks to perform and constraints on the order of processings such as data dependencies.

Many workflow representation formats and execution managers have been proposed in the literature with very different properties [15]. The emblematic task-based workflow manager is the CONDOR Directed Acyclic Graph Manager (DAGMan) [16], on top of which the Pegasus system is built [17]. Based on the static description of such a workflow, many different optimization strategies for the execution have been proposed [18]. The service-based approach has been implemented in different workflow managers such as the Kepler system [19], the Taverna workbench [20], Triana [21] and the MOTEUR enactor developed in our team [22], which aims at optimizing the execution of data intensive applications.

The main interest for using grid infrastructures is to exploit the potential application parallelism thanks to the availability of the grid resources. There are three different levels of parallelism that can be exploited in service-based workflows [32]. Service grouping strategies have to cautiously take care of them in order to avoid execution slow down.

Workflow parallelism. The intrinsic workflow parallelism depends on the application graph topology. For instance if we consider the application example presented in figure 7, services `Baladin` and `Yasmina` can be executed in parallel.

Data parallelism. Data segments are processed independently from each other. Therefore, different input data segments can be processed in parallel on different resources. This may lead to considerable performance improvements given the high level of parallelism achievable in many scientific applications.

Service parallelism. The processings of two different data sets by two different services are totally independent from each other. This pipelining model, very successfully exploited inside CPUs, can be adapted to sequential parts of service-based workflows. Considering the workflow represented on figure 7, services `crestLines` and `crestMatch` may be run in parallel on independent data sets. In practice this kind of parallelism strongly improves the workflow execution on production grids.

3 Generic Web-Service wrapper

3.1 Wrapping application codes

To ease the embedding of legacy or non-instrumented codes in the service-based framework, an application-independent job submission service is required. In this section, we briefly review systems that are used to wrap legacy code into services to be embedded in service-based workflows.

The Java Native Interface (JNI) has been widely adopted for the wrapping of legacy codes into services. Wrappers have been developed to automate this process. In [24], an automatic JNI-based wrapper of C code into Java and the corresponding type mapper with Triana [21] is presented: JACAW generates all the necessary java and C files from a C header file and compiles them. A coupled tool, MEDLI, then maps the types of the obtained Java native method to Triana types, thus enabling the use of the legacy code into this workflow manager. Related to the ICENI workflow manager [25], the wrapper presented in [26] is based on code re-engineering. It identifies distinct components from a code analysis, wraps them using JNI and adds a specific CXML interface layer to be plugged into an ICENI workflow.

The WSPeer framework [27], interfaced with Triana, aims at easing the deployment of Web-Services by exposing many of them at a single endpoint. It differs from a container approach by giving to the application the control over service invocation. The Soaplab system [28] is especially dedicated to the wrapping of command-line tools into Web-Services. It has been largely used to integrate bioinformatics executables in workflows with Taverna [20]. It is able to deploy a Web-Service in a container, starting from the description of a command-line tool. This command-line description, referred to as the metadata of the analysis, is written for each application using the ACD text format file and then converted into a corresponding XML format. Among domain specific descriptions, the authors underline that such a command-line description format must include (i) the description of the executable, (ii) the names and types of the input data and parameters and (iii) the names and types of the resulting output data. As described latter, the format we used includes those features and adds new ones to cope with requirements of the execution of legacy code on grids.

The GEMLCA environment [29] addresses the problem of exposing legacy code command-line programs as Grid services. It is interfaced with the P-GRADE portal workflow manager [30]. The command-line tool is described with the LCID (Legacy Code Interface Description) format which contains (i) a description of the executable, (ii) the name and binary file of the legacy code to execute and (iii) the name, nature (input or output), order, mandatory, file or command line, fixed and regular expressions to be used as input validation. A GEMLCA service depends on a set of target resources where the code is going to be executed. Architectures to provide resource brokering and service migration at execution time are presented in [31].

Apart from this latest early work, all of the reviewed existing wrappers are static: the legacy code wrapping is done offline, before the execution. This is hardly compatible with our approach, which aims at optimizing the whole application execution at run time. As we will see in section 5.2, our design is exploiting a dynamic grouping strategy optimization that is enacted through a service factory called at execution time by the MOTEUR workflow manager. We thus developed a specific grid submission Web-Service, which can wrap an executable at run time.

3.2 A generic Web-Service wrapper

We designed a generic application code wrapper compliant with the Web-Services specification. It enables the execution of a legacy executable through a standard service interface. This service is generic in the sense that it is unique and it does not depend on the executable code to submit. It exposes a standard interface that can be used by any Web-Service compliant client to invoke the execution. It completely hides the grid infrastructure from the end user as it takes care of the interaction with the grid middleware. This interface plays the same role as the ACD and LCID files quoted in the previous section, except that it is interpreted at the execution time.

To accommodate to any executable, the generic service is taking two different inputs: a descriptor of the legacy executable command line format, and the input parameters and data of this executable. The production of the legacy code descriptor is the only extra work required from the application developer. It is a simple XML file which describes the legacy executable location, command line parameters, input and output data.

3.3 Legacy code descriptor

The command line description has to be complete enough to allow a dynamic composition of the command line from the list of parameters at the service invocation time and to access the executable and input data files. As a consequence, the executable descriptor contains:

(1) The name and access method of the executable. In our current implementation, access methods can be a URL or a Grid File Name (GFN). The wrapper is responsible for fetching the data according to different access modes.
(2) The access method and command-line option of the input data. As our approach is service-based, the actual name of the input data files is not mandatory in the description. Those values will be defined at the execution time. This feature differs from various job description languages used in the task-based middlewares. The command-line option allows the service to dynamically build the actual command-line at the execution time.
(3) The command-line option of the input parameters: parameters are values of the command-line that are not files and therefore which do not have any access method.
(4) The access method and command-line option of the output data. This information enables the service to register the output data in a suitable place after the execution. Here again, in a service-based approach, names of output data files cannot be statically determined because output file names are only generated at execution time.
(5) The name and access method of the sandboxed files. Sandboxed files are external files such as dynamic libraries or scripts that may be needed for the execution although they do not appear on the command-line.

