Kollaps: Emulação Dinâmica e Descentralizada de Topologias de Rede

Paulo Gouveia[1], João Neves[1], Luca Liechti[2], Valerio Schiavoni[2], and Miguel Matos[1]

[1] INESC ID & IST, University of Lisbon, Portugal.
Email: miguel.marques.matos@tecnico.ulisboa.pt
[2] University of Neuchâtel, Switzerland.
Email: valerio.schiavoni@unine.ch

Resumo O comportamento de aplicações distribuídas de grande escala é fortemente influenciado pelas propriedades da rede como latência, largura de banda e perda de pacotes. É, portanto, fundamental averiguar o desempenho destas aplicações de uma forma sistemática e reproduzível em cenários controlados e potencialmente adversos como perda de pacotes em restrições de rede.
Infelizmente, o estado da arte em emulação de redes e plataformas de teste apresentam várias lacunas tais como escalabilidade limitada (ex.: MiniNet), falta de suporte a dinamismo na rede (ex.: EmuLab), ou pelo foco exclusivo no plano de dados (ex.: CrystalNet).
Neste artigo propomos o Kollaps, um emulador descentralizado, agnóstico da aplicação e protocolo de transporte, e capaz de escalar a milhares de processos mantendo uma precisão equivalente às abordagens estado-da-arte centralizadas. O Kollaps ultrapassa as limitações no estado da arte partindo de dois princípios. Primeiro, para as aplicações o importante são as propriedades fim-a-fim e não o estado dos *routers* e *switches* que compõem a rede. Segundo, esta observação permite-nos construir um modelo de emulação simplificado que pode ser mantido de uma forma distribuída, e escalando assim com o número de máquinas requeridas pela aplicação. A nossa avaliação usando micro-benchmarks e aplicações reais demonstra a escalabilade e precisão do Kollaps.

1 Introdução

Avaliar sistemas distribuídos de grande escala é difícil, lento e dispendioso. Imaginemos que se quer executar e avaliar o impacto que uma dada alteração numa aplicação tem sobre o seu desempenho ou no comportamento da rede. O problema da execução é parcialmente resolvido por containers (*e.g.*, Docker [29], Linux LXC [7]) e orquestradores de containers (*e.g.*, Docker Swarm [4], Kubernetes [6]) que permitem fazer um *deployment* de modo simplificado. O maior desafio é, portanto, a avaliação sistemática de aplicações. A inerente variabilidade da rede torna difícil averiguar o impacto de alterações nas aplicações. No

entanto, é fundamental conseguir responder a questões como: Será que a melhoria de desempenho foi realmente proveniente das alterações efetuadas ou foi apenas uma execução que, por sorte, apanhou melhores condições de rede?

Estas mesmas questões surgem no dilema da reprodutibilidade que atualmente atormenta a comunidade de sistemas distribuídos [11,33]. Avaliando o mesmo sistema podemos obter resultados diferentes não só por as experiências serem executadas sobre ambientes diferentes, não totalmente controláveis, mas também porque *testbeds* como o Emulab [20], CloudLab [36] ou PlanetLab [13] onde tais experiencias são conduzidas tendem a ficar sobrecarregadas por volta das datas de conferências [23]. Existe, portanto, a necessidade de ferramentas capazes de garantir reprodutibilidade e uma avaliação sistemática de aplicações de larga escala. . Uma abordagem possível é simulação que se baseia em modelos que capturam algumas propriedades específicas do ambiente e sistema alvo [9]. Os simuladores oferecem completo controlo dos sistemas e ambiente (e por isso são completamente reproduzíveis) e possibilitam o estudo dos sistemas sobre diversos cenários. No entanto, os simuladores sofrem de alguns problemas bastante conhecidos como a significativa distância entre o modelo simulado e o sistema real o que resulta em vários comportamentos observados no mundo real não serem capturados pela simulação [12,19,18,32].

