Containers versus Máquinas Virtuais para Emulação de Controlo de Redes

Bruno Anjos e Nuno Morais

Orientados por: Paulo Lopes e José Legatheaux Martins

NOVA LINCS, Departamento de Informática
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
2829–516 Caparica, Portugal
{b.anjos@campus., nm.morais@campus, paulo.lopes@,
jose.legatheaux@}fct.unl.pt
http://di.fct.unl.pt

Resumo Para o estudo do comportamento de sistemas sem recorrer à sua prévia implementação, cada vez mais surge a necessidade de emulação dos mesmos, geralmente recorrendo a tecnologias de virtualização. Esta necessidade é particularmente aguda em sistemas distribuídos de larga escala. Entre as soluções mais populares de virtualização estão os *containers* e as máquinas virtuais (VMs). Com a generalização do *Cloud Computing* e a cada vez maior utilização de virtualização, criou-se uma indústria à volta da otimização destas soluções e do *deployment* de sistemas distribuídos escaláveis. No caso das VMs a solução mais popular em infraestruturas empresariais é a implementada pela VMWare com o *hypervisor* ESXi, e para *containers* o Docker com o *Docker Engine*. Neste projeto procurou-se analisar o desempenho da emulação de um sistema distribuído escalável em ambas as tecnologias e concluir qual a melhor solução de virtualização para este tipo de sistema.

1 Introdução

Dada a complexidade e as dificuldades associadas ao teste de protocolos distribuídos para controlo de equipamentos de rede num contexto real, é comum recorrer a emulação para teste dos mesmos. Os primeiros emuladores recorriam a máquinas reais, interligadas através de um sistema de emulação do tempo de trânsito e da perda de pacotes numa rede real [14].

Com o desenvolvimento das tecnologias de virtualização, seja esta através de máquinas virtuais ou de *containers*, tornou-se atractivo substituir as máquinas reais e os equipamentos de comutação por sistemas virtualizados. Um dos sistemas mais populares que seguem a abordagem com base em *containers* é o sistema MiniNet [12], para emulação de redes estruturadas segundo o paradigma Software Defined Networking [10] e o protocolo OpenFlow [13]. Outros sistemas, que utilizam outros protocolos e outros tipos de equipamentos de comutação, utilizam para o mesmo efeito máquinas virtuais [4].

No quadro de um projecto em que se pretende testar protocolos de controlo de equipamentos de rede diferentes do OpenFlow, e que utilizam protocolos distribuídos com semântica de falhas bem definidas, da classe dos usados em geo-replicação e bases de dados distribuídas, coloca-se o problema de decidir qual a tecnologia de virtualização a utilizar: máquinas virtuais (VMs) ou *containers*.

Este artigo apresenta um estudo empírico comparativo das tecnologias: máquinas virtuais implementadas sobre o *hypervisor* ESXi da VMWare e *containers* Docker suportados no interior de VMs. A execução de *containers* no interior de VMs, parecendo um contra-senso, não o é de facto pois a maioria das plataformas de grande escala (e.g., *clouds*) hoje acessíveis, não disponibiliza acesso nativo partilhado a *containers* por duas ordens de razões: segurança e isolamento dos diferentes *tenants* quando usam *containers*, e custo associado à existência de uma infraestrutura não virtualizada separada da ”principal”.

Para proceder ao estudo aqui apresentado foi necessário desenvolver de raiz um *benchmark* novo, pois os *benchmarks* geralmente usados em estudos semelhantes, não estudam nem a escalabilidade do número de nós emulados, nem a escalabilidade da solução quando existe um enorme volume de tráfego na rede. Tanto quanto é do nosso conhecimento, estes são traços que diferenciam este estudo de outros semelhantes.

O artigo está assim organizado. Na seção 1 apresenta-se uma breve comparação entre as tecnologias de virtualização de *containers* e VMs. De seguida apresenta-se o *benchmark* construído e clarifica-se a motivação para o seu desenvolvimento. Na quarta seção são expostos os testes realizados e apresentados os seus resultados. Posteriormente são discutidos os resultados obtidos na seção anterior e retiradas conclusões. Na seção 6 são analisados trabalhos relacionados. Por último apresentam-se as conclusões, refletindo sobre possíveis melhoramentos e trabalho futuro.

