Orchestration d'expériences à l'aide de processus métier

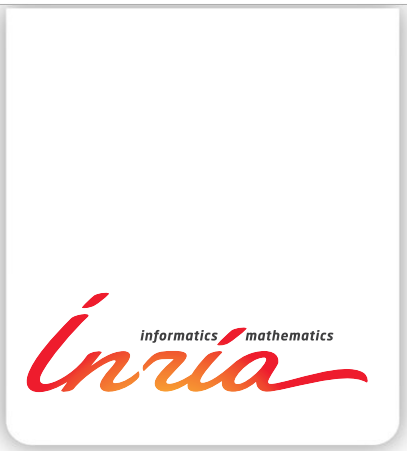
Tomasz Buchert

Orchestration d'expériences à l'aide de processus métier

Tomasz Buchert

RESEARCH REPORT
N° 8129
October 2012
Project-Teams Algorille

ISSN 0249-6399 ISRN INRIA/RR--8129--FR+ENG



Orchestration d'expériences à l'aide de processus métier

Tomasz Buchert

Équipes-Projets Algorille

Rapport de recherche n° 8129 — October 2012 — 11 pages

Résumé : Malgré une activité de recherche sur les systèmes distribués très importante et très active, les expériences dans ce domaine sont souvent difficiles à concevoir, décrire, mener et reproduire. Surmonter ces difficultés pourrait permettre à ce domaine d'être encore plus stimulé, et aux résultats de gagner en crédibilité, à la fois dans le domaine des systèmes distribués. Les facteurs principaux responsables de cette situation sont techniques (bugs logiciels, problèmes matériels), méthodologiques (mauvaises pratiques), et sociaux (réticence à partager son travail). Dans cet article, les approches existantes pour la description et la conduite d'expériences sur les systèmes distribués sont décrites, et une nouvelle approche, utilisant le *Business Process Management (BPM)*, est présentée pour répondre à leurs limitations. Puis diverses questions se posant lors de l'utilisation d'une telle approche sont discutées. Nous montrons que cette approche peut être une meilleure alternative à la manière traditionnelle de conduire des expériences, qui encourage de meilleures pratiques scientifiques, et qui résulte en une recherche et des publications de meilleure qualité. Pour finir, notre plan de travail est décrit, et des applications possibles de ce travail dans d'autres domaines sont décrites.

Mots-clés : expérimentation, validation expérimentale ; démarche scientifique ; workflows scientifiques ; expériences à grande échelle

RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Orchestration of experiments with the help of business workflows

Abstract: While rapid research on distributed systems is observed, experiments in this field are often difficult to design, describe, conduct and reproduce. By overcoming these difficulties the research could be further stimulated and add more credibility to results in distributed systems research. The key factors responsible for this situation are technical (software bugs and hardware errors), methodological (incorrect practices), as well as social (reluctance to share work). In this paper, the existing approaches for the management of experiments on distributed systems are described and a novel approach using business process management (BPM) is presented to address their shortcomings. Then, the questions arising when such approach is taken, are addressed. We show that it can be a better alternative to the traditional way of performing experiments as it encourages better scientific practices and results in more valuable research and publications. Finally, a plan of our future work is outlined and other applications of this work are discussed.

Key-words: experimentation; experimental validation; scientific method; scientific workflows; large-scale experiments

1 Introduction

La reproductibilité d'expériences dans la recherche sur les systèmes complexes est en danger. Ces systèmes sont construits de nombreuses couches logicielles et matérielles. Chaque couche est une abstraction et les abstractions, dans une certaine mesure, sont imparfaites et poreuses. Dans les systèmes contemporains, les erreurs peuvent se produire à n'importe quel niveau et sont très difficiles à cerner. En particulier, la complexité sans cesse croissante des systèmes distribués et des systèmes à grande échelle entrave le développement présent. Pire encore - une fiabilité d'exécution d'expériences souvent n'est pas assurée. Nous pourrions penser que l'expérimentation de systèmes distribués ou utilisant des systèmes distribués est de mauvaise qualité compte-tenu des propriétés intrinsèques de ces systèmes qui confèrent un comportement non prédictible. Cela est la grande douleur, parce que les systèmes distribués sont une abstraction nécessaire pour continuer le progrès dans l'informatique.

