Using business workflows to improve control of experiments in distributed systems research

Tomasz Buchert

advised by:
Lucas Nussbaum and Jens Gustedt

Table of Contents

What is a distributed system?

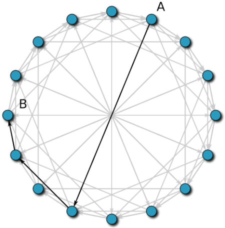
A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

Leslie Lamport

What is a distributed system? (2)

Examples of well-known distributed systems:

- DNS
- BitTorrent
- Gmail
- HPC applications



What is a distributed system? (3)

Distributed systems are:

- complex
- erroneous
- difficult to control
- nondeterministic

Moreover, they:

- span many networks
- span many geographical locations
- consist of thousands of machines

Research in distributed systems

Can we make better BitTorrent, Gmail, MPI, etc.?

Solution: **experimentation** (like in natural sciences)

The "classic way" is difficult, but other solutions exist:

- simulation
- benchmarking
- emulation

	Real application	**Modeled application**
Real platform	Standard approach	Benchmarking
Modeled platform	Emulation / Virtualization	Simulation

Research in distributed systems

Can we make better BitTorrent, Gmail, MPI, etc.?

Solution: **experimentation** (like in natural sciences)

The "classic way" is difficult, but other solutions exist:

- simulation
- benchmarking
- emulation

	Real application	**Modeled application**
Real platform	Standard approach	Benchmarking
Modeled platform	Emulation / Virtualization	Simulation

Experimentation

We all know how frustrating experimenting can be.

That's because experiments in distributed systems are:

- time-consuming
- difficult to do correctly
- complex and incomprehensible
- failure-prone

We have to properly **control** the experiments.

Experimentation tools

Many tools to manage experiments exist:

- Naive way (SSH + Bash + ...)
- Expo
- g5k-campaign
- OMF
- Plush
- ... among many others

They are based on different paradigms.

Our main goal

To improve the research, the experimentation framework has to:

- improve **descriptiveness** of the experiments
- give a **modular** way to build experiments
- handle **unexpected**, but **inevitable** errors
- ensure **scalability** of experiments
- ensure **reproducibility** (or at least **repeatability**)

In the end, we want to

improve experimentation on testbeds like Grid'5000 and PlanetLab.

Bottom-up vs. top-down approach

The majority of tools mentioned use **bottom-up design**.

What about a **top-down** approach?

1. Start with high-level description of the experiment.
2. Implement low-level details.
3. Run the experiment.
4. Improve if necessary and reiterate.

There already exists an approach like this.

Business Process Management

Business Process Management is about:

- **understanding** an organization
- modeling its processes as **workflows**
- **executing** and **monitoring** processes
- **improving** organizational **activities**
- redesigning **processes** to make them:
  - cheaper
  - faster
  - less defective

BPM vs. WfM

Business Process Management:

- **not** a technology
- **nothing** to do with computer science

BPM is a **management discipline**.

Workflow Management:

- a set of technologies supporting BPM
- a computer science discipline

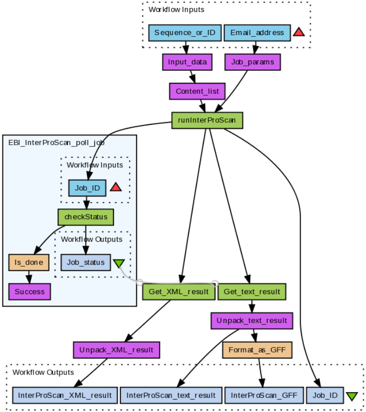
BPM vs. Scientific workflows

Business Process Management:

- workflows describe *control flow*
- arbitrary flow graphs are possible

Scientific workflows:

- *data flows* transforming inputs
- constrained to DAGs
- computing platform is abstracted



The goal of my thesis

Can BPM improve the experimentation with distributed systems?

(and if the answer is "yes", then how?)

