Aplicação da técnica *Paramount Iteration* nas aplicações BLAST e DNN-ROM na nuvem computacional

William Felipe da Cunha Tavares[1], **Lucas Oliveira Pimenta dos Reis**[1], **Jeferson Rech Brunetta**[1], **Edson Borin**[1]

[1]Instituto de Computação - Unicamp
Campinas - SP - Brasil

Abstract. *The growing trend of cloud computing brings new challenges to the community of high-performance computing. Due to the large number of resources, predicting the best configuration for a specific application is a costly and time consuming task. The* paramount iteration *technique consists of running a portion of the application to determine the expected behavior in this computational environment when fully executed. This article validates and uses the* paramount iteration *technique for BLAST and DNN-ROM applications, making possible to determine the best cloud computing environment for them.*

Resumo. *O crescimento da tendência da computação em nuvem traz novos desafios para a comunidade de computação de alto desempenho. Por possuir um amplo número de recursos, predizer a melhor configuração para uma aplicação específica é uma tarefa custosa e de alto consumo de tempo e principalmente financeiro. A técnica* paramount iteration *consiste em executar uma parcela da aplicação a fim de determinar o comportamento esperado neste ambiente computacional quando executado por completo. Este artigo valida e utiliza a técnica* paramount iteration *para as aplicações BLAST e DNN-ROM, sendo possível determinar o melhor ambiente de computação em nuvem para estas.*

1. Introdução

A nuvem computacional se tornou um recurso alternativo aos *cluster* de processamento principalmente devido à acessibilidade e flexibilidade, apresentando benefícios como baixo tempo e custo de aquisição. Além disso, a variedade de ambientes computacionais disponíveis e o provisionamento dinâmico escalável chamou a atenção da comunidade que trabalha com algoritmos paralelos e aplicações de alto desempenho.

Neste contexto, a grande quantidade de recursos disponíveis configura-se como um desafio para os usuários. Definir qual configuração computacional apresenta o melhor custo-benefício não é uma tarefa fácil, pois para isso deve-se saber de antemão a quantidade de recursos que uma aplicação necessita para atingir seu pico de desempenho [4]. Mesmo possuindo plena compreensão sobre as aplicações, o usuário pode não dispor da percepção de alguns detalhes de funcionamento da nuvem computacional – percepção da interação de sua aplicação com as camadas de abstração da nuvem, detalhes do *hardware* que executa a máquina virtual na nuvem ou consequências de compartilhar o ambiente da nuvem com outros usuários [11, 10].

Em busca do cenário ideal, onde o usuário possa executar sua aplicação com o melhor custo-benefício, é desejável que o desempenho possa ser estimado de forma

especulativa. Esta tarefa pode ser realizada com técnicas de modelagem de desempenho, as quais exigem alto nível de conhecimento sobre a aplicação e/ou ferramentas de instrumentação [5, 6]. Uma alternativa para determinar a melhor configuração possível com um baixo custo é executar a aplicação com parâmetros de entrada que representam uma pequena fração da execução desejada. No entanto, tais execuções podem não representar de forma precisa o desempenho da aplicação quando executada com entradas maiores.

Para contornar as dificuldades da escolha do melhor recurso computacional a ser usado, empregamos a técnica de execução parcial *paramount iteration*.Dada a gama de variações de configurações dos recursos na nuvem computacional, o processo de escolha poderia se tornar financeiramente impraticável se execuções completas fossem feitas para cada uma das possíveis instâncias a fim de encontrar a que melhor atende a aplicação. A técnica *paramount iteration* propõe executar parcialmente a aplicação e analisar o tempo de execução dos laços iniciais para determinar a melhor configuração.

Neste trabalho, a técnica *paramount iteration* é aplicada e validada em duas aplicações distintas: BLAST [9], um algoritmo conhecido e comumente utilizado na bioinformática, e DNN-ROM [7], um simulador de fluidos que utiliza redes neurais e modelo de ordem reduzida. Com este trabalho exploramos o paralelismo e escalabilidade das aplicações BLAST em máquinas *multicore* e DNN-ROM utilizando processamento em *Graphics Processing Unit* (GPU) e *multicore* submetidos a diferentes configurações na nuvem computacional AWS (*Amazon Web Services*) [2]. Mostramos que o uso da técnica *paramount iteration* permite a escolha otimizada de ambiente de execução - tanto determinando o conjunto de máquinas quanto a escalabilidade da aplicação; executando uma pequena fração da aplicação em diferentes conjuntos de instâncias para avaliar tempo e custo estimados de execução.

A Seção 2 apresenta os trabalhos relacionados, a técnica *paramount iteratio*ne detalha as aplicações avaliadas. A Seção 3 descreve os materiais e métodos utilizados na avaliação experimental. A Seção 4 apresenta e discute os resultados experimentais. Por fim, a Seção 5 apresenta as conclusões e considerações finais.

2. Base teórica

Esta seção apresenta os trabalhos relacionados, a técnica *paramount iteration* e as aplicações BLAST e DNN-ROM.