Parfois, le problème est le design d'expérience et souvent le facteur humain. Une solution évidente et fructueuse dans l'administration de grands systèmes ou le génie logiciel, consiste dans une automatisation de l'ensemble du processus et l'exécution d'expériences sans surveillance. Pourtant, cela n'est pas toujours possible, parce qu'une évaluation humaine peut être nécessaire.

La complexité des systèmes n'est pas seulement une difficulté pour la reproductibilité, mais également pour les chercheurs. Étonnamment souvent, une réalisation d'expérience est autant complexe qu'incompréhensible, même pour l'auteur. Ce type d'expériences est rarement reproductible, sans parler de la confiance dans les résultats obtenus.

Dans ce travail nous essayons de prouver que la situation courante peut être améliorée. La contribution principale de cet article consiste dans une réalisation pratique d'un moteur de conduite d'expériences basé sur la modélisation et le pilotage de processus métier. Ce moteur peut gérer des expériences complexes avec fiabilité et de façon reproductible. De plus, une description abstraite permet de mieux les comprendre.

L'article est structuré comme suit. La Section 2 contient une description de notre moteur de conduite d'expériences. Pour l'évaluer, dans la Section 3, l'expérience conduite avec le moteur est décrite et notre approche est contrastée avec une liste d'exigences. Dans la Section 4, l'état de l'art est présenté et comparé avec nos résultats. Pour finir, nous concluons notre travail dans la Section 5 et envisageons les prochaines questions à attaquer.

2 Moteur de conduite d'expériences

2.1 Description

Nous avons mis en œuvre un prototype de moteur de conduite d'expériences. L'approche se base sur la modélisation et le pilotage de processus métier. L'idée d'utilisation de processus métier pour améliorer l'état d'art dans un domaine d'expérimentation était déjà envisagé et évalué dans [2]. Dans cet article les exigences importantes et souvent manquantes dans les solutions étaient définies. L'analyse a montrée que l'approche basée sur processus métier est très prometteuse. Alors, notre implantation est une continuation est une réalisation de cet approche. Notre prototype est écrit en Ruby 1.9, mais quand-même garde la compatibilité avec la version 1.8 d'interpréteur.

Traditionnellement, dans un domaine de processus métier, les chercheurs définissent les concepts de base, y compris **les processus** et **les activités**. Ces sont également deux types de constructions qui servent à décrire les expériences dans notre solution.

Les processus font une description de haut niveau – ils lancent et coordonnent les activités et sous-processus, gèrent les erreurs d'exécution, etc. Leur description utilise un langage dédié

pour faciliter le travail d'expérimentateur. Le langage dédié est essentiellement un langage pour écrire de façon formel les workflows comme celui dans la Figure 1. Le langage dédié utilisé par notre moteur est parallèle, néanmoins les situations de compétition (*race condition*) ou d'un interblocage (*deadlock*) sont impossibles : d'abord les variables ne peuvent être assignées qu'une fois (comme en Erlang) et les threads qui les lisent sont implicitement synchronisés. Les processus peuvent être lancés avec les paramètres et peuvent retourner les résultats.

Par contre, **les activités** font partie de l'expérience de bas niveau – elles réservent les ressources, lancent les scripts, interprètent les résultats, etc. Normalement, un langage de programmation impératif est utilisé pour les écrire. Les activités peut accepter des paramètres et retourner des valeurs, comme les fonctions dans n'importe quel langage de programmation. Pour éviter des problèmes typiques de programmation concurrente, les activités doivent être *thread-safe*. Il n'existe aucune restriction sur une imbrication des activités et des processus : l'activité peut être lancée à partir du processus et vice versa.

L'exécution des activités et des processus est minutieusement suivie et les temps des événements, comme le démarrage et la terminaison d'exécution, sont enregistrés. Cette fonctionnalité permet de voir le progrès de l'expérience au fil du temps (cf. Figure 2) et de trouver les goulots d'étranglement.

Les processus et les activités peuvent être groupés dans **une bibliothèque** (nous allons voir une bibliothèque pour accéder Grid'5000 dans la Section 2.3). Les bibliothèques peuvent être importées dans un espace de noms global ou local, mais cette dernière option est conseillée pour éviter les conflits.

Une activité peut enregistrer les actions à faire si une erreur se produit pendant l'exécution d'expérience. Par exemple, la réservation des nœuds est supprimée dans ce cas-là.

