Avaliação de Estratégias de gerenciamento de contêineres para uma Aplicação de Bioinformática na AWS*

Rafael Amparo[1], **Cristina Boeres**[1], **Vinod E.F. Rebello**[1]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói – RJ – Brasil

ramparo@id.uff.br, {boeres,vinod}@ic.uff.br

Abstract. *This work evaluates the execution of a bioinformatics application on the AWS cloud, considering different container services and execution configurations. The ECS-EC2 Spot, ECS-Fargate, and EC2-Docker approaches were compared, considering performance, cost, and ease of management. The experiments explored different levels of parallelism, varying the number of threads and containers. The results indicate that ECS-EC2 Spot offers a balance between cost and performance, although subject to the instability of the Spot market. ECS-Fargate prioritizes practicality, but with higher overheads and costs. Finally, EC2-Docker presents the best efficiency, but with greater management effort.*

Resumo. *Este trabalho avalia a execução de uma aplicação de bioinformática na nuvem AWS considerando diferentes serviços e configurações de execução em contêineres. Foram comparadas as abordagens ECS-EC2 Spot, ECS-Fargate e EC2-Docker, considerando desempenho, custo e facilidade de gerenciamento. Os experimentos exploraram diferentes níveis de paralelismo, variando o número de threads e contêineres. Os resultados indicam que ECS-EC2 Spot oferece equilíbrio entre custo e desempenho, embora sujeito à instabilidade do mercado Spot. ECS-Fargate prioriza praticidade, mas com maior sobrecarga e custo, enquanto EC2-Docker apresenta a melhor eficiência, mas com maior esforço de gerenciamento.*

1. Introdução

A computação em nuvem consolidou-se como paradigma dominante de infraestrutura, oferecendo escalabilidade, elasticidade e redução de custos operacionais em comparação a uma infraestrutura física local. A virtualização tradicional, por meio de máquinas virtuais, permitiu melhor aproveitamento de recursos, mas com sobrecargas de inicialização e execução que podem ser significativas. Por outro lado, os contêineres emergem como alternativa mais leve e eficiente, oferecendo isolamento, portabilidade e inicialização rápida [Randal 2020].

Com o objetivo de analisar diferentes configurações e serviços de conteinerização no ambiente de Nuvem AWS, a aplicação MASA-OpenMP (*Multi-Platform Architecture for Sequence Aligner*) [De O. Sandes et al. 2016] foi considerada como objeto de estudo. O MASA-OpenMP realiza alinhamentos de sequências com mais de 200 milhões

*Financiado pelo projeto CNPq/AWS (processo 421828/2022-6).

de nucleotídeos. Através da execução no ambiente de Nuvem AWS, foi investigado como decisões de gerenciamento de contêineres, estratégias de paralelismo e a escolha de serviços impactam no desempenho, custo e a usabilidade do serviço. Foram definidas três combinações experimentais para comparação: (i) execução de contêineres *Docker* em instâncias EC2 (EC2-Docker), (ii) orquestração via Amazon ECS sobre instâncias EC2 no mercado *Spot* (ECS-EC2 Spot) e (iii) execução via Amazon ECS sobre AWS Fargate (ECS-Fargate). Essas combinações avaliam diferentes níveis de abstração, do controle manual de contêineres à delegação de seu gerenciamento pelos serviços de orquestração, e seus impactos no ciclo de vida das tarefas, nas sobrecargas em tempo de execução e nos modelos de cobrança.

2. Experimentos e sua análise

No conjunto de experimentos realizado, 9 configurações foram especificadas variando o número t de *threads* e número de contêineres c, em grupos de sete sequências de tamanhos $g \in \{200K, 300K, 400K\}$ nucleotídeos. Foram capturados tempos de execução ($T_e(t, g)$) do alinhamento dentro de um contêiner com t *threads*, tempo de cobrança ($T_c(g, c, t)$) que corresponde ao tempo decorrido desde da criação até a finalização dos c contêineres que executam a tarefa em t *threads* considerando alinhamentos de tamanho g. O percentual do *overhead* de execução nos c contêineres é definido como $Po(g, c, t) = \frac{T_c(g,c,t) - \max_{\forall c} T_e(g,t)}{T_c(g,c,t)} * 100$. O overhead de orquestração dos contêineres foi obtido pela diferença entre o tempo total de execução, que inclui desde a criação dos contêineres até o seu término, e o tempo de execução medido internamente pela aplicação dentro dos contêineres. Foram calculados ainda os custos monetários associados às combinações (g, c, t). A divisão de carga e a alocação de recursos foram organizadas de forma integrada: para cada g, foram executados 28 alinhamentos par-a-par, divididos entre c contêineres, de modo que cada um executou $\frac{28}{c}$ alinhamentos em sequência, sendo cada alinhamento foi processado com t *threads*. A nível de infraestrutura, para a definição de tarefa foram especificados 16 vCPUs e 32 GB de memória, recursos esses divididos entre os c contêineres. A Tabela 1 mostra os tempos $Tc()$ e $\max_{\forall c} Te()$ e os custos associados para cada modelo. A Tabela 2 mostra o *Speedup*,

