Permitindo Maior Reprodutibilidade de Experimentos em Ambientes Distribuídos com Nodos de Baixa Confiabilidade

Nelson Antônio Antunes Junior, Weverton Luis da Costa Cordeiro, Luciano Paschoal Gaspary

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

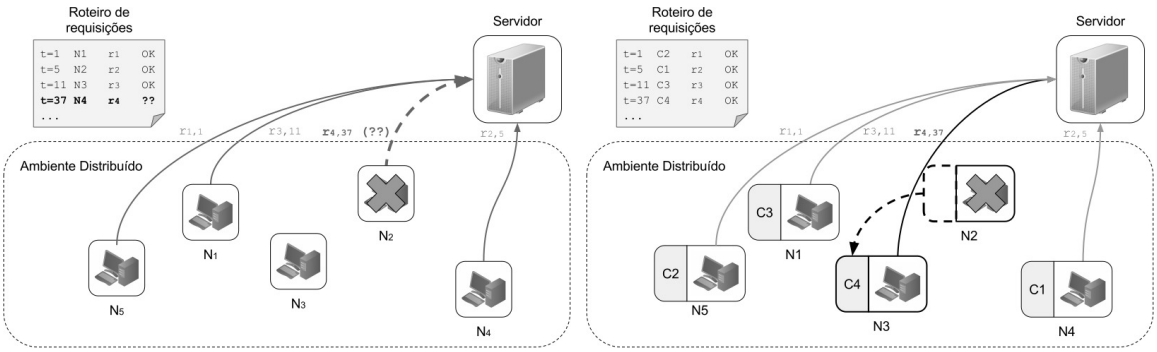
{naajunior, wlccordeiro, paschoal}@inf.ufrgs.br

Resumo. *A reprodutibilidade de experimentos, essencial para a verificação da eficácia/eficiência de contribuições científicas, é particularmente desafiadora no contexto de sistemas distribuídos de larga escala. Falhas não programadas durante os experimentos (sejam nos nodos que fazem parte do sistema, ou de comunicação entre eles) podem dificultar a obtenção de significância estatística nos resultados, ou a verificação da validade dos mesmos. Para abordar esse problema propõe-se* EASYEXP*, uma arquitetura tolerante a falhas para garantir a reprodutibilidade de experimentos em testbeds distribuídos de baixa confiabilidade. No* EASYEXP*, nodos do ambiente de experimentação "interpretam" trabalhadores e executam ações previstas para os mesmos, seguindo o roteiro pré-definido para o experimento. Na falha de um nodo, substitui-se o mesmo por outro funcional, mantendo o contexto de execução do trabalhador interpretado por ele. Resultados obtidos mostram que o* EASYEXP *é capaz de manter menor variação (desvio padrão de 1.6%) e maior precisão (95.7%) entre múltiplas execuções de um mesmo experimento, quando comparado àqueles executados de forma tradicional (desvio de até 25% e precisão de apenas 72%).*

Abstract. *Experiment reproducibility, essential for the verification of effectiveness/efficiency of scientific contributions, is particularly challenging in the context of large-scale distributed systems. Non-programmed failures (either at nodes that compose the system, or in the communication between them) may make it difficult for one to achieve statistical significance in the results, or to verify their validity. To address this problem, we propose* EASYEXP*, a fault-tolerant architecture to ensure the reproducibility of experiments in non-reliable distributed testbeds. In* EASYEXP*, nodes in the experiment environment "interpret" workers and execute actions that are expected for them, following a predefined schedule. In the event of failure of a node, it is replaced by another functional one, keeping the execution context of the worker interpreted by it. Results obtained show that* EASYEXP *is able to maintain a lower variation (standard deviation of 1.6 %) and higher precision (95.7 %) among multiple runs of the same experiment, when compared to those performed in a traditional way (25% deviation and 72% accuracy only).*

1. Introdução

A reprodutibilidade de experimentos é uma das principais formas de se verificar a eficácia/eficiência de contribuições científicas [Baker 2016]. O método aplicado, as métricas utilizadas, as condições de operação, os dados de entrada e de saída, entre outros, estão entre os requisitos mínimos necessários para garantir que os resultados de



(a) Sem recuperação (b) Com recuperação

Figura 1. Execução de experimentos em *testbeds* sem e com recuperação de falhas não programadas.

avaliação de uma contribuição científica sejam reprodutíveis. Diversos aspectos de reprodutibilidade têm sido discutidos recentemente, incluindo suporte por parte de ambientes de experimentação [Nussbaum 2017] entre outros desafios [Bajpai et al. 2017]. A importância desse tópico inclusive motivou um workshop no ACM SIGCOMM 2017 sobre reprodutibilidade [Bonaventure et al. 2017].

Para aplicações distribuídas (por exemplo de *live streaming*, distribuição de conteúdos, etc.), a reprodutibilidade é particularmente desafiadora. O longo período de operação dessas aplicações as torna mais suscetíveis a instabilidades durante a experimentação, devido à probabilidade cumulativa de falhas não programadas transientes ou permanentes (nos nodos que fazem parte do *testbed*, ou de comunicação entre os mesmos). Para ilustrar, considere a avaliação de um protocolo para redes de distribuição de conteúdos, para o qual se deseja aferir sua eficácia e eficiência. O ambiente de experimentação é formado por um servidor central S, e por vários nodos N_i distribuídos geograficamente. Cada nodo N_i deverá enviar um conjunto de requisições ao servidor central em um determinado instante, a partir das quais interações subsequentes deverão ocorrer. O objetivo da avaliação é analisar o tempo necessário para que todas as requisições sejam completadas com sucesso e o tempo médio de resposta das solicitações.

Para definir uma avaliação que gere resultados reprodutíveis, faz-se necessário padronizar os parâmetros da mesma, por exemplo a sequência de requisições $r_{j,t}$ que cada nodo N_i deverá realizar ao servidor em um instante t. O desafio, nesse caso, é diminuir a influência de falhas não programadas do ambiente na avaliação do desempenho da proposta sob análise. Por exemplo, considere o caso ilustrado na Figura 1(a). A figura descreve uma instância do cenário recém descrito, no qual um roteiro de requisições é seguido pelos nodos do ambiente, que enviam requisições ao servidor. Nesse caso, as requisições $r_{1,1}$, $r_{2,5}$ e $r_{3,11}$ foram enviadas com sucesso pelos nodos N_5, N_1 e N_4, respectivamente. A requisição $r_{4,37}$, no entanto, não foi enviada por falha do nodo N_2 (que deveria tê-la enviado). Caso a falha seja permanente, nenhuma outra requisição alocada para N_2 será enviada. A principal consequência negativa é que o resultado da avaliação não será fiel ao roteiro base, e novas execuções dessa mesma avaliação gerarão resultados não passíveis de comparação. Em resumo, avaliações da mesma natureza que a descrita no exemplo impõem aos pesquisadores um desafio adicional para se alcançar um bom grau de reprodutibilidade.

