Apoio à Utilização de Análise de Dados em Aplicações CSE por meio de Contêineres *

Liliane Neves[1], Débora Pina[1], Daniel de Oliveira[2], Marta Mattoso[1]

[1]COPPE – Universidade Federal do Rio de Janeiro (UFRJ)

[2]Instituto de Computação – Universidade Federal Fluminense (IC/UFF)

{lneves,dbpina, marta}@cos.ufrj.br, danielcmo@ic.uff.br

Abstract. *Large-scale scientific applications are characterized by invoking several software libraries and producing large amounts of data through scripts. Container-based virtualization is a way that is not widely adopted by the HPC community to support the portability of applications, especially approaches for data analysis. To assist the adoption of containers, this article presents ProvDeploy with a model aimed at incorporating services into applications developed for PAD environments.*

Resumo. *Aplicações científicas em larga escala se caracterizam por invocar diversas bibliotecas de software e produzir grandes quantidades de dados por meio de scripts. A virtualização baseada em contêineres é uma maneira ainda pouco utilizada pela comunidade de HPC para auxiliar a portabilidade dessas aplicações, em especial aquelas que possuem abordagens para análise de dados acopladas. Para auxiliar a adoção de tecnologia de contêineres esse artigo apresenta a* $ProvDeploy$ *com um modelo voltado a incorporar serviços a aplicações desenvolvidas para ambientes PAD.*

1. Introdução

Aplicações de Ciência Computacional e Engenharia (*Computational Science and Engineering* - CSE) comumente consomem e produzem uma quantidade maciça de dados e se utilizam de uma grande quantidade de ferramentas, bibliotecas e serviços de apoio. Um exemplo é a análise de dados científicos para a sintonia fina de parâmetros e configuração dessas aplicações que pode ser enriquecida por meio da coleta de dados de proveniência (dados históricos de execuções passadas das aplicações) [Davidson and Freire 2008]. Sendo assim, além de invocar bibliotecas com *softwares* de terceiros para métodos numéricos, visualização, perfilagem de código e monitoramento, torna-se necessário invocar sistemas de coleta de dados de proveniência e de análise de dados.

Tais componentes para coleta e análise de dados podem ter dependências em relação à máquinas virtuais Java, bibliotecas de compressão de dados, *etc*. Tomemos como exemplo abordagens que necessitam de um Sistema de Gerência de Banco de Dados (SGBD), como serviços de proveniência, para armazenar dados e prover consultas. Tal SGBD é responsável por armazenar os dados, e por permitir ao usuário elaborar consultas (*e.g.,* em SQL) para realizar suas análises durante a execução da simulação. Em muitos casos, o processo de adição de chamadas a bibliotecas de *software* à pilha de componentes *software* da aplicação CSE do usuário poder ser um processo longo, laborioso e propenso à erros, se executado manualmente.

Os ambientes de processamento de alto desempenho (PAD) possuem uma arquitetura sofisticada com objetivo de atender a múltiplos usuários, ofertando alto desempenho e vazão

*O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001, CNPq e FAPERJ.

à execução de aplicações, ao mesmo tempo que precisam garantir segurança e estabilidade de todo o ambiente [Vahi et al. 2019]. Com isto, a gerência de aplicações não é simples e não se dá de maneira similar à gerência em computadores pessoais em que o usuário possui todos os privilégios. As aplicações CSE crescem em complexidade e em tecnologias (*e.g.,* bibliotecas e componentes que visam a otimizar seu desempenho), o que gera a demanda de ambientes cada vez mais customizáveis [Vahi et al. 2019, Kurtzer et al. 2017]. Nos últimos anos, com vistas a facilitar o desenvolvimento e implantação de sistemas, a tecnologia de contêineres começou a ser amplamente utilizada [Preeth et al. 2015]. A portabilidade oferecida pela conteinerização se mostra cada vez mais necessária aos ambientes de PAD. Apesar de existirem soluções de conteinerização voltadas para PAD [Kurtzer et al. 2017, Gerhardt et al. 2017, Priedhorsky and Randles 2017] e para facilitar o seu uso [Vahi et al. 2019], os contêineres ainda são pouco utilizados pelo usuário de PAD [McMillan 2018]. Uma das principais razões para a dificuldade nessa adoção é que mesmo com contêineres, ainda é complexo realizar a configuração das múltiplas bibliotecas necessárias ao ambiente PAD e os problemas de configuração do ambiente podem ser similares dentro do do contêiner quando se tem pilhas de *software* muito complexas. Além disso, o tempo necessário para executar tal tarefa pode ser significativo [McMillan 2018]. Dessa forma, é desejável que exista uma abordagem que auxilie a adição de serviços à pilha de *software* complexas, diminuindo a complexidade da fase de configuração e preparação do ambiente

Observado isto, este artigo apresenta a ferramenta $ProvDeploy$, que tem como objetivo auxiliar a organização de contêineres com acesso a dados de SGBDs em aplicações de PAD. A $ProvDeploy$ adota uma organização especifica para seus contêineres com diferentes estruturas para acesso e persistência de dados e execução de aplicações para facilitar o uso de múltiplas bibliotecas, em especial bibliotecas de coleta de dados de proveniência com armazenamento em SGBDs. A avaliação se dá a partir de um modelo de custos baseado no total de passos para configuração da pilha de *software* de uma aplicação.