Tabela 1. Tempo cobrado T_c (em seg.), maior tempo de execução T_e (em seg.) e custo em USD associados às combinações g, c, t.

			ECS-EC2 (c7g.8xlarge)			ECS-Fargate (16 vCPU/32GB)			EC2-Docker (c7g.8xlarge)		
g	c	t	T_c	$\max T_e$	Custo	T_c	$\max T_e$	Custo	T_c	$\max T_e$	Custo
200k	1	16	190.61	185.14	0.0141	203.79	185.59	0.0358	190.60	187.30	0.0232
	2	8	192.04	177.46	0.0142	209.18	177.61	0.0367	181.24	178.71	0.0221
	4	4	197.24	173.53	0.0146	208.06	173.62	0.0367	177.73	175.11	0.0216
300k	1	16	413.48	408.37	0.0307	423.72	406.69	0.0744	414.66	410.70	0.0505
	2	8	416.71	396.00	0.0309	423.22	394.75	0.0743	400.53	398.20	0.0488
	4	4	412.82	389.75	0.0306	437.06	389.26	0.0767	394.66	391.90	0.0480
400k	1	16	722.40	717.11	0.0536	733.80	715.44	0.1288	725.24	721.20	0.0883
	2	8	707.49	700.57	0.0525	730.12	697.42	0.1282	705.90	703.30	0.0859
	4	4	712.95	692.86	0.0529	710.30	688.45	0.1247	697.66	694.20	0.0849

calculado como $Sp(g, c, t) = \frac{T_e(g,1,t)}{T_e(g,c,t)}$. Para os testes utilizando **ECS**, foi seguido o framework de experimentação definido por [Cavalcante et al. 2024].

Tabela 2. *Speedup* e *Po* associados às combinações g, c, t

Serviços	g	c	$Sp(g, c, t)$	*Po(g, c, t)*
EC2-Docker (c7g.8xlarge)	**200k**	1	–	1.73%
		2	1.048	1.40%
		4	1.069	1.47%
	300k	1	–	0.95%
		2	1.031	0.58%
		4	1.048	0.70%
	400k	1	–	0.56%
		2	1.025	0.37%
		4	1.039	0.54%
ECS-Fargate (16 vCPU / 32 GB)	**200k**	1	–	8.93%
		2	1.045	15.09%
		4	1.069	16.55%
	300k	1	–	4.02%
		2	1.030	6.73%
		4	1.045	10.94%
	400k	1	–	2.50%
		2	1.026	5.09%
		4	1.039	3.06%
ECS-EC2 Spot (c7g.8xlarge)	**200k**	1	–	2.86%
		2	1.043	7.59%
		4	1.067	12.02%
	300k	1	–	1.23%
		2	1.031	4.96%
		4	1.048	5.59%
	400k	1	–	0.73%
		2	1.025	0.97%
		4	1.035	2.82%

Enquanto o aumento de *threads* por contêiner apresentou ganhos de desempenho de execução (observáveis nos valores de *speedup* da Tabela 2), cenários multi-contêineres só demonstram vantagem quando o tempo de processamento útil por contêiner é capaz de amortizar os *overheads* de orquestração. Em **ECS-EC2 Spot**, $T_c()$ aproximou-se de $T_e()$ somente para sequências da classe $g = 400K$, refletindo overheads moderados neste caso (não mais que 3%) e boa relação custo-benefício. Entretanto, existe a imprevisibilidade do preço do *Spot* e o risco de revogação de instâncias. Já em **ECS-Fargate**, a parcela de *overhead* foi, em média, maior e mais variável, o que reduz o grau de vantagem de uso do paralelismo multi-contêiner, especialmente para sequencias menores: para $g = 200K$ ficou entre 8 a 16%, $g = 300K$ entre 6 a 10% e $g = 400K$ entre 3 a 5%.