2 Máquinas virtuais versus *containers*

A virtualização, nomeadamente sob a forma de Máquinas Virtuais (VM), é uma tecnologia que se encontra presente em múltiplos sistemas em produção, particularmente em centros de dados. Esta tem progredido maioritariamente devido à sua vasta utilização no contexto empresarial. As primeiras ocorrências de virtualização surgiram na década de 60 e atualmente ainda se encontram em desenvolvimento [6].

No entanto, recentemente surgiram novas tecnologias de virtualização com a introdução dos *Linux containers* em 2008 [1]. Esta nova tecnologia trouxe consigo uma solução mais leve em termos dos recursos exigidos, sacrificando isolamento e segurança.

Uma das soluções mais populares de virtualização por *containers* é disponibilizada pela plataforma Docker [2] e executada num ambiente Linux. A figura 1, retirada de [3], apresenta uma visualização sintética das duas tecnologias e das principais diferenças, que são em seguida apresentadas em mais detalhe.

2.1 Máquinas Virtuais

Num ambiente de virtualização com recurso a máquinas virtuais o *hypervisor* é o componente principal. Como podemos observar na figura 1, este componente é fundamental

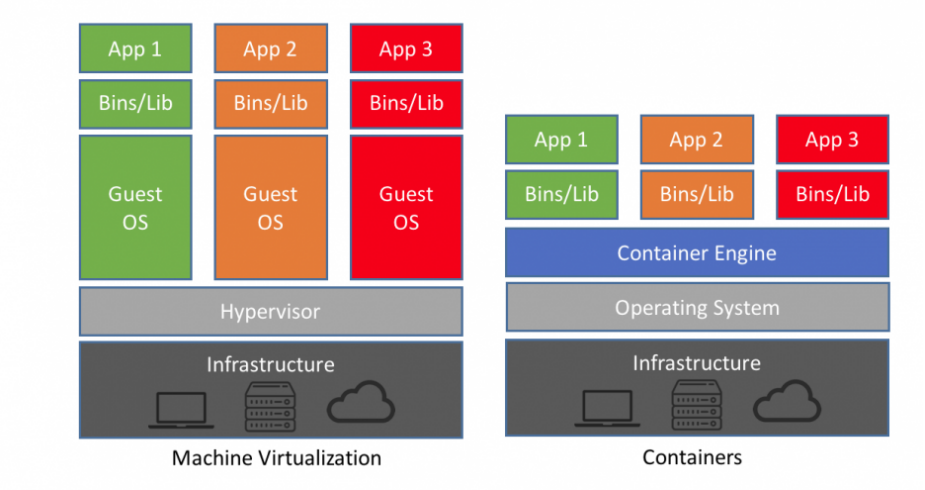

Figura 1: Arquitetura da virtualização em VMs e *containers*
[3]

pois é responsável por ocultar o *hardware* real e disponibilizar sob a forma de *hardware* virtual (que nalguns casos, mas não em todos, tem exatamente as mesmas propriedades que o real) os recursos afetados à máquina virtual. O isolamento entre máquinas virtuais é um ponto forte desta solução. Este isolamento garante (à partida) que se um serviço for comprometido, outras máquinas virtuais instanciadas no mesmo servidor, e até o próprio *hypervisor* não serão também comprometidas.

De uma forma simplificada, podemos dizer que a execução de aplicações numa VM decorre na sua maior parte sem intervenção do *hypervisor*; contudo, a execução de certas instruções privilegiadas obriga à sua intervenção, o que contribui com algum *overhead* na execução.

2.2 *Containers*

A tecnologia dos *containers* baseia-se numa virtualização ao nível do sistema de operação (SO), inicialmente baseada nos *control groups* e *kernel namespaces* presentes no Linux. Isto permite que várias instâncias de *containers* possam ser executadas sobre o mesmo *kernel*, evitando assim a camada de virtualização do *hardware* presente nas máquinas virtuais (observável na Figura 1). Assim, e ao contrário do que acontece nas VMs, a execução de tais instruções privilegiadas, *e.g.*, as utilizadas em leituras do disco, são efetuadas diretamente no SO do *host*. A tecnologia Docker (*Docker engine*) domina atualmente o mercado no que toca a este tipo de virtualização. No entanto a segurança da mesma ainda é um fator de preocupação devido à falta de isolamento entre processos.

