dispel4py: A Python Framework for Data-Intensive eScience

PyHPC2015
15 November 2015
Outline

Introduction
dispel4py features
dispel4py basic concepts
dispel4py advanced concepts
dispel4py workflows
Evaluations
Current work
Conclusions and future work
What it is dispel4py?

User-friendly tool

Develop scientific methods and applications on local machines

Run them at scale on a wide range of computing resources without making changes
What it is dispel4py?

Open source project: [www.dispel4py.org](http://www.dispel4py.org) & [https://github.com/dispel4py/dispel4py](https://github.com/dispel4py/dispel4py)

Publications:

- 11th IEEE eScience Conference, 2015
- Book Chapter in “Conquering Big Data Using High Performance”, 2015

Users:

- Computational Seismologists
- Astrophysicists
- BioInformatics

Contributors:

- University of Edinburgh
- KNMI
- LMU
- University of Malaysia
dispel4py features

Stream-based

- Tasks are connected by streams
- Multiple streams in & out
- Optimisation based on avoiding IO

Python for describing tasks and connections

Modular

Multiple enactment systems
Processing element

Es represent the basic computational unit.
Data transformation, scientific method, service request.
Es are the “Lego bricks” of tasks and users can assemble them into a workflow as they wish.

General PE features

**Consumes** any number and types of input streams
**Produce** any number and types of output streams
Graph

- Topology of the workflow: connections between PEs
- Users focus on the algorithm to implement or the service to use
PE Instance

- Executable copy of a PE that runs in a process.
- Each PE is translated into one or more instances in run-time.
Groupings

Grouping by” a feature (MapReduce)

- All data items that satisfy the same features are guaranteed to be delivered to the same instance of a PE

dispel4py basic concepts – Groupings
Groupings

-One-To-All

**P3 - grouping “all”:**
P2 instances send copies of their output data to all the connected instances

-Global

**P3 - grouping “global”:**
All the instances of P2 send all the data to one instance of P3
Composite PE

- Sub-workflow in a PE
- Hides the complexity of an underlying process
- Treated like any other PE
Composite PE and partition

Partition

- PEs wrapped together
- Run several PEs in a single process
Example of a dispel4py workflow

```
dispel4py.workflow_graph import WorkflowGraph

FilterTweet()
CounterHashTag()
CounterLanguage()
Statistics()

graph = WorkflowGraph()

connect(pe1,'hash_tag',pe2,'input')
connect(pe1,'language',pe3,'input')
connect(pe2,'hash_tag_count',pe4,'input1')
connect(pe3,'language_count',pe4,'input2')
```

Users only have to implement:
- PEs
- Connections
Example of a PE

```python
filterTweet(GenericPE):
    def __init__(self):
        GenericPE.init(self)
        self.add_output('hash_tags')
        self.add_output('language')

    def process(self, inputs):
        twitterData = inputs['input']
        for line in twitterData:
            tweet = json.loads(line)
            language = tweet['lang'].encode('utf-8')
            text = tweet['text'].encode('utf-8')
            hashtags = re.findall(r'\w+(?:(?:\w+)?,)*', text)

            self.write('hash_tags', hashtags)
            self.write('language', language)
```

Users only have to implement:
- PEs
- Connections
- Logic of PE
- Stream out data
Mappings

**Sequential**
- Sequential mapping for local testing
- Ideal for local resources: Laptops and Desktops

**Multiprocessing**
- Python’s multiprocessing library
- Ideal for shared memory resources

**MPI**
- Distributed Memory, message-passing parallel programming model
- Ideal for HPC clusters

**STORM**
- Distributed Real-Time computation System
- Fault-tolerant and scalable
- Runs all the time

**SPARK (Prototype)**
Users can select which metadata to store. Searches over products metadata within and across runs. Data download and preview. Capturing of Errors for Diagnostic purposes. Data Fabric: Multi directional navigations across data dependencies.
The VERCE project provides a framework to the seismological community to exploit the increasingly large volume of seismological data:

- Support data-intensive and HPC applications
- e-Science Gateway for submitting applications
- Distributed and diversified data sources
- Distributed HPC resources on Grid, Cloud and HPC clusters

Use cases – dispel4py:
- Seismic Noise Cross-Correlation
- Misfit calculation
Seismology, Cross Correlation

- Data intensive problem and it is commonly used in seismology.