2.1. Trabalhos Relacionados

Muitos esforços estão sendo despendidos para a execução de aplicações de alto desempnho, ou **HPC** (*High Performance Computing*), na nuvem computacional. Netto e outros [13] elencam alguns dos principais desafios em se mover aplicações de alto desempenho para a nuvem, destacando que os pontos fracos de tal infraestrutura são gargalo da rede e custos. Eles apresentam uma abordagem para decidir ou não migrar as aplicações da infraestrutura adquirida para a nuvem baseada no histórico de execuções.

Zhang e outros [19] também desenvolveram um mecanismo de predição, utilizado pequenas execuções. Eles preveem o tempo de execução da aplicação combinando a frequência de ocorrências das instruções com informações de rede. O problema desta abordagem é que não se sabe de antemão a quantidade de iterações de todos os laços, o que

impediria o cálculo da frequência. A frequência então, pode ser obtida executando-se uma versão simplificada da aplicação, porém, em muitos casos, ela depende da computação e não pode ser obtida rapidamente.

Mariani e outros [12] apresentaram uma terceira abordagem de predição. A técnica consiste em criar um modelo da infraestrutura e um modelo independente de *hardware* da aplicação e utilizar ambos como entrada para um algoritimo *Random Forest* para determinar a melhor infraestrutura. A principal desvantagem desta abordagem é o requerimento de duas execuções para a coleta das informações.

O uso de tais técnicas não contempla a variabilidade de desempenho das instâncias na nuvem, conforme apontado por Gupta e outros [8].

2.2. *Paramount Iteration*

A técnica *paramount iteration* utiliza execuções parciais para estimar o comportamento de execução de uma aplicação sem a necessidade de executá-la em sua completude. Pensada para o uso de aplicações em ambientes de computação de alto desempenho, as quais demandam um tempo significativo de computação, ela é especialmente útil para especular o desempenho sem a necessidade da execução total. A técnica é baseada nas premissas de Yang e outros [17] de que as aplicações científicas têm uma característica naturalmente repetitiva. Ou seja, a fase de computação, responsável pela maior proporção de tempo gasto, é executada de maneira repetida. Caso olhemos para um pequeno conjunto dessas repetições (iterações de um laço principal), o tempo gasto nessas iterações poderia refletir no desempenho relativo da aplicação em um determinado ambiente de execução.

Sua utilização consiste em executar uma pequena parcela das iterações do laço principal da aplicação e analisar o tempo consumido por cada uma das iterações. Ao realizar este procedimento em diferentes ambientes computacionais, é possível classificar o desempenho esperado pela aplicação sem a necessidade de aguardar sua execução completa, evitando desperdício de recursos e reduzindo custos.

Apesar de apresentar um alto potencial para medição de aplicações de alto desempenho, é esperado que a técnica *paramount iteration* não consiga prever relações de desempenho em programas que apresentam laços não determinísticos. Como exemplo, podemos citar laços dependentes de variáveis aleatórias ou que escalam sua execução conforme a aplicação se desenvolve. Portanto, medidas devem ser tomadas para garantir que a aplicação não apresente este tipo de comportamento durante a avaliação. Além disso, não há garantia de confiabilidade nos dados se a aplicação for medida com entradas ou quantidade de nós diferentes, já que o seu tempo de execução costuma variar também com a quantidade de CPUs que executam a aplicação.

2.3. BLAST

BLAST (*Basic Local Alignment Search Tool*) é um algoritmo capaz de encontrar regiões com similaridade local para sequências biológicas. Para isso, sequências nucleotídicas ou proteicas são comparadas para identificar semelhanças em diferentes base de dados e calcular a significância estatística das combinações encontradas. Esta aplicação pode ser usada para encontrar relações evolutivas entre sequências, assim como ajudar a identificar membros de famílias genéticas. Por executar sobre uma vasta quantidade de dados, o

algoritmo é considerado depende do desempenho da memória volátil. [15]. A NCBI (*National Center for Biotechnology Information*) fornece acesso a um conjunto de programas BLAST, como o `blastn` que trabalha com sequências de DNA.

Por se tratarem de aplicações consolidadas no meio científico, há um grande interesse em extrair o máximo de seu desempenho. Dentre estudos que buscam este objetivo, já foram realizados experimentos que comparam desempenho em diferentes modelos de programação, como em GPU[14] e *Field-programmable Gate Array* (FPGA)[18], e utilizando arquiteturas como a em *grid*[1]. Neste trabalho, a técnica *paramount iteration* será validada para a aplicação `blastn` rodando em múltiplas CPUs, possibilitando esclarecer o ambiente computacional que permite extrair o melhor desempenho.

2.4. DNN-ROM

DNN-ROM (*Deep Feedforward Neural Networks - Reduced Order Model*) é uma aplicação que simula o fluxo de fluidos utilizando redes neurais para construir modelos de ordem reduzida. Pode ser aplicada em diferentes sistemas dinâmicos não-lineares, como: osciladores canônicos não-lineares amortecidos e o fluxo que passa por um cilindro circular com baixo número de Reynolds. As simulações utilizam um resolvedor de fluxo diferencial finito compacto de alta ordem. Por ser tratar de uma implementação[1] recente, não há estudos que avaliem seu desempenho. No entanto, a etapa de regressão usando DNN - foco deste trabalho, é muito estudada na literatura [16]. Com a aplicação da técnica *paramount iteration* nesta aplicação, será possível avaliar diferentes configurações para realizar esse tipo de computação.