Para superar esse desafio, nesse artigo propõe-se EASYEXP, uma arquitetura tole-

rante a falhas para garantir a reprodutibilidade de experimentos em *testbeds* distribuídos de baixa confiabilidade. No EASYEXP, os nodos "interpretam" trabalhadores e executam ações previstas para os mesmos, de acordo com o roteiro base. Para ilustrar, considere o exemplo da Figura 1(b). O roteiro base define as ações a serem executadas por *trabalhadores* C_i, os quais são "interpretados" por nodos N_i do ambiente. Na falha de um nodo (no exemplo, N_2), substitui-se o mesmo por outro funcional (no exemplo, para C_4 utiliza-se o próximo nodo disponível, N_3), mantendo o contexto de execução do trabalhador interpretado por ele. Com o EASYEXP, o progresso de um experimento não está atrelado ao nodo onde a parte do experimento distribuído é executado, mas sim a uma *abstração* de trabalhadores. Essa abstração pode ainda ser chamada de contexto, com nodos sendo designados para instanciar os contextos listados no experimento.

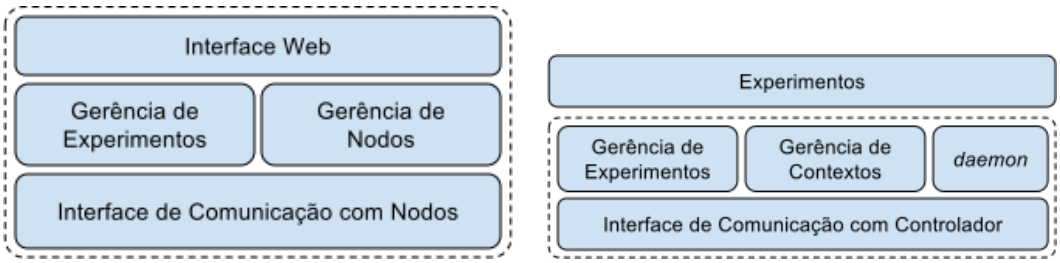
Os resultados obtidos reproduzindo experimentos com o EASYEXP mostraram que a arquitetura proposta é capaz de manter menor variação (desvio padrão de 1.6%) e maior precisão (95.7%) entre múltiplas execuções, quando comparado àqueles executados de forma tradicional (desvio de até 25% e precisão de apenas 72%). Mais ainda, esses resultados são alcançados sem intervenção manual durante a execução do experimento. Isso evidencia a efetividade dos mecanismos de detecção de falhas e de salvamento e recuperação de contexto dos trabalhadores.

O restante do artigo está organizado como segue. Na Seção 2 são revisados os principais trabalhos relacionados. A Seção 3 descreve a arquitetura proposta, e discorre sobre os desafios de pesquisa abordados para a sua materialização. Na Seção 4 são discutidos aspectos de implementação da arquitetura, enquanto que na Seção 5 são apresentados os principais resultados obtidos com a reprodutibilidade de um experimento genérico. Por fim, a Seção 6 fecha o artigo com uma breve discussão sobre aplicabilidade do EASYEXP, limitações da arquitetura proposta, e perspectivas de trabalhos futuros.

2. Trabalhos Relacionados

A instabilidade de nodos (e da comunicação entre os mesmos) em *testbeds* distribuídos tem sido alvo de várias investigações recentes [Santos et al. 2014, Costa et al. 2015, Garrett et al. 2017]. Costa *et al.* [Costa et al. 2015] e Garrett *et al.* [Garrett et al. 2017] abordam esse desafio, por exemplo, com algoritmos de seleção das máquinas menos propensas a falhas como alternativa para melhorar a reprodutibilidade de experimentos. Em resumo, os autores baseiam-se na análise de métricas como tempo de resposta entre subconjuntos de nodos pertencentes ao *testbed*, e aplicação de estratégias de seleção dos nodos mais estáveis para execução de experimentos de longa duração. Apesar dos resultados positivos alcançados, não há garantias de que os nodos escolhidos, embora mais estáveis no passado, não falharão no futuro.

Em paralelo, vários *frameworks* foram desenvolvidos para facilitar a configuração de experimentos e, por consequência, diminuir o esforço necessário para alcançar reprodutibilidade nesses ambientes. Gush (GENI User Shell) [Albrecht and Huang 2010], anteriormente conhecido com Plush (PlanetLab User Shell) [Albrecht et al. 2007] são alguns dos exemplos mais proeminentes. O objetivo do Gush/Plush é gerenciar experimentos em plataformas de ambientes distribuídos, oferecendo configuração via arquivos XML, e permitindo aos usuários gerenciarem os nodos remotos como recursos a serem descobertos e alocados dinamicamente, seja usando SSH (Secure Shell) ou APIs das plataformas suportadas (ex., GENI). Apesar de permitir a substituição de nodos falhos durante experimentos, não há recuperação do contexto de execução dos nodos. Nesse caso, nodos que substituem outros falhos podem inclusive repetir todas as ações já executa-



(a) Controlador (b) Trabalhador

Figura 2. Visão conceitual do controlador e do trabalhador no EASYEXP.

das pelo nodo substituído (antes da sua falha). Essa limitação torna o Gush/Plush menos interessante para uso em experimentos sensíveis à perda de contexto por parte dos nodos.

Outros exemplos de *frameworks* são o Splay [Leonini et al. 2009], EXPO [Ruiz et al. 2013] e EXECO [Imbert et al. 2013]. O Splay [Leonini et al. 2009], por exemplo, foi desenvolvido para avaliação de algoritmos distribuídos executados em ambientes controlados. No Splay, cada nodo remoto do sistema possui um *daemon* que deve ser conectado ao controlador (entidade que gerencia todas as operações). O *daemon* delimita os recursos disponíveis ao experimento (em cada nodo) e fornece uma interface para utilização dos mesmos, via linguagem Lua. A execução remota se baseia em RPC (Remote Procedure Call) através de XML; em outras palavras, não há transferência de arquivos envolvidos, apenas execução de algoritmos. Apesar das potencialidades, o Splay não lida nativamente com tolerância a falhas nos nodos que compõem o experimento.