3.4 Example

An example of a legacy code description file is presented in figure 1. It corresponds to the description of the service `crestLines` of the workflow depicted in figure 7. It describes the script `CrestLines.pl` which is available from the server `legacy.code.fr` and takes 3 input arguments: 2 files (options `-im1` and `-im2` of the command-line) that are already registered on the grid as GFNs and 1 parameter (option `-s` of the command-line). It produces 2 files that will be registered on the grid. It also requires 3 sandboxed files that are available from the server (`Convert8bits.pl`, `copy` and `cmatch`).

The command-line description format presented here might have some limitations when applications that are not pure command-line are taken into consideration. For instance, some applications may require input from the stdin or even ask for a graphical interaction with the user. To cope with these limitations, our description format could easily be extended. Yet, it will be dependent on the ability of the grid middleware to handle such interactive jobs.

3.5 Discussion

This generic service highly simplifies application development because it is able to wrap any legacy code with a minimal effort. The application developer only needs to write the executable descriptor for her code to become service aware.

But its main advantage is in enabling the sequential service grouping optimization that will be described in section 5. Indeed, as the workflow enactor has access to the executable descriptors, it is able to dynamically create a virtual service, composing the command lines of the codes to be invoked, and submitting a single job corresponding to this sequence of command lines invocation.

It is important to notice that our solution remains compatible with the services standards. The workflow can still be executed by other enactors, as we did not introduce any new invocation method. Those enactors will make standard service calls (e.g. SOAP ones) to our generic wrapping service. However, the optimization strategy described in the next section is only applicable to services including the descriptor mentioned in section 3.3. We call those services MOTEUR services, referring to our workflow manager presented in section 2.4.

```
<description>
  <executable name="CrestLines.pl">
    <access type="URL">
      <path value="http://legacy.code.fr"/>
    </access>
    <value value="CrestLines.pl"/>
    <input name="floating_image" option="-im1">
      <access type="GFN"/>
    </input>
    <input name="reference_image" option="-im2">
      <access type="GFN"/>
    </input>
    <input name="scale" option="-s"/>
    <output name="crest_reference" option="-c1">
      <access type="GFN"/>
    </output>
    <output name="crest_floating" option="-c2">
      <access type="GFN"/>
    </output>
    <sandbox name="convert8bits">
      <access type="URL">
        <path value="http://legacy.code.fr"/>
      </access>
      <value value="Convert8bits.pl"/>
    </sandbox>
    <sandbox name="copy">
      <access type="URL">
        <path value="http://legacy.code.fr"/>
      </access>
      <value value="copy"/>
    </sandbox>
    <sandbox name="cmatch">
      <access type="URL">
        <path value="http://legacy.code.fr"/>
      </access>
      <value value="cmatch"/>
    </sandbox>
  </executable>
</description>
```

Fig. 1. Legacy code descriptor example for our service wrapper. The location of the executable is first described. Then, inputs and outputs participating in the command-line generation are specified. Finally, external dependencies (such as dynamic libraries) are described in the sandbox section.

4 Workflow manager SOA

The generic Web-Service wrapper introduced in section 3 drastically simplifies the embedding of legacy code into application services. However, it mixes two different roles:

- the legacy command line generation
- the grid submission.

Submission is only dependent on the target grid and not on the application service itself. In a SOA it is preferable to split these two roles into two independent services for several reasons. First, the code handling the job submission does not need to be replicated in all application services. Second, the submission role can be transparently and dynamically changed (to submit to a different infrastructure) or updated (to adapt to middleware evolutions).

Figure 2 illustrates the resulting SOA design through a simple workflow deployment example. The workflow manager orchestrates three different services P_1, P_2 and P_3. These are standard Web-Services: either legacy code wrapping services (P_1 and P_2, in blue) or any Web-Service (P_3) that the workflow manager can invoke. The services may submit jobs to a grid infrastructure. In particular, MOTEUR services are interfaced with a submission service (in red). There may exist various submission services corresponding to several grid infrastructures. Services may thus use different infrastructures and even dynamically change the submission target during the execution (*e.g.* taking into account the infrastructure load). In the next section, we will see how the flexibility of this SOA can be exploited to dynamically optimize the execution of an application workflow.

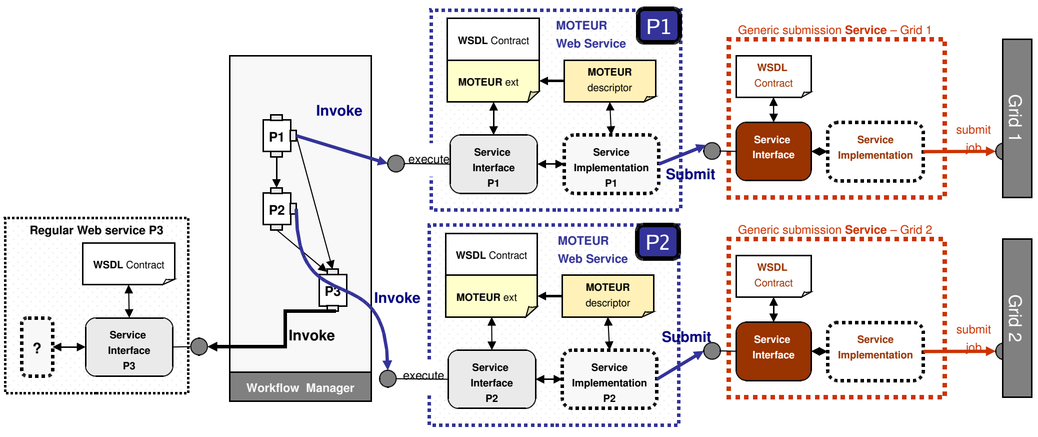

Fig. 2. Workflow manager SOA.

