



Orchestration d'expériences à l'aide de processus métier

Tomasz Buchert

► To cite this version:

Tomasz Buchert. Orchestration d'expériences à l'aide de processus métier. ComPAS: Conférence d'informatique en Parallélisme, Architecture et Système., Jan 2013, Grenoble, France. hal-00749601v2

HAL Id: hal-00749601

<https://inria.hal.science/hal-00749601v2>

Submitted on 7 Dec 2012 (v2), last revised 28 Feb 2013 (v4)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Orchestration d'expériences à l'aide de processus métier

Tomasz Buchert

INRIA – LORIA – Université de Lorraine – CNRS
Équipe AlGorille - 615, rue du Jardin Botanique
54500 Villers-lès-Nancy - France
tomasz.buchert@inria.fr

Résumé

La recherche sur les systèmes distribués a quelques caractéristiques qui la rendent intéressante, difficile et unique. D'abord, les systèmes modernes sont souvent trop compliqués pour permettre une analyse formelle et théorique. Ainsi, l'approche la plus courante est l'analyse empirique, par exemple à l'aide d'expériences. Malheureusement, à cause de la complexité des systèmes informatiques, réaliser des expériences est très difficile à faire correctement, avec une rigueur scientifique suffisante et de façon reproductible.

Cet article présente une idée novatrice pour écrire, représenter et conduire les expériences scientifiques impliquant une multitude des nœuds hétérogènes connectés par un réseau. L'idée centrale consiste à utiliser la modélisation et le pilotage de processus métier (*Business Process Modeling* et *Business Process Management*). Notre solution a de nombreux avantages, dont les plus importants sont la facilité d'écrire et de comprendre une description d'expérience, et la robustesse d'exécution d'expériences même en présence de problèmes de traitement. Après la présentation de notre approche, un prototype du moteur de conduite d'expériences est validé à l'aide d'un cas d'utilisation de référence.

Mots-clés : expérimentation ; validation expérimentale ; démarche scientifique ; processus métier ; expériences à grande échelle

1. Introduction

Les systèmes distribués 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.

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 comparée 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 conçu un prototype de moteur de conduite d'expériences. L'approche se base sur la modélisation et le pilotage de processus métier. L'idée d'utiliser 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 suite et une réalisation de cette approche.

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

Les processus sont 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 formelle les workflows comme celui de 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 des 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. Un langage de programmation impératif (Ruby) 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 : les activités peuvent être lancées à partir de 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 des exécutions, sont enregistrés. Cette fonctionnalité permet de voir la progression de l'expérience au fil du temps (cf. Figure 2) et de déterminer les goulots d'étranglement.

Les processus et les activités peuvent être groupés dans **une bibliothèque** (nous présentons 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 peut alors être supprimée automatiquement.

Actuellement, le moteur se compose d'environ 5000 lignes de code en Ruby (120 KB).

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 le domaine des processus métiers [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 sont 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 le comportement normal,
- *parallel* – lance des activités en même temps et attend qu’elles soient toutes finies,
- *sleep* – interrompt l’exécution,
- *log* – enregistre 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 présenté 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 un sous-processus. Dans la Section 3.4, nous montrons 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 perdu. L’expérimentateur doit relancer l’expérience dès le début, gaspillant du temps et des ressources.

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 échoue pendant l’exécution de l’application (ligne 21), elle peut être relancée depuis la sauvegarde d’instantané : *prepared*. La sauvegarde d’instantané contient des variables du workflow sérialisées et à la suite une complexité en temps et en espace de sauvegarde est proportionnelle à la mémoire utilisée par ces variables. Pour l’instant, seul l’état du moteur est sauvegardé alors que une réservation de nœuds et leur configuration ne sont pas considérées. Nous voulons aborder ce problème dans nos travaux futurs.

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 des réservations et les attendre, distribuer des fichiers, se connecter aux nœuds et exécuter des 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 sur les 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 la plate-forme Grid’5000. Le code source de l’expérience, ainsi que le moteur d’expérience utilisé, est librement disponible.