3. Materiais e Métodos

Nesta seção são descritos em detalhes os experimentos realizados para as duas aplicações, além dos materiais utilizados.

3.1. BLAST

A aplicação `blastn` é constituída por um laço que realiza a busca de sequências. A quantidade de iterações não é conhecida e o tempo de cada uma não é constante, pois depende da quantidade de combinações encontradas nas sequências genéticas e a significância obtida nestas combinações.

Dois dos parâmetros da aplicação consistem em uma base de dados para realizar a busca e um arquivo FASTA como entrada. A partir de genomas humanos disponíveis em Ensembl[2] e com auxílio da aplicação `makeblastdb` do BLAST, foi criada uma base de dados de $1,5$GB. O arquivo `Homo_sapiens.GRCh38.cds.all.fa` foi utilizado como entrada e também está disponível na mesma página.

Para realizar os experimentos foram utilizadas diferentes instâncias da nuvem computacional. Na Tabela 1 encontram-se as especificações. Outros detalhes podem ser encontrados na página da AWS [3]. Foi utilizada a imagem Ubuntu Server 18.04 LTS (HVM), SSD Volume Type (64-bit x86) com 40GB de armazenamento. A aplicação foi executada de forma paralela com o número de *threads* igual ao número de núcleos computacionais presente em cada instância.

[1] https://github.com/hugolui/ROM_code

[2] https://www.ensembl.org/Homo_sapiens/Info/Index

instancia	vCPUs	RAM (GiB)	Preço *on demand* (USD/h)
m5.2xlarge	8	32	$0,384
m5a.2xlarge	8	32	$0,344
m4.2xlarge	8	32	$0,4
c5.4xlarge	16	32	$0,68
r5.xlarge	4	32	$0,252
r5.2xlarge	8	64	$0,504
r5.4xlarge	16	128	$1,008
r5a.xlarge	4	32	$0,226
r5a.2xlarge	8	64	$0,452
r4.xlarge	4	30,5	$0,244
r4.2xlarge	8	61	$0,532
x1e.2xlarge	8	244	$1,668

Tabela 1. Especificações das instâncias para os experimentos com `blastn`

O código da aplicação `blastn` foi instrumentado para permitir realizar a interrupção da execução a partir de um certo número de iterações. Cada experimento foi executado uma vez. Por não ser viável sua execução total, considerando o tempo e custo dos experimentos, a aplicação foi executada até 400 iterações em cada uma das instâncias descritas na Tabela 1.

De forma a permitir a reprodutibilidade, foi desenvolvida uma receita para gerar uma imagem de *container* com a plataforma Singularity, contendo o BLAST e suas dependências devidamente instaladas. O passo a passo de como executar os experimentos estão disponíveis em um repositório no GitHub[3].

3.2. DNN-ROM

A aplicação DNN-ROM é composta por várias etapas. De acordo com testes preliminares, a regressão por DNN representa a maior parcela do tempo total de execução da aplicação. Por este motivo, os experimentos foram realizados nesta etapa. Além disto, por fazer uso da biblioteca Tensorflow, é possível paralelizá-lo com GPU ou *multicore*.

Todos os experimentos são realizados na base de dados `qSpanAvg.cgns` (16GB), fornecida pelo autor da aplicação. Foram utilizadas instâncias `p2.xlarge`, `p3.2xlarge` e `g3s.xlarge`, que possuem GPUs, e a família de instâncias `c5`, que são otimizadas para uso intensivo de computação com CPU. Por limitações da implementação, não é possível utilizar mais de uma GPU, reduzindo o experimento às instâncias mais básicas das famílias `p2`, `p3` e `g3`. A Tabela 2 apresenta a lista de instâncias utilizadas nos experimentos com a aplicação DNN-ROM. As instâncias `p2` apresentam processadores Intel Xeon E5-2686 v4 @ 2,2 GHz e GPU NVIDIA K80, enquanto as instâncias `p3` e `g3` apresentam o mesmo modelo de processador com GPU NVIDIA Tesla V100 e NVIDIA Tesla M60, respectivamente. Já as instâncias da família `c5` possuem modelos variados de processadores, apresentando o modelo Intel Xeon Scalable Segunda Geração @ 3,6 GHz (3,9 GHz turbo) para os modelos `c5.12xlarge` e `c5.24xlarge` e processador Intel Xeon Platinum série 8000 @ 3,4 GHz (3,5 GHz turbo) para os outros modelos menores.

Para a execução da aplicação em instâncias `p2`, `p3` e `g3` foi utilizada a imagem Deep Learning Base AMI (Ubuntu) Version 19.0, e para as instâncias da família `c5` foi utilizada a imagem Ubuntu Server 18.04 LTS (HVM), SSD Volume Type. Ambas utilizam 70 GB de armazenamento.