2. Abordagem Proposta: a $ProvDeploy$

Com o objetivo de atenuar os problemas relativos à configuração e execução quando se deseja integrar diferentes pilhas de *software*, propõe-se que seja utilizada uma abordagem com uso de múltiplos contêineres, a $ProvDeploy$. Objetiva-se reutilizar as imagens presentes nos diferentes registros públicos (DockerHub, Singularity Hub). A $ProvDeploy$ pode ser descrita como um serviço de apoio à adoção automatizada de abordagens e bibliotecas para análise de dados, *e.g.,* coleta de dados de proveniência, com múltiplos contêineres. A $ProvDeploy$ disponibiliza o coletor de dados desejado pelo usuário e por meio do uso de contêineres executa a aplicação principal em conjunto com o sistema de coleta de dados. A $ProvDeploy$ utiliza imagens preparadas para execução de diferentes aplicações com coleta dados de proveniência. Esta coleta é incorporada de modo sistemático e buscando aproveitar imagens de contêineres presentes nos registros públicos para diminuir o tempo para criação de contêineres.

A arquitetura da $ProvDeploy$ é composta de quatro componentes principais (conforme apresentado na Figura 1): o *Catálogo*, o *Instrumentador*, o *Deployer* e o *Inicializador* (um *script* de inicialização dos componentes). O *Catálogo* armazena as referências para diferentes coletores de dados de proveniência, ferramentas de armazenamento e ambientes de execução conteinerizados. No *Catálogo* também ficam armazenadas as informações relativas ao versionamento de arquivos de configuração de contêineres. No *Catálogo*, as imagens podem ser cadastradas pelo usuário e podem ser locais ou remotas. A $ProvDeploy$ permite ao usuário consultar as imagens existentes no *Catálogo* e solicitar uma execução utilizando um valor mnemônico na chamada do *script* de inicialização. O *Instrumentador* tem papel de identificar

dentro do *script* pontos de interesse do usuário e adicionar as anotações para coleta de dados de proveniência, quando necessário. O *Deployer* invoca o modelo de custo (detalhado em [Neves 2020]) e apresentar ao usuário o cenário com menor custo (consideramos como custo o número de passos necessários para configurar a aplicação CSE e todas suas dependências). O *Inicializador* é responsável por conectar os componentes da $ProvDeploy$, este tem por papel acessar o catálogo de modo a consultar e adicionar imagens, inicializar a ferramenta de coleta de dados de proveniência, prover acesso aos dados, enviar ao *Deployer* os dados necessários para acesso remoto e quais arquivos serão copiados ou transferidos.

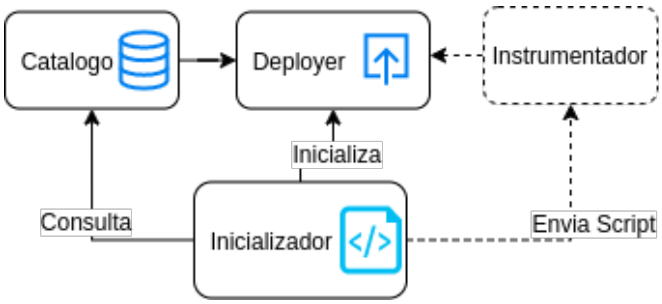

Figura 1. Componentes da $ProvDeploy$

A $ProvDeploy$ trata os contêineres de coletores de dados de proveniência como contêineres que encapsulam toda aplicação junto com suas dependências, apenas fazendo chamadas para sua execução. Apesar da possibilidade de existência de mais de um serviço de coleta de dados de proveniência no *Catálogo*, apenas um destes pode ser definido como padrão. O serviço de coleta padrão será executado pela $ProvDeploy$ automaticamente, caso seja o desejo do usuário. Os contêineres de armazenamento devem conter aplicações voltadas para realizar leitura e escrita como SGBDs e ferramentas de otimização destes processos como o FastBit [Wu et al. 2009]. Para execução, a $ProvDeploy$ trabalha com contêineres que encapsulam um grupo de configurações que pode ser utilizado para execução de diferentes aplicações, onde o *script* funciona como uma entrada para o contêiner. Com relação aos contêineres voltados exclusivamente à execução de um serviço de coleta de proveniência ou banco de dados, tudo necessário é encapsulado no contêiner e para inserir modificações, é necessário acesso direto em modo escrita.