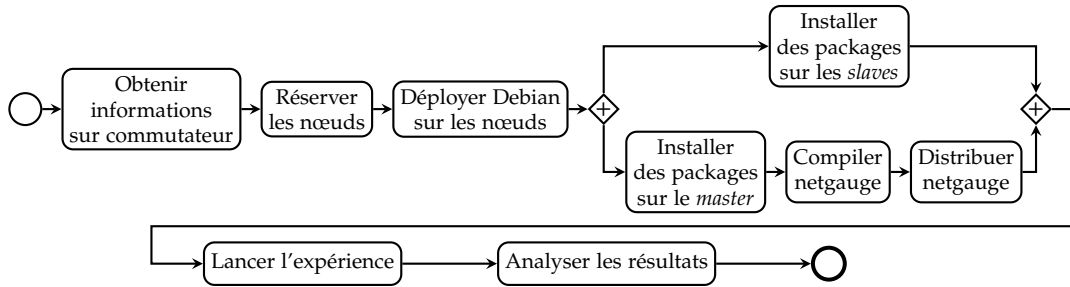


FIGURE 1 – Représentation de l’expérience de la Section 3.2 en formalisme BPMN (*Business Process Modeling Notation*).

3.2. Description

Selon le motif de communication, les commutateurs réseau présentent des comportements différents. Par exemple, la bande passante maximale est rarement obtenue si tous les ports du 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 bisection réseau ou, autrement dit, la bande passante entre deux ensembles disjoints de nœuds connectés au commutateur. Puisque la multitude de combinaisons possibles ne peuvent toutes être vérifiées, on définit l’*effective bisection bandwidth* [6] par la bande passante moyenne si on mesure la performance de la bisection 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, le 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 à la plate-forme Grid’5000 est un ensemble de commandes dans un shell sécurisé (SSH). Malheureusement, cette interface est difficile à automatiser. Il n’est par exemple pas trivial d’obtenir et d’analyser la topologie réseau des sites. De plus, se connecter aux nœuds de la réservation peut poser quelques problèmes techniques (comme la configuration de clés SSH). Gérer les données de l’expérience de manière efficace est également difficile. Notre moteur utilise donc l’API REST fournie par Grid’5000 pour manipuler la plate-forme.

Lors du déploiement de l’environnement, nous avons également utilisé l’API REST Grid’5000. Un élément novateur est toutefois 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). D’ailleurs, 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 de l’expérience profite également de notre approche. Grâce à l’intégration de l’analyse des 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 de l’expérience dans le langage dédié est présentée dans le Listing 1 et les détails d’une exécution sont visibles dans la Figure 2. La version finale de l’expérience se compose de 141 lignes (dont 24 lignes de script R).

3.3. Observations et résultats

Les résultats obtenus sont présentés dans la Figure 3 (résultats bruts et densité empirique de probabilité calculée avec R^2). La bande passante moyenne par nœud est égale à 164.96 MiB/s.

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

1. <http://www.unixer.de/research/netgauge/ebb/>

2. <http://www.r-project.org/>

```

1 engine.process :per_site do |site, switch_name|
2   switch = run g5k.switch, site, switch_name
3   ns = run g5k.nodes, switch
4   r = run g5k.reserve_nodes,
5       :nodes => ns, :time => '2h',
6       :site => site, :type => :deploy
7   master, rest = (first_of ns), (tail_of ns)
8   run g5k.deploy, r, :env => 'squeeze-x64-nfs'
9   checkpoint :deployed
10  parallel :retry => true do
11    forall rest do |slave|
12      run :install_pkgs, slave
13    end
14    sequence do
15      run :install_pkgs, master
16      run :build_netgauge, master
17      run :distribute_netgauge, master, rest
18    end
19  end
20  checkpoint :prepared
21  output = run :netgauge, master, ns
22  checkpoint :finished
23  run :analysis, output, switch
24 end

```

```

1 engine.activity :install_pkgs do |node|
2   log 'Installing packages on ', node
3   run 'g5k.bash', node do
4     aptget :update
5     aptget :upgrade
6     aptget :purge, 'mx'
7   end
8 end