[3]https://github.com/theorangewill/blast-paramount-iteration/

instancia	vCPUs	RAM (GiB)	GPUs	Memória de GPU (GiB)	Preço *on-demand* (USD/h)
`c5.2xlarge`	8	16	-	-	$0,34
`c5.4xlarge`	16	32	-	-	$0,68
`c5.9xlarge`	36	72	-	-	$1,53
`c5.12xlarge`	48	96	-	-	$2,04
`c5.18xlarge`	72	144	-	-	$3,06
`c5.24xlarge`	96	192	-	-	$4,08
`g3s.xlarge`	4	30.5	1	8	$0,75
`p2.xlarge`	4	61	1	12	$0,90
`p3.2xlarge`	8	61	1	16	$3,06

Tabela 2. Especificações das instâncias `p2`, `p3`, `g3` e `c5` para os experimentos com a aplicação DNN-ROM

Fundamentalmente, a aplicação consiste em uma busca de redes neurais com parâmetros aleatórios. Assim, para garantir a validade da experimentação entre diferentes instâncias, uma semente é alimentada na função de geração de números aleatórios, assegurando que as sequências de redes geradas em cada experimento sejam idênticas. Para esta base, são treinadas 50 redes neurais aleatórias com número variado de camadas (método de RandomSearch). O treinamento de cada rede consiste em uma iteração da aplicação. Cada experimento foi executado uma vez.

De forma a garantir a reprodutibilidade, foi desenvolvida a imagem de um *container* com a plataforma Singularity, contendo todas as dependências necessárias para execução da aplicação DNN-ROM. O código fonte da aplicação, bem como suas instruções de instalação e a imagem do *container* estão disponíveis em um repositório do GitHub[4].

4. Resultados experimentais

4.1. BLAST

As iterações da aplicação `blastn` não possuem tempo constante ou evolutivo, como mostrado pela Figura 1. O gráfico desta figura mostra o tempo de execução de cada iteração para duas instâncias do experimento. Também é possível verificar equivalência dos tempos entre os diferentes ambientes.

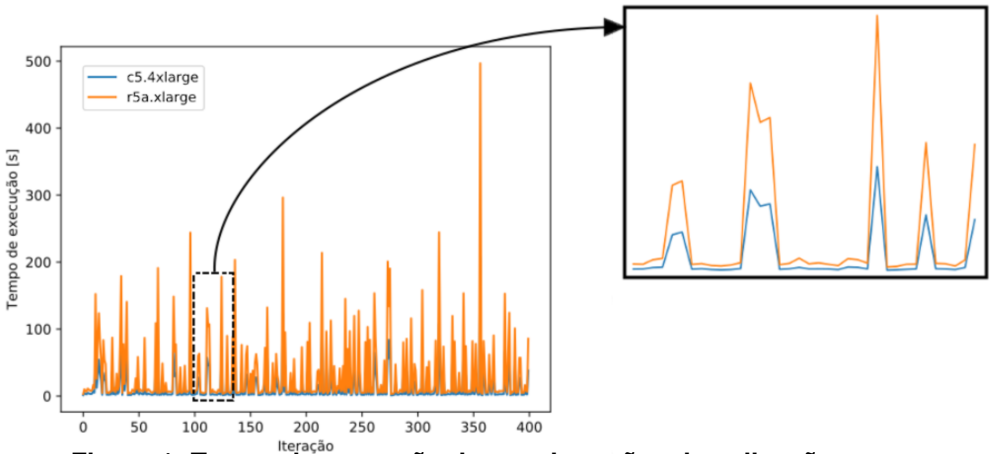


Figura 1. Tempo de execução das 400 iterações da aplicação `blastn`

A Tabela 3 apresenta a sobrecarga de desempenho entre as configurações experimentadas para diferentes quantidades de iterações. A sobrecarga de desempenho é cal-

[4]https://github.com/lucasreis1/ROM_code-paramout-iteration

culada em relação à instância que apresentou o melhor tempo de execução da aplicação (`c5.4xlarge`).

A predição realizada pelo *paramount iteration* nas diferentes quantidades de iterações é bem precisa. De fato, a técnica mostra que não há muita discrepância nas sobrecargas de desempenho para cada iteração em relação à execução total de 400 iterações, salvo algumas exceções destacadas em negrito na Tabela 3. Além disso, a partir de 1 iteração a instância `c5.4xlarge` já se mostra a melhor configuração para executar a aplicação.

Para a execução das 400 iterações na instância `c5.4xlarge` foram necessários pouco menos de 91 minutos. Em contrapartida, 10 iterações foram executadas em 38 segundos. Para a instância com o pior desempenho, 400 iterações executaram por 211 minutos e 10 iterações por 89, 7 segundos. Isto mostra que é possível utilizarmos o desempenho do início da aplicação para determinar o ambiente computacional em que será extraído o melhor desempenho, diminuindo custo financeiro e também o tempo de execução.

Através da Tabela 5, também é possível visualizar a escalabilidade da aplicação. Isto pode ser verificado mais facilmente pela família de instâncias `r5`, quanto maior o número de cores presente, melhor é o desempenho. Por limitações de paralelismo da própria aplicação, em função do tamanho da base de dados de entrada, os 16 núcleos presentes na instância `r5.4xlarge` não são completamente utilizados. Por esta razão, os desempenhos próximos.

