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

Tomasz Buchert, Lucas Nussbaum, Jens Gustedt
Research in distributed systems (1)

Large-scale systems do not submit themselves to formal analysis:
- too much complexity
- only 4% of papers use formal analysis
- researchers turn to experimental science

Proofs may not be possible!
Large-scale experiments are difficult:

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

Up to 26% empirical papers in CS have no evaluation at all!
The standard way to experiment

Traditionally, the experiments are written as a set of scripts.

In this approach:

- the experiment is poorly documented
- the provenance of results is difficult to track
- the same experimental problems are addressed repeatedly
- the scalability problems arise
- the work is difficult to reproduce or extend
- the results are tied to a particular platform

Can we do better than that?
Three axes of experiment control systems

Consider 3 different aspects of experimentation:

- experiment description
- experiment modularity
- execution robustness

All three are crucial for experimentation.
Different ways to provide description exist:

- imperative (Expo) – a standard programming language
- declarative (Plush) – a high-level description
- mixed (OMF, Splay)
Experiment modularity

Existing tools are not very modular or easy to extend, because:

- they focus on one-time studies (Splay)
- they focus on a single platform (Plush)
- they are not designed with that in mind
Execution robustness

Some tools can cope with execution failures automatically:
  - failing nodes (Plush)
  - adherence to platform specification (OMF)
These features tend to be platform-specific.
However, most of the tools require manual failure handling.
We propose a new way to model and structure experiments.

We use a workflow representation based on Business Processes to model experiments:

- they are build from simple, independent blocks
- standard patterns from Business Process Management are used
- workflows are built with a domain-specific language

In the paper, we show:
- advantages of workflow-based description
- modularity and extensibility of our approach
- the robustness of execution

http://www.loria.fr/~buchert/  T. Buchert, L. Nussbaum, J. Gustedt  A workflow-inspired, modular and robust approach to exp. in DS
A workflow approach (cont.)

```plaintext
process :setup_experiment do |clients, server|
  parallel do
    forall clients do |node|
      run :setup_client, node
    end
  end
  sequence do
    run :setup_server, server
    run :start_server, server
    log "Server started"
  end
end
```

http://www.loria.fr/~buchert/

T. Buchert, L. Nussbaum, J. Gustedt
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 flows are possible
- difficult to reproduce

Scientific workflows:
- data flows transforming inputs
- constrained to DAGs
- computing platform is abstracted
- usually reproducible

http://www.loria.fr/~buchert/  T. Buchert, L. Nussbaum, J. Gustedt  A workflow-inspired, modular and robust approach to exp. in DS 13 / 25
Advantages of workflow-based description

The advantages of workflow-based description include:

- power of expressiveness
- automatic analysis
- graphical representation
- modularity
- integrated monitoring
- special workflow patterns ensure robustness
Modularity of the approach

Workflows are modular by design.

We show the modularity of our approach by:

- running our experiment in 3 different testbeds
- building the experiment from reusable, smaller components
Modularity of the approach (2)

The experiment was run on:

- the author’s machine using Linux Containers
- on Grid’5000 testbed
- on cloud testbed deployed on Grid’5000

The results were different for each testbed.
The exemplary experiment consists of:

- an HTTP benchmark
- a *minimal sample experiment* that adaptively retries the scalability experiment to achieve a desired precision
- a *scalability* experiment with varying number of clients

**RUN THE BENCHMARK s TIMES**
The exemplary experiment consists of:

- an HTTP benchmark
- a *minimal sample experiment* that adaptively retries the scalability experiment to achieve a desired precision
- a *scalability* experiment with varying number of clients

![Diagram of minimal sample experiment]

1. **Draw initial sample**
2. **Estimate min. sample size \((s)\)**
3. **Run the benchmark \(s\) times**
4. **Return data vector**
The exemplary experiment consists of:

- an HTTP benchmark
- a *minimal sample experiment* that adaptively retries the scalability experiment to achieve a desired precision
- a *scalability* experiment with varying number of clients
Failures may change the outcome of the experiment.

Consider two executions equivalent if they lead to the same outcome.

How can we ensure that executions are equivalent despite failures?
Workflow properties

To address that we define a few properties of workflows:

- \textit{restartability}
- \textit{idempotency}
- \textit{eventual success}
- \textit{eventual idempotency}

These properties are not composable!

However, workflows with such properties may be composed and succeed despite execution failures.
Failure handling patterns

Two basic handling patterns exist in our approach:

- workflow retry pattern
- checkpoint (of the workflow state, not platform)

Some properties are required:

- retried workflow should be *eventually successful*
- checkpoint is followed by an *eventually idempotent* workflow

With these properties, any workflow can be executed and eventually succeed.
In our exemplary, we measure scalability of the Nginx web server:

- with ApacheBench benchmark
- using Debian OS
- tested on 3 different testbeds
- we measure requests per second (TPS) as number of clients increases
3 experimental testbeds

An author’s machine:
- used to prototype
- very low performance and scale

Grid’5000 testbed:
- uses real machines
- very good performance
- limited scale (199 nodes in our case)

Cloud testbed:
- uses KVM
- limited performance
- scalable, we reached up to 2,033 clients in parallel
Workflow of the experiment

1. **Deploy Grid'5000 nodes**
   - **Install master**
     - Copy files
     - Configure the node

2. **Install slaves**
   - Copy files
   - Copy SSH key
   - Copy KVM image
   - Configure nodes

3. **Start nginx server**
   - Stop all VMs
   - Start server VM

4. **Launch instances**
   - Stop all VMs
   - Start client VMs

http://www.loria.fr/~buchert/
T. Buchert, L. Nussbaum, J. Gustedt
A workflow-inspired, modular and robust approach to exp. in DS
The results of the exemplary experiment

![Graph showing the number of requests per second vs. the number of simultaneous clients. The graph compares two testbeds: Grid’5000 and Cloud-like. The Grid’5000 testbed shows a sharp increase in requests as the number of clients increases, while the Cloud-like testbed remains relatively stable.]
In our work, we have shown that our workflow approach:

- has a formal and malleable description
- enables modular structure of experiments
- provides useful special patterns to handle failures

Future work will include:

- provenance and visualization of results
- distributed execution of experiments
- release of XPFlow (our workflow engine)

**Thank you for your attention. Questions?**

Contact: tomasz.buchert@inria.fr