```

```

1 engine.activity :netgauge do |master, nodes|
2   log 'Running experiment...'
3   run 'g5k.bash', master do
4     cd '~/netgauge-build'
5     mpirun nodes, './netgauge'
6   end
7 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.

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

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

3.4. Évaluation de la méthodologie

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

À ce stade, parmi les exigences identifiées, nous satisfaisons au moins partiellement : *Descriptiveness*, *Modularity*, *Reusability*, *Maintainability*, *Support for common patterns*, *Snapshotting*, *Error handling*, *Integration of lower-level tools* et *Monitoring*. Toutefois, les fonctionnalités ne sont pas complètes et il y a beaucoup de détails à améliorer. D'autres fonctions (*Human interaction*, *Instrumentation* et *Data analysis*) ne sont pas encore assurées par notre méthode. Une description plus précise est fournie dans la Table 1.

4. État de l'art

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

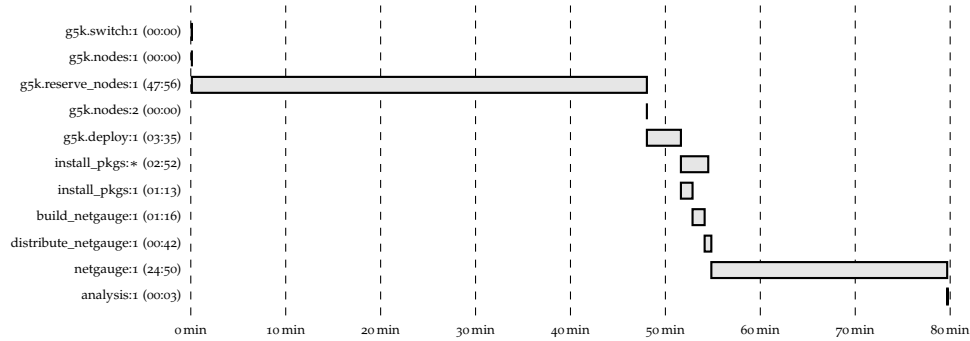


FIGURE 2 – Diagramme de Gantt obtenu pendant l'exécution de l'expérience présentée dans la Section 3.2 et le Listing 1. Les activités exécutées plus d'une fois ont le nombre ordinal supérieur à 1 (cf. *g5k.nodes*). Les activités aux nombreuses instances (*install_pkgs*) ont été groupées dans une tâche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

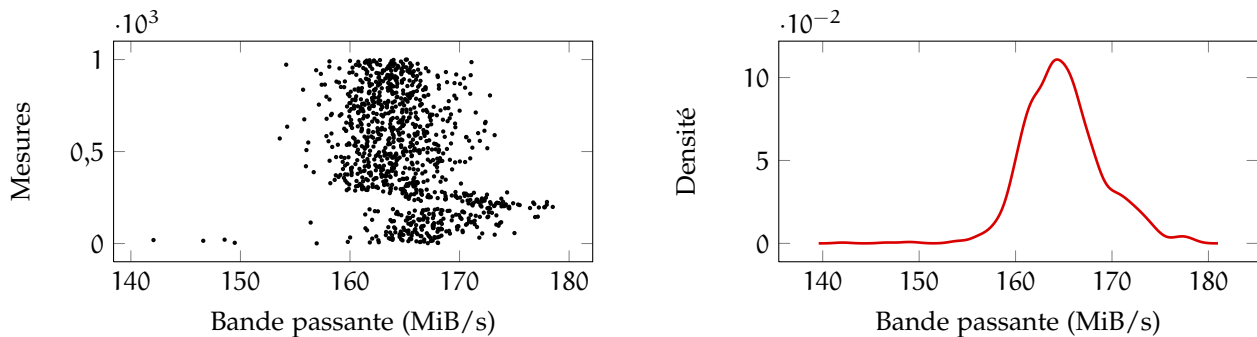


FIGURE 3 – Résumé des résultats d'expérience dans la Section 3.2. Les données brutes sont présentées sur la gauche et une densité empirique de probabilité sur la droite.

Les scripts bruts Une méthode traditionnelle consiste à écrire des scripts d'expérience dans un langage de programmation impératif, souvent en langage Bash.

Bien que ce soit la méthode la plus populaire, elle rend les expériences difficiles à comprendre, ne permet que très difficilement de gérer les erreurs, et possède tous les problèmes d'une approche de bas niveau (dont une susceptibilité aux bogues).

OMF *cOntrol and Management Framework* [8] est un ensemble de logiciels pour une gestion, un contrôle et une prise de mesures sur des *testbed* réseaux. Les expériences sont décrites avec un langage dédié 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 récupération des 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 visent le même objectif, tandis que OMF est un projet plus complet. Par contre, nous croyons que notre approche basée sur l'approche des processus métier est plus générique et permet une meilleure structure des expériences. De 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 plateformes 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 leurs résultats.

Contrairement à notre solution, Expo n'offre pas de moyens avancés pour exprimer les expé-

riences complexes. Un langage dédié est impératif et souffre d'une expressivité limitée.

g5k-campaign Un logiciel dédié à Grid'5000, *g5k-campaign*³, est un moteur expérimental conduit par les événements (réalisé 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. De 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 performances et l'essai de logiciels sur les clusters ou les grilles, ZENTURIO [7], utilise 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 celle-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 le problème posé (il existe pour tester les applications distribuées, pas pour les évaluer), mais il inclut quelques idées similaires.