[por discutir] → rever isto, elaborar com a tabela, falar de Copy On write

[1] O ambiente Docker está transportado para outros sistemas de operação mas geralmente recorre igualmente a máquinas virtuais, podendo ser considerada uma solução mista

Tópico	*Containers*	Máquinas Virtuais
Virtualização	Virtualização ao nível do sistema de operação	Virtualização ao nível do *hardware*
Inicialização	Cerca de 500ms	Cerca de 20s
Versatilidade	Numa solução nativa é restrito ao ambiente Linux [1]	Independente do sistema de operação
Isolamento	Baseado em *namespaces* e *Cgroups*	Isolamento total
Maturidade da tecnologia	Pouco empregue em contexto empresarial	Presente na maioria dos sistemas de virtualização em produção
Chamadas de sistema	Chamadas de sistema diretas ao SO do *host*	Chamadas ao SO *guest* e posteriormente ao *hypervisor*
Utilização de disco	Apenas possui dependências dos executáveis e de um *filesystem*	Possui todos os ficheiros presentes num sistema de operação

Tabela 1: Principais diferenças entre *containers* e máquinas virtuais

2.3 Conclusões

Em suma, as principais diferenças entre as duas opções de virtualização resumem-se no consumo de recursos por instância, isolamento entre instâncias e no nível de abstração. É importante salientar que, embora não sejam aspetos decisivos, o tempo de arranque, o espaço consumido em disco e ferramentas disponibilizadas pelas tecnologias *e.g.*, Docker Hub e VMWare vSphere são fatores a considerar ao escolher uma destas tecnologias.

3 Benchmarks e metodologia de testes

3.1 Motivação

O controlo de redes ao nível do chamado *Control Plane* (protocolos e subsistemas responsáveis pelo encaminhamento, gestão, optimização das redes) baseia-se, por razões relacionadas com a necessidade de um elevado desempenho e de simplicidade, em protocolos de coordenação entre as diferentes componentes de rede (e.g. *switches, routers, controladores* . . .) com níveis de consistência do tipo "consistência eventual". Por essa razão, o estado propagado ou computado pelas diferentes componentes da rede transita entre estados consistentes, passando por períodos de inconsistência mais ou menos prolongados como é comum quando se utilizam protocolos como OSPF, IS-IS, [7], BGP [11,9], OpenFlow [13].

O trabalho descrito neste documento integra-se num projecto em que se procura substituir esse nível de *consistência eventual* por protocolos baseados em níveis de consistência com propriedades bem definidas e sem períodos intermédios de inconsistência, em redes baseadas no paradigma SDN (*Software Defined Networking*). Nomeadamente, no que diz respeito aos protocolos de troca de estado, eventos e comandos entre componentes. Um dos objectivos do projecto passa por substituir as funcionalidades em parte desempenhadas pelo protocolo OpenFlow, por protocolos da mesma natureza que os protocolos usados para replicar estado em sistemas de bases de dados, numa abordagem como a descrita em [5].

Dados os objectivos do projecto, e tendo em consideração que o principal obstáculo a vencer será o de conseguir responder ao desafio de obter um desempenho adequado, a avaliação realista do desempenho, em redes de média ou alta complexidade, é de uma grande importância. Para isso serão necessários emuladores de rede recorrendo a virtualização, dado ser esta a forma mais realista de proceder a essa avaliação.

O sistema a emular deverá ter um elevado número de *switches, routers,* controladores . . . a comunicarem intensamente entre si, utilizando protocolos da mesma natureza que os dos sistemas de bases dados. Para fazer o estudo comparativo da adequação da emulação com base em máquinas virtuais versus baseadas em *containers*, revelou-se necessário utilizar um *benchmark* específico, com características bastante diferentes dos em que esses estudos comparativos têm geralmente lugar, *c.f.* Secção 6. Assim, essa aplicação de teste deve ser caracterizada por uma troca intensa de dados entre as diferentes componentes do sistema, através de protocolos do tipo dos usados em base de dados, e simultaneamente, executar em cada componente operações também da mesma natureza que as requeridas pela execução de actualizações de tabelas de bases de dados.