2.2 Modèles de structuration des expériences

Pour décrire les processus, un expérimentateur utilise des modèles de structure fournis. Certains d'entre eux sont basés sur les modèles identifiés dans un domaine de processus métier [10]. En revanche, nous avons mis en œuvre quelques méthodes additionnelles, qui sont utiles pour écrire les expériences. Ce qui suit n'est en aucun cas une description complète, et seuls les éléments les plus importants vont être présentés.

Les constructions de base, utilisées presque partout dans des description d'expériences, sont :

- `run` – lance l'activité ou le sous-processus,
- `sequence` – lance les activités l'une après l'autre ; c'est un comportement normal,
- `parallel` – lance des activités en même temps et attend qu'elles soient toutes finies,
- `sleep` – interrompt l'exécution,
- `log` – registre un message,
- `on, switch` – les équivalents des instructions conditionnelles `if` et `switch` dans le langage C ; cependant, leur usage n'est pas recommandé car elles compliquent l'analyse du workflow.

En plus, le moteur offre les modèles concernant l'exécution d'une multitude de tâches similaires :

- `foreach` – crée une boucle séquentielle.
- `forall` – crée une boucle parallèle et attend que tous les itérations soient finies.

Finalement, pour faciliter le travail d'expérimentateur ou exprimer les workflows plus complexes, il existe quelques commandes spéciales :

- `checkpoint` – sauvegarde l'état d'exécution,
- `try` – en cas d'erreur, relance un sous-workflow ; disponible aussi comme un paramètre pour les autres constructions (`parallel`, par exemple),
- `many` – lance des activités en parallèle et attend la première qui se terminera (cf. `parallel`).

Un exemple réel d'utilisation est présenter dans le Listing 1.

Le langage présenté permet de créer les workflows de n'importe quelle complexité. Si le niveau de complexité devient trop important, les actions peuvent être extraites dans le sous-processus. Dans la Section 3.4, on voit comment notre modèle satisfait les exigences avec succès.

2.3 Détails de la mise en œuvre

Sauvegarde d'instantané (*snapshots*) Si une expérience est écrite comme un script monolithique et qu'une erreur se produit, l'état d'exécution est souvent perdu. Après, l'expérimentateur doit relancer l'expérience dès le début, gaspillant du temps et des ressources. Une méthode standard consiste à diviser la code en quelque parties et à les lancer manuellement l'une après l'autre. C'est compliqué et cela prend du temps.

Dans notre approche, l'état du processus (introduit dans la Section 2.1) est sauvegardé sur un disque dur et peut être utilisé en cas d'erreur. Par exemple, comme nous pouvons le voir dans le Listing 1, si l'expérience est brusquement terminée pendant l'exécution d'application (ligne 21), l'expérience peut être relancée depuis la sauvegarde d'instantané `:prepared`. Par défaut, le moteur choisit l'état le plus récent. La sauvegarde d'instantané contient seulement les données vraiment nécessaires – précisément, les variables déclarées *avant* et utilisées *après* le point de sauvegarde. Facultativement, la durée de vie des sauvegardes d'instantané peut être précisée.

Bibliothèque Grid'5000 Avec notre moteur, nous distribuons un ensemble d'activités pour accéder la plate-forme expérimentale Grid'5000 [3]. Cependant, l'interface est complètement indépendante.

Quelques dizaines d'activités ont été définies : nous pouvons interroger la plate-forme, soumettre les réservations et les attendre, distribuer les fichiers, se connecter aux nœuds et exécuter les commandes capturant leur sortie. La bibliothèque gère les erreurs afin qu'un expérimentateur puisse se concentrer sur l'expérience elle-même.

En plus de la capacité d'exécution des commandes directement sur les machines (avec SSH), la bibliothèque offre un langage dédié basé sur le shell Bash. Les avantages de cette approche sont : une meilleure gestion des erreurs, un maintien du contexte local (répertoire courant, par exemple) et l'intégration avec un langage de haut niveau.

3 Expérience conduite

3.1 Introduction

Pour évaluer l'utilité de notre approche, nous avons conçu une expérience et l'avons décrite en utilisant notre logiciel de conduite d'expériences. Il est important de noter que dans le cas d'utilisation qui suit, il faut se concentrer sur l'exécution de l'expérience elle-même et non aux résultats particuliers. L'exemple ci-dessous n'est là que pour illustrer l'expérience typique dans un domaine des réseaux, des systèmes distribués ou des systèmes à grande échelle.