Results

My current work is represented by the following contributions:

- analysis of requirements for an experimentation engine
- development of an approach to control of experiments (and its implementation XPFLOW)
- use of the approach to conduct a large-scale, complicated experiments in a reliable and modular way
- design of a framework to evaluate experiment control tools and evaluation of existing ones

Publications

The following papers are about our approach:

- **A workflow-inspired, modular and robust approach to experiments in distributed systems**
  CCGrid 2014, *T. Buchert, L. Nussbaum, J. Gustedt*
- **Orchestration d'expériences à l'aide de processus métier**
  ComPAS 2013, *T. Buchert, L. Nussbaum*
- **Leveraging business workflows in distributed systems research for the orchestration of reproducible and scalable experiments**
  MajecSTIC 2012, *T. Buchert*

The following papers use our approach:

- **Emulation at Very Large Scale with Distem**
  CCGrid Scale Challenge 2014, *T. Buchert, E. Jeanvoine, L. Nussbaum*
- **Scalable and Reliable Data Broadcast with Kascade**
  HPDIC 2014, *S. Martin, T. Buchert, P. Willemet, O. Richard, E. Jeanvoine, L. Nussbaum*

The remaining papers are:

- **A taxonomy of experiment management tools for distributed systems**
  (unpublished yet), *T. Buchert, C. Ruiz, L. Nussbaum, O. Richard*

Publication 1

Leveraging business workflows in distributed systems research for the orchestration of reproducible and scalable experiments

In this article we defined goals and requirements (*desiderata*) for an experiment engine and positioned our approach among existing ones.

We showed that BPM can indeed help.

Publication 1 (cont.)

Design

Descriptiveness
Modularity
Reusability
Maintainability
Support for common patterns

Execution

Snapshotting
Error handling
Integration with lower-level tools
Human interaction

Monitoring

Monitoring
Instrumentation
Data analysis

Publication 2

Orchestration d'expériences à l'aide de processus métier

In this article we validated our approach by:

- describing our early implementation
- presenting our language to describe experiments
- testing the new approach with an MPI experiment

Publication 3

A workflow-inspired, modular and robust approach to experiments in distributed systems

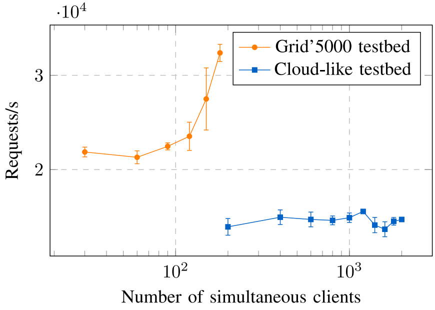
In this article we showed a large-scale, cloud-like experiment managed with XPFLOW, by benchmarking Nginx web server with three different testbeds:

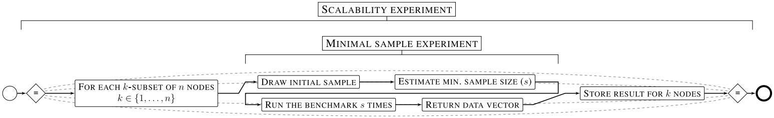
- with a local one built with Linux Containers
- with the Grid'5000 testbed
- with a cloud-like testbed (using KVM) hosted on the Grid'5000 testbed

We showed:

- formal description of workflow execution and error handling
- modularity, by easily switching testbeds and composing experiments
- robustness, by handling failures during the experiment

Publication 3 (cont.)





Publication 4

Scalable and Reliable Data Broadcast with Kascade

The article features a comprehensive performance evaluation of a file distribution method.

Nearly all experiments were run with XPFLOW, showing its usefulness.

Publication 4 (cont.)

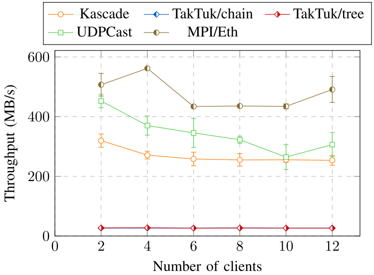


Fig. 8. The performance of the methods using 10 Gbit/s Ethernet network. No method is able to saturate the available bandwidth.

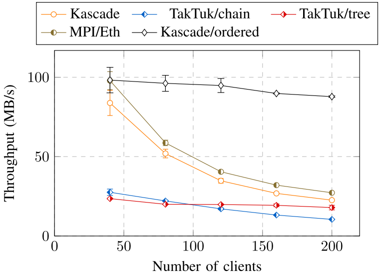


Fig. 10. The performance of the methods with random ordering of nodes. Kascade relies on a proper ordering of nodes and just like other methods shows low performance in this artificial scenario.