O EXPO [Ruiz et al. 2013] e o EXECO [Imbert et al. 2013], por sua vez, permitem o envio de experimentos de forma eficiente, graças a algoritmos de divisão de trabalho e paralelização em topologia de árvore na comunicação entre nodos. A diferença entre ambas está na forma que experimentos são definidos e instanciados. Enquanto EXPO adota um arquivo de configuração com sua DSL (Domain Specific Language), EXECO funciona através de uma biblioteca Python, sendo possível configurar máquinas destinos programaticamente. No entanto, ambas se limitam a registrar falhas na experimentação no relatório final do experimento, não havendo mecanismos para recuperação delas.

3. Visão Conceitual do EASYEXP

Considerando a lacuna existente na literatura, e a importância e necessidade de uma arquitetura que permita a reprodutibilidade de experimentos em ambientes distribuídos de baixa confiabilidade, nesta seção descreve-se uma visão conceitual do EASYEXP.

O EASYEXP baseia-se fundamentalmente nos conceitos de *controlador* e *trabalhador*. O controlador é a entidade coordenadora dos experimentos, e reúne as funcionalidades de gerência e de interação com os trabalhadores. Um trabalhador, por sua vez, é uma abstração de entidade que de fato executa os experimentos (como um conjunto de tarefas). Um trabalhador é "interpretado" por um nodo do ambiente, cabendo ao controlador distribuir papéis de trabalhadores aos nodos, e coordenar a execução das tarefas pelos mesmos. A seguir são descritos os componentes que compõem o controlador e os trabalhadores, ambos ilustrados conceitualmente na Figura 2.

3.1. Controlador

Conforme ilustrado na Figura 2(a), a *interface web* é o ponto de entrada para que o usuário possa registrar experimentos para serem executados, e cadastrar nodos do ambiente dis-

poníveis para executá-los. Uma vez registrado um experimento e nodos do ambiente, o controlador ficará responsável por coordenar a execução distribuída dos experimentos pelos nodos. No EASYEXP, cada experimento possui um pacote com arquivos que implementam o experimento, um roteiro de tarefas (incluindo o instante em que cada uma deverá ser executada, medido a partir de $t = 0$), e um conjunto de trabalhadores que ficarão responsáveis por executá-las.

O componente *gerência de experimentos* é o principal responsável por manter os experimentos. Esse componente recupera a lista de nodos disponíveis no ambiente (devidamente registrados pelo usuário via *interface web*) e aloca aos mesmos os papéis de trabalhadores, também definidos pelo usuário autor do experimento. Uma vez alocados os papéis aos nodos, esse componente distribui tarefas do experimento entre os trabalhadores, conforme o roteiro base.

O componente *gerência de nodos* é o principal responsável por monitorar o desempenho dos nodos e a execução das tarefas alocadas pelo componente *gerência de experimentos*. Quando um nodo é adicionado via *interface web*, o módulo de *gerência de nodos* comunica-se com o mesmo (usando as credenciais fornecidas pelo usuário) para verificar se o mesmo está operacional. Quando um nodo é selecionado pelo *gerência de experimentos* para interpretar um trabalhador, esse componente comunica-se novamente com o nodo para fazer o envio do pacote do experimento e o conjunto de tarefas a ser executado pelo mesmo. Além disso, o componente *gerência de nodos* é responsável por monitorar os nodos, receber o contexto atual de cada nodo (o qual é enviado periodicamente pelos mesmos), e substituir nodos falhos e restaurar contextos nos nodos substitutos (de acordo com o trabalhador que estava sendo interpretado pelo nodo falho).

Toda a comunicação com os nodos é feita via *interface de comunicação com nodos*. Esse componente reúne as funções que os componentes de gerência de experimentos e de nodos necessitam para se conectar aos trabalhadores e acessá-los. Em geral, essas funções incluem *b*) acessar comandos *shell* remotamente (conforme a interface de execução de processos disponível no ambiente), *b*) transferir arquivos entre controlador e trabalhador, *c*) receber informação sobre o estado de conexões, e *d*) notificar o início de experimentos. Observe que as tarefas dos experimentos são definidas utilizando o instante relativo $t = 0$. Portanto, cabe ao componente de gerência de experimentos determinar o instante absoluto no qual o experimento será iniciado, e ao componente de gerência de nodos comunicar a todos os trabalhadores o início do mesmo.

3.2. Trabalhador

No EASYEXP, um experimento corresponde a um conjunto de tarefas que devem ser executadas de forma distribuída por um conjunto de *trabalhadores*, onde cada trabalhador executa uma ou mais tarefas do experimento. Cada trabalhador é "interpretado" por um nodo do ambiente, e mantém como informações de contexto: *a*) o conjunto de tarefas a ser executado e o instante de tempo para execução de cada, *b*) a tarefa atualmente em execução e *c*) o resultado de execução de cada tarefa. No evento de falha do nodo que interpreta o trabalhador, esse contexto é enviado a um nodo substituto (despachado automaticamente pelo controlador), o qual continuará a execução das tarefas.

A arquitetura dos trabalhadores é ilustrada na Figura 2(b). O componente *gerência de experimentos* é responsável por manter localmente os experimentos em execução pelo nodo, por exemplo inicializando-os ou terminando-os conforme instruções recebidas do controlador. Observe nesse caso que um mesmo nodo pode interpretar vários trabalhadores, cada um de um experimento diferente. Um experimento é colocado em execução

a partir da criação de um processo filho do trabalhador. Para cada experimento em execução, o componente *gerência de contextos* salva periodicamente o seu contexto e envia ao controlador. Esse mesmo componente também é responsável por restaurar o contexto de outro nodo, caso tenha sido selecionado para substituir um nodo falho. Por fim, o componente *daemon* é responsável por manter comunicação com o controlador (via *interface de comunicação com controlador*) e trocar mensagens periódicas de *heartbeat* com o mesmo (discutida na subseção a seguir).

3.3. Monitoração dos Trabalhadores e Manutenção de Contextos

Para que as aplicações que implementam experimentos possam tirar proveito da capacidade de recuperação de falhas do EASYEXP, faz-se necessário que as mesmas implementem um indicador de progresso de execução das tarefas, de modo que essa informação possa ser salva como parte do contexto, e assumida por um nodo substituto em caso de falhas. Para isso, o projetista do experimento deve implementar tal capacidade em cada aplicação. No EASYEXP, a funcionalidade de recuperação de contexto é disponibilizada através de uma classe chamada *snapshot*. O código que implementa o experimento deve ser desenvolvido herdando a classe original e utilizando ela e seus métodos para salvar e consultar as variáveis de contexto.