3.2 Descrição do *benchmark*

Visando este projecto a comparação das soluções de virtualização de um ambiente distribuído com as características atrás indicadas, foi desenhada de raiz uma aplicação representativa de um sistema dessa natureza. Como indicado na seção 6, não foi possível utilizar os mesmos *benchmarks* que são usados em estudos semelhantes. A aplicação desenvolvida baseia-se num conjunto de nós em que cada nó possui uma base de dados local. Todos os nós têm acesso à base de dados local, como também à base de dados de todos os outros.

Após uma fase de inicialização, cada nó começa a inserir tuplos nas bases de dados de todos os nós, incluindo a sua cópia local. A aplicação termina quando todos os nós tiverem na base de dados local a totalidade dos tuplos inseridos por todos os nós, os quais são necessariamente indênticos.

Ao variar o débito das inserções e o número de nós do sistema distribuído, é possível analisar a capacidade de emulação de uma rede virtual como também observar quais os limites desta.

O servidor onde foram efetuados os *benchmarks* é composto por dois processadores *Intel Xeon E5-2670 v3* a 2.30GHz com 24 núcleos físicos (48 lógicos) cada, 128GB de RAM DDR4 2400MHz. Este executa o *hypervisor* VMWare ESXi 6.5.

Para os testes com *containers*, foi instanciada no mesmo servidor uma VM que permite os testes correrem num ambiente semelhante a um sistema *cloud*, onde os *contai-*

ners executam sobre uma VM instanciada por questões de segurança. As especificações desta VM consistem em 24 *cores* físicos (48 lógicos) e 94GB de RAM. Foi observado que o sistema, durante os testes, não saturava a memória alocada. O sistema de operação da VM onde foram executados os *benchmarks* foi o CentOS 7. Nos *containers* é usado MySQL 14.14, Python 3.5.3 e como sistema operativo o Debian 9. As versões de MySQL são distintas.

Para os testes de VMs, foram instanciadas tantas VMs quantos os números de nós parametrizados. Cada VM possuía 4 núcleos do CPU e 3 GB RAM. As ferramentas empregues foram: MySQL 15.1, Python 3.4.8 e como sistema de operação CentOS 7.

Em ambos os ambientes foi implementado um *ramdisk* na diretoria de escrita do MySQL, de forma a eliminar o *overhead* introduzido por leituras e escritas no disco.

3.3 Funcionamento da aplicação

Os seguintes parâmetros caracterizam uma execução do *benchmark*:

- Seja D o débito da aplicação, isto é, o número de tuplos gerados por unidade de tempo em cada instância
- Seja AG o fator de agregação dos tuplos inseridos de uma só vez nas bases de dados remotas
- Seja N o número de nós que compõe o *benchmark*; a Figura 2 ilustra a topologia da rede com $N = 6$
- Seja *master node* o elemento do sistema cujo endereço IP é o menor
- Seja T um tuplo composto por dois elementos, uma chave aleatória com probabilidade de colisão computacionalmente nula, e um número aleatório de 0 a 100

A aplicação desenrola-se em 3 fases:

Sincronização A fase de sincronização possui como objetivo coordenar todas as instâncias, visando a entrada sincronizada do sistema no ciclo principal. O processo de sincronização consiste no estabelecimento de ligações entre os N elementos (formando uma rede em malha). Posteriormente cada nó efetua uma inserção na tabela de coordenação pertencente à base de dados do *master node*. Em seguida cada nó consulta periodicamente esta tabela. No momento em que o número de entradas for igual ao número de nós, a fase de inicialização termina.

Ciclo principal O ciclo principal tem como objetivo simular um sistema distribuído que efetua uma grande quantidade de trocas de mensagens continuamente. De acordo com o débito D, serão gerados tuplos e inseridos na base de dados local. A cada AG inserções locais serão inseridos os mesmos tuplos nas restantes $N - 1$ bases de dados remotas.