L'expérience concerne la mesure de bande passante du commutateur Ethernet (débit de fond de panier). Cette expérience était conduite sur plate-forme Grid'5000. Le code source de l'expérience, ainsi que le moteur d'expérience utilisé, est librement disponible.

3.2 Description

Selon le motif de communication, les commutateurs réseau présentent des comportement différents. Par exemple, la bande passante maximale est rarement obtenue si tous les ports du

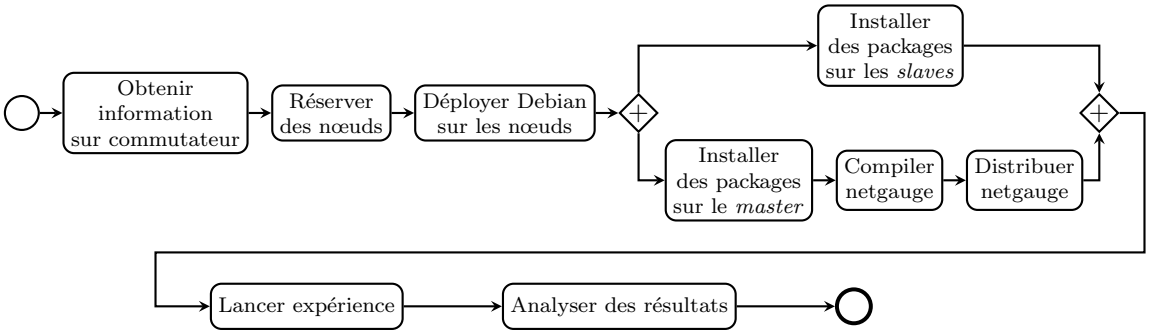


FIGURE 1 – Une représentation d'expérience dans la Section 3.2 en forme du workflow BPMN (*Business Process Modeling Notation*).

commutateur sont utilisés en même temps et dépend de détails sur l'organisation de la communication. Nous allons nous intéresser à la bande passante de la bissection réseau ou, autrement dit, la bande passante entre deux ensembles disjoints de nœuds connectés au commutateur. Puisque la une multitude de combinaisons possibles est impossible à vérifier, on définit l'*effective bisection bandwidth* [6] par la bande passante moyenne si on mesure la performance de la bissection aléatoire de la communication.

La mesure elle-même est prise par le logiciel *netgauge*[1] (version 2.4.6). C'est un logiciel utilisant le standard MPI dont la mise en œuvre utilisée est OpenMPI (version 1.4.3). Finalement, un système d'exploitation utilisé comme environnement expérimental est Debian 6.0 avec noyaux Linux 2.6.32.

Une représentation de haut niveau est présentée dans la Figure 1. Le processus complet peut être divisé en trois parties principales : l'interaction avec la plate-forme Grid'5000, le déploiement d'environnement expérimental et l'expérience elle-même.

L'interface la plus utilisée pour accéder la plate-forme Grid'5000 est un ensemble de commandes dans un shell sécurisé (SSH). Même si très utile, l'interaction avec Grid'5000 est difficile à automatiser. Par exemple, il n'est pas trivial d'obtenir et d'analyser une topologie de réseau. En plus, se connecter aux nœuds de la réservation peut poser quelques problèmes techniques (comme la nécessité d'une distribution des clefs SSH). Finalement, une distribution des fichiers efficace est également difficile. Pourtant, comme décrit dans la Section 2.3, nous avons intégré l'interface fidèle qui utilise notre modèle.

Concernant la partie du déploiement d'environnement, elle est également compliquée. Nous avons toujours utilisé l'interface Grid'5000 pour se connecter aux nœuds et les configurer, mais un élément novateur consiste dans l'exécution complexe de quelques actions parallèles. Dans notre exemple, l'installation de packages sur les *slaves* peut s'exécuter en même temps que l'installation de packages et la compilation du logiciel sur le *master* (cf. Figure 2). Au juste, même la distribution du logiciel compilé vers les nœuds peut coexister avec l'installation de packages. C'est une vraie amélioration dans beaucoup de cas qui permet de gagner de temps et est difficile à faire manuellement.

Finalement, la dernière partie d'expérience profite de notre approche également. Grâce à l'intégration d'analyse de résultats, l'expérience est plus reproductible et la fonctionnalité de sauvegarde d'instantané permet rapidement d'analyser les résultats sans avoir à redémarrer l'expérience.