5 Service grouping optimization strategy

In this section, we propose a service grouping strategy to optimize the execution time of a workflow. Grouping services of a workflow may reduce the total overhead induced by the submission, scheduling, queuing and data transfers time because it reduces the number of submitted jobs required to run the application. This impact is particularly important on production infrastructures, where this overhead can be very high (several minutes) due to the large scale and multi-users nature of those platforms. Consider the simple workflow represented on the left side of figure 3. On top, services P_1 and P_2 are invoked independently. Data transfers are handled by each service and the connection between the output of P_1 and the input of P_2 is handled at the workflow engine level. On the bottom, P_1 and P_2 are grouped into a virtual single service. This service is capable of sequentially invoking the code embedded in both services, thus resolving the data transfer and independent code invocation issues.

Conversely, grouping services may also reduce the parallelism and we have to take care of the grouping strategy in order to avoid performance losses. In particular, grouping sequentially linked services is interesting because they do not benefit from any parallelism. Those groupings can be done at the services level, *i.e* they will be available for each data item processed by the workflow. For example, considering the workflow of our application presented on figure 7, services `crestLines` and `crestMatch` can be grouped without parallelism loss as well as services `PFMatchICP` and `PFRegister`.

From the middleware point of view, grouping strategies may also be interesting because it reduces the total number of jobs to handle, thus decreasing the global load imposed on the infrastructure. Yet, grouping services leads to the submission of longer jobs, which may also increase the average queuing time as a damaging side effect.

5.1 Grouping strategy

Service grouping can lead to significant speed-ups, especially on production grids that introduce high overheads, as it will be demonstrated in the next section. However, it may also slow down the execution by limiting one of the 3 levels of parallelism described in section 2.4. We thus have to determine efficient strategies to group services.

In order to determine a grouping strategy that does not introduce any slowdown, neither from the user point of view, nor from the infrastructure one, we impose the two following constraints:

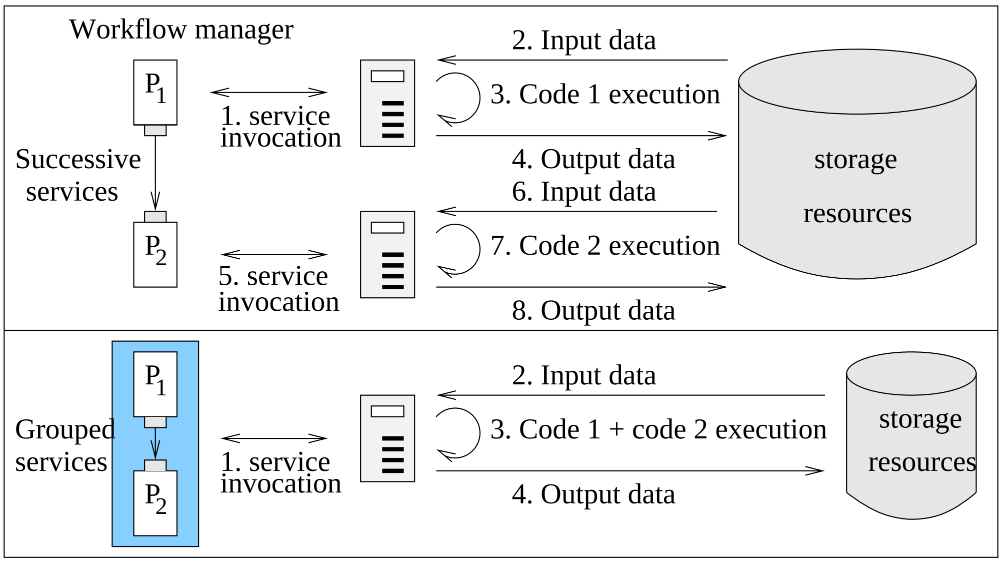

Fig. 3. Classical services invocation (top) and service grouping (bottom).

- the grouping strategy must not limit any kind of parallelism (user point of view)
- during their execution, jobs cannot communicate with the workflow manager (infrastructure point of view).

The second constraint prevents a job from holding a resource just waiting for one of its ancestor to complete. An implication of this constraint is that if services A and B are grouped together, the results produced by A will only be available once B will have completed.

A workflow may include both MOTEUR Web-Services (*i.e.* services that are able to be grouped) and classical ones, that could not be grouped. Assuming those two constraints, we can prove the following rule:

> Let A be a MOTEUR service of the workflow and $\{B_0,...B_n\}$ its children in the service graph. Grouping B_i and A does not lead to any parallelism loss IF and ONLY IF:
> (1) B_i is an ancestor of every B_j for every $i \neq j$ and
> (2) each ancestor C of B_i is an ancestor of A or A itself.

Let us first prove that (1) and (2) are *necessary* conditions to avoid parallelism loss. If (1) is not respected, then there exists a child B_j of A which is not a descendant of B_i. If A and B_i are grouped, then workflow parallelism is broken between B_i and B_j because B_j has to wait for B_i to complete before starting. Similarly, if (2) is not respected, then there exists an ancestor C of B_i that is not an ancestor of A and workflow parallelism is broken between A and C when A and B_i are grouped.

(1) and (2) are also *sufficient* to avoid any parallelism break in the workflow.

Let us first notice that grouping services does not break **data parallelism** because this kind of parallelism only concerns a single service of the workflow. Moreover, **service parallelism** relies on the independence of the processings of two different data segments by two successive services. As service grouping does not prevent B_i from processing a given piece of data while A is processing another one (assuming that data parallelism is not broken, which is the case here), service grouping does not break service parallelism. Thus, we are left to prove that (1) and (2) guarantee that **workflow parallelism** is not broken by grouping A and B_i. (1) guarantees that there is no workflow parallelism between B_i and every B_j. Workflow parallelism is thus likely to concern B_i only for services that are not children of A and thus cannot be broken by grouping A and B_i. Similarly, (2) guarantees that there is no workflow parallelism between A and every other ancestor of B_i. Workflow parallelism is thus likely to concern A only for services that are not ancestors of B_i and thus cannot be broken by grouping A and B_i. ■

Our grouping strategy tests this rule for each MOTEUR service of the workflow. Groups of more than two services may be recursively composed by successive matches of the grouping rule.