A alternativa é recorrer a emulação. Nos emuladores o sistema real é executado sobre um modelo da rede que replica o comportamento do mundo real modelando a topologia da rede com os seus componentes internos como *switches*, *routers* e o comportamento destes. Emulação permite, portanto, chegar a conclusões quanto ao comportamento do sistema real num cenário concreto em vez de apenas quanto a um modelo do sistema. Infelizmente, os emuladores do estado da arte atual sofrem de várias limitações, das quais se destaca a escalabilidade.

O Kollaps ultrapassa as limitações do estado-da-arte através de três pontos. Primeiro, da perspetiva de aplicações distribuídas, as propriedades de fim-a-fim como latência, largura de banda ou perda de pacotes, são mais importantes para o seu comportamento que o estado de cada componente da rede que origina essas propriedades. Segundo, é possível manter este modelo de emulação de forma distribuída e com precisão, o que permite a emulação escalar com o número de nós da aplicação sem sacrificar precisão. E finalmente, este modelo simplificado permite rápidas alterações ao mesmo, o que possibilita a emulação de eventos dinâmicos como remoção de ligações ou adição de tráfego em segundo plano.

Contribuições. As nossas contribuições principais são: *(i)* Kollaps, o primeiro emulador de topologias de rede que permite a avaliação de aplicações de larga-escala em redes dinâmicas; *(ii)* a integração do Kollaps com Docker Swarm [4] e Kubernetes [6] permitindo execuçar e avaliar aplicações baseadas em containers sem modificações; *(iii)* avalia sobre vários cenários, incluindo topologias dinâmicas de aplicações reais; *(iv)* uma amostragem dos novos tipos de experiências que o Kollaps permite. Nomeadamente, um exemplo com Cassandra que mostra como esse sistema é afetado por diferentes características da rede como se executado no AWS [2].

Organização. Este artigo está organizado da seguinte forma. Examinamos trabalho relacionado em §2. A arquitetura do KOLLAPS é descrita em §3. Em §4 apresentamos a avaliação do KOLLAPS. Por fim, apresentamos uma breve conclusão e as limitações atuais em §5.

2 Trabalho Relacionado

Nesta secção, por falta de espaço, detalhamos apenas como o KOLLAPS se compara com sistemas representativos e consideramos simuladores (*e.g.*, NS-3 [37], PeerSim [30]) fora do âmbito deste artigo.

Abordagens *user-space*. O Trickle [16] usa as funcionalidades de *linking* dinâmico e pré-carregamento de sistemas baseados em Unix para inserir o seu código entre binários não modificados e chamadas de sistema à API das sockets. O Trickle implementa atrasos e manipulação de largura de banda antes de passar o trafego nas chamadas às sockets. Apesar de múltiplas instancias poderem cooperar entre si, configurar um sistema com varias máquinas para emular redes de grandes dimensões envolve significativo trabalho manual uma vez que não existe um ponto central a partir de onde executar experiências. Em contraste, o KOLLAPS é independente da aplicação a ser testada porque trabalha com binários não modificados, quer dinâmica ou estaticamente ligados.

O SPLAYNET [39] possibilita a emulação de topologias de rede arbitrarias, executadas sobre múltiplas máquinas físicas de forma completamente distribuída. Para emular a topologia, SPLAYNET analisa o grafo da rede e utiliza algoritmos de emulação distribuídos, colapsando a topologia interna e trocando pacotes diretamente entre nós emulados. No entanto, está limitado a uma linguagem específica da plataforma, não sendo compatível com aplicações existentes, nem suporta redes dinâmicas. O KOLLAPS também adota uma abordagem descentralizada mas ultrapassa todas as limitações do SPLAYNET.