La description d'expérience dans le langage dédié est présentée dans le Listing 1 et les détails

1. `http://www.unixer.de/research/netgauge/ebb`

```
engine.process :per_site do |site, switch_name|
    switch = run g5k.switch, site, switch_name
    ns = run g5k.nodes, switch
    r = run g5k.reserve_nodes,
        :nodes => ns, :time => '2h',
        :site => site, :type => :deploy
    master, rest = (first_of ns), (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 :distribute_netgauge, master, rest
        end
    end
    checkpoint :prepared
    output = run :netgauge, master, ns
    checkpoint :finished
    run :analysis, output, switch
end
```

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

```
engine.activity :netgauge do |master, nodes|
    log 'Running experiment...'
    run 'g5k.bash', master do
        cd '~/netgauge-build'
        mpirun nodes, './netgauge'
    end
end
```

Listing 1 – Description de l'expérience présentée dans la Section 3.2. La définition du processus métier est sur la gauche, deux exemples d'activités sur la droite.

d'une exécution sont visibles dans la Figure 2.

3.3 Observations et résultats

Les résultats obtenus lors d'une exécution de l'expérience sont présentés dans la Figure 3.

On présente les résultats bruts et la densité empirique de probabilité calculée avec R [2]. La bande passante moyenne par le nœud est égale à 164.96 MiB/s. Le commutateur peut fournir la bande passante complète de bissection parce que la bande passante ne dépend pas d'une partition de nœuds.

Dans la section suivante, nous allons analyser et évaluer notre moteur de conduite d'expériences par rapport à l'approche proposée.

3.4 Évaluation de la méthodologie

Dans [2], les fonctions manquantes dans la plupart de solutions aidant l'expérimentation sont identifiées. Notre prototype du moteur de conduite d'expériences utilise la méthodologie basée sur des processus métier et continue ce travail.

À ce moment, entre les exigences identifiées, nous avons accompli, au moins partiellement : *Descriptiveness*, *Modularity*, *Reusability*, *Maintainability*, *Support for common patterns*, *Snapshotting*, *Error handling*, *Integration of lower-level tools* et *Monitoring*. Il faut remarquer que la fonctionnalité n'est pas complète et il y a toujours beaucoup de détails à améliorer. En plus, *Human interaction*, *Instrumentation* et *Data analysis* sont les grandes exigences qui ne sont pas encore assurées par notre méthode. Le résumé plus précis est dans la Table 1.

Même s'il n'a pas toutes les fonctionnalités attendues, il est une vraie amélioration par rapport aux méthodes précédentes.

2. `http://www.r-project.org/`

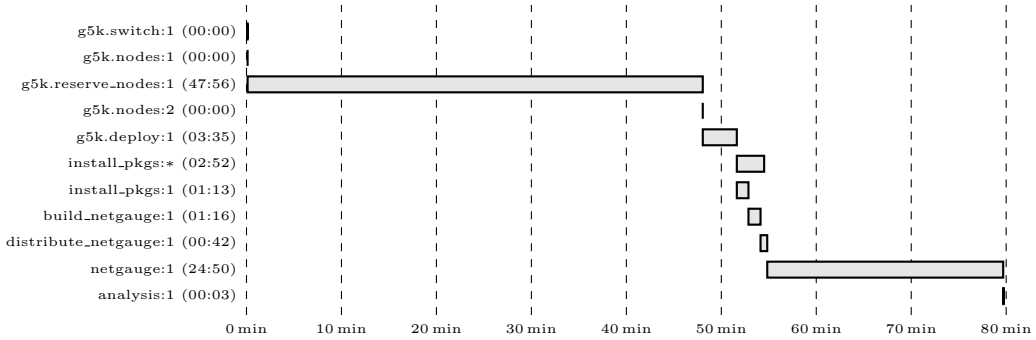


FIGURE 2 – Diagramme de Gantt obtenu pendant l'exécution de l'expérience présentée dans la Section 3.2 et le Listing 1. Les activités exécutées plus d'une fois ont le nombre ordinal supérieur à 1 (cf. *g5k.nodes*). Les instances nombreux d'activité *install_pkgs* ont été groupées dans une tache *install_pkgs :∗*.