Um desafio abordado no desenvolvimento do EASYEXP é como lidar com falhas *transientes* e *permanentes*. Em resumo, falhas transientes ocorrem durante um (curto) período de tempo, após o qual o nodo retorna ao estado funcional, e geralmente são relacionadas à ruptura na comunicação entre o trabalhador e o controlador. As falhas permanentes, por sua vez, são aquelas de efeito duradouro, por exemplo término anormal da aplicação que implementa o experimento ou travamento/desligamento do próprio nodo. Os desafios, nesse caso, são (*i*) como monitorar de forma eficiente os nodos que interpretam os trabalhadores, (*ii*) como diferenciar falhas transientes de permanentes, e (*iii*) como lidar com inconsistências devido a falhas transientes nas quais o nodo desconecta-se do controlador, porém continua executando as tarefas do experimento alocadas para si.

O mecanismo empregado pelo EASYEXP para monitoração dos trabalhadores baseia-se em *heartbeats*. Enviadas periodicamente ao controlador, as mensagens de *heartbeat* informam o estado de atividade do trabalhador (contexto) e permitem que falhas transientes e permanentes sejam detectadas durante a execução do experimento. A detecção da falha baseia-se no tempo de resposta das mensagens, identificado como uma falha quando o limite estabelecido pela aplicação é ultrapassado. O *daemon* nos trabalhadores é responsável por enviar as mensagens de *heartbeat* e receber suas confirmações, enquanto que o componente de gerência de nodos do controlador é responsável pelo recebimento dessas mensagens, e a partir delas verificar o estado das aplicações. O protocolo empregado pelo EASYEXP para manter a consistência na execução do experimento durante falhas transientes consiste no trabalhador interromper a execução de suas tarefas ao não receber confirmações das mensagens de *heartbeat* enviadas. Essa abordagem é efetiva uma vez que não há motivo para manter o funcionamento de um trabalhador quando não há garantias que seu contexto ou execução estão sendo recebido pelo controlador.

Outro recurso para garantir a eficácia na execução dos experimentos é o envio, junto com a mensagem de *heartbeat*, dos recursos disponíveis no nodo trabalhador para a execução da aplicação (por exemplo espaço em disco, memória e uso médio do processador). Assim, o controlador poderá decidir, utilizando como base um script definido pelo usuário operador do experimento, se um determinado nodo é adequado ou não para interpretar um trabalhador no experimento em questão.

4. EASYEXP: Permitindo Maior Reprodutibilidade de Experimentos

Após apresentar uma visão conceitual do EASYEXP, a seguir discorre-se sobre aspectos de implementação do mesmo[1]. O EASYEXP foi implementado em Python (versão 2.7.13), principalmente pela instalação simplificada de pacotes através do PIP (o que facilita a integração com vários ambientes como GENI e PlanetLab), e a possibilidade de se unificar todos os componentes da arquitetura do EASYEXP de forma mais simples. O projeto possui atualmente 3.050 linhas de código.

As subseções a seguir descrevem em detalhes aspectos de implementação dos componentes do controlador, começando pelo armazenamento de dados na Seção 4.1, composto pelo servidor Apache ZooKeeper em conjunto com o sistema de arquivos. A Seção 4.2 descreve como o Protocolo SSH e o ZooKeeper permitem a interação entre controlador e trabalhadores. A Seção 4.3 discorre sobre os gerenciadores de experimentos e de nodos, enquanto que na Seção 4.4 apresenta-se uma visão geral da interface web. Por fim, a Seção 4.5 discorre sobre aspectos de implementação dos trabalhadores.

4.1. Armazenamento de Dados

Para armazenamento dos dados mantidos pelos componentes do controlador e dos contextos dos trabalhadores, considerou-se o Apache ZooKeeper [Hunt et al. 2010]. Entre as vantagens do ZooKeeper interessantes para o projeto, menciona-se a possibilidade de *a*) armazenar dados como nodos em uma estrutura similar a sistema de arquivos (permitindo assim manter uma organização hierárquica dos dados dos experimentos), *b*) configurar notificações sobre modificações nos dados armazenados (*watches* do ZooKeeper), e *c*) sincronizar cópias dos dados entre múltiplas instâncias de trabalhadores (nesse caso, informações de contexto, tarefas dos experimentos e resultados obtidos).

O módulo cliente do controlador serve como um intermediário entre a interface web e o *daemon*, e o Kazoo. O Kazoo é uma API que permite a interação com o servidor ZooKeeper via linguagem de programação Python. A comunicação com o servidor consiste em operações simples, como criar, recuperar e editar nodos e seus parâmetros. Com o apoio do módulo cliente, outras funções mais complexas podem ser realizadas, por exemplo adicionar os dados de um trabalhador ou experimento ao servidor. O cliente também implementa uma camada de transformação dos dados, entre o formato necessário para manter os dados dos experimentos, e o formato nativo do ZooKeeper.

4.2. Comunicação de Trabalhadores e Controlador

Os componentes de comunicação compreende quatro conjuntos de funções: *a*) notificação de eventos, *b*) gerenciamento de mensagens de *heartbeat*, *c*) envio de comandos shell remotos, e *d*) transferências de arquivos. Os dois primeiros são implementados via servidor ZooKeeper, enquanto que os dois últimos são implementados via protocolo SSH.

Para a notificação de eventos, utiliza-se a criação de nodos no ZooKeeper, aproveitando-se do fato de que modificações nos dados mantidos pelo ZooKeeper geram notificações. Nesse caso, cada trabalhador monitora pela criação de nodos com localização e nomes específicos na hierarquia de nodos mantida pelo ZooKeeper. Para a troca de mensagens de *heartbeat*, por outro lado, utiliza-se nodos efêmeros do ZooKeeper. Criados por cada um dos trabalhadores conectados ao EASYEXP, a existência de um

[1]Uma versão estável do EASYEXP está amplamente disponível para uso e pode ser obtida a partir de `https://github.com/naajuniorr/expeeker`.

nodo efêmero com uma nomenclatura específica na hierarquia de nodos mantida pelo ZooKeeper indica uma conexão ativa. Essa premissa baseia-se na exigência do ZooKeeper que seus clientes (no caso, os nodos interpretando os trabalhadores) mantenham conexão constante com o servidor. O EASYEXP mantém um parâmetro para o intervalo entre mensagens de *heartbeat* que, quando expirado, faz com que o trabalhador seja considerado desconectado. Através da interface de comunicação, o componente de gerência de nodos tem acesso ao estado atual de um trabalhador, verificando seu nodo de conexão.