Finalização No final da fase anterior cada nó remove o tuplo inserido durante a fase de sincronização (presente na tabela de coordenação do *master node*). Em seguida irá consultar periodicamente essa tabela até que esta esteja vazia. Atingido este ponto, cada nó termina a sua execução. O benchmark é concluído após todos os nós terminarem.

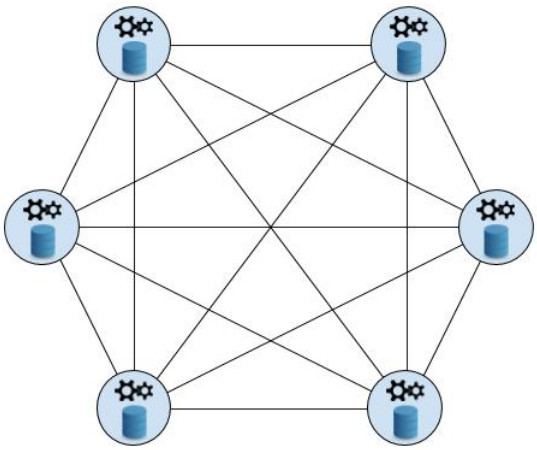

Figura 2: Topologia da rede do *benchmark* com $N = 6$

4 Testes

N

Foram realizados diversos testes que estudam o impacto da variação do débito e do número de nós do sistema no tempo de execução. Um conjunto de testes foi efetuado num modo que designamos por *baseline*. Este consiste em eliminar toda a comunicação entre nós do sistema, visando analisar o *overhead* introduzido pela troca de mensagens (e consequentemente, da emulação da rede virtual). No entanto, a actividade de geração e inserção de tuplos na base de dados é preservada, realizando-se um número total equivalente de gerações e inserções de tuplos.

Em todos os testes foi fixado o fator de agregação ($AG = 100$), referido em **??**, uma vez que se concluiu que este parâmetro apenas é relevante para valores baixos (de 1 a 5).

Todos os testes foram executados em ambos os ambientes (*containers* e VMs). Os parâmetros de teste são os seguintes:

- Variação de débito com 15 nós e todas as inserções locais (**teste 1**: *containers*, D variável, $N = 15$ e modo *baseline*)
- Variação do número de nós com cada nó a gerar 5000 tuplos por minuto que insere N vezes localmente (**teste 2**: *containers*, D = 5000, N variável, e modo *baseline*)
- Variação de débito (**teste 3**: *containers*, D variável, $N = 15$)
- Variação do número (**teste 4**: *containers*, D = 5000, N variável)

O comportamento esperado do sistema varia de acordo com o número de operações que este executa por unidade de tempo. Este número é, no essencial, proporcional ao número de inserções na base de dados. A equação que define o número total de inserções por unidade de tempo é a seguinte:

$$I = N^2 * D \tag{1}$$

4.1 Resultados

Nesta secção serão apresentados os resultados obtidos pelos testes apresentados acima. Cada gráfico relaciona o parâmetro que varia com o desvio (em percentagem) do tempo de execução esperado, *i.e.*, o tempo definido pelo débito D admitindo que as operações são realizadas sem atrasos.

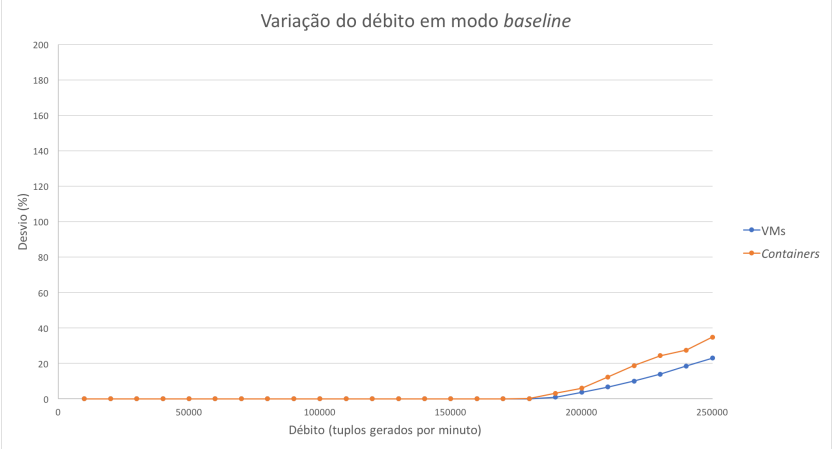

Figura 3: **Teste 1** (*containers* e VMs, D variável, $N = 15$ e modo *baseline*)