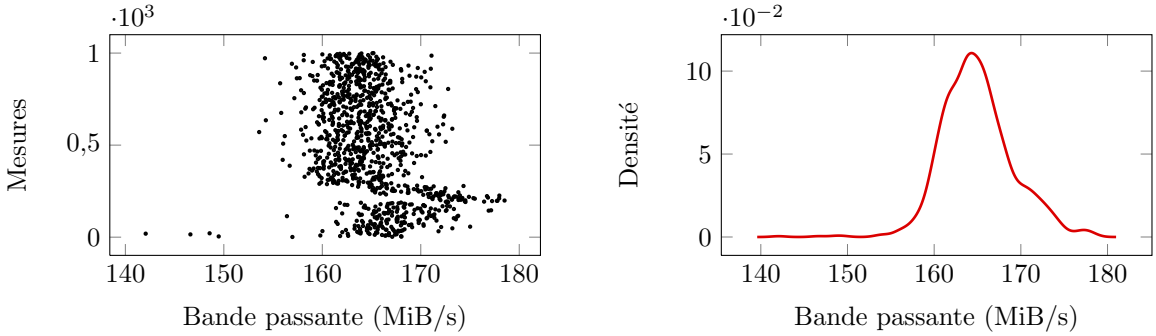


FIGURE 3 – Résumé des résultats d'expérience dans la Section 3.2. Les données brutes sont présentée sur la gauche et une densité empirique de probabilité sur la droite.

4 État de l'art

Un contrôle précis et robuste d'expériences est un problème depuis le début de recherche sur les systèmes distribués. Ainsi, cela n'est pas surprenant qu'il y ait beaucoup d'approches et de logiciels qui ont pour mission de l'améliorer. Ci-dessous sont listés et analysés les efforts existants.

OMF *cOntrol and Management Framework* [8] est l'ensemble du logiciel pour une gestion, un contrôle et prend des mesures aux bancs des tests réseau. Les expériences sont décrites avec un langage dédiée fondé sur les événements (« les nœuds sont prêts » ou « augmente le trafic du réseau toutes les 15 secondes », par exemple). Les ressources expérimentales requises peuvent être précisées en détail et une collection de résultats dans la base de données relationnelle est intégrée. OMF est utilisé par plusieurs installations et publié sous licence *open source*.

Notre solution et OMF vident le même objectif, tandis que OMF est un projet plus complet. Par contre, nous croyons que notre approche basée sur l'idée de processus métier est plus générique et permet une meilleure structure d'expérience. En plus, nous envisageons quelques applications sans rapport avec l'expérimentation.

Expo Le moteur d'expériences *Expo* [11] était conçu pour gérer l'exécution d'expériences sur les

Apport attendu	Moyen d'atteindre cet objectif
Descriptiveness	Le langage dédié puissant et expressif
Modularity	L'abstraction d'activités
Reusability	L'abstraction d'activités et d'une bibliothèque
Maintainability	Les abstractions claires, enregistrement des événements
Support for common patterns	Le langage basé sur les processus métier
Snapshotting	La fonctionnalité de sauvegarde d'instantanés
Error handling	La politique paramétrable et claire de la gestion d'erreurs
Integration of lower-level tools	L'abstraction d'activités et d'une bibliothèque
Monitoring	Suivre l'état et des événements importantes

TABLE 1 – Le résumé d'évaluation de la méthodologie présentée.

plate-formes dédiées (Grid'5000, principalement). Les expériences sont décrites dans un langage dédié très accessible qui simplifie l'interaction avec une plate-forme. Un expérimentateur peut réserver et contrôler les nœuds, ainsi que lancer des tâches en parallèle, capturant leur résultats.

Contrairement à notre solution, Expo n'offrit pas les moyens avancés pour exprimer les expériences complexes. Un langage dédié est impératif et souffre de l'expressivité limitée.

g5k-campaign Un logiciel dédié à Grid'5000, *g5k-campaign*[3], est un moteur expérimental conduit par les événements (réalisée sous la forme d'une classe en langage Ruby).

L'usage de *g5k-campaign* automatise quelques étapes du travail avec Grid'5000, mais requiert beaucoup d'attention aux détails de bas niveau, et une gestion d'erreurs manuelle. En plus, c'est un outil accessible seulement sur la plate-forme Grid'5000.