Para o envio de comandos *shell* remotos, utilizou-se o protocolo SSH e o pacote Paramiko. Esse pacote disponibiliza um cliente SSH dentro da linguagem Python, com suporte a transferência de arquivos via SFTP (SSH File Transfer Protocol). Após testes, percebeu-se que a maioria das máquinas utilizadas não ofereciam suporte ao SFTP, mas apenas ao SCP (Secure Copy), onde um pacote de mesmo nome permite seu uso através do Python. Uma biblioteca intermediária Canal foi criada então para não ser necessário sempre instanciar ambos clientes SSH e SCP na implementação, pois o uso de um normalmente está relacionado com o outro.

4.3. Gerência de Experimentos e de Nodos

Tanto o controlador quanto os trabalhadores mantêm *threads* para gerenciamento de nodos e de trabalhadores (o que inclui adição, substituição e remoção de nodos que interpretam os mesmos), e processamento de mensagens de *heartbeat*. Para isso, a verificação será realizada periodicamente por um processo filho do principal, que ficará responsável por conectar-se aos trabalhadores.

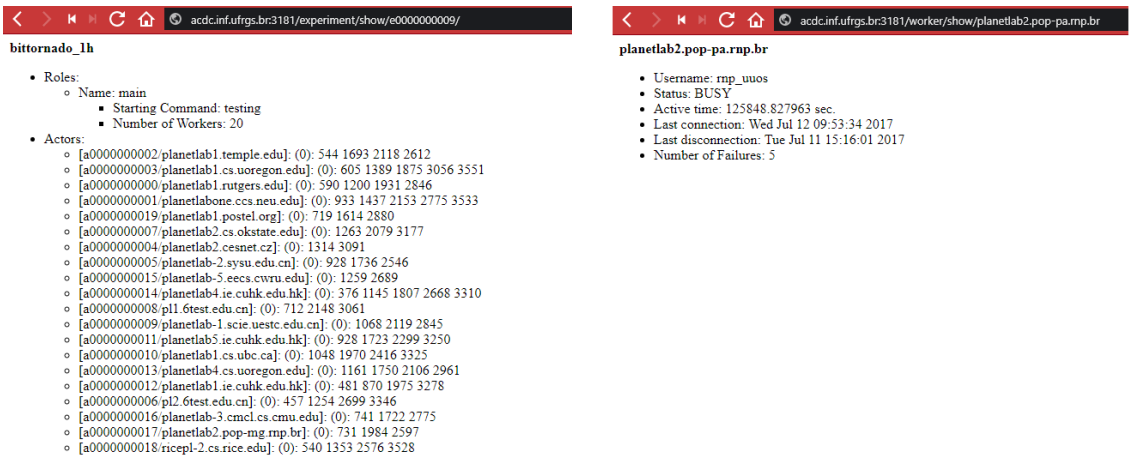
Para gerenciar as operações com trabalhadores, é utilizada uma fila de tarefas mantida no ZooKeeper, onde cada tarefa é identificada por um nodo sequencial, definindo sua ordem de inserção e permitindo seguir o conceito FIFO com a ordenação dos nodos pelo seu nome. A tarefa irá representar as ações que o trabalhador deve realizar. Assim que finalizada, o nodo da respectiva tarefa é removido da fila e a próxima é realizada. ZooKeeper *watches* são usados para notificar a chegada de novas tarefas. Enquanto não houver tarefas, essa *thread* de execução se manterá a espera de notificações.

O controlador aceita três tarefas: *a*) instalação de trabalhador, que engloba a instalação do *software* de controle necessário para manter a conexão com sua máquina e suas dependências, *b*) envio e execução de experimento, no qual é necessário alocar o número de trabalhadores necessários, transferir o arquivo do experimento e iniciar sua execução, e *c*) recuperação de trabalhadores perdidos, na qual uma tentativa de reconexão e reativação é feita, e em caso de falha, um novo trabalhador será alocado para substituir execução na máquina perdida, realizando as mesmas operações da tarefa *b*) descrita.

4.4. Interface Web

Para implementar a interface web do EASYEXP foi utilizado o *framework* Django, com o objetivo de facilitar o acesso às funcionalidades oferecidas pelos demais módulos bem como para recuperação de dados mantidos pelo sistema. Para prova de conceito do EASYEXP, foram implementadas funcionalidades para adicionar, listar e consultar dados de trabalhadores e experimentos.

A Figura 3(a) representa a janela de consulta de um experimento, onde é descrito quais papéis foram declarados e as saídas de cada trabalhador que o executou. A Figura 3(b), por sua vez, representa a janela de consulta de trabalhador, onde dados como seu estado, tempo ativo, data da última conexão e desconexão com o sistema e número de falhas ocorridas são apresentadas.



(a) Experimento (b) trabalhador

Figura 3. Janelas contendo detalhes de um experimento, incluindo os nodos trabalhadores participantes (esq.) e de um trabalhador específico (dir.).

4.5. Trabalhadores

Assim como o controlador, o módulo de *software* que executa nos trabalhadores é implementado em Python. O pacote Kazoo é utilizado para implementar a comunicação com o controlador, via estabelecimento de conexão com o ZooKeeper executando no mesmo. O *daemon* no trabalhador utiliza *watches* para verificar a designação de novos experimentos, feitas durante a alocação do trabalhador. Quando notificado, os dados e arquivos do experimento são carregados e preparados para execução. Assim, um segundo *watch* é ativado para verificar o início do experimento. Quando recebido, o experimento é executado. O *daemon* no trabalhador mantem sua execução até que seja terminado, e sua saída é retornada ao controlador. Erros de execução no experimento não são tratados, apenas reportados ao controlador como erro gerado. Como não é possível averiguar se a origem do erro é de código mal implementado ou falha do sistema, optou-se por uma implementação mais simples, na qual delega-se ao operador do experimento analisar a saída e tomar as devidas providências.