Figura 4: **Teste 2** (*containers* e VMs, $D = 5000$, N variável, e modo *baseline*)

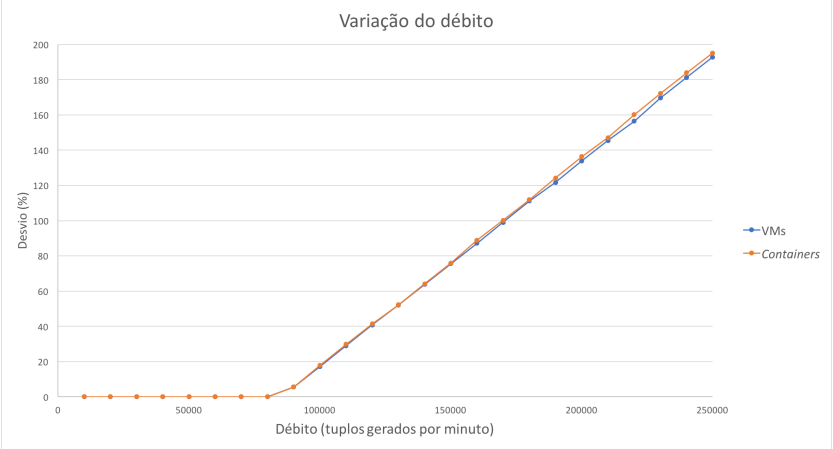

Figura 5: **Teste 3** (*containers* e VMs, D variável, $N = 15$)

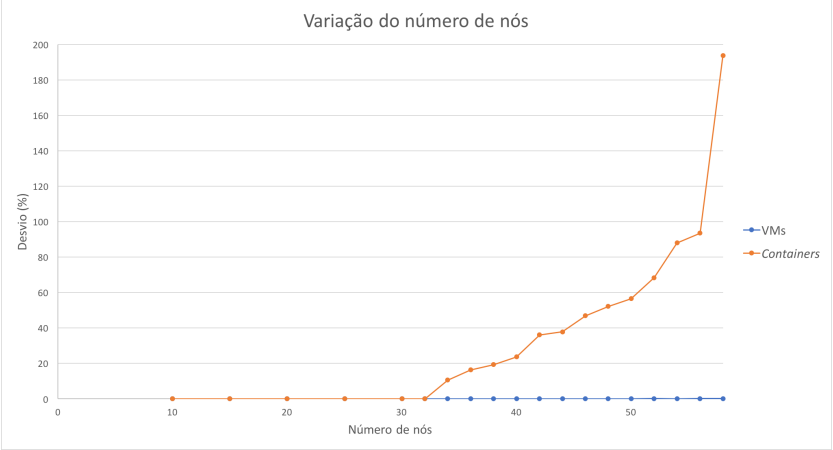

Figura 6: **Teste 4** (*containers* e VMs, $D = 5000$, N variável)

5 Discussão de resultados

5.1 *Baseline*

Na Figura 3 encontra-se representado o comportamento de ambos os ambientes de emulação fazendo variar as gerações e inserções por minuto sem comunicação. Pode-se verificar que fixando o número de nós, os testes exibem um incremento linear, tal como esperado de acordo com a equação anteriormente apresentada. A diferença entre as duas soluções manifesta-se quando o débito é superior a 180.000 tuplos gerados por minuto. Esta deve-se, provavelmente, à camada adicional de virtualização (*containers* sobre uma VM) que introduz *overhead* suplementar.

O gráfico da Figura 4 não apresentam qualquer desvio do tempo de execução esperado. Visto isto, nos gráficos onde é introduzida a comunicação, caso os resultados se distanciem dos obtidos nos testes *baseline*, conclui-se que a diferença se deve á emulação da rede virtual.

