LoonyBin: Keeping Language Technologists Sane through Automated Management of (Hyper)Workflows

Jonathan Clark and Alon Lavie
Carnegie-Mellon University
LREC 2010
Thursday, May 20, 2010
Outline

• Empirical NLP Research
  • Day-to-day issues
  • Current problems
  • LoonyBin’s solutions

• Workflows
• HyperWorkflows
Empirical NLP

- Plumbing: Gluing (Linux) tools together
- Recording results
- Sanity checking
- Running variations
- Moving between clusters & schedulers
Empirical NLP

• Plumbing: Gluing (Linux) tools together
• Recording results
• Sanity checking
• Running variations
• Moving between clusters & schedulers
Empirical NLP

- Plumbing: Gluing (Linux) tools together
- Recording results
- Sanity checking
- Running variations
- Moving between clusters & schedulers
Empirical NLP

- Plumbing: Gluing (Linux) tools together
- Recording results
- Sanity checking
- Running variations
- Moving between clusters & schedulers
Empirical NLP

- Plumbing: Gluing (Linux) tools together
- Recording results
- Sanity checking
- Running variations
- Moving between clusters & schedulers
Empirical NLP

- Plumbing: Gluing (Linux) tools together
- Recording results
- Sanity checking
- Running variations
- Moving between clusters & schedulers
Empirical NLP

- Plumbing: Gluing (Linux) tools together
- Recording results
- Sanity checking
- Running variations
- Moving between clusters & schedulers
Empirical NLP

• Plumbing: Gluing (Linux) tools together
• Recording results
• Sanity checking
• Running variations
• Moving between clusters & schedulers
Empirical NLP

- Plumbing: Gluing (Linux) tools together
- Recording results
- Sanity checking
- Running variations
- Moving between clusters & schedulers
Empirical NLP

- Plumbing: Gluing (Linux) tools together
- Recording results
- Sanity checking
- Running variations
- Moving between clusters & schedulers
Empirical NLP

- Plumbing: Gluing (Linux) tools together
- Recording results
- Sanity checking
- Running variations
- Moving between clusters & schedulers
Empirical NLP

- Plumbing: Gluing (Linux) tools together
- Recording results
- Sanity checking
- Running variations
- Moving between clusters & schedulers
Empirical NLP

- Plumbing: Gluing (Linux) tools together
- Recording results
- Sanity checking
- Running variations
- Moving between clusters & schedulers
Empirical NLP

- Plumbing: Gluing (Linux) tools together
- Recording results
- Sanity checking
- Running variations
- Moving between clusters & schedulers
Empirical NLP

- Plumbing: Gluing (Linux) tools together
- Recording results
- Sanity checking
- Running variations
- Moving between clusters & schedulers
Proposed Solution: HyperWorkflow Management
LoonyBin

- Define the tools
  (inputs/outputs/parameters → shell commands)
- Define the workflow
  (DAG of steps and dependencies)
- Generate & run a shell script
LoonyBin

• Define the tools
  (inputs/outputs/parameters → shell commands)

• Define the workflow
  (DAG of steps and dependencies)

• Generate & run a shell script
Drag and Drop

Available Tools
Available Tools

Drag and Drop
Drag and Drop

Available Tools
Available Tools

Drag and Drop

Tooltips for Params
Generating a Script for

**INPUTS**
- foreignCorpus
- nativeCorpus

**PARAMETERS**
- fertility

**OUTPUTS**
- alignments

**Diagram:**
- A → B → W

**Python Tool**
- Descriptor
Generating a Script for

<table>
<thead>
<tr>
<th>INPUTS</th>
<th>OUTPUTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>foreignCorpus</td>
<td>A’s output “x”</td>
</tr>
<tr>
<td>nativeCorpus</td>
<td>B’s output “y”</td>
</tr>
</tbody>
</table>

PARAMETERS

- fertility: 0.01

Parameters & dependencies from workflow

Python Tool Descriptor
Generating a Script for

**INPUTS**

| foreignCorpus | A’s output “x” | …/inputs/f |
| nativeCorpus  | B’s output “y” | …/inputs/n |

**PARAMETERS**

| fertility     | 0.01           |

**OUTPUTS**

| alignments   | …/outputs/wa |

LoonyBin assigns paths

Parameters & dependencies from workflow

Python Tool Descriptor
Generating a Script for foreignCorpus and nativeCorpus

<table>
<thead>
<tr>
<th>INPUTS</th>
<th>OUTPUTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>foreignCorpus</td>
<td>alignments</td>
</tr>
<tr>
<td>nativeCorpus</td>
<td>…/outputs/wa</td>
</tr>
<tr>
<td>A’s output “x”</td>
<td></td>
</tr>
<tr>
<td>B’s output “y”</td>
<td></td>
</tr>
<tr>
<td>…/inputs/f</td>
<td></td>
</tr>
<tr>
<td>…/inputs/n</td>
<td></td>
</tr>
</tbody>
</table>

PARAMETERS

- fertility: 0.01

Python Tool Descriptor

LoonyBin assigns paths

Parameters & dependencies from workflow

java edu.cmu.Tokenizer ../inputs/f

../inputs/n > ../outputs/wa
So far...