Um nodo é considerado ocupado quando um experimento estiver em execução neles, e é função do *daemon* indicar ao controlador seu estado atual. Seu estado é representado a partir do nodo efêmero que representa sua conexão no ZooKeeper. Dependendo onde esse nodo for criado, ele indicará o estado do trabalhador: se for filho dos nodos ociosos, é um trabalhador ocioso, se for filho de um nodo ocupado, é um trabalhador ocupado, e caso não esteja em nenhum dos dois, é um trabalhador desconectado. Assim, o controlador consegue determinar com facilidade quais trabalhadores estão disponíveis para alocação ou não, apenas recuperando os filhos do nodo ocioso. Logo, o trabalhador deve ser capaz de modificar seu estado atual de acordo com a chegada e finalização de experimentos, removendo o nodo do estado anterior e instanciando o novo.

5. Avaliação Experimental

Para avaliar a eficácia e a efetividade do EASYEXP, um conjunto de experimentos genéricos foram executados no PlanetLab, ambiente particularmente reconhecido pela instabilidade na disponibilidade dos nodos que o compõe. A Subseção 5.1 descreve o

cenário de avaliação, enquanto que as Subseções 5.2 e 5.3 apresentam os resultados obtidos com baterias de experimentos de 2 e 24 horas, respectivamente.

5.1. Descrição do Ambiente e Experimento

A avaliação considera uma aplicação cliente-servidor na qual um servidor HTTP recebe requisições POST de clientes distribuídos geograficamente. O arquivo com as requisições é registrado no EASYEXP, que o processa para definir o conjunto de tarefas (requisições) que deverá ser executado por cada trabalhador. Para aferir a eficácia do EASYEXP, falhas são injetadas nos clientes para emular instabilidade no ambiente de execução. A falha consiste em enviar ao processo executando no cliente um sinal de término de processo (SIGTERM em sistemas UNIX). Observe que, além das falhas injetadas, falhas naturais do ambiente de execução (por exemplo desconexão do nodo ou término anormal do processo) também podem ocorrer.

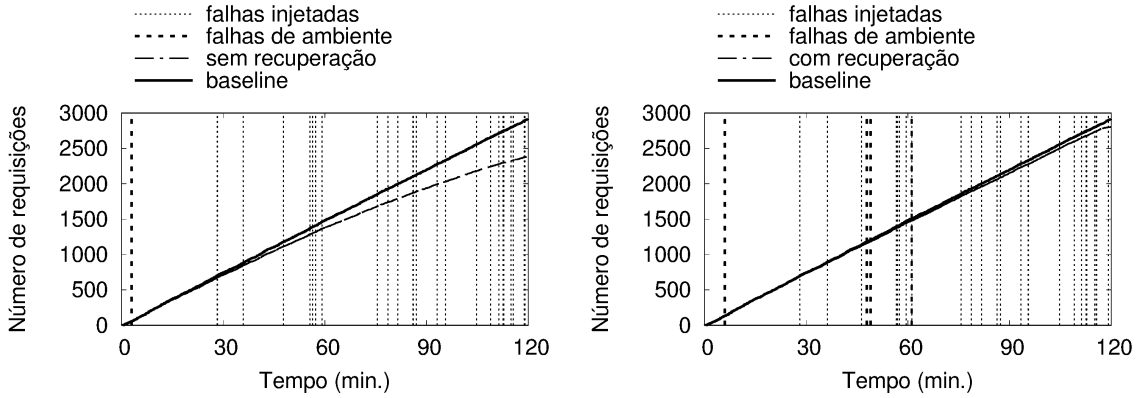
Os nodos empregados no experimento possuem sistema operacional Linux Fedora 8, com casos de Linux Fedora 12 e Linux Centos 6, a maioria em versões de 32-bits. Os processadores disponíveis nos nodos são (em ordem decrescente de frequência) Intel Xeon, Intel Core 2 Duo, Intel Pentium D, Intel Core 2 Quad, Intel i5 e AMD Opteron. A memória RAM disponível variou entre 1GB e 16GB, com moda em 4GB. Para o controlador, foi usado um Dell PowerEdge R815 quatro processadores AMD Opteron 6276, totalizando 64 núcleos a 2.3 GHz, e 64GB de RAM. Seu sistema operacional é o Linux Ubuntu 16.04.2 LTS com kernel 4.4.0-66-generic. Os *apps* utilizados foram Kazoo 2.4, Paramiko 2.2.1, SCP 0.10.2, Django 1.10.5, PIP 9.0.1 e Apache ZooKeeper 3.4.9.

Os parâmetros da avaliação foram definidos como segue. Em ambas as baterias de experimentos, foram utilizados 70 nodos do ambiente PlanetLab, sendo 50 para interpretar os clientes (trabalhadores), e outras 20 disponíveis para substituição em caso de falhas. Os intervalos entre mensagens de *heartbeat*, entre salvamento de contextos, e de verificação do estado dos trabalhadores pelo controlador, foram definidos como 30 segundos cada um. O roteiro de experimento de 2 horas compreende 2.915 requisições; o número de requisições por cliente e o intervalo entre requisições foram distribuídos uniformemente. A injeção de falhas é distribuída aleatoriamente, com um total de 24 falhas cobrindo 48% dos 50 nodos do experimento. O roteiro de 24 horas, por sua vez, possui 47.114 requisições distribuídas seguindo uma distribuição gaussiana, com centros em 6 e 18 horas. Essa distribuição foi escolhida para simular dois momentos distintos de picos de requisições dos clientes. A injeção de falhas segue uma gaussiana com centro em 12 horas, sendo injetadas 99 falhas com uma cobertura de 96% das máquinas.

5.2. Resultados da Bateria de 2 Horas

A Figura 4 apresenta os resultados obtidos para execuções sem e com recuperação de falhas. As curvas "baseline" incluídas nos gráficos denotam o comportamento do experimento em um cenário teórico ideal de latência desprezível e sem ocorrência de falhas. Além disso, as linhas verticais denotam o instante de ocorrência de uma falha (que pode ser injetada ou ocorrida naturalmente) em algum nodo participante do experimento. Observe na Figura 4(a) que na medida em que as falhas vão ocorrendo, ocorre uma degradação no número de requisições completadas, e por consequência a curva "sem recuperação" vai se distanciando da "baseline". Em números, de 2.915 requisições esperadas no período do experimento, foram completadas 2.380 (82%). No período foram registradas 25 falhas, sendo 24 injetadas e 1 natural.

O uso do EASYEXP melhora a qualidade dos resultados do experimento de forma substancial. Observe na Figura 4(b) que a diferença entre as curvas "com recuperação" e



(a) Sem recuperação (b) Com recuperação

Figura 4. Avaliação da aplicação em ambientes sem e com recuperação de falhas, baseado em requisições distribuídas ao longo de 2 horas.