A bordagens *kernel-space*. De seguida detalhamos emuladores de rede que requerem explicito suporte do sistema operativo e kernel. O Dummynet [38] opera diretamente sobre uma dada interface de rede. É usado como uma ferramenta de baixo nível para construir *testbeds* com o Modelnet [40]. O Modelnet é uma *testbed* que permite o teste de aplicações não modificadas. Estas aplicações são executadas em nós *edge* e todo o tráfego é redirecionado por um conjunto de *core routers* – máquinas dedicadas que coletivamente emulam as propriedades da rede desejada antes de entregarem os pacotes aos nós de destino. O KOLLAPS usa o Linux's Traffic Control (`tc`) [8] para oferecer semelhantes capacidades de controlo da rede, mas (1) sem necessitar de máquinas dedicadas e (2) ao mesmo tempo oferecendo integração com ferramentas de orquestração de containers de grande escala.

O Emulab [42] é uma testbed de emulação que suporta a execução de sistemas operativos fornecidos pelos utilizadores. O Emulab é capaz de correr grandes topologias distribuídas por um cluster, mantendo alocados os recursos escolhidos pelos utilizadores e um agendamento ideal. A técnica utilizada pelo Emulab para comprimir grafos é similar com a abordagem de colapsamento de topologias

do Kollaps, no entanto, não suporta dinamismo de rede e requer um *cluster* dedicado.

Abordagens *container-based*. Por último, abordamos ferramentas de emulação baseadas em containers. O Mininet [25] emula topologias de rede numa única máquina. Ele depende dos mecanismos de virtualização do Linux (*i.e.*, `cgroups`) para emular máquinas separadas na rede. O Mininet é capaz de emular centenas de dispositivos numa única máquina física onde *switches* e *routers* são processos independentes, mas está limitado a uma única máquina física impedindo que este seja usado para testar aplicações de grande escala e com uma utilização intensiva de recursos.

O Maxinet [41] estende o Mininet para suportar containers de Docker e permitir execução de testes em clusters, tirando partido de protocolos de *tunneling* para esconder das instancias o facto de estarem a correr em máquinas separadas. No entanto, isto requer que todos os nós que se ligam ao mesmo *switch* estejam a correr na mesma máquina. O Maxinet possuí também uma fraca escalabilidade [27]. O ContainerNet [34,35] estende também o Mininet para adicionar suporte para containers de Docker e topologias dinâmicas. No entanto, continua limitado por apenas uma única máquina física.

O CrystalNet [27] emula com precisão o *control-plane* de redes de larga escala (*e.g.* tabelas de encaminhamento, software dos *switches* ou *firmware* dos dispositivos) mas não pode emular o *data-plane* (*e.g.* latência, largura de banda), e por isso não é capaz de emular o comportamento de sistemas distribuídos de larga escala.

Tanto quanto conseguimos apurar, o Kollaps é o único sistema que pode ser usado para testar aplicações não modificadas em redes emuladas sem nenhum nó central, mantendo uma precisão ao nível de outros sistemas existentes do estado-da-arte.

3 Kollaps

Nesta secção descrevemos o design do Kollaps. Um ponto chave da nossa abordagem é que o modelo de emulação é mantido de forma completamente distribuída. Em vez de emular diretamente elementos da rede como routers, switches e o seu estado interno, emulamos as propriedades equivalentes da topologia nos nós terminais. A ideia por trás desta abordagem é colapsar a topologia num grupo de caminhos mais curtos entre todos os nós deixando de fora todos os elementos internos da rede e expor à aplicação apenas as resultantes propriedades das ligações originais.