Tomemos por exemplo de utilização do modelo, uma rede neural implementada com Tensorflow 1.14 e executada no *cluster* Lobo Carneiro[1]. Implementado em Python, o Tensorflow requer um grupo de bibliotecas Python muito atual. Além do Tensorflow, a implementação utilizada faz uso do módulo *numpy*, do *scikit-learn* para divisão dos conjuntos de treino e teste e da biblioteca TfLearn para otimização de desempenho. A imagem do catálogo da $ProvDeploy$ que foi utilizada para execução não contêm apenas as bibliotecas do Tensorflow, mas também bibliotecas utilizadas em aprendizado de máquina para manipular, analisar e visualizar dados como Pandas e MatPlotLib, bibliotecas científicas e matemáticas como *Scipy* e *Numpy* e outras bibliotecas muito utilizadas em aprendizado de máquina como *scikit-learn*. Deste modo, caso o usuário faça modificações ao seu *script*, a mesma imagem pode ser utilizada.

Utilizando o modelo de custos, temos que o conjunto de $passos$ para executar esta aplicação com coleta de proveniência é dado pelo conjunto de dependências *D*={*Pyenv, pyenv-virtualenv, Python 3.6, Tensorflow, scikit-learn, Numpy, dfa-lib-python, DfAnalyzer, MonetDB,*

[1] O LoboC é um *cluster* SGI ICE-X com 504 CPUs Intel Xeon E5-2670v3 (Haswell), totalizando 6.048 processadores. Cada nó computacional contém 24 processadores (adicional de mais 24 processadores com Hyper-Threading) com 64GB de memória RAM. Os nós computacionais são interconectados com a tecnologia InfiniBand FDR –56 Gbs (Hypercube)

Tabela 1. Custo estimado para configurar um ambiente com e sem a ProvDeploy

Sem ProvDeploy		Com ProvDeploy
– PyEnv (≈ 195s) • clonar o repositório (≈ 150s) • definir (2) variáveis de ambiente (≈ 10s) • adicionar os comandos de inicialização ao *shell script*(≈ 30s) • reiniciar o *shell* (≈ 5s) – PyEnv-virtualenv (≈ 185s) • clonar o repositório (≈ 150s) • adicionar os comandos de inicialização ao *shell script* (≈ 30s) • reiniciar o *shell* (≈ 5s) – Python 3.6 (≈ 185s) • baixar a versão Python com Pyenv (≈ 180s) • criar o ambiente Python com virtualenv (≈ 5s) – dfa-lib-py (≈ 125s) • carregar o ambiente Python criado no passo anterior (≈ 5s) • instalar coverage(≈ 30s), • instalar pytest (≈ 30s) • executar script de instalação da dfa-lib-py(≈ 60s) – DfAnalyzer (≈ 65s) • baixar (≈ 30s); • ativar módulo Java (≈ 5s); • modificar dfa.properties (≈ 30s).	– MonetDB (≈ 915s) • definir (6) variáveis de ambiente (≈ 30s); • criar o diretório do MonetDB (≈ 5s); • baixar com *wget*(≈ 300s); • descompactar (≈ 120s); • criar um diretório para compilação (≈ 5s); • ativar modulo gcc(≈ 5s); • executar o *./configure* (≈ 120s); • executar *make*(≈ 300s); • verificar erros (≈ 10s); • executar *make install* (≈ 300s); • adicionar os caminhos do MonetDB às variáveis de ambiente no arquivo de bash do usuário (≈ 30s). – fastbit (≈ 1560s) • wget no arquivo (≈ 180s) • executar *./configure* (≈ 600s) • executar *make* (≈ 600s) • verificar a instalação com *make test* (≈ 180s) – demais bibliotecas(Tensorflow, scikit, TfLearn)(≈ 660 s) • instalar Tensorflow (≈ 300s) • instalar scikit-learn (≈ 240s) • instalar TfLearn (≈ 120s)	– PyEnv (≈ 195s) – PyEnv-virtualenv (≈ 185s) – Python 2.7+ (≈ 215) • baixar a versão python com Pyenv (≈ 180s); • criar o ambiente Python com virtualenv (≈ 5s); • instalar Paramiko(≈ 30s). – Contêiner de execução (≈ 970): • selecionar imagem de registro público (≈ 60s); • escrever arquivo de definição (≈ 120s); • compilar arquivo de definição(≈ 310s); • adicionar ao catálogo da $ProvDeploy$(≈ 5s).

FastBit$\}$. Utilizando Pyenv, os passos para instalação do Tensorflow, scikit-learn são agrupados em um único passo, onde a instalação do scikit-learn já engloba a Numpy. As ações de instalação do Tensorflow e do scikit-learn são demorados devido ao grande número de bibliotecas que trazem consigo. Cada aplicação listada representa um passo no modelo de custos da $ProvDeploy$. O conjunto de ações e os tempos estimados associados a cada passo e ação são apresentados na coluna **sem ProvDeploy** da Tabela 1.