5.2 Variação do débito *D*

Observando a Figura 5 verifica-se que mantendo um número constante de nós o desvio exibe um incremento linear, de acordo com a equação (1). Comparando os dois gráficos, observa-se que a utilização das duas tecnologias de virtualização testadas não modifica significativamente os resultados obtidos. Desse modo pode concluir-se que o fator que limita a fiabilidade da emulação tem o mesmo peso nos testes realizados nos dois ambientes de virtualização.

5.3 Variação do número de nós *N*

Na figura 6 verifica-se que a partir de 32 nós o desvio assemelha-se a uma curva quadrática, semelhante á equação (1). Em contraste, a Figura **??** não apresenta o mesmo comportamento, mantendo-se o desvio quase nulo. Uma possível explicação deste resultado, é o *overhead* suplementar introduzido devido aos *containers* estarem a executar sobre uma máquina virtual. Essa arquitetura parece prejudicar a solução de *containers* usada nos ambientes de *cloud* para assegurar um nível superior de isolamento. Outra possível origem da disparidade dos dois resultados é uma virtualização de rede de qualidade inferior no caso do *Docker Engine*.

Conclui-se, com base nestes gráficos, que a solução de máquinas virtuais possui, neste ambiente, uma escalabilidade superior quando o número de nós aumenta.Tal constitui, aparentemente, um resultado surpreendente e contrário à intuição geralmente prevalente na literatura.

6 Trabalho relacionado

A tecnologia da virtualização está em constante desenvolvimento, e o amadurecimento da tecnologia de conteinerização foi acompanhado de muitos estudos comparativos entre a mesma e a tecnologia, mais madura, de máquinas virtuais. No entanto, a maioria desses estudos senão todos, focam-se na análise comparativa do desempenho de ambas as tecnologias, utilizando *benchmarks* que avaliam, por exemplo, os custos da escrita em disco, da compressão de ficheiros, do desempenho de algoritmos no consumo da CPU, da gestão da memória e seu dinamismo, do acesso à rede, etc. procurando sempre realizar a comparação focando-se num recurso isolado e usando aplicações centralizadas ou quase sempre centralizadas.

São exemplos destes estudos os seguintes: ????????

São também comuns comparações entre a execução de aplicações em modo nativo, directamente sobre o hardware, com as mesmas aplicações a executarem em máquinas virtuais, como por exemplo os estudos referidos em [6]. ?????? ..... No entanto este tipo de comparação não se adequa aos nossos objectivos, pois a comparação deveria

incidir sobre o comportamento de sistemas distribuídos escaláveis, o que exige o uso de múltiplos nós.

Existe uma tradição de utilização de *containers* suportados directamente no sistema Linux para emulação de redes. O exemplo mais conhecido é o Mininet [12], cuja adequação e escalabilidade tem sido bastante estudada [8]. No entanto, este sistema tem diferenças relativamente aos objectivos do nosso estudo. Por um lado, o sistema depende directamente de módulos no Kernel do Linux que implementem *switches* virtuais controlados directamente por OpenFlow. Desta forma ficamos limitados a um único tipo de *control plane*, isto é, um que se baseie em SDN implementada com o protocolo OpenFlow. Por outro lado, tal como no nosso caso, os estudos não abrangem a problemática do isolamento, isto é, não avaliam comparativamente a utilização de *containers* com ou sem VM de suporte. Adicionalmente, o ambiente de orquestração das componentes da rede emulada é específico e provavelmente menos geral e flexível que o disponibilizado pela plataforma Docker.

7 Conclusões e trabalho futuro

Neste projeto foram avaliadas duas soluções de emulação de sistemas distribuídos escaláveis. Em ambos os casos foi testada a eficiência da implementação da rede virtual através de testes que simulavam uma troca intensiva de mensagens pela rede.

No ambiente de testes usado, verificou-se um comportamento semelhante das duas soluções de virtualização quando não se utiliza a emulação da rede. Em contrapartida, quando esta emulação é usada, as VMs têm um desempenho superior quando o número de nós aumenta. A explicação deste resultado necessita de ser aprofundada, mas tal não nos impede de retirar desde já várias conclusões imediatas.

