Software composition for scientific workflows

Johan Montagnat
CNRS / UNS, I3S, MODALIS
April 17, 2009

GWENDIA ANR-06-MDCA-009
http://gwendia.polytech.unice.fr
Grid infrastructures

• Scientific production
  – Europe: EGEE (www.eu-egee.org), USA: OSG (www.opensciencegrid.org), NAREGI (www.naregi.org)...

EGEE European infrastructure:
  > 250 computing centers in 45 countries
  > 80 000 CPU cores
  > 13 000 users
  > 250 000 jobs / day

• behavior
  – Very large scale (WAN)
  – Coarse grain parallelism, loose coupling
  – High heterogeneity, low reliability, large latencies
Images analysis pipelines

Preprocessing:
- Non-uniformity correction
- Intensity normalisation

Registration and resampling:
- Stereotaxic registration
- Resampling

Segmentation:
- Brain segmentation
- Classification

Analysis:
- Quantification
- Statistical tests

Multimodal MRI

4 classes:
- White matter
- Gray matter
- CSF
- Lesions
Images analysis pipelines

Clinical experiment: inferon beta drugs trials for multiple sclerosis
Population: 200 subjects (patients and normal control)
Images: 2400 longitudinal acquisition (1.3 TB)
Processing pipeline: 0h45 to 2h30 per image
Complete experiment: 7 days using 30 CPUs

Multimodal MRI

Segmentation

Classification

Analysis

Quantification

Statistical tests

4 classes:
- White matter
- Gray matter
- CSF
- lesions
Images analysis pipelines

**Clinical experiment:** inferon beta drugs trials for multiple sclerosis
**Population:** 200 subjects (patients and normal control)
**Images:** 2400 longitudinal acquisition (1.3 TB)
**Processing pipeline:** 0h45 to 2h30 per image
**Complete experiment:** 7 days using 30 CPUs

**Other compute intensive health applications:**
- Clinical and statistical studies
- Epidemiology
- Anatomical and physiological models design
- Validating medical image analysis procedures
- Image-based medical treatment validation
  ...

**Multimodal MRI**

**Analysis**

**Statistical tests**

**Images**

**Non-uniformity correction**

**Intensity normalisation**

**Preprocessing**

**Registration and resampling**

**Stereotaxic registration**

**Resampling**

**Segmentation**

**Classification**

**4 classes:**
- White matter
- Gray matter
- CSF
- Lesions

**MR Images**

(T1, T2, PD)

GWENDIA ANR-06-MDCA-009

Software composition day

Johan Montagnat

April 17, 2009
Scientific application workflows

- Application logic description
  - Coarse grain, data defined independently

Filtering, initialization
Image processing
Quantification
Visualization, Decision taking
• **Application community**
  – Compute and data intensive applications
  – Non-expert end users
  – Distributed (medical centers)

• **Coarse grain parallelism**
  – Grid computing
  – Massive data parallelism

• **Platform independence**
  – Common representation / submission interface to
    - Different grids
    - Multiple grids

• **Pipelines are pure data flows**
  – Successive image processing filters
  – Data intensive and *data driven*
Dynamic data flow composition

- Data flow resolved dynamically
- Services invoked multiple times

Diagram:

- Data flow resolved dynamically
- Services invoked multiple times

GWENDIA ANR-06-MDCA-009
• **Cardiac sequences example**

The input data set is a 4D image composed of 2 volumes (labelled green and red). Each volume is composed of 4 slices.

- **Image reader**
- **Patient identifier**
- **4D image represented as a set of slices**
- **Individual slices processing**
- **Slice stacks recomposition**
- **Individual volumes processing**

**Image Crop**

**Interpolation**

**Gradient computing**

**Motion estimation**
Data flows composition

Grid Workflow Efficient Enactment for Data Intensive Applications

\begin{itemize}
\item \textit{cross iterator: all-to-all}
\item \textit{dot iterator: one-to-one}
\end{itemize}

\begin{itemize}
\item \{A\_1, A\_2, A\_3\} \{B\_1, B\_2, B\_3\}
\item \{A\_1, A\_2, A\_3\} \{B\_1, B\_2, B\_3\}
\end{itemize}
Scufl data flow definition

• **Graph of services (+ data)**

![Graph of services](image)

- **input0**
- **Service0**
- **Service1**
- **Service2**
- **Service3**

**DAG of tasks**

- **Job0**
- **Job1**
- **Job2**
- **Job3**
• Graph of services (+ data)  