3.1 Emulation Manager

O *Emulation Manager* é o elemento principal da arquitetura do Kollaps. Este é composto por dois componentes: o *Emulation Core* que mantem o modelo emulado durante a experiência, e a *Traffic Control Abstraction Layer* (TCAL), que é responsável por aplicar restrições da topologia calculadas pelo *Emulation*

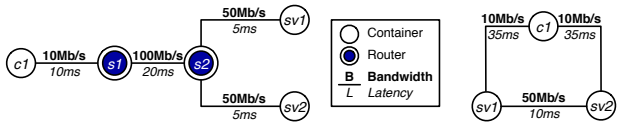

Figura 1: Examplo da colapsação de uma simples topologia.

core. A TCAL implementa uma biblioteca sobre o Traffic Control (tc) do Linux [21]. É responsável pela configuração inicial da rede, recolher informação sobre o uso de, e modificar os máximos disponíveis para a largura de banda nas ligações. Esta biblioteca mantem uma hierarquia de queuing disciplines (`qdiscs`) para aplicar as restrições de rede para cada destino. Por cada destino existe uma `netem qdisc` [17], usada para definir latência, *jitter* e perda de pacotes, e uma `htb qdisc` (hierarchical token bucket) [15], usada para aplicar restrições largura de banda. O *Emulation Core* mantem o modelo da emulação e computa as restrições de rede a impor.

A Figura 1 é um exemplo de colapsação de topologias. Consideremos a completa topologia no lado esquerdo com um cliente (*c1*) e dois servidores (*sv1* e *sv2*). O caminho entre os nós $c1$ e $sv1$ é composto por três ligações diferentes com diferentes propriedades. Estas ligações são colapsadas numa única ligação, cujas propriedades são equivalentes às propriedades fim-a-fim resultantes das três ligações originais. A resultante topologia colapsada pode ser vista no lado direito da Figura 1. Mais precisamente, para a latência e perda de pacotes são somadas e multiplicadas respetivamente as propriedades das ligações para obter as propriedades da ligação virtual. No caso do *jitter* são somadas as variâncias de cada ligação. A largura de banda máxima num caminho é determinada pela ligação com menor largura de banda. No entanto, a largura de banda disponível numa ligação não depende apenas da capacidade física dos caminhos, mas também dos fluxos de pacotes em cada ligação. Em particular, quando a largura de banda requerida por cada fluxo ultrapassa a máxima disponível observa-se congestão e como tal necessitamos um mecanismo para garantir uma justa alocação da largura de banda por cada fluxo. No mundo real quando existe congestão na rede os *routers* e *switches* começam a perder pacotes. Em protocolos de transporte não fiáveis, como o UDP, a perda de pacotes é ignorada. No entanto, em protocolos de transporte fiáveis, como o TCP, a perda de pacotes serve como um sinal para ajustar a taxa de transferência da aplicação fazendo assim com que fluxos concorrentes usem uma percentagem da largura de banda justa. No Kollaps, usamos o modelo RTT-Aware Min-Max [22,28], inspirado pelo TCP Reno [31] para dividir justamente a largura de banda. Este modelo dá-nos uma percentagem da largura de banda que cada fluxo pode usar, sendo esta inversamente proporcional ao *round-trip time do fluxo*.

O modelo é concretizado pelo *Emulation Core* que está implementado em Python v3.7. A execução está dividida em duas fases: inicialização e ciclo de emulação. A fase de inicialização consiste em construir o grafo da topologia colapsada e inicializar a TCAL. Este grafo é a principal estrutura de dados sobre a qual o *Emulation Core* computa, entre outros, os limites de largura de banda.

O ciclo de emulação mantem dados adicionais com a largura de banda usada por cada nó e periodicamente executa os seguintes passos: *(i)* limpar o estado dos fluxos ativos; *(ii)* usar a TCAL para obter a largura de banda utilizada; *(iii)* partilhar com as outras instâncias a largura de banda usada; *(iv)* computar a largura de banda usada em cada caminho e os seus links com a informação recebida no passo anterior; *(v)* aplicar as restrições da largura de banda.