Para começar, é importante frisar que o desenvolvimento do projeto provou ser mais ágil na plataforma Docker. Em contraste, no ambiente de VMs foi necessário criar vários *scripts* auxiliares para automatizar o processo de instalação e de realização dos testes.

No decorrer do desenvolvimento foram estudadas e testadas várias ferramentas disponibilizadas pelas plataformas, como por exemplo Docker Swarm, Docker Compose, bem como o sistema de operação Photon OS da VMWare. No entanto, estas ferramentas não foram utilizadas pois foi possível alcançar um controlo mais fino através de soluções específicas desenvolvidas por nós.

Surgiram obstáculos relativos á utilização do MySQL devido á natureza ACID das transações, o que provocou um *bottleneck* na utilização do disco. Foi discutido o aumento do número de *threads* de escrita na base de dados visando melhorias do desempenho. No entanto, a ideia foi descartada devido ao facto de que o número de *threads* total ser já consideravelmente maior que o número de núcleos do processador. Visto que a persistência da base de dados não era um fator importante, foi utilizado um *ramdisk* com o objetivo de remover este *bottleneck*.

Um outro imprevisto surgiu na fase de sincronização das máquinas virtuais que demorava cerca de 40 vezes mais do que no Docker, devido a problemas na resolução de nomes. Este imprevisto resultou no atraso da obtenção de resultados na fase mais crítica do trabalho.

Existe lugar para trabalho futuro imediato, nomeadamente testar os *containers* sobre PhotonOS e sem VM intermédia. Estes testes permitirão aprofundar a origem do *overhead* suplementar verificado, assim como aclarar a problemática da viabilidade da plataforma Docker com ou sem isolamento forte. Relativamente às ferramentas desenvolvidas, a documentação das mesmas deve ser melhorada antes de as disponibilizarmos publicamente. Adicionalmente, a bateria de testes poderá também ser alargada de modo a clarificar o comportamento do sistema quando o desvio do tempo de execução começa a aumentar.

Tal como alertados pelos orientadores, a investigação em sistemas reveste-se de inúmeros obstáculos e problemas concretos a resolver, antes de se chegarem a conclusões credíveis e verificáveis publicamente sobre o comportamento dos sistemas, o que foi para nós algo completamente novo.

Referências

1. `https://en.wikipedia.org/wiki/LXC`, 2018. [accessed 25-June-2018].
2. `https://www.docker.com`, 2018. [accessed 25-June-2018].
3. `https://blog.netapp.com/blogs/containers-vs-vms/`, 2018. [accessed 25-June-2018].
4. G. Apostolopoulos and C. Hassapis. V-em: A cluster of virtual machines for robust, detailed, and high-performance network emulation. In *14th IEEE International Symposium on Modeling, Analysis, and Simulation*, pages 117–126, Sept 2006.
5. B. Davie, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. Gude, A. Padmanabhan, T. Petty, K. Duda, and A. Chanda. A database approach to sdn control plane design. *SIGCOMM Comput. Commun. Rev.*, 47(1):15–26, Jan. 2017.
6. W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, volume 00, pages 171–172, March 29-31 2015.
7. P. Francois, C. Filsfils, J. Evans, and O. Bonaventure. Achieving sub-second igp convergence in large ip networks. *ACM SIGCOMM Computer Communication Review*, 35(3):35–44, 2005.
8. N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, pages 253–264, New York, NY, USA, 2012. ACM.
9. E. K.-B. e. a. JP John. Consensus routing: The internet as a distributed system. In *Proceedings of the 5th NSDI USENIX Symposium*, NSDI '08. USENIX, 2008.
10. K. Kirkpatrick. Software-defined networking. *Commun. ACM*, 56(9):16–19, Sept. 2013.
11. C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed internet routing convergence. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '00, pages 175–187, New York, NY, USA, 2000. ACM.
12. B. Lantz, B. Heller, and N. McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.
13. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

14. A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. *SIGOPS Oper. Syst. Rev.*, 36(SI):271–284, Dec. 2002.