The constraints applied by the matching rule are illustrated on three different grouping examples in figure 4. This simplified workflow was extracted from our medical imaging application (see figure 7). It is made of four MOTEUR services. As it can be seen from the workflow graph, the data dependencies will enforce a sequential execution of these four services. It is therefore expected that the four services are grouped into a single one in order to minimize the job submission overhead. On this figure, notations nearby the services correspond to the ones introduced in the grouping rule. For each of the 3 examples of figure 4, the grouping of the two services outlined by a blue box is studied:

(1) On the left of figure 4, the tested MOTEUR service A is `crestLines`. A is connected to the workflow inputs and it has two children: B_0 and B_1. B_0 is a father of B_1 and it only has as single ancestor which is A. Thus, the rule matches: A and B_0 can be grouped. If there were a service C ancestor of B_0 but not of A as represented on the figure, the rule would not match: A and C would have to be executed in parallel before starting B_0. Similarly, if there were a service D child of A but not of B_0, then the rule would not match as the workflow manager would need to communicate results during the execution of the grouped jobs in order to allow workflow parallelism between B_0 and D.
(2) In the middle of figure 4, the tested service A is now `crestMatch`. A has a single child: B_0. B_0 has two ancestors, A and C. The rule matches because C is an ancestor of A. A and B_0 can then be grouped.
(3) On the right of figure 4, A is the `PFMatch` service. It has only one child B_0 which only has a single ancestor, A. The rule matches and those services

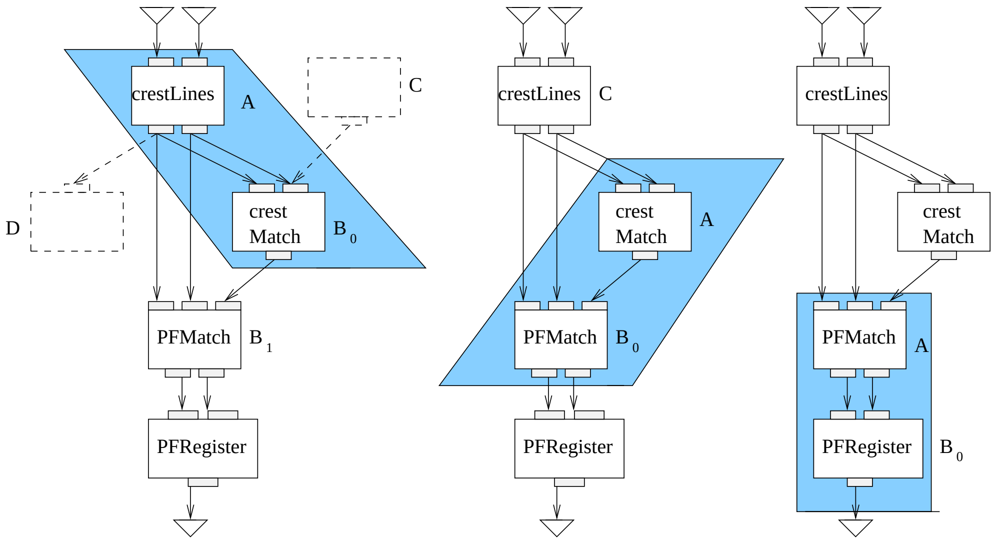

Fig. 4. Service grouping examples. On this workflow, the grouping rule matches 3 times (once for each green box), thus resulting in a single service wrapping those 4. On the left part of the figure, service C or D would prevent the grouping between `crestLines` and `crestMatch` because it would break workflow parallelism between A and C and between B_0 and D.

can thus be grouped.

Finally, when A is the `PFRegister` service, the grouping rule does not match because it does not have any child. Note that in this example, the recursive grouping strategy leads to a single job submission, as expected.

5.2 Dynamic generic service factory

In practice, grouping jobs in the task-based approach is straightforward whereas it is usually not possible in the service-based approach given that:

- the services composing the workflow are totally independent from each other
- the grid infrastructure handling the jobs does not have any information concerning the workflow and the job dependencies.

That is why a new architecture has to be designed to allow service grouping.

To do that, an advantage of the SOA design of our workflow engine is that it can dynamically enable service grouping by analyzing the workflow and generating grouped services on the fly. A service factory is added to the architecture. Its role is to instantiate both the legacy code wrapping services and the grouped services. The complete architecture is diagrammed on figure 5.