"baseline" é praticamente desprezível; 2.803 requisições foram completadas com sucesso, ou 96% do esperado. No período, 28 falhas ocorreram, sendo 4 naturais (uma falha não pode ser injetada devido à ocorrência de uma falha natural pouco antes). Ainda assim, o experimento foi concluído com todos os 50 clientes operacionais.

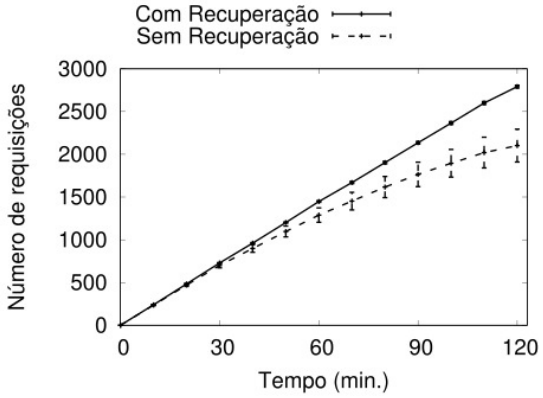


Figura 5. Média dos resultados de 30 avaliações.

Foram realizadas ainda 30 execuções do experimento sem e com recuperação, e calculadas a média das requisições completadas com sucesso e intervalo de confiança com precisão de 95% para intervalos de 10 minutos. A Figura 5 apresenta um resumo dos resultados. Observe que a curva "com recuperação" apresenta intervalos de confiança menores que a curva "sem recuperação". Isso evidencia que a variabilidade nos resultados obtidos em cada experimento é menor nos cenários em que o EASYEXP é utilizado.

A Tabela 1 apresenta uma visão mais detalhada dos resultados obtidos nas 30 execuções de experimentos. Note que intervalo de confiança no instante de 120 minutos dos experimentos com recuperação é uma variação de apenas 16 requisições, enquanto que o para sem recuperação, esse intervalo aumenta para uma variação de 192 requisições. A consistência dos resultados obtidos com recuperação mostra a eficácia do EASYEXP em permitir a maior reprodutibilidade de experimentos realizados em ambientes pouco confiáveis como o PlanetLab.

Tabela 1. Média e intervalos de confiança de 30 avaliações em ambientes com e sem recuperação de falhas, detalhados em janelas de 10 minutos.

Tempo (min.)	Sem Recuperação			Com Recuperação		
	Média (μ)	$\mu - 1.96(\sigma/\sqrt{n})$	$\mu + 1.96(\sigma/\sqrt{n})$	Média (μ)	$\mu - 1.96(\sigma/\sqrt{n})$	$\mu + 1.96(\sigma/\sqrt{n})$
10	236	229.5	242.4	239	235	242.9
20	470	454.7	485.2	484	480	487.9
30	700	671.5	728.4	729	723.1	734.8
40	898	853.3	942.6	961	954.9	967
50	1097	1033.6	1160.3	1202	1194.1	1209.8
60	1289	1205.1	1372.8	1448	1437.6	1458.3
70	1452	1348.7	1555.2	1670	1656.4	1683.5
80	1617	1493.4	1740.5	1903	1887.8	1918.1
90	1765	1621.7	1908.3	2135	2119.8	2150.1
100	1895	1733.6	2056.3	2365	2348.1	2381.8
110	2020	1840.4	2199.5	2600	2582.4	2617.5
120	2102	1910	2199.5	2792	2775.8	2808.1

Tabela 2. Número de falhas injetadas e de ambiente detectadas durante as 30 avaliações em um ambiente com recuperação.

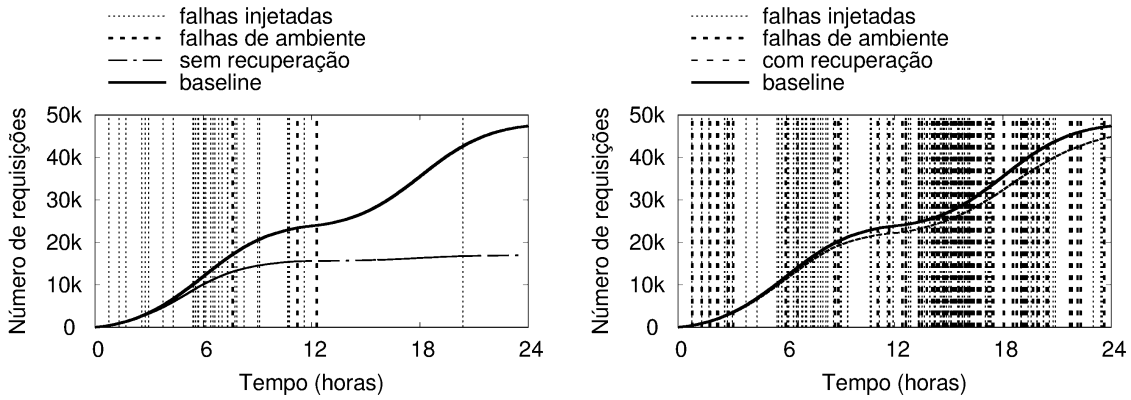
Tempo (min.)	Falhas Injetadas				Falhas de Ambiente			
	Média (μ)	Desvio Padrão (σ)	Mínimo	Máximo	Média (μ)	Desvio Padrão (σ)	Mínimo	Máximo
10	1	0	1	2	3	6.9	0	34
20	2	0	2	3	4	6.8	0	34
30	5	0	4	6	7	9.1	0	34
40	6	0	6	8	8	11.7	0	52
50	7	0	7	9	10	12.5	0	52
60	12	1	10	14	12	15	0	60
70	14	0	13	16	14	16.4	0	60
80	15	0	14	17	14	16.5	0	60
90	20	1.4	16	23	14	16.8	0	60
100	21	1.4	16	24	14	16.9	0	60
110	21	1.4	16	24	14	17	0	60
120	21	1.4	16	24	14	17.1	0	60

A Tabela 2 apresenta uma visão de sobre falhas ocorridas durante a experimentação, medidas de forma acumulada a cada intervalo de 10 minutos. As falhas são classificadas entre injetadas (aquelas inseridas para avaliar a robustez da solução proposta) e falhas de ambiente (aquelas que ocorrem naturalmente, e que são o foco primário da pesquisa reportada no presente artigo). Note que enquanto um dos experimentos não teve qualquer falha de ambiente, outro teve até 60 falhas, o que mostra a natureza imprevisível e aleatória das mesmas. Quanto às falhas injetadas, algumas delas não puderam sê-lo devido a alguma falha de ambiente que já havia ocorrido no nodo no qual a falha deveria ter sido injetada.