É notório, no entanto, que na combinação **EC2-Docker** os menores *overheads* foram registrados, com $T_c()$ frequentemente próximo de $T_e()$ (não mais do que 1.8% $\forall g$). A não utilização dos serviços de orquestração reduz tempo, mas transfere ao desenvolvedor a responsabilidade por todo o provisionamento e gerenciamento do processo de conteinerização, perdendo a facilidade e abstração oferecidas em serviços como ECS.

Configurações de um contêiner com 16 *threads* mostraram menor *overhead* relativo para g menores. Já a divisão em múltiplos contêineres com menos *threads* explorou o paralelismo distribuído de forma eficaz apenas para grupos de sequências maiores ($g \in 300K, 400k$), quando o ganho pela distribuição de tarefas compensa o tempo adicional de gerenciamento. A análise de custo explicitou diferenças estruturais: no AWS

Fargate a fórmula de cobrança soma custo de memória e vCPU ao longo do tempo de execução (tornando tarefas curtas mais caras), enquanto em EC2 o custo está associado ao tempo de instância (com economia no mercado *Spot* que pode sofrer revogações).

3. Conclusão

Os experimentos mostraram um *trade-off* entre custo, desempenho e facilidade de uso dos serviços de conteinerização aqui considerados. A execução de contêineres no modelo (**EC2-Docker**) apresentou os menores *overheads* de provisionamento, com a maioria dos $T_c()$ próximos de $T_e()$, evidenciando que, se eliminada a camada de orquestração, a maior parte do tempo cobrado é computação útil. Porém, essa opção exige esforço operacional maior (provisionamento de contêineres e gerenciamento de sua execução) e, no caso do mercado *Spot*, há a necessidade de tornar a aplicação robusta quanto a revogações. Por outro lado, o modelo (**ECS-EC2 Spot**) alcançou o equilíbrio mais favorável entre custo e eficiência, apresentando overheads moderados e menor custo médio nas execuções registradas; sua principal limitação é, também, a volatilidade do preço e o risco de preempção das instâncias, além de não aproveitar completamente o paralelismo de contêineres se comparado à opção sem serviço de orquestração. Finalmente, o **ECS-Fargate** destacou-se pela máxima simplicidade operacional, porém *overhead* e custo por tarefa substancialmente maiores, especialmente em tarefas curtas, porque a cobrança engloba todo o ciclo de provisionamento.

A análise do paralelismo nos dois níveis (*threads*/alinhamento e multi-contêineres) mostrou impactos não-lineares em relação ao Speedup e *overhead*. Observou-se que a estratégia multi-contêiner compensa em alinhamentos de sequências maiores (maiores cargas) por contêiner, amortizando custo de provisionamento. Na prática, a escolha deve refletir a finalidade: se a meta principal for redução de custo médio e há tolerância a alguma variabilidade de disponibilidade e tempo de execução mais longo, **ECS-EC2 Spot** oferece mais equilíbrio; se a prioridade for simplicidade operacional, **ECS-Fargate** é a opção indicada, ciente do custo maior por tarefa; se o foco é em desempenho de execução e maior aproveitamento de recursos alocados, **EC2-Docker** é a combinação mais adequada, desde que existam mecanismos de gerenciamento para evitar ociosidade e lidar com possíveis revogações, caso o mercado *Spot* seja escolhido.

Como trabalho futuro, essa pesquisa pretende analisar o uso de outras ferramentas de orquestração para ambientes multi-contêiner, como o *Kubernetes*, ampliando essa análise para execuções em ambientes de servidores *on-premises*.

Referências

Cavalcante, S., Boeres, C., and Rebello, V. (2024). Explorando a eficiência de serviços de conteinerização aws. In *Anais da IX Escola Regional de Alto Desempenho do Rio de Janeiro*, pages 34–36. SBC.

De O. Sandes, E. F., Miranda, G., Martorell, X., Ayguade, E., Teodoro, G., and De Melo, A. (2016). MASA: A multiplatform architecture for sequence aligners with block pruning. *ACM Trans. on Par. Comp.*, 2(4).

Randal, A. (2020). The ideal versus the real: Revisiting the history of virtual machines and containers. *ACM Comput. Surv.*, 53(1).