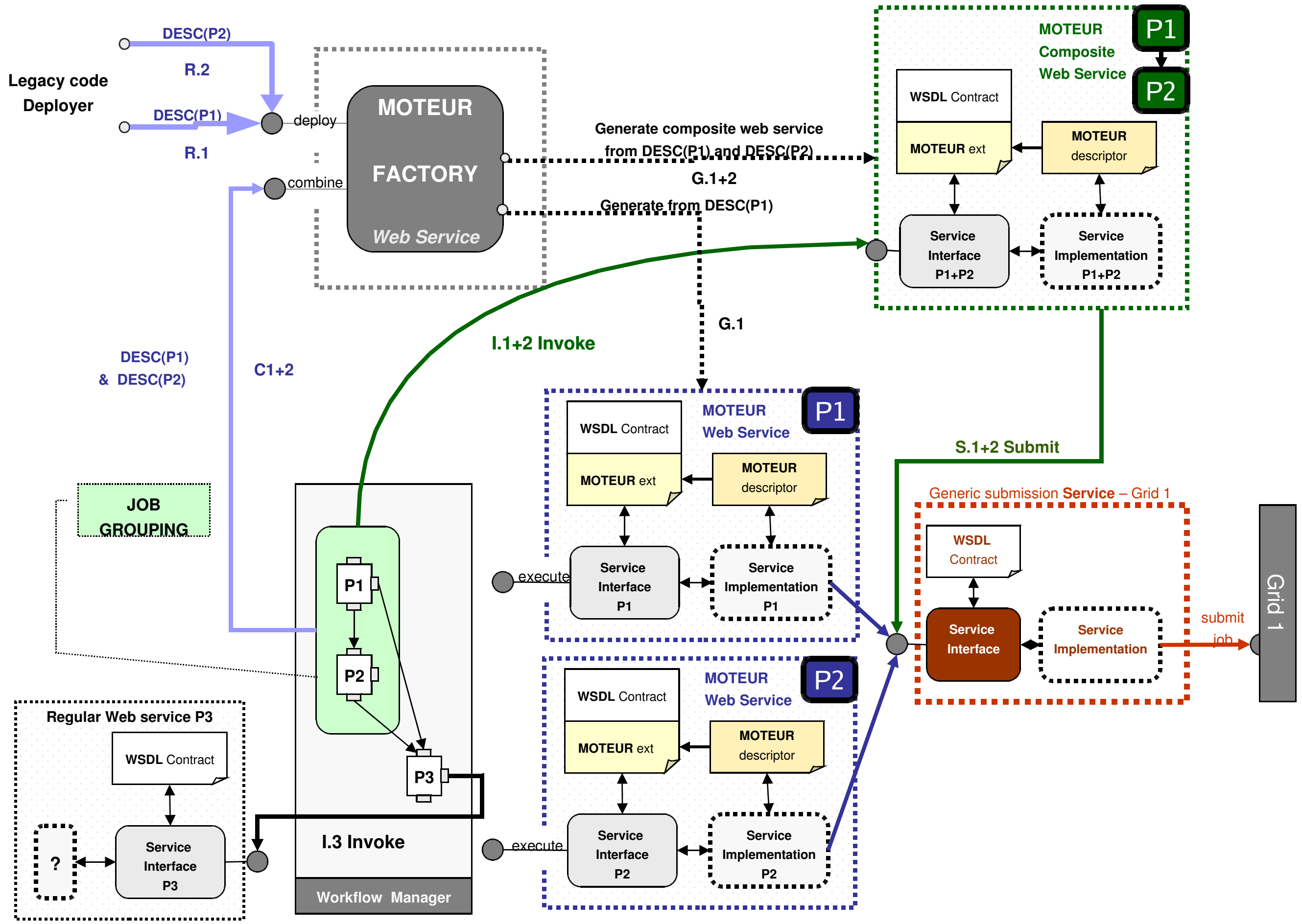

Fig. 5. Services factory enabling service grouping. The MOTEUR factory is able to deploy a Web-Service from the description of an executable (see figure 1 for an example of such a description). To group services, the workflow engine (MOTEUR) dynamically invokes the services factory with the description of the algorithms to group (DESC(P1) and DESC(P2)). The factory then deploys a composite Web-Service P1+P2 that can be directly invoked by the workflow engine.

The service factory is responsible for dynamically generating and deploying application services. The aim of this factory is to achieve two antagonist goals:

- To expose legacy codes as autonomous Web-Services respecting the main principles of SOA.
- To enable the grouping of two of these Web-Services as a unique one for optimizing the execution.

On one hand, the specific Web-Service implementation details (*i.e.* the execution of the wrapped code on a grid infrastructure) are hidden to the consumer. On the other hand, when the consumer is a workflow manager which can group jobs, it needs to be aware of the real nature the Web-Services (the encapsulation of a MOTEUR descriptor) so that it could merge them at run time. We choose to use the WSDL XML Format extension mechanism which allows to insert user defined XML elements in the WSDL content itself. We thus strictly conform to the WSDL standard while enabling our optimization strategy.

On figure 5, we exemplify the architecture through a usage scenario:

R.1 First, the legacy code provider registers a MOTEUR XML descriptor P1 to the MOTEUR factory.
G.1 The factory, then dynamically generates a Web-Service which wraps the submission of the legacy code to the grid via the generic service wrapper.
R.2 Another provider do the same with the descriptor of P2.

The resulting Web-Services expose their WSDL contracts to the external world with a specific extension associated with the WSDL operation. For instance, the WSDL contract resulting of the deployment of the `crestLines` legacy code described on figure 1 is printed on figure 6. This WSDL document defines two types (`CrestLines-request` and `CrestLines-response`) corresponding to the descriptor inputs and outputs and a single `Execute` operation. Notice that in the binding section, the WSDL document contains an extra `MOTEUR-descriptor` tag pointing to the URL of the legacy code descriptor file (`location`) and a binding to the Execute operation (`soap:operation`).

Suppose now that the workflow manager identifies a service grouping optimization (*e.g.* P1 and P2, displayed in green in figure 5). Because of its ability to discover the extended nature of these two services, the engine can retrieve the two corresponding MOTEUR descriptors.

C.1+2 The workflow manager can ask the factory to *combine* them and
G.1+2 generate a single composite Web-Service which exposes an operation taking its inputs from P1 (and P2 inputs coming from other external services) and returning the outputs defined by P2 (and P1 outputs going to other external services).