Observando que o usuário executa os todos passos de instalação em um mesmo período de tempo, passos como instalação do Tensorflow, Numpy e scikit-learn são considerados ações atômicas pois o ambiente Python já se encontra ativado e atualizado. O tempo em segundos para execução de cada passo é dado pelo conjunto $C(D) = \{195, 185, 185, 125, 65, 915, 1560, 660\}$ e o custo $C(D)$ de cumprir essas dependências convertido em minutos é 64,8 minutos.

Para executar esta mesma aplicação com coleta de proveniência com uso da $ProvDeploy$ o conjunto de $passos$ para execução é dado pelo seguinte conjunto de dependências $D_{ProvDeploy}$=$\{$ *PyEnv, Pyenv-Virtualenv, Python 2.7+, Paramiko, Singularity, Contêiner de execução*$\}$. O Singularity é uma dependência já instalada no Loboc e por isso, esta dependência não é contabilizada no custo total. O Paramiko é uma biblioteca Python, utilizada pela $ProvDeploy$ para realizar conexões remotas e assim realizar transferência de arquivos caso o usuário queira realizar uma execução remota. Nesse conjunto de passos o tempo de compilação do arquivo de definição das imagens de contêineres para execução da ferramenta de coleta de proveniência é desconsiderado, por que esses já são disponibilizados pela $ProvDeploy$, bastando a imagem de contêiner para execução da rede neural. Nesse caso,

o indicado é que seja utilizada a imagem oficial do Tensorflow como imagem base, pois isto é indicado pela documentação[2] e adicionar via arquivo de definição a *dfa-lib-py*. O conjunto de ações e os tempos estimados associados a cada passo e ação são apresentados na coluna **com ProvDeploy** da Tabela 1.

Estimando o custo da ação de compilação das imagens contêineres como o custo acumulado das diferentes dependências instaladas no contêiner, temos que o tempo em segundos de execução de cada passo com a $ProvDeploy$ é dado pelo conjunto $C(D_{ProvDeploy}) = \{195, 185, 215, 970\}$ e o custo $C(D_{ProvDeploy})$ de cumprir essas dependências convertido em minutos é 26,08 minutos.

3. Conclusões

Aplicações CSE se caracterizam pelo grande volume de dados gerados e por invocarem muitos componentes externos, que integram sua pilha de *software*. Como parte da solução para incentivar o uso de contêineres na gerencia de dados foi proposta a $ProvDeploy$, que possui um modelo de custos próprio para analisar as vantagens e desvantagens do ponto de vista de ganho de tempo e diminuição de passos para o usuário cumprir uma configuração. O exemplo apresentado demonstra ganho no tempo de configuração, o que é vantajoso para pilhas de *software* complexas.

Referências

Davidson, S. B. and Freire, J. (2008). Provenance and scientific workflows: challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1345–1350.

Gerhardt, L., Bhimji, W., Fasel, M., Porter, J., Mustafa, M., Jacobsen, D., Tsulaia, V., and Canon, S. (2017). Shifter: Containers for hpc. In *J. Phys. Conf. Ser.*, volume 898, page 082021.

Kurtzer, G. M., Sochat, V., and Bauer, M. W. (2017). Singularity: Scientific containers for mobility of compute. *PloS one*, 12(5).

McMillan, S. (2018). Making containers easier with hpc container maker. In *In HPCSYSPROS18: HPC System Professionals Workshop, Dallas, TX*.

Neves, L. (2020). Provdeploy: Apoio à coleta de dados de proveniência em scripts de execução de códigos científicos.

Preeth, E., Mulerickal, F. J. P., Paul, B., and Sastri, Y. (2015). Evaluation of docker containers based on hardware utilization. In *2015 International Conference on Control Communication & Computing India (ICCC)*, pages 697–700. IEEE.

Priedhorsky, R. and Randles, T. (2017). Charliecloud: Unprivileged containers for user-defined software stacks in hpc. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–10.

Vahi, K., Rynge, M., Papadimitriou, G., Brown, D. A., Mayani, R., da Silva, R. F., Deelman, E., Mandal, A., Lyons, E., and Zink, M. (2019). Custom execution environments with containers in pegasus-enabled scientific workflows. *arXiv preprint arXiv:1905.08204*.

Wu, K., Ahern, S., Bethel, E. W., Chen, J., Childs, H., Cormier-Michel, E., Geddes, C., Gu, J., Hagen, H., Hamann, B., et al. (2009). Fastbit: interactively searching massive data. In *Journal of Physics: Conference Series*, volume 180, page 012053. IOP Publishing.

[2]https://www.tensorflow.org