**Scufl data flow definition**

Grid Workflow Efficient Enactment for Data Intensive Applications

- **Graph of services (+ data)**
  - Service0
  - Service1
  - Service2
  - Service3

- **DAG of tasks**
  - Job0
  - Job1
  - Job2
  - Job3

4 data segments
• Graph of services (+ data)  

DAG of tasks

4 data segments
This data representation allows to:
- Handle dot products iteration strategies if data segments are puzzled
- Retrieve results provenance

Data graph
MOTEUR workflow manager

Grid Workflow Efficient Enactment for Data Intensive Applications

- **Open source workflow enactor**
  - Code + docs + tutorial: [http://egee1.unice.fr/MOTEUR](http://egee1.unice.fr/MOTEUR)
  - Developed at the I3S CNRS laboratory
  - With the support of French national projects
    - AGIR: [http://www.agir.org](http://www.agir.org)
    - GWENDIA: [http://gwendia.polytech.unice.fr](http://gwendia.polytech.unice.fr)

- **Targets**
  - Ease of use, flexibility, service-oriented approach
  - Performance, transparent exploitation of application parallelism

- **Supports**
  - Scufl language (from myGrid/Taverna)
  - Service based invocation (WS)
  - Grid middlewares (EGEE / Grid'5000)
• **Batch-oriented infrastructure:** no deployment
  - EGEE / gLite middleware
  - Grid'5000 / OAR, using idle resources

• **Service oriented infrastructure:** pre-deployment
  - Web Services
  - DIET Services
- Provide service wrapper to non instrumented code
- Handle data transfers (references to grid data)

```xml
<description>
  <executable name="CrestLines.pl">
    <access type="URL">
      <path value="http://colors.unice.fr:80/"/>
    </access>
    <value value="CrestLines.pl"/>
  </executable>
  <input name="image" option="-im1">
    <access type="LFN"/>
  </input>
  <input name="scale" option="-s"/>
  <output name="crest_lines" option="-c2">
    <access type="LFN"/>
  </output>
  <sandbox name="convert8bits">
    <access type="URL">
      <path value="http://colors.unice.fr:80/"/>
    </access>
    <value value="Convert8bits.pl"/>
  </sandbox>
</description>
```
- **Generic Application Service Wrapper**
  - Provide service wrapper to non instrumented code
  - Handle data transfer (references to grid data)

- **Execution scheme:**

  ![Diagram](image)

  - User Web Server
  - **WSDL Contract**
  - **Code descriptor**
  - **Required file**
  - **Storage Resource**
  - **Executable**
  - **Input file**
  - **Grid submission service**
  - **Computing resource**

---

GWENDIA ANR-06-MDCA-009
- **Generic Application Service Wrapper**
  - Provide service wrapper to non instrumented code
  - Handle data transfer (references to grid data)

- **Execution scheme:**

  - **GASW host**
    - Code descriptor
    - wget library
    - grid-get executable
    - grid-get input file
    - command-line
    - grid-put output file

  - **User Web Server**
    - WSDL Contract
    - Code descriptor
    - Required library

  - **HTTP**

  - **Grid submission service**

  - **Storage Resource**
    - Executable
    - Input file

  - **Computing resource**
• **Generic Application Service Wrapper**
  - Provide service wrapper to non instrumented code
  - Handle data transfer (references to grid data)

• **Execution scheme:**

![Diagram showing the components and flow of the execution scheme.

**User Web Server**
- WSDL Contract
- Code descriptor
- Required library

**GASW host**

**Grid submission service**

**Grid Job**
- Executable
- Input file
- grid-get executable
- grid-get input file
- grid-put output file
- wget library

**Storage Resource**
• **Generic Application Service Wrapper**
  – Provide service wrapper to non instrumented code
  – Handle data transfer (references to grid data)

• **Execution scheme:**
Dynamic wrapping

- **Generic Application Service Wrapper**
  - Provide service wrapper to non instrumented code
  - Handle data transfer (references to grid data)

- **Execution scheme:**

  ![Diagram](image)

  - User Web Server
    - WSDL Contract
    - Code descriptor
    - Required library
  - Storage Resource
    - Executable
    - Input file
  - Grid submission service
  - Grid Job
    - Required file
    - Executable
    - Input file
    - Output file
    - command-line
• **Generic Application Service Wrapper**
  - Provide service wrapper to non instrumented code
  - Handle data transfer (references to grid data)

• **Execution scheme:**

![Diagram showing components of grid workflow efficient enactment for data intensive applications]

- **User Web Server**
  - **WSDL Contract**
  - **Code descriptor**
  - **Required library**