- Phase 1- Preprocess: Time series data (traces) from seismic stations are preprocessed in parallel.

- Phase 2: Cross-Correlation: Pairs all of the stations and calculates the cross-correlation for each pair (complexity $O(n^2)$).

Input data:
1000 stations as input data (150MB)

Output data:
499,500 cross-correlations (39GB)
Seismology, Misfit Computation

Phase 1 – Preprocess: Align and prepare traces
Phase 2 – Misfit: Compare synthetic and observed data
dispel4py workflows – Misfit visualisation

time frequency

pyflex windows
## Computing resources

<table>
<thead>
<tr>
<th>Computing resources</th>
<th>Terracorrelator</th>
<th>SuperMUC</th>
<th>Amazon EC2</th>
<th>EDIM1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Shared-memory</td>
<td>Cluster</td>
<td>Cloud</td>
<td>Cloud</td>
</tr>
<tr>
<td>Enactment systems</td>
<td>MPI, multi</td>
<td>MPI, multi</td>
<td>MPI, Storm, multi</td>
<td>MPI, Storm, multi</td>
</tr>
<tr>
<td>Nodes</td>
<td>1</td>
<td>16</td>
<td>18</td>
<td>14</td>
</tr>
<tr>
<td>Cores per Node</td>
<td>32</td>
<td>16</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Total Cores</td>
<td>32</td>
<td>256</td>
<td>36</td>
<td>14</td>
</tr>
<tr>
<td>Memory</td>
<td>2TB</td>
<td>32GB</td>
<td>4GB</td>
<td>3GB</td>
</tr>
<tr>
<td>Workflows</td>
<td>xcorr, int_ext,</td>
<td>xcorr,</td>
<td>xcorr</td>
<td>xcorr, int_ext,</td>
</tr>
</tbody>
</table>
## Performance measures

<table>
<thead>
<tr>
<th>Mode</th>
<th>Terracorrelator (32 cores)</th>
<th>SuperMuc (256 cores)</th>
<th>Amazon (36 cores)</th>
<th>EDIM1 (14 cores, 4 shared)</th>
</tr>
</thead>
<tbody>
<tr>
<td>multi</td>
<td>1332.20 (~23 minutes)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Storm</td>
<td>27898.89 (~8 hours)</td>
<td>120077.123 (~33 hours)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Int_ext:**
1050 galaxies

**xcorr:**
- 1000 stations
- Input 150MB
- Output 39GB
Current work

Diagnosis tool

- How to partition the workflow automatically
- How many processes execute each partition

Run-time Stream Adaptive Compression
dispel4py – Monitoring

6ac525ff-a88b-45df-95c6-d1384bf11311

Processing Element: TestOneInOneOut2
Process: 1
Method: process
Number of calls: 100

Refresh

Graph showing process 1 with a timeline from 0 to 90 and y-axis ranging from 0.000000 to 0.000035.
Conclusions and Future work

Python library for streaming and data-intensive processing

• Users express their computational activities
• Same workflow executed in several parallel systems
• Easy to use and open

Future

• Support for PE failures
• Select the best computing resource and mapping
Installations and Links

- This is all you need:
  
  `pip install dispel4py`


- GitHub: [https://github.com/dispel4py/dispel4py](https://github.com/dispel4py/dispel4py)

- Documentation: [http://dispel4py.org/documentation/](http://dispel4py.org/documentation/)
Contact emails

- Amy Krause: a.krause@epcc.ed.ac.uk
- Rosa Filgueira: rosa.filgueira@ed.ac.uk
- Malcolm Atkinson: Malcolm.Atkinson@ed.ac.uk