ZENTURIO Un outil consacré à l'étude des paramètres, l'analyse de performance et l'essai du logiciel sur les clusters ou les grilles, ZENTURIO [7], emplit un langage spécial pour préciser les détails d'expérience et s'interface avec les systèmes de grille. D'ailleurs, l'interface du web est disponible pour suivre le progrès et visualiser les résultats.

ZENTURIO n'aide pas au design d'expériences directement - il les traite comme une boîte noire. En revanche, notre solution peut exprimer l'expérience et l'exécution de celui-ci aussi.

DART *Distributed Automated Regression Testing for Large-Scale Network Applications* [4] est un outil pour découvrir les régressions dans les applications distribuées. Il contrôle l'exécution d'expérience et simule les situations exceptionnelles (les défauts du réseau, par exemple).

DART n'est pas vraiment une solution pour un problème posé (il existe pour tester les applications distribuées, pas pour les évaluent), mais il inclut quelques idées similaires.

Plush (Gush) Originellement, Plush et son successeur Gush [1], étaient dédiés pour le banc de test PlanetLab. Gush vise à devenir un système extensible de gestion d'exécution d'expériences. Le traitement d'erreurs est bien intégrée et avec le framework nous pouvons surveiller le système et modifier ses paramètres à la volée.

Gush ne se focalise ni sur la reproductibilité ni sur l'expression d'expériences avec les modèles expérimentaux communs, contrairement à notre approche.

Weevil La cible du projet Weevil [12] est l'amélioration d'expérimentations avec les systèmes distribués. La fonctionnalité de base consiste dans l'exécution facile d'expériences et la génération de la charge artificielle.

Les deux caractéristique unique du Weevil sont : l'injection de fautes et la génération de la charge artificielle. Pourtant, l'expressivité d'expériences est limitée.

3. `http://g5k-campaign.gforge.inria.fr/`

NXE *Network eXperiment Engine* [4] facilite la conduite d'expériences de réseau et l'interaction avec les nœuds considérés dans l'étude. Le logiciel fournit le workflow prédéfini.

NXE possède une fonctionnalité intéressante, mais il est plutôt simple et inflexible. Par exemple, le workflow ne peut pas être modifié, sans parler d'exprimer les workflows plus complexes.

Experimentation workbench for Emulab Emulab est un banc de test qui offre sa propre solution : une interface graphique avec contrôle de base d'expériences. Adressant restrictions de cet interface, *Experimentation workbench for Emulab* [5] était conçu. Il encourage une recherche reproductible, aidant le design d'expériences et leur exécution. Un stockage automatique des résultats, bien que la possibilité du travail collaboratif soient disponible.

Cette solution possède quelques caractéristiques intéressantes et est plutôt mature. Cependant, la solution rencontre des problèmes avec passage à l'échelle. En plus, les auteurs mentionnent qu'un modèle d'expérimentation implanté est responsable des problèmes de la convivialité.

Les workflows scientifiques Les autres solutions utilisant le concept de processus métier, les workflows scientifiques (VisTrails, Taverna, Kepler) [13], sont fructueusement utilisées dans la biologie, la médecine, etc. L'avantage principal est que les workflows peuvent être écrits par des chercheurs qui ne sont pas d'informaticiens. Les workflows sont exécutés facilement ou partagés entre les chercheurs, parfois à l'aide d'une plate-forme dédiée, comme myExperiment [5].

Les workflows scientifiques se concentre sur la modélisation d'interdépendance entre les données et les tâches. Même s'ils aident l'exécution du calcul sur les plate-formes distribuées, celles plate-formes sont seulement un outil, pas l'objet d'étude.

5 Conclusion et travaux futurs

Dans cet article, nous avons introduit un premier prototype de notre moteur de conduite d'expériences. Bien qu'il soit encore un projet jeune, il a déjà prouvé son utilité. Certaines caractéristiques importantes manquent encore et sa fiabilité peut être améliorée.

En dehors de la liste ci-dessus, l'ajout de plusieurs fonctionnalités est envisagé :

- les moyens d'intégration avec les autres outils expérimentaux,
- l'écriture des activités dans n'importe quel langage de programmation (cf. Org-mode [9]),
- le planning d'expérience avancé (*design of experiments*, DOE),
- l'interface graphique en forme de site web qui expose les détails d'expérience et le progrès,
- l'intégration d'une bibliothèque avec les activités collectant et analysant les résultats,
- la possibilité d'interaction humaine au cours de l'expérience (l'expérience avec surveillance).