Instância	1 iteração	5 iterações	10 iterações	400 iterações
`c5.4xlarge`	1.00	1.00	1.00	1.00
`m5.2xlarge`	**1.27**	**1.29**	**1.27**	1.21
`r5.4xlarge`	1.24	1.24	1.21	1.22
`r5.2xlarge`	1.26	1.26	1.24	1.24
`r5a.2xlarge`	1.45	1.46	1.46	1.46
`m5a.2xlarge`	1.49	1.49	1.47	1.50
`m4.2xlarge`	1.53	**1.61**	**1.59**	1.51
`r4.2xlarge`	1.53	1.59	1.57	1.53
`x1e.2xlarge`	1.58	1.67	1.65	1.59
`r5.xlarge`	1.94	1.96	1.96	1.88
`r4.xlarge`	**2.44**	**2.54**	**2.55**	2.32
`r5a.xlarge`	2.34	2.36	2.36	2.33

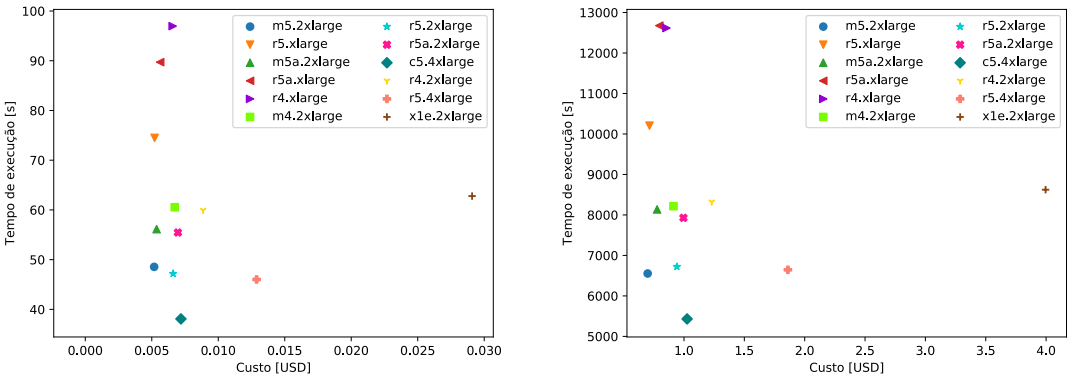
Tabela 3. Sobrecarga de desempenho para o `blastn` nas diferentes instâncias

Na Tabela 4 é apresentada a sobrecarga de custo das execuções em relação à instância que apresentou o menor custo (`m5.2xlarge`). Novamente a predição da técnica *paramount iteration* é correta (salvo duas exceções em negrito). Todos os custos são calculados levando em consideração o preço *on-demand* de cada instância, multiplicado pelo tempo necessário para execução da aplicação.

A partir dos experimentos, pode-se construir um gráfico que apresenta o custo-benefício da execução da aplicação `blastn` nas diferentes configurações computacionais. Este gráfico encontra-se na Figura 2, mostrando que a instância `c5.4xlarge` apresenta o menor tempo de execução, assim como `m5.2xlarge` com o menor custo. Além de evidenciar boas configurações, é notado o baixo desempenho das instâncias `r5a.xlarge` e `r4.xlarge` e o custo elevado da instância `x1e.2xlarge`.

Instância	1 iteração	5 iterações	10 iterações	400 iterações
m5.2xlarge	1.00	1.00	1.00	1.00
r5.xlarge	1.00	1.00	1.01	1.02
m5a.2xlarge	1.05	1.04	1.04	1.11
r5a.xlarge	1.08	1.08	1.09	1.14
r4.xlarge	1.22	1.26	1.27	1.22
m4.2xlarge	1.26	**1.31**	**1.30**	1.31
r5.2xlarge	1.30	1.28	1.28	1.35
r5a.2xlarge	1.34	1.34	1.34	1.42
c5.4xlarge	1.40	1.38	1.39	1.47
r4.2xlarge	1.67	1.72	1.71	1.76
r5.4xlarge	2.57	2.53	2.49	2.66
x1e.2xlarge	5.41	5.63	5.62	5.72