Plush (Gush) Originellement, Plush et son successeur Gush [1], étaient dédiés à la plate-forme PlanetLab. Gush vise à devenir un système extensible de gestion d'exécution d'expériences. Le traitement d'erreurs est bien intégré, 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. Weevil offre 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. Cependant, l'expressivité d'expériences est limitée.

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

NXE possède des fonctionnalités intéressantes, 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 une plate-forme expérimentale qui offre sa propre solution : une interface graphique avec contrôle de base d'expériences. Visant à contourner des restrictions de cette interface, *Experimentation workbench for Emulab* [5] a été conçu. Il encourage une recherche reproductible, aidant le design d'expériences et leur exécution.

Cette solution possède quelques caractéristiques intéressantes et est plutôt mature. Cependant, la solution rencontre des problèmes avec le passage à l'échelle.

Les workflows scientifiques pour des grilles informatiques Les autres solutions utilisant le concept de workflows, les workflows scientifiques (VisTrails, Taverna, Kepler) [13], sont utilisées avec succès dans la biologie, la médecine, etc. L'avantage principal est que les workflows peuvent être écrits par des chercheurs qui ne sont pas des informaticiens. Les workflows sont exécutés facilement et partagés entre les chercheurs, parfois à l'aide d'une plate-forme dédiée (cf. myExperiment⁵).

Les workflows scientifiques se concentrent sur la modélisation d'interdépendance entre les données et les tâches. Même s'ils aident l'exécution du calcul sur les plate-formes distribuées, ces plate-formes sont seulement un outil pour effectuer le calcul. Par contre, dans notre travail nous considérons des systèmes distribués, y compris les grilles, comme les objets propres d'étude.

3. <http://g5k-campaign.gforge.inria.fr/>

4. <http://ens-lyon.fr/LIP/RESO/Software/NXE/>

5. <http://www.myexperiment.org/>

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 :

- l'intégration avec les autres outils facilitant la réalisation d'expériences,
- la sauvegarde d'instantané plus avancée,
- l'écriture d'activités dans n'importe quel langage de programmation (cf. Org-mode [9]),
- le planning d'expériences avancé (*design of experiments*, DOE),
- une interface graphique qui expose les détails de l'expérience et sa progression,
- l'intégration d'une bibliothèque avec les activités collectant et analysant les résultats,
- la possibilité d'interaction avec l'utilisateur au cours de l'expérience.

Nous allons travailler sur ces questions dans le futur.

Bibliographie

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.
2. Buchert (T.) et Nussbaum (L.). – Leveraging business workflows in distributed systems research for the orchestration of reproducible and scalable experiments. *Majestic 2012*, Oct 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, controllable 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, n4, 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, n3, 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, n1, 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.