- **Storage Resource**
  - **Executable**
  - **Input file**
  - **Output file**

- **GASW host**

- **Grid submission service**

- **Computing resource**
• A workflow naturally provides application parallelization
• 3 kinds of parallelism exploited

**Workflow parallelism**

- \( D_0', D_1 \)
- \( S_1 \)
- \( S_2, S_3 \)

**Data parallelism**

- \( D_0', D_1 \)
- \( D_3, D_6 \)
- \( S_2, S_3 \)

**Service parallelism**

- \( D_0', D_1 \)
- \( D_1 \)
- \( S_2, D_0 \)

• Data sets are composed dynamically
• **GASW service is recognized by MOTEUR**
  – Not a real blackbox

• **Submission can be directly made to the grid**
  – Grid mode: Web Service call bypassed
  – No difference from a user point of view (no change in Scufl)
Optimizing the execution through dynamic code wrapping

- **High latencies strongly penalizes job execution**
  - Latency ≈ 5 minutes +/- 5 minutes on the EGEE production grid

- **High variability requires to reduce the number of jobs**
  - The higher the number of submitted jobs, the higher the probability to get outliers

- **Services grouping reduces the number of job submissions**
• Goal: find a grouping rule that does not break parallelism
• Let A be a service of the workflow and \{B_0, \ldots, B_n\} its children
• For grouping A and B_{i0}: no parallelism loss \iff
  – (1) B_{i0} is an ancestor of every B_j
  – (2) Every ancestor of B_{i0} is an ancestor of A (or A itself)
• No parallelism loss =>
  – \neg(1) => parallelism between B_j and B_{i0} is broken
  – \neg(2) => parallelism between A and C is broken
• (1) & (2) => no parallelism loss
• This rule is recursively applied on the workflow graph
Job Grouping Experiments

- 6 services – 2 groups
- 4 job submissions/input data set submission/input data set

- 4 services - 3 groups
- Recursive application
Performance results

- On the EGEE infrastructure

![Graph showing performance results for different workload enablers: JG = Job Grouping, DP = Data Parallelism, SP = Service Parallelism (pipelining). The graph illustrates the execution time (in hours) for various numbers of images processed, highlighting the benefits of combining different enablers.]
• Two basic registration orchestrations to be merged
• Merging services:
• Merging variables: =

[Diagram of workflows transformation with various nodes and arrows indicating merging and non-merging of variables.]
• Merging variables (II):
Workflows transformation

Merging outputs:

A fully automatic merging procedure is not suitable
Graph representation

OMSM representation in Prolog

[C. Nemo et al, SCC'07]

%Orchestration declaration
orch(pfm,[im0,im1,fN,method,transfo,cl,pm,pr],
[i1,i2,i3,i4,...]).

% Instructions
instr(i1,[im0,im1,cl],clOut,invoke(cl)).
instr(i2,[clOut],c1,assign(pfmC1)).
instr(i3,[clOut],c2,assign(pfmC2)).
instr(i4,
[pm,pfmC1,pfmC2,transfo],pfmOut,invoke(pfm)).
...

% Constraints
pred(i1,i2).
pred(i1, i3).
pred(i2, i4).
pred(i3, i4).
...
• **Consistency properties:**
  – P1: orchestrations have at most one invocation to a given basic service
    ▪ Multiple invocations are wrapped in a complex invocation
  – P2: orchestrations have at most one input and one output
    ▪ Multiple input / outputs are encapsulated in a structure
  – P3: orchestrations satisfy a set of constraints:
    ▪ No concurrent write access to the variables
    ▪ No cycle in instructions invocation order
• **Potential unification points are detected because they break either P1 or P2**
• **If an orchestration breaks P3, it is rejected from the process**
Grid Workflow Efficient Enactment for Data Intensive Applications

Conflicts detected

1. ref
2. $s$
3. float
4. file
5. method

- CL
- CM
- Convert
- Write
- Eval

Conflicting
Grid Workflow Efficient Enactment for Data Intensive Applications

Inputs / outputs merging

Grid Workflow Efficient Enactment: for Data Intensive Applications

CL
CM
Convert
Write
Eval

Conflictual

ref
float

file

method 1

pm pr
method 2

Convert
Write
Eval

GWENDIA ANR-06-MDCA-009
Software composition day
Johan Montagnat
April 17, 2009
Conclusions

• Scientific workflows
  – Implicit parallelism
  – Complex data flows

• Software composition
  – Code reuse
  – Heterogeneous codes assembling

• Preoccupations
  – Performance
  – Ease of use
  – Separate application logic and data sets
  – Compact representation of complex processes