Nous allons travailler sur ces questions dans le futur.

Références

[1] Albrecht (J.), Tuttle (C.), Braud (R.), Dao (D.), Topilski (N.), Snoeren (A. C.) et Vahdat (A.). – Distributed Application Configuration, Management, and Visualization with Plush. *ACM Transactions on Internet Technology*, vol. 11, décembre 2011, pp. 6 :1–6 :41.

4. `http://ens-lyon.fr/LIP/RESO/Software/NXE/`
5. `http://www.myexperiment.org/`

[2] Buchert (T.) et Nussbaum (L.). – Leveraging business workflows in distributed systems research for the orchestration of reproducible and scalable experiments. *Majecstic 2012*, octobre 2012.

[3] Cappello (F.), Desprez (F.), Dayde (M.), Jeannot (E.), Jégou (Y.), Lanteri (S.), Melab (N.), Namyst (R.), Primet (P.), Richard (O.), Caron (E.), Leduc (J.) et Mornet (G.). – Grid'5000 : a large scale, reconfigurable, controlable and monitorable Grid platform. *In : 6th IEEE/ACM International Workshop on Grid Computing (Grid)*.

[4] Chun (B. N.). – DART : Distributed Automated Regression Testing for Large-Scale Network Applications. *In : Proceedings of the 8th International Conference on Principles of Distributed Systems*.

[5] Eide (E.), Stoller (L.) et Lepreau (J.). – An Experimentation Workbench for Replayable Networking Research. *In : Proceedings of the 4th Symposium on Networked System Design and Implementation*.

[6] Hoefler (T.), Schneider (T.) et Lumsdaine (A.). – Multistage Switches are not Crossbars : Effects of Static Routing in High-Performance Networks. *In : Proceedings of the 2008 IEEE International Conference on Cluster Computing*. – IEEE Computer Society.

[7] Prodan (R.) et Fahringer (T.). – ZENTURIO : An Experiment Management System for Cluster and Grid Computing. *In : Proceedings of the 4th International Conference on Cluster Computing (CLUSTER 2002)*. pp. 9–18. – IEEE Computer Society Press.

[8] Rakotoarivelo (T.), Ott (M.), Jourjon (G.) et Seskar (I.). – OMF : a control and management framework for networking testbeds. *ACM SIGOPS Operating Systems Review*, vol. 43, n° 4, Jan 2010, pp. 54–59.

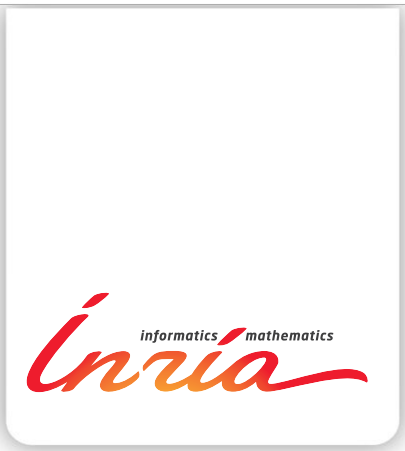
[9] Schulte (E.), Davison (D.), Dye (T.) et Dominik (C.). – A Multi-Language Computing Environment for Literate Programming and Reproducible Research. *Journal of Statistical Software*, vol. 46, n° 3, 1 2012, pp. 1–24.

[10] Van Der Aalst (W. M. P.), Ter Hofstede (A. H. M.), Kiepuszewski (B.) et Barros (A. P.). – Workflow Patterns. *Distrib. Parallel Databases*, vol. 14, n° 1, juillet 2003, pp. 5–51.

[11] Videau (B.) et Richard (O.). – Expo : un moteur de conduite d'expériences pour plates-forme dédiées. *In : Conférence Française en Systèmes d'Exploitation (CFSE)*.

[12] Wang (Y.), Rutherford (M. J.), Carzaniga (A.) et Wolf (A. L.). – Automating Experimentation on Distributed Testbeds. *In : Proceedings of the 20th IEEE/ACM International Conference On Automated Software Engineering (ASE)*. pp. 164–173. – New York, NY, USA, 2005.

[13] Yu (J.) et Buyya (R.). – A Taxonomy of Scientific Workflow Systems for Grid Computing. *SIGMOD Record*, vol. 34, September 2005, pp. 44–49.



RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399