Tabela 4. Sobrecarga de custo para o `blastn` nas diferentes instâncias



(a) Execução de 10 iterações **(b) Execução de 400 iterações**

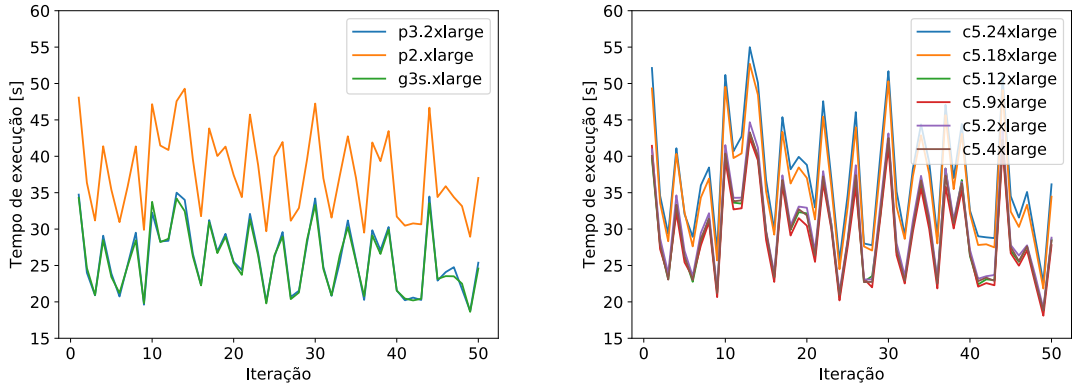
Figura 2. Relação tempo de execução e custo para aplicação `blastn`

4.2. DNN-ROM

A Figura 3 apresenta os gráficos de medição do tempo individual de cada iteração de todas as instâncias testadas, separadas entre execuções de instâncias que utilizam GPU (famílias `p2, p3 e g3`) na Figura 3a e instâncias que utilizam CPU (família `c5`) na Figura 3b. Essa separação se deve pela leve diferença encontrada no comportamento dos dois grupos de instâncias citados. Portanto, as análises subsequentes dos resultados serão divididas entre estes dois conjuntos de instâncias, almejando também uma melhor divisão dos dados.

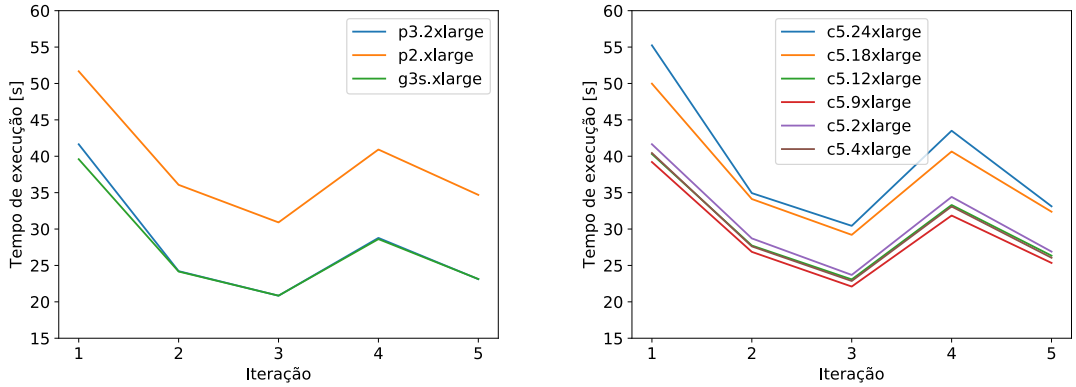
Uma análise da Figura 3 também demonstra a falta de correlação entre iterações, que não aparentam seguir qualquer tendência. Isso se deve pela aleatoriedade das redes geradas. Também observando os gráficos, é possível verificar equivalência de tempo de execução de cada iteração entre as instâncias. Na Figura 4, é possível observar o tempo de execução de 5 iterações da aplicação. Nela, fica evidenciada a mesma tendência entre instâncias presente na execução total da aplicação.

A Tabela 5 mostra a sobrecarga de desempenho obtida utilizando a execução total da aplicação em relação à utilização da técnica *paramount iteration*. A sobrecarga de desempenho é calculada em relação às instâncias que apresentam o melhor tempo de execução da aplicação para cada conjunto de iterações contabilizadas. Apesar de pequenas discrepâncias nos valores absolutos, a comparação entre diferentes instâncias é fiel para todos os intervalos de iterações exibidos, sem erros de predição.



(a) Famílias `p2`, `p3` e `g3` **(b)** Família `c5`

Figura 3. Média do tempo de execução total para a aplicação DNN-ROM



(a) Famílias `p2`, `p3` e `g3` **(b)** Família `c5`

Figura 4. Média do tempo de execução de 5 iterações da aplicação DNN-ROM utilizando a técnica *paramount iteration*

Instância	1 iteração	5 iterações	10 iterações	50 iterações
`g3s.xlarge`	1.00	1.00	1.00	1.00
`p3.2xlarge`	1.05	1.01	1.00	1.00
`p2.xlarge`	1.30	1.42	1.45	1.46
`c5.9xlarge`	1.00	1.00	1.00	1.00
`c5.12xlarge`	1.02	1.03	1.01	1.02
`c5.4xlarge`	1.03	1.03	1.01	1.02
`c5.2xlarge`	1.06	1.06	1.04	1.04
`c5.18xlarge`	1.27	1.28	1.22	1.22
`c5.24xlarge`	1.40	1.35	1.26	1.27

Tabela 5. Sobrecarga de desempenho para o DNN-ROM nas diferentes instâncias

Mesmo levando em consideração a imprevisibilidade no custo de execução de cada iteração, uma análise do código-fonte da aplicação mostra que os possíveis parâmetros de seleção de uma rede aleatória possuem intervalos pequenos, fazendo com que o número de iterações necessárias para representar uma execução completa seja pequeno.