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions ...>
 <types>
  <schema>
   <element name="CrestLines-request">
     <complexType>
      <sequence>
       <element name="floating_image"
                type="string"... />
       <element name="reference_image"
                type="string"... />
       <element name="scale" type="string"... />
      </sequence>
     </complexType>
    </element>
    <element name="CrestLines-response">
      <complexType>
        <sequence>
          <element name="crest_reference"
                   type="string"... />
          <element name="crest_floating"
                   type="string"... />
        </sequence>
      </complexType>
    </element>
  </schema>
 </types>
 <message name="ExecuteSoapIn">
  <part name="parameters"
        element="CrestLines.pl-request" />
 </message>
 <message name="ExecuteSoapOut">
  <part name="parameters"
        element="CrestLines.pl-response" />
 </message>
 <portType name="CrestLines.plSoap">
  <operation name="Execute">
   <input message="ExecuteSoapIn" />
   <output message="ExecuteSoapOut" />
  </operation>
 </portType>
 <binding ...>
  <soap:binding transport="http://..." />
   <operation name="Execute">
    <soap:operation soapAction="http://.../Execute"
          style="document" />
     <MOTEUR-descriptor xmlns="urn:...">
      <location>http://...</location>
     </MOTEUR-descriptor>
     ....
   </operation>
 </binding>
</definitions>
```

Fig. 6. Extended WSDL generated by the factory for the code introduced in figure 1

I.1+2 The workflow manager can invoke this composite Web-Service. It is of the same type than any regular legacy code wrapping service and it is accessible through the same interface.

S.1+2 It also delegates the grid submission to the generic submission Web-Service by sending the composite MOTEUR descriptor and the input link of P1 and P2 in the workflow.

6 Experiments on a production grid

To quantify the speed-up introduced by service grouping on a real workflow, we made experiments on the EGEE production grid infrastructure. The EGEE system is a pool of thousands computers (standard PCs) and storage resources accessible through the gLite middleware. The resources are assembled in computing centers, each of them running its internal batch scheduler. Jobs are submitted from a user interface to a central Resource Broker which distributes them to the available resources. The access to EGEE grid resources is controlled for each Virtual Organizations (VOs). For our VO, about 3000 CPUs accessible through 25 batch queues were available at the time of the experiments. The large scale and multi-users nature of this infrastructure makes the overhead due to submission, scheduling and queuing time of the order of 5 to 10 minutes. Limiting job submissions by service grouping is therefore highly suitable on this kind of production infrastructure.

6.1 Experimental workflows

We made experiments on a medical image analysis application which is made from 6 legacy algorithms developed by the Asclepios team of INRIA Sophia-Antipolis [33,34]. The workflow of this application is represented on figure 7. It aims at assessing the accuracy of 4 registration algorithms, namely `crestMatch`, `PFMatchICP/PFRegister`, `Baladin` and `Yasmina`. A number of input image pairs constitute the input of the workflow (`floating image` and `reference image`). Those pairs are first registered by the `crestMatch` method and this result initializes the 3 remaining algorithms. At the end of the workflow, the `MultiTransfoTest` service is a statistical step that computes the accuracy of each algorithm from all the previously obtained results. `crestLines` is a preprocessing step for `crestMatch` and `PFMatchICP`. The total CPU time consumed by this workflow is about 15 minutes *per input data set*. For 126 input images, the CPU time is thus 31.5 hours, which motivates the use of grids for this application.

To show how service grouping is able to speed-up the execution on highly sequential applications, we also considered a sub-workflow of our application, as shown in figure 7. It is made of 4 services that correspond to the `crestLines`, `crestMatch`, `PFMatchICP` and `PFRegister` ones in the application workflow. Our grouping rule groups those 4 services into a single one, as it has been detailed in the example of figure 4. It is important to notice that even if this sub-workflow is sequential, and thus does not benefit from workflow parallelism, its execution on a grid does make sense because of data and service parallelisms. To evaluate the impact of our grouping strategy on the perfor-

Number of input image pairs	Sub-workflow (figure 4)			Whole application (figure 7)		
	Number of jobs		Speed-up	Number of jobs		Speed-up
	Regular	Grouping		Regular	Grouping	
12	48	12	2.91	72	48	1.42
66	264	66	1.72	396	264	1.34
126	504	126	2.30	756	504	1.23

Table 1
Grouping strategy speed-ups.

mance, we compared the execution times of those workflows with and without the grouping strategy.

Those experiments involved 3060 jobs, corresponding to a total CPU time of 4.5 days. For each number of data items, only a single workflow execution was performed. Yet, to guarantee that the grid status was the same between the regular (without grouping) and optimized strategies, we submitted those two cases simultaneously for each data set and using the same Resource Broker. The optimized and regular executions were thus compared in similar conditions.

The scheduling of the jobs submitted by these workflows is completely delegated to the EGEE grid middleware. On this infrastructure, two different levels of scheduling are performed. First, a particular computing center is selected by the Resource Broker, according to the job's requirements and the load of the sites. Second, a batch scheduler is responsible for the node allocation at the site level. Consequently, we had no control on the scheduling and each job may be executed on any of the nodes available for our VO. Several grid sites may be used by a single workflow. The overall EGEE scheduling policy is not centrally defined but results from the interactions of largely autonomous policies.