Publication 5

Emulation at Very Large Scale with Distem

The article is a submission to Scale challenge that focuses on very large-scale experiments.

Owing to Distem tool and robustness of XPFLOW control, we achieved an experiment involving 40000 nodes.

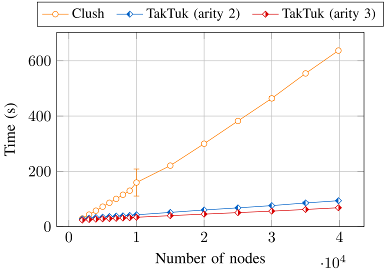


Figure 4. The time required to execute a command using various methods (large number of nodes). Clush is increasingly outperformed by both variations of TakTuk which use a tree overlay to execute commands.

Publication 6

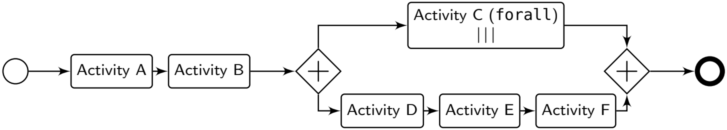
A taxonomy of experiment management tools for distributed systems

The article defines a verbose and useful framework to evaluate experiment control engines like XPFLOW and uses it to evaluate the most prominent ones.

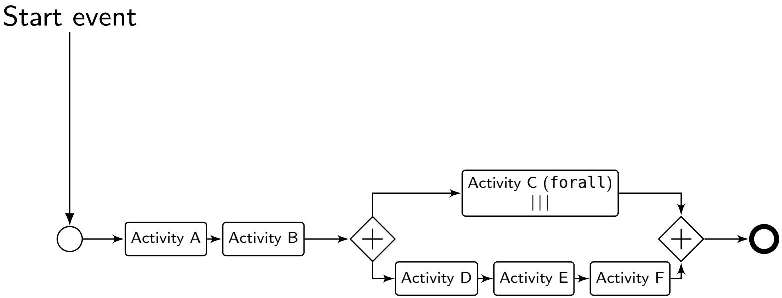
Publication 6 (cont.)

- Description Language
  - Representation — Imperative / Declarative / Workflow / Scripts
  - Modularity — Yes / No
  - Expressiveness — Yes / No
  - Low entry barrier — Yes / No
- Type of Experiments
  - Application type — Real / Model
  - Platform type — Real / Model
  - Intended use — Distributed applications / Wireless / Services / Any
- Interoperability
  - Testbed independence — Yes / No
  - Support for testbed services — Yes / No
  - Resource discovery — Yes / No
  - Software interoperability — Yes / No
- Reproducibility
  - Provenance tracking — Yes / No
  - Fault injection — Yes / No
  - Workload generation — Yes / No
- Fault Tolerance
  - Checkpointing — Yes / No
  - Failure handling — Yes / No
  - Verification of configuration — Yes / No
- Debugging
  - Interactive execution — Yes / No
  - Logging — Yes / No
  - Validation — Yes / No
- Monitoring
  - Experiment monitoring — Yes / No
  - Platform monitoring — Yes / No
  - Instrumentation — Yes / No
- Data Management
  - Deployment — Yes / No
  - File management — Yes / No
  - Analysis of results — Yes / No
- Architecture
  - Control structure — Centralized / Distributed
  - Low resource requirements — Yes / No
  - Simple installation — Yes / No
  - Efficient operations — Yes / No
  - Interface — CLI / GUI / API