A Tabela 6 apresenta a sobrecarga de custo obtida em relação ao custo de utilização das instâncias, utilizando a mesma metodologia mencionada. É possível observar que a utilização da técnica *paramount iteration* prevê corretamente a relação de

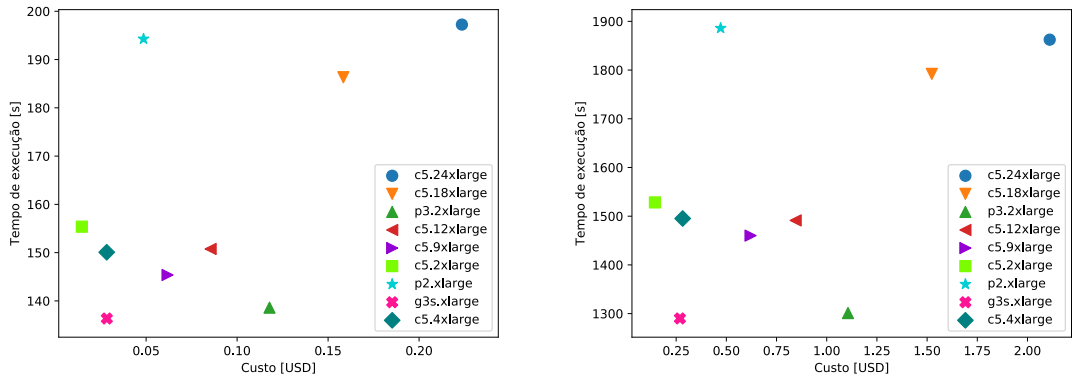
custo entre as máquinas, com exceção de uma única instância quando se trata da previsão utilizando apenas uma iteração, como evidenciado em negrito. Todos os custos são calculados levando em consideração o preço *on-demand* de cada instância, multiplicado pelo tempo necessário para execução da aplicação.

Instância	1 iteração	5 iterações	10 iterações	50 iterações
`c5.2xlarge`	1.00	1.00	1.00	1.00
`g3s.xlarge`	**2.09**	1.93	1.88	1.86
`c5.4xlarge`	1.94	1.93	1.94	1.95
`p2.xlarge`	3.28	3.31	3.29	3.26
`c5.9xlarge`	4.23	4.21	4.32	4.29
`c5.12xlarge`	5.80	5.82	5.80	5.85
`p3.2xlarge`	9.00	8.02	7.70	7.66
`c5.18xlarge`	10.80	10.80	10.60	10.55
`c5.24xlarge`	15.91	15.23	14.65	14.62

Tabela 6. Sobrecarga de custo para o DNN-ROM nas diferentes instâncias

Para a execução das 50 iterações na instância `c5.4xlarge`, foram necessários pouco menos de 25 minutos. Já para a execução das 5 iterações, a execução durou menos de 3 minutos. Para a instância com pior desempenho, 50 iterações executaram por mais de 30 minutos, enquanto as primeiras 5 iterações são executadas por pouco mais de 3 minutos. Levando em consideração os resultados apresentados nas figuras e na Tabela 5, é possível afirmar que a técnica *paramount iteration* apresenta alta confiabilidade para este ambiente de execução e pode prover uma alta economia de recursos.

A Figura 5 demonstra a diferença dos tempos de execução entre as diferentes instâncias, além de apresentar o custo de cada execução. Ainda no contexto da validação da técnica *paramount iteration*, é possível perceber algumas disparidades entre os dois gráficos da figura. Para a execução total, a instância `p2.xlarge` apresenta desempenho melhor que a instância `c5.24xlarge`, o que não se reflete na Figura 5a. Além disso, a diferença de tempo de execução entre as instâncias `p3.2xlarge` e `g3s.xlarge` e as instâncias `c5.2xlarge`-`c5.12xlarge` se torna menos significativa quando se analisa apenas a Figura 5a.



(a) Execução de 5 iterações **(b) Execução de 50 iterações**

Figura 5. Relação entre tempo de execução e custos para a aplicação DNN-ROM

5. Conclusão

A técnica *paramount iteration* apresenta-se como um preditor de desempenho para aplicações científicas e uma alternativa para uma boa escolha de configuração na nuvem

computacional. Neste trabalho, esta técnica foi validada em duas aplicações, BLAST e DNN-ROM, utilizando o ambiente da AWS. Foi possível observar que tanto a predição de desempenho quanto a de custo se mostraram precisas e permitiram a escolha da configuração mais apropriada para a execução completa das aplicações, mesmo levando-se em consideração um conjunto limitado de iterações na análise.

Apesar das diferenças de comportamento das aplicações, a técnica *paramount iteration* manteve-se válida. Enquanto a aplicação BLAST não possui uma quantidade de iterações conhecida e o tempo entre as iterações não é constante ou equivalente, o DNN-ROM é uma aplicação com número de iterações conhecido e com tempo equivalente. Também foram observados resultados interessantes em relação às melhores instâncias para cada aplicação. Para o DNN-ROM, os resultados apresentados por instâncias otimizadas para computação *multicore* foram surpreendentes, quando se considera que a aplicação foi projetada para ser acelerada com GPUs.