6.2 Results

Table 1 presents the speed-ups induced by our grouping strategy for a growing number of input image pairs and for the two experimental workflows described above. This speed-up is computed as the ratio of a regular grid execution time (where each service invocation leads to a job submission) over the execution time using the grouping strategy. We can notice on those tables that service grouping does effectively provide a significant speed-up on the workflow execution. This speed-up is ranging from 1.23 to 2.91.

The speed-up values are greater on the sub-workflow than on the whole application one. Indeed, on the sub-workflow, 4 services are grouped into a single one, thus saving 3 job submissions for each input data set. On the whole application workflow, the grouping rule is applied only twice, thus only saving 2 job submissions for each input data set, as depicted on figure 7.

The overhead of the service grouping optimization proposed in this paper remains negligible with respect to the grid overhead. Indeed, grouping services is done once for all the data items, at the beginning of the workflow execution. It only consists in searching for workflow services for which the rule described in section 5.1 matches. Thus, the overhead of service grouping is in the order of a few seconds, whereas the grid introduces an overhead of several minutes per job.

It is true that the service grouping results presented in this section are limited to the scope of our particular application. Investigating how the workflow topology and the nature of the submitted jobs would impact the speed-up values would be an interesting perspective to this work. Nonetheless, being able to forecast the performance of a workflow on a production grid such as EGEE is definitely a non-trivial problem. This kind of infrastructure is highly variable and non stationary so that deterministic models are hardly usable. We started investigating probabilistic models in order to be able to predict the impact of a given optimization on the execution time of a workflow (*e.g* in [35]).

7 Conclusion

In this paper, we discussed the advantages of the service-oriented approach for enabling scientific application codes on a grid infrastructure. We described an application-independent non intrusive legacy code wrapper that works at run time, by interpreting a command-line description file. We designed a workflow manager SOA taking advantage of this wrapper to enact complex scientific applications on a grid. Any legacy code-based application can thus be instantiated by only defining textual MOTEUR descriptors.

We then introduced a workflow optimization strategy based on an extension of the wrapper. This strategy consists in grouping services that do not benefit from any parallelism in order to reduce the impact of the grid overhead. We took advantage of the flexibility of the workflow architecture to introduce a new service factory enabling dynamic and automated instantiation of legacy code wrapping and the service grouping.

We showed results on a real medical imaging application workflow deployed on the EGEE production grid infrastructure. Our grouping strategy is able to

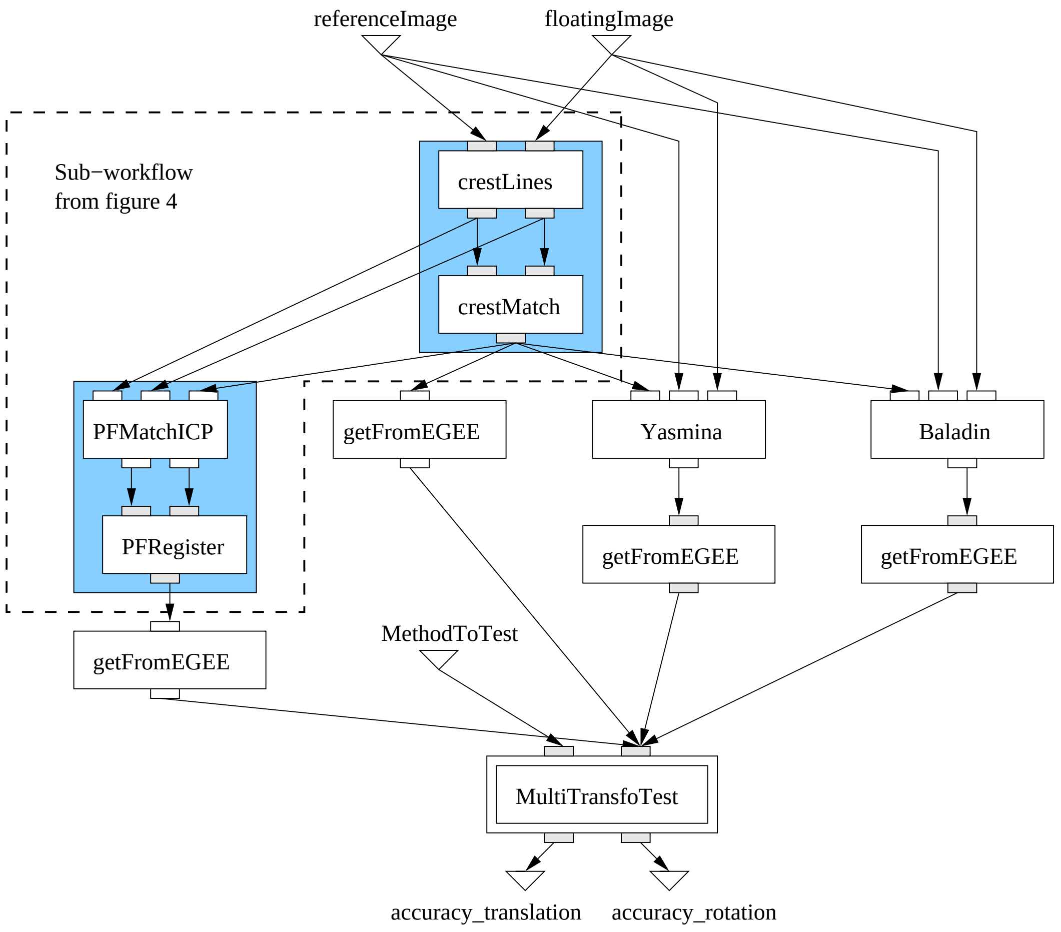

Fig. 7. Workflow of the application. Services to be grouped are squared in blue. The extracted sub-workflow is grouped into a single service, as detailed on figure 4. Only `crestLines`, `crestMatch`, `PFMatchICP`, `PFRegister`, `Yasmina` and `Baladin` lead to a grid job submission. The other services are computed locally.

provide significant speed-ups in the range 1.2 to 2.9 on a real application. On more sequential workflows, the speed-up increases to almost 3.

It is important to notice that the grouping strategy presented in this chapter is very unlikely to slow down the application because it does not break any parallelism (even if it is true that some side-effects resulting from an increase of the job size may limit the expected speed-up). Future directions in service grouping could be to limit parallelism at some point, thus further reducing the number of submitted jobs and the impact of the grid overhead on the execution. In this case, a compromise would have to be found between parallelism loss and overhead reduction. We started investigating such a strategy in [36] where data parallelism is restricted in order to limit the impact of the latency. Breaking workflow parallelism to reduce the number of submitted jobs may also be envisaged.

Acknowledgments

This work is partially funded by the AGIR project (http://www.aci-agir.org/) from the French research program "ACI-Masse de données" and the GWENDIA project (http://gwendia.polytech.unice.fr, contract ANR-06-MDCA-009) from the French National Agency for scientific Research (ANR). We are grateful to the EGEE European project for providing the grid infrastructure and user assistance.

References

[1] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, A. Savva, Job Submission Description Language (JSDL) Specification, Version 1.0, Tech. rep., GGF (nov 2005).
URL http://www.gridforum.ord/documents/GFD.56.pdf

[2] Portable Batch System (PBS), http://www.openpbs.org/.

[3] C. Albing, Cray NQS: Production batch for a distributed computing world, in: 11th Sun User Group Conference and Exhibition, Brookline, MA, USA, 1993, pp. 302–309.

[4] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Marti, A batch scheduler with high level components, in: International Symposium on Cluster Computing and the Grid (CCGrid'05), Vol. 2, 2005, pp. 776– 783.

[5] I. Foster, C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications 11 (2) (1997) 115–128.