• Complaints about current implementation of empirical NLP experiments
• Define the tools (inputs/outputs/parameters)
• Define the workflow (DAG of steps and dependencies)
• Generate & run a shell script
HyperWorkflows

- **HyperWorkflows**: Shared substructure in experiments
- Encode small variations in a HyperDAG
HyperWorkflows

- **HyperWorkflows**: Shared substructure in experiments
- Encode small variations in a HyperDAG
HyperWorkflows

- **HyperWorkflows**: Shared substructure in experiments
- Encode small variations in a HyperDAG
HyperWorkflows

- **HyperWorkflows**: Shared substructure in experiments
- Encode small variations in a HyperDAG
HyperWorkflows

- **HyperWorkflows**: Shared substructure in experiments
- Encode small variations in a HyperDAG

Don’t re-run

Packing Node

Realizations

Moses Phrase Table Training

Minimum Error Rate Training

Decode Sentences

{syntax-st, syntax-ch, moses}
• **HyperWorkflows**: Shared structure in experiments

• Encode small variations in a HyperDAG

Don’t re-run

Organized directory structure & easy-to-parse logs

Realizations

HyperWorkflows

Filter

Corpus

Target Language Corpus

Build Language Model

Charniak Parser

Build Syntactic Translation Model

{st,ch}

Minimum Error Rate Training

{syntax-st, syntax-ch, moses}

Decode Sentences

Moses Phrase Table Training

Decide Sentences

{syntax-st, syntax-ch, moses}

Stanford Parser

Packing Node

Build Language Model

st

ch
Multiple Machines and Schedulers

Design Machine

Java
Multiple Machines and Schedulers

Design Machine

Java

Home Execution Machine

UNIX

Manually Copy Bash Script
Multiple Machines and Schedulers

- Design Machine
  - Java
  - Manually Copy Bash Script

- Home Execution Machine
  - UNIX

- Remote Execution Machine
  - UNIX
  - Passwordless SSH

- Remote Execution Machine
  - UNIX
  - Passwordless SSH

- Remote Execution Machine
Multiple Machines and Schedulers

- Design Machine
  - Manually Copy Bash Script
- Home Execution Machine
- Condor
- Remote Execution Machine
  - Passwordless SSH
- Remote Execution Machine
  - Passwordless SSH
- Remote Execution Machine
  - Bash
- Sun Grid Engine
- Java
  - UNIX
  - UNIX
  - UNIX
Other Things to Make Life Easier

- Sanity checking at each step (embedded in Tool Descriptors)
- Copying of files (including to HDFS)
- Text-based workflow definition (in SVN)
- Open-source LGPL License
WANTED

- Users & Contributors
- Machine Translation Toolpack (released)
- Corpus Processing Toolpack?
- Parsing Toolpack?
- Question Answering Toolpack?
- Resource Directory Toolpack?
- Speech Recognition Toolpack?
Conclusion

- Make your life easier
- Automation
- Sanity Checking
- Logging
- Make your colleagues’ lives easier
- Reproducibility
- Modularity
Questions?

Tutorial & Software at

http://loonybin.sourceforge.net
Home Machine

Base Directory

100-local

100-local-myRealization.loon
100-local-otherRealization.loon
200-remote-myRealization.loon
...

100-local

myRealization
otherRealization
...

200-remote

myRealization

working
final

Remote Machine 1

Target Working Directory

200-second

Lists files even on remote machines

myRealization

...
Practical Issues

• How does LoonyBin know when to run/rerun a step?
  • Each vertex x realization has a loon log file. If the file does not exist, the step is (re)run

• What if I don’t want that many steps?
  • Workflows have many granularities!
Recommendations

• Store your workflow files in SVN

• Store your log files in SVN -- experimental data is useful long after we get annoyed with size of data files!

• Log the SVN revision of frequently changing tools in your Loon logs -- Build them from SVN every time to ensure you’re executing that version
Future Work

• Default parameters -- Short-term

• Asynchronous DAG execution (currently all steps are run in serial) -- Mid-Term

• Workflow monitoring and reprioritization during execution -- Long-term

• Encapsulation of workflows as “tools” (hierarchical tools) -- Long-term

• Automatic file compression -- Long-term