5.3. Resultados da Bateria de 24 Horas

A Figura 6(a) apresenta o resultado da execução em um ambiente sem recuperação de trabalhadores. Com o número maior de falhas concentrados mais no centro da execução, ao invés de ao longo de toda ela como nos resultados anteriores, é possível ver o seu efeito no número de requisições, onde seu crescimento para a partir do ponto em que a maioria dos trabalhadores pararam de funcionar. Inclusive é possível ver isso com o número reduzido de falhas após a metade do experimento. Das 47.114 requisições esperadas, apenas 16.951 foram recebidas, 35.7% do total. Das 48 falhas totais, 15 foram causadas pelo ambiente, logo, apenas 2 trabalhadores terminaram a execução.

A Figura 6(b) apresenta os resultados obtidos utilizando a recuperação de trabalhadores. Os resultados indicam a eficácia do sistema, onde mesmo sob 333 falhas, onde



(a) Sem recuperação (b) Com recuperação

Figura 6. Avaliação a aplicação em ambientes sem e com recuperação de falhas, baseado em requisições distribuídas ao longo de 24 horas.

239 são do ambiente, foi capaz de realizar 44.863 das requisições do roteiro, 94.6% do total. Nem todas as máquinas foram capazes de finalizar a execução, 6 não foram corretamente recuperadas, indicando que o protótipo ainda possui problemas a serem corrigidos. Nota-se o número excessivo de falhas do ambiente, comprovando a instabilidade do PlantetLab, aproximadamente 10 falhas por hora.

6. Considerações Finais

A experimentação de novas soluções em ambientes distribuídos é particularmente desafiadora, principalmente quando o ambiente não oferece garantias mínimas de estabilidade dos nodos que o compõem. Apesar das soluções existentes para facilitar a execução de experimentos, nenhuma oferece suporte nativo à recuperação de falhas que possam impactar na qualidade dos resultados. Para superar essa lacuna, neste artigo propõe-se EASYEXP, uma arquitetura capaz de recuperar trabalhadores falhos e armazenar seus contextos, garantindo maiores possibilidades de reprodutibilidade de experimentos. Os resultados obtidos foram extremamente satisfatórios, e evidenciam a capacidade do EASYEXP de permitir a reprodução de experimentos, dados os recursos necessários.

Para tirar proveito do EASYEXP, a aplicação deve ser necessariamente distribuída, com clientes operando de forma independente, com um roteiro bem definido de eventos, e projetada de modo que o contexto de cada entidade participante possa ser salvo periodicamente. Como mencionado anteriormente, tal pode ser feito pelo operador da aplicação estendendo a estrutura de dados “snapshot”. Exemplos de aplicações incluem aplicações *peer-to-peer* em geral, como *live streaming* e de distribuição de conteúdo.

É importante destacar que, tal como proposto, o EASYEXP não é capaz de fornecer um ambiente totalmente tolerante a falhas, ou compatível com os mais diversos experimentos em ambientes distribuídos. Além disso, o EASYEXP requer que o controlador se mantenha ativo em todos os momentos, por ser o ponto central da arquitetura. Quando qualquer tipo de falha atingir este componente, todos os trabalhadores conectados a ele serão desativados. Uma maior disponibilidade do controlador pode ser alcançada através da sua replicação em diversas máquinas, obrigando que todos os dados também sejam replicados constantemente. Esse aspecto é deixado como direção de trabalhos futuros.

Agradecimentos

A pesquisa reportada no artigo foi financiada pelo Projeto 462091/2014-7 concedido pelo CNPq (Edital MCTI/CNPQ/Universal 14/2014).

Referências

Albrecht, J. and Huang, D. Y. (2010). Managing distributed applications using gush. In *Int'l Conference on Testbeds and Research Infrastructures*, pages 401–411. Springer.

Albrecht, J. R., Braud, R., Dao, D., Topilski, N., Tuttle, C., Snoeren, A. C., and Vahdat, A. (2007). Remote control: Distributed application configuration, management, and visualization with plush. In *Large Installation System Administration (LISA)*, volume 7, pages 1–19.

Bajpai, V., Kühlewind, M., Ott, J., Schönwälder, J., Sperotto, A., and Trammell, B. (2017). Challenges with reproducibility. In *SIGCOMM Reproducibility Workshop*, Reproducibility '17, pages 1–4, New York, NY, USA. ACM.

Baker, M. (2016). 1,500 scientists lift the lid on reproducibility. *Nature*, 533(7604):452–454.

Bonaventure, O., Iannone, L., and Saucez, D. (2017). *Proceedings of the ACM SIGCOMM Reproducibility Workshop*. ACM, New York, NY, USA.

Costa, L. L., Bona, L. C., and Duarte Jr, E. P. (2015). Melhorando a precisão e repetibilidade de experimentos no planetlab. In *Simpósio Brasileiro de Redes de Computadores e de Sistemas Distribuídos (SBRC 2015)*. SBC.

Garrett, T., Bona, L. C., and Duarte Jr, E. P. (2017). Improving the performance and reproducibility of experiments on large-scale testbeds with k-cores. *Computer Communications*.

Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. (2010). Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX Annual Technical Conference*, volume 8, page 9. Boston, MA, USA.

Imbert, M., Pouilloux, L., Rouzaud-Cornabas, J., Lèbre, A., and Hirofuchi, T. (2013). Using the execo toolkit to perform automatic and reproducible cloud experiments. In *Int'l Conference on Cloud Computing Technology and Science (CloudCom 2013)*, volume 2, pages 158–163. IEEE.

Leonini, L., Rivière, É., and Felber, P. (2009). Splay: Distributed systems evaluation made simple (or how to turn ideas into live systems in a breeze). In *Networked Systems Design and Implementation (NSDI)*, volume 9, pages 185–198.

Nussbaum, L. (2017). Testbeds support for reproducible research. In *SIGCOMM Reproducibility Workshop*, Reproducibility '17, pages 24–26, New York, NY, USA. ACM.

Ruiz, C. C., Richard, O. A., Iegorov, O., and Videau, B. (2013). Managing large scale experiments in distributed testbeds. In *Int'l Association of Science and Technology for Development (IASTED)*, pages 628–636.

Santos, M., Fernandes, S., and Kamienski, C. (2014). Conducting network research in large-scale platforms: Avoiding pitfalls in planetlab. In *Advanced Information Networking and Applications (AINA)*, pages 525–532. IEEE.