[6] D. Thain, T. Tannenbaum, M. Livny, Distributed computing in practice: the Condor experience., Concurrency and Computation: Practice & Experience 17 (2–4) (2005) 323–356.

[7] The gLite middleware, http://glite.web.cern.ch/glite/.

[8] I. Foster, C. Kesselman, J. Nick, S. Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Open Grid Service Infrastructure WG, Global Grid Forum (Jun. 2002).

[9] W. World Wide Web Consortium, Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl (mar 2001).

[10] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, H. Casanova, A GridRPC Model and API for End-User Applications, Tech. rep., Global Grid Forum (GGF) (jul 2005).
URL http://www.gridforum.org/documents/GFD.52.pdf

[11] E. Caron, F. Desprez, DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid, International Journal of High Performance Computing Applications 20 (3) (2006) 335–352.

[12] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, S. Matsuoka, Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing, Journal of Grid Computing (JGC) 1 (1) (2003) 41–51.

[13] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, S. Vadhiyar, Users' Guide to NetSolve V1.4.1, Tech. Rep. ICL-UT-02-05, University of Tennessee, Knoxville (jun 2002).

[14] I. Foster, Globus Toolkit Version 4: Software for Service-Oriented Systems, in: International Conference on Network and Parallel Computing (IFIP), Springer-Verlag LNCS 3779, 2006, pp. 2–13.

[15] J. Yu, R. Buyya, A taxonomy of scientific workflow systems for grid computing, ACM SIGMOD Record 34 (3) (2005) 44–49.

[16] M. Livny, Direct Acyclic Graph Manager (DAGMan), http://www.cs.wisc.edu/condor/dagman/.

[17] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, S. Koranda, Mapping Abstract Complex Workflows onto Grid Environments, Journal of Grid Computing (JGC) 1 (1) (2003) 9–23.

[18] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, K. Kennedy, Task Scheduling Strategies for Workflow-based Applications in Grids, in: International Symposium on Cluster Computing and the Grid (CCGrid'05), Cardiff, UK, 2005, pp. 759–767.

[19] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, Y. Zhao, Scientific Workflow Management and the Kepler System, Concurrency and Computation: Practice & Experience 18 (10) (2006) 1039–1065.

[20] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, P. Li, Taverna: A tool for the composition and enactment of bioinformatics workflows, Bioinformatics journal 17 (20) (2004) 3045–3054.

[21] I. Taylor, I. Wand, M. Shields, S. Majithia, Distributed computing with Triana on the Grid, Concurrency and Computation: Practice & Experience 17 (9) (2005) 1197–1214.

[22] T. Glatard, J. Montagnat, X. Pennec, An optimized workflow enactor for data-intensive grid applications, Tech. Rep. I3S/RR-2005-32-, I3S, Sophia-Antipolis (oct 2005).

[23] J. Montagnat, T. Glatard, D. Lingrand, Data composition patterns in service-based workflows, in: Workshop on Workflows in Support of Large-Scale Science (WORKS'06), Paris, France, 2006.

[24] Y. Huang, I. Taylor, D. M. Walker, R. Davies, Wrapping Legacy Codes for Grid-Based Applications., in: 17th International Parallel and Distributed Processing Symposium (IPDPS), Washington, DC, USA, IEEE Computer Society, 2003, p. 139.

[25] N. Furmento, A. Mayer, S. McGough, S. Newhouse, T. Field, J. Darlington, ICENI : Optimisation of component applications within a Grid environment, Journal of Parallel Computing 28 (12) (2002) 1753–1772.

[26] J. Li, Z. Zhang, H. Yang, A Grid Oriented Approach to Reusing Legacy Code in ICENI Framework, in: IEEE International Conference on Information Reuse and Integration (IRI'05), Las Vegas, Nevada, USA, 2005, pp. 464– 469.

[27] A. Harrison, I. Taylor, Dynamic Web Service Deployment Using WSPeer, in: Proceedings of 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies, 2005, pp. 11–16.

[28] M. Senger, P. Rice, T. Oinn, Soaplab - a unified Sesame door to analysis tool, in: UK e-Science All Hands Meeting, Nottingham, 2003, pp. 509–513.

[29] T. Delaitre, T. Kiss, A. Goyeneche, G. Terstyanszky, S. Winter, P. Kacsuk, GEMLCA: Running Legacy Code Applications as Grid Services, Journal of Grid Computing (JGC) 3 (1-2) (2005) 75–90.

[30] P. Kacsuk, G. Dzsa, J. Kovcs, R. Lovas, N. Podhorszki, Z. Balaton, G. Gombs, P-GRADE: A Grid Programing Environment, Journal of Grid Computing (JGC) 1 (2) (2003) 171–197.

[31] G. Kecskemeti, Y. Zetuny, T. Kiss, G. Sipos, P. Kacsuk, G. Terstyanszky, S. Winter, Automatic Deployment and Interoperability of Grid Services, in: UK e-Science All Hands Meeting, Nottingham, UK, 2005, pp. 729–736.

[32] T. Glatard, J. Montagnat, X. Pennec, Efficient services composition for grid-enabled data-intensive applications, in: IEEE International Symposium on High Performance Distributed Computing (HPDC'06), Paris, France, 2006, pp. 333–334.

[33] T. Glatard, X. Pennec, J. Montagnat, Performance evaluation of grid-enabled registration algorithms using bronze-standards, in: Medical Image Computing and Computer-Assisted Intervention (MICCAI'06), LNCS, Copenhagen, Denmark, 2006, pp. 152–160.

[34] S. Nicolau, X. Pennec, L. Soler, N. Ayache, Evaluation of a New 3D/2D Registration Criterion for Liver Radio-Frequencies Guided by Augmented Reality, in: International Symposium on Surgery Simulation and Soft Tissue Modeling (IS4TM'03), Vol. 2673 of LNCS, INRIA Sophia Antipolis, Springer-Verlag, Juan-les-Pins, 2003, pp. 270–283.

[35] T. Glatard, J. Montagnat, X. Pennec, Optimizing jobs timeouts on clusters and production grids, in: International Symposium on Cluster Computing and the Grid (CCGrid'07), IEEE, Rio de Janeiro, 2007, pp. 100–107.

[36] T. Glatard, J. Montagnat, X. Pennec, Probabilistic and dynamic optimization of job partitioning on a grid infrastructure, in: 14th euromicro conference on Parallel, Distributed and network-based Processing (PDP06), Montbéliard-Sochaux, 2006, pp. 231–238.