A disseminação de metadados é um aspeto crítico no sistema uma vez que queremos que seja eficiente e rápido, mesmo para aplicações de grande escala. Estes metadados são disseminados usando Aeron [1], um protocolo eficiente e fiável para o transporte de mensagens UDP e IPC. Para *containers* na mesma máquina física, os metadados são trocados por memória partilhada, que é rápido e não tem impacto no uso da rede. A partilha de metadados entre máquinas físicas é feita através de mensagens de reliable unicast UDP do Aeron. Cada uma destas mensagens contem a seguinte informação: *(i)* número de fluxos, 2 bytes; *(ii)* lista com as larguras de banda usadas por cada fluxo, 4 bytes por fluxo; *(iii)* número de ligações; *(iv)* lista com os identificadores das ligações. Estes metadados são usados no passo *(iv)* acima para atualizar o modelo de emulação.

4 Avaliação

Nesta secção avaliamos o Kollaps com uma série de micro e macro experiências no nosso cluster. Para verificar a validade da nossa abordagem face a cenários realistas, comparámos o comportamento de aplicações em Kollaps e EC2 da Amazon. No geral os nossos resultados mostram que: *(1)* Kollaps escala linearmente com o número de fluxos e containers e tem um custo constante independente da largura de banda usada pelas aplicações *(2)* executar uma aplicação num cluster com Kollaps ou no EC2 da Amazon gera resultados semelhantes *(3)* a precisão da emulação do Kollaps é comparável com outras ferramentas que emulam todo o estado da rede como o Mininet.

Configurações de avaliação. O nosso cluster é composto por 5 servidores 5 Dell PowerEdge R630, com 64 cores Intel Xeon E5-2683v4 com frequência 2.10 GHz, 128 GB de RAM, com Ubuntu Linux 16.04 LTS, kernel v4.4.0-127-generic e ligados por um switch Dell S6010-ON de 40 GbE. Os testes executados na EC2 da Amazon usam instâncias `r4.16xlarge`, o mais perto das configurações de hardware do nosso cluster. Nos testes usamos a versão estável mais recente do Mininet (v2.2.2) e do Maxinet (v1.2).

4.1 *Overhead* do tráfego de Metadados

O Kollaps depende de disseminação de metadados para modelar e emular restrições de largura de banda para fluxos concorrentes. Nesta secção estudamos o custo da disseminação de metadados com uma topologia em altere onde todos os caminhos entre os lados opostos do altere partilham um link. Usamos iPerf [5] para gerar um continuo fluxo de TCP de $50Mbit/s$, a capacidade máxima do link partilhado. Cada configuração é identificada por um tuplo `F/C` onde `F` é o

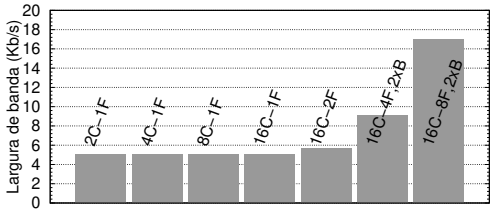

Figura 2: Uso de largura de banda para um incremental número de fluxos e containers (1 fonte e 1 destino por fluxo). `F`: fluxos concurrentes, `C` número total de containers.

número de fluxos concorrentes e `C` o número total de containers. Os resultados para um cluster de 2 máquinas é mostrado na Figura 2.

Podemos observar que a largura de banda não está relacionada com o número de containers uma vez que as mensagens são trocadas apenas entre *Media Drivers*. A largura de banda usada para troca de metadados aumenta com o número de fluxos, uma vez que aumentar o número de fluxos significa que existe mais informação a ser partilhada. No entanto, a largura de banda usada pela troca de metadados não aumenta com a largura de banda usada pela aplicação pois as mensagens têm o mesmo tamanho, apenas valores diferentes.

4.2 Regulação Descentralizada de Largura de Banda

O período de disseminação influencia o tempo de reação para regulação do tráfego, em particular quando o número de fluxos concorrentes varia com o tempo. Para medir isto usamos uma simples topologia em altere com 3 servidores e 3 clientes, ilustrada na Figura 3. A ligação