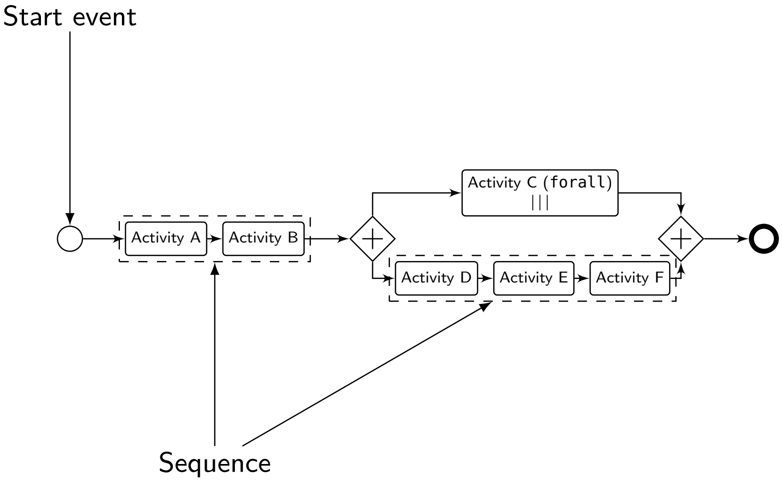
Crash course in workflows



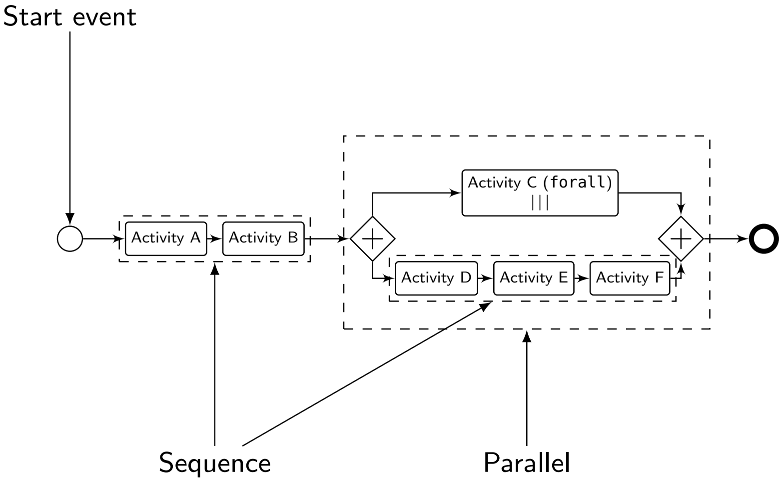
Crash course in workflows



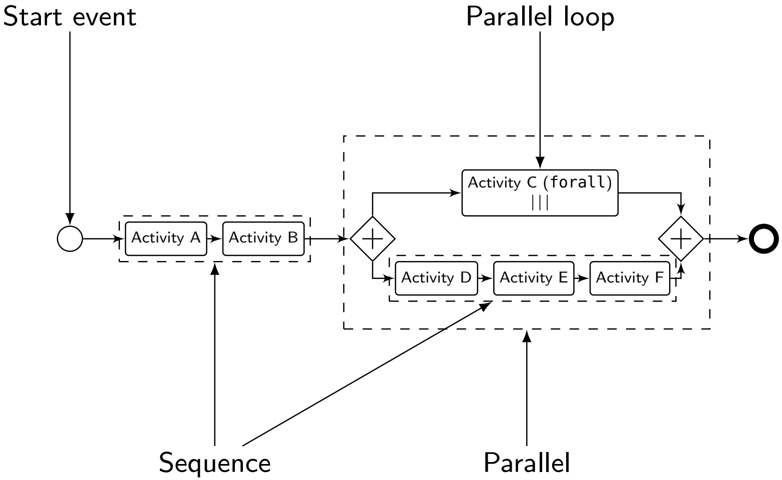
Crash course in workflows



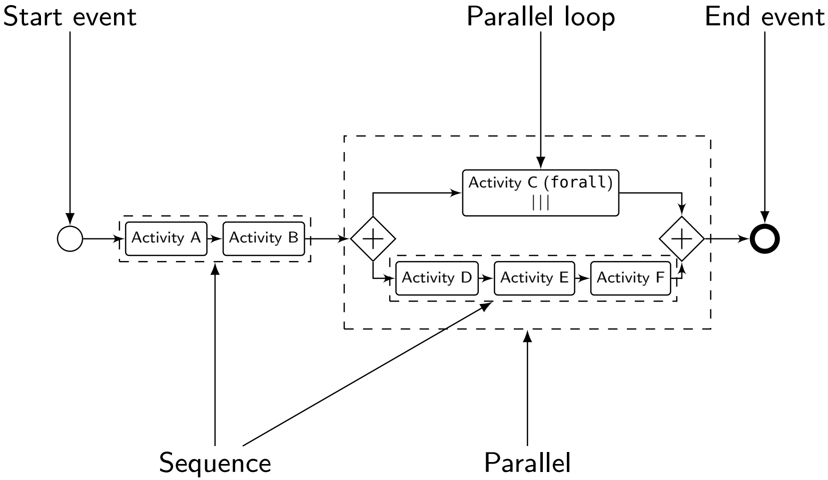
Crash course in workflows



Crash course in workflows



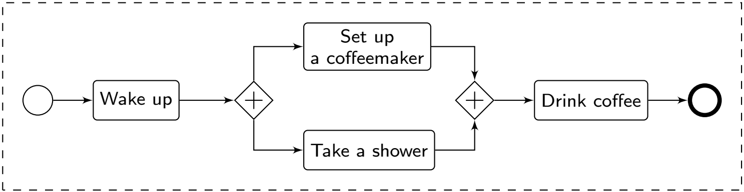
Crash course in workflows



Main concepts

There are 2 main concepts in our approach:

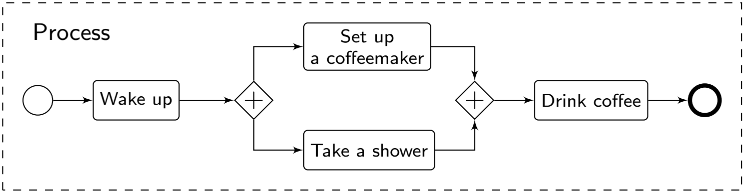
- Processes – high-level description of an experiment:
  - workflows written in a DSL
  - orchestrate other processes and activities
- Activities – low-level building blocks of experiments:
  - do real hard work
  - written in a standard programming language



Main concepts

There are 2 main concepts in our approach:

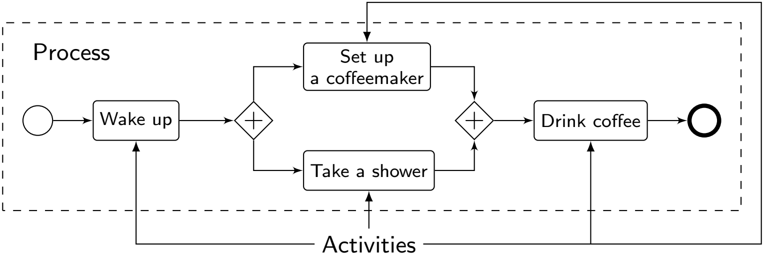
- Processes – high-level description of an experiment:
    - workflows written in a DSL
    - orchestrate other processes and activities
- Activities – low-level building blocks of experiments:
    - do real hard work
    - written in a standard programming language



Main concepts

There are 2 main concepts in our approach:

- Processes – high-level description of an experiment:
  - workflows written in a DSL
  - orchestrate other processes and activities
- Activities – low-level building blocks of experiments:
  - do real hard work
  - written in a standard programming language



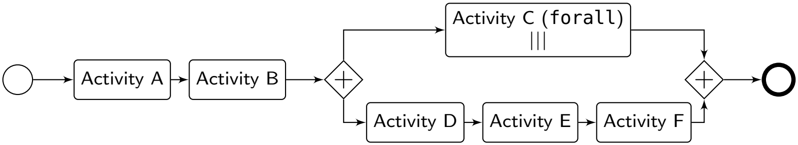
Domain-specific language for processes

The DSL for processes features different **workflow patterns**:

- running activities and other processes (`run`),
- running activities in order or in parallel (`sequence`, `parallel`),
- conditional expressions (`if`, `switch`)
- running sequential and parallel loops (`loop`, `foreach`, `forall`),
- error handling (`try`, `checkpoint`).