É válido reforçar que, para a técnica *paramount iteration* possa ser usada de forma eficaz, a computação realizada durante as interações deve ser determinística. O DNN-ROM, que possui um fator aleatório, teve que ser modificado de forma a manter as diferentes execuções equivalentes. O mesmo vale para aplicações de comportamento semelhante.

Explorar a técnica em outras instâncias da AWS tanto quanto em outros provedores de serviço de nuvem é um dos possíveis trabalhos futuros. Avaliar a técnica em outras aplicações é de bastante interesse, principalmente em aplicações que possuam características diferentes das tratadas neste trabalho. O uso da plataforma Singularity facilitou a reprodutibilidade, no entanto ela pode gerar sobrecarga de desempenho na execução; é relevante avaliar o impacto desta plataforma na técnica *paramount iteration*.

Agradecimentos

Os autores gostariam de prestar os devidos agradecimentos ao pesquisador William Wolf e seu aluno Hugo Lui, pelo desenvolvimento da aplicação DNN-ROM, bem como o fornecimento de uma base de dados significativa e pelo auxílio na tarefa de compreender e executar a aplicação.

Também gostariam de agradecer o pesquisador Marcelo Carazzolle, pela orientação na geração de uma base de dados representativa a partir de sequências genéticas reais, bem como auxílio no entendimento da execução e funcionamento da aplicação BLAST.

Por fim, agradecemos à Petrobras, ao CNPq (313012/2017-2) e à FAPESP (CEPID 2013/08293-7) pelo apoio financeiro.

Referências

[1] E. Afgan and P. Bangalore. Performance characterization of BLAST for the grid. In *7th International Symposium on BioInformatics and BioEngineering*, 2007.

[2] Amazon. Amazon Web Services webpage. `https://aws.amazon.com/pt/ec2/`.

[3] Amazon. Tipos de instância do Amazon EC2. `https://aws.amazon.com/pt/ec2/instance-types/`.

[4] R. Aversa, B. Di Martino, M. Rak, S. Venticinque, and U. Villano. Performance prediction for HPC on clouds. In *Cloud Computing: Principles and Paradigms*, 2011.

[5] A. Bhattacharyya and T. Hoefler. PEMOGEN: Automatic adaptive performance modeling during program runtime. In *23rd International Conference on Parallel Architectures and Compilation*, 2014.

[6] R. Escobar and R. V. Boppana. Performance prediction of parallel applications based on small-scale executions. In *23rd International Conference on High Performance Computing*, 2016.

[7] H. F. S. Lui and W. Wolf. Construction of reduced order models for fluid flows using deep feedforward neural networks, 2019.

[8] A. Gupta, P. Faraboschi, F. Gioachin, L. V. Kale, R. Kaufmann, B. Lee, V. March, D. Milojicic, and C. H. Suen. Evaluating and improving the performance and scheduling of hpc applications in cloud. *IEEE Transactions on Cloud Computing*, 4(3), 2016.

[9] M. Johnson, I. Zaretskaya, Y. Raytselis, Y. Merezhuk, S. McGinnis, and T. L. Madden. NCBI BLAST: a better web interface. *Nucleic Acids Research*, 36, 2008.

[10] G. Mariani, A. Anghel, R. Jongerius, and G. Dittmann. Predicting cloud performance for HPC applications: A user-oriented approach. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2017.

[11] G. Mariani, A. Anghel, R. Jongerius, and G. Dittmann. Predicting cloud performance for HPC applications before deployment. *Future Generation Computer Systems*, 2018.

[12] G. Mariani, A. Anghel, R. Jongerius, and G. Dittmann. Predicting cloud performance for HPC applications before deployment. *Future Generation Computer Systems*, 87, 2018.

[13] M. A. S. Netto, R. L. F. Cunha, and N. Sultanum. Deciding when and how to move HPC jobs to the cloud. *Computer*, 48(11), 2015.

[14] S. Rani and O. P. Gupta. CLUS_GPU-BLASTP: accelerated protein sequence alignment using GPU-enabled cluster. *The Journal of Supercomputing*, 2017.

[15] S. Sawyer, M. Horton, C. Burdyshaw, G. Brook, and B. Rekapalli. HPC-BLAST: Distributed BLAST for modern HPC clusters. In *Proceedings of 11th International Conference on Bioinformatics and Computational Biology*, 2019.

[16] S. Shi, Q. Wang, and P. Xu. Benchmarking state-of-the-art deep learning software tools. In *7th International Conference on Cloud Computing and Big Data*, 2016.

[17] L. T. Yang, Xiaosong Ma, and F. Mueller. Cross-platform performance prediction of parallel applications using partial execution. In *SC: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, 2005.

[18] M. Yoshimi, C. Wu, and T. Yoshinaga. Accelerating BLAST computation on an FPGA-enhanced PC cluster. In *2016 Fourth International Symposium on Computing and Networking*, 2016.

[19] W. Zhang, M. Hao, and M. Snir. Predicting HPC parallel program performance based on LLVM compiler. *Cluster Computing*, 20(2), 2017.