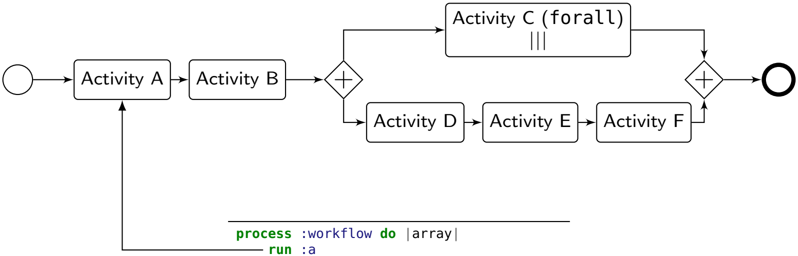
Some of them are taken directly from BPM.

DSL example



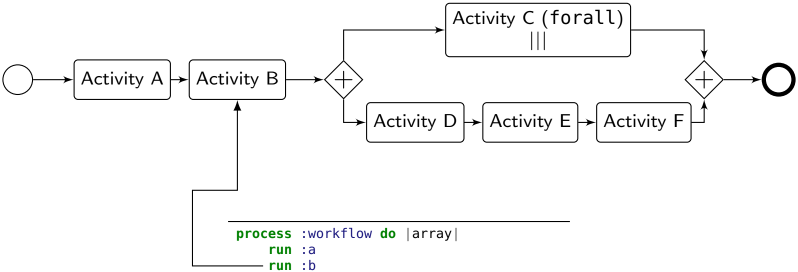

```
process :workflow do |array|
    run :a
    run :b
    parallel do
        forall array do |x|
            run :c, x
        end
        sequence do
            run :d
            run :e
            run :f
        end
    end
end
```

DSL example



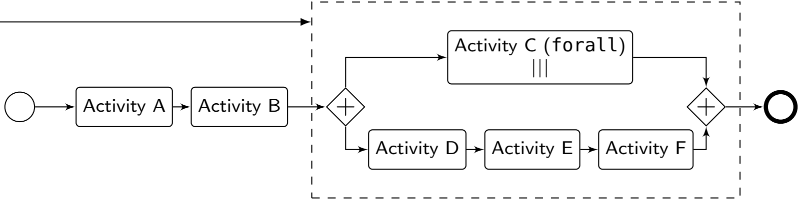

```
process :workflow do |array|
    run :a
    run :b
    parallel do
        forall array do |x|
            run :c, x
        end
        sequence do
            run :d
            run :e
            run :f
        end
    end
end
```

DSL example



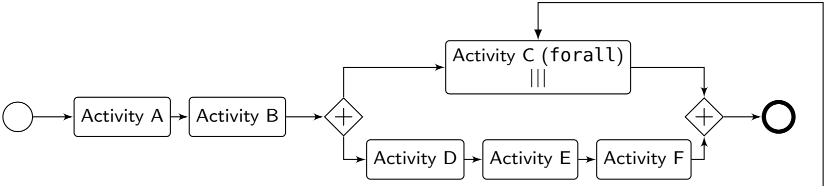

```
process :workflow do |array|
    run :a
    run :b
    parallel do
        forall array do |x|
            run :c, x
        end
        sequence do
            run :d
            run :e
            run :f
        end
    end
end
```

DSL example



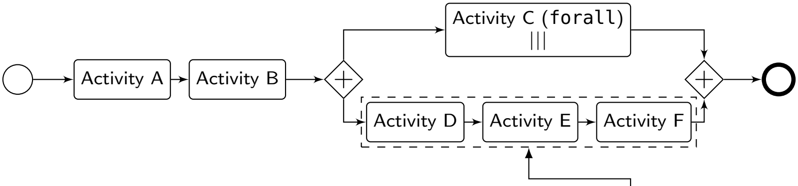

```
process :workflow do |array|
    run :a
    run :b
    parallel do
        forall array do |x|
            run :c, x
        end
        sequence do
            run :d
            run :e
            run :f
        end
    end
end
```

DSL example



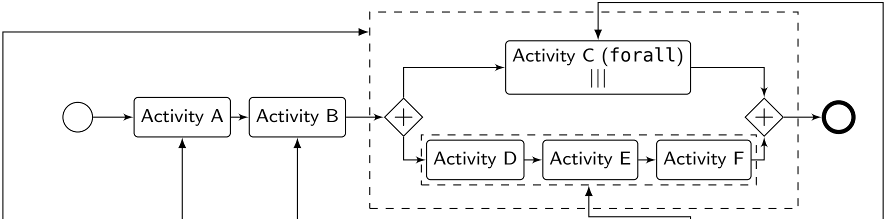

```
process :workflow do |array|
    run :a
    run :b
    parallel do
        forall array do |x|
            run :c, x
        end
        sequence do
            run :d
            run :e
            run :f
        end
    end
end
```

DSL example




```
process :workflow do |array|
    run :a
    run :b
    parallel do
        forall array do |x|
            run :c, x
        end
        sequence do
            run :d
            run :e
            run :f
        end
    end
end
```

DSL example




```
process :workflow do |array|
    run :a
    run :b
    parallel do
        forall array do |x|
            run :c, x
        end
        sequence do
            run :d
            run :e
            run :f
        end
    end
end
```

Error handling

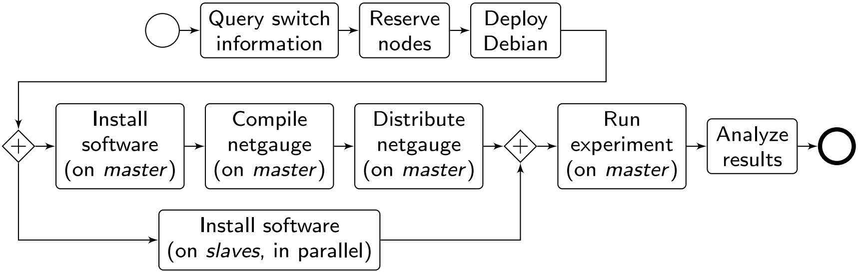
XPflow gives some means to cope with failures:

- snapshotting:
  - saves a state of an experiment for future use
  - shortens a development's cycle
- retry policy:
  - retries a failed subprocess execution
  - manages timeouts of operations
  - improves reliability

```
process :snapshotting do
    run :long_deployment
    checkpoint :d
    run :experiment
end
```

```
process :retrying do
    try :retry => 5 do
        run :tricky_activity
    end
end
```

An experiment (once again)



An experiment workflow - DSL representation

```
process :exp do |site, switch|
    s = run g5k.switch, site, switch
    ns = run g5k.nodes, s
    r = run g5k.reserve_nodes,
        :nodes => ns, :time => '2h',
        :site => site, :type => :deploy
    master = (first_of ns)
    rest = (tail_of ns)
    run g5k.deploy,
        r, :env => 'squeeze-x64-nfs'
    checkpoint :deployed
    parallel :retry => true do
        forall rest do |slave|
            run :install_pkgs, slave
        end
        sequence do
            run :install_pkgs, master
            run :build_netgauge, master
            run :dist_netgauge,
                master, rest
        end
    end
    checkpoint :prepared
    output = run :netgauge, master, ns
    checkpoint :finished
    run :analysis, output, switch
end
```

An experiment workflow - DSL representation

```
process :exp do |site, switch|
    s = run g5k.switch, site, switch
    ns = run g5k.nodes, s
    r = run g5k.reserve_nodes,
        :nodes => ns, :time => '2h',
        :site => site, :type => :deploy
    master = (first_of ns)
    rest = (tail_of ns)
    run g5k.deploy,
        r, :env => 'squeeze-x64-nfs'
    checkpoint :deployed
    parallel :retry => true do
        forall rest do |slave|
            run :install_pkgs, slave
        end
        sequence do
            run :install_pkgs, master
            run :build_netgauge, master
            run :dist_netgauge,
                master, rest
        end
    end
    checkpoint :prepared
    output = run :netgauge, master, ns
    checkpoint :finished
    run :analysis, output, switch
end
```

Activity :install_pkgs

```
activity :install_pkgs do|node|
    log 'Installing packages on ', node
    run 'g5k.bash', node do
        aptget :update
        aptget :upgrade
        aptget :purge, 'mx'
    end
end
```

An experiment workflow - DSL representation

```
process :exp do |site, switch|
    s = run g5k.switch, site, switch
    ns = run g5k.nodes, s
    r = run g5k.reserve_nodes,
        :nodes => ns, :time => '2h',
        :site => site, :type => :deploy
    master = (first_of ns)
    rest = (tail_of ns)
    run g5k.deploy,
        r, :env => 'squeeze-x64-nfs'
    checkpoint :deployed
    parallel :retry => true do
        forall rest do |slave|
            run :install_pkgs, slave
        end
        sequence do
            run :install_pkgs, master
            run :build_netgauge, master
            run :dist_netgauge,
                master, rest
        end
    end
    checkpoint :prepared
    output = run :netgauge, master, ns
    checkpoint :finished
    run :analysis, output, switch
end
```

Activity :build_netgauge

```
activity :build_netgauge do |master|
    log "Building netgauge on #{master}"
    run 'g5k.copy', NETGAUGE, master, '~'
    run 'g5k.bash', master do
        build_tarball NETGAUGE, PATH
    end
    log "Build finished."
end
```

An experiment workflow - DSL representation

```
process :exp do |site, switch|
    s = run g5k.switch, site, switch
    ns = run g5k.nodes, s
    r = run g5k.reserve_nodes,
        :nodes => ns, :time => '2h',
        :site => site, :type => :deploy
    master = (first_of ns)
    rest = (tail_of ns)
    run g5k.deploy,
        r, :env => 'squeeze-x64-nfs'
    checkpoint :deployed
    parallel :retry => true do
        forall rest do |slave|
            run :install_pkgs, slave
        end
        sequence do
            run :install_pkgs, master
            run :build_netgauge, master
            run :dist_netgauge,
                master, rest
        end
    end
    checkpoint :prepared
    output = run :netgauge, master, ns
    checkpoint :finished
    run :analysis, output, switch
end
```

Activity :dist_netgauge

```
activity :dist_netgauge do |m, s|
    master, slaves = m, s
    run 'g5k.dist_keys', master, slaves
    run 'g5k.bash', master do
        distribute BINARY,
            DEST, 'localhost', slaves
    end
end
```

An experiment workflow - DSL representation

```
process :exp do |site, switch|
    s = run g5k.switch, site, switch
    ns = run g5k.nodes, s
    r = run g5k.reserve_nodes,
        :nodes => ns, :time => '2h',
        :site => site, :type => :deploy
    master = (first_of ns)
    rest = (tail_of ns)
    run g5k.deploy,
        r, :env => 'squeeze-x64-nfs'
    checkpoint :deployed
    parallel :retry => true do
        forall rest do |slave|
            run :install_pkgs, slave
        end
        sequence do
            run :install_pkgs, master
            run :build_netgauge, master
            run :dist_netgauge,
                master, rest
        end
    end
    checkpoint :prepared
    output = run :netgauge, master, ns
    checkpoint :finished
    run :analysis, output, switch
end
```

Activity :netgauge

```
activity :netgauge do |master, nodes|
    log "Running experiment..."
    out = run 'g5k.bash', master do
        cd PATH
        mpirun nodes, "./netgauge"
    end
    log "Experiment done."
end
```

Current activities

Among my current activities are:

- validating the approach with large-scale experiments:
  - the advantages of workflow representation
  - ensuring reliability of experiment execution
  - interoperability with external tools and services
- provenance of data in experiment (related to data analysis)
- provenance of experiment description
- early validation of experiment description
- preparing release of XPFLOW

Provenance

Provenance tries to answer many questions:

- how the data were produced?
- when the data were produced?
- which nodes were involved?

Provenance information can be used to:

- verify the experimental results
- reproduce them
- find problems with the platform (e.g., a reason for outliers)
- cache parts of the experiment

Provenance (cont.)

Provenance in *data flows* (e.g., scientific workflows) is (mostly) a solved problem:

- DAG nodes have a well-defined *parent-child* relation
- data flows are (generally) deterministic
- data is a primary concern of workflows

As far as we know, the provenance collection is nearly non-existing in BPM tools and approaches.

My current work is to introduce provenance to our BPM-based approach.

Conclusions

In this talk I presented my work:

- my research topic and the main goals
- 6 related publications
- overview of the implementation (XPFLOW)
- my current activities

Future work:

- extending the idea to provide reproducibility, provenance, design of experiments, easy data analysis and visualization, validation, monitoring, etc.
- work on lower-level services for efficient and scalable experiments
- release of our software (during the next Grid'5000 school)

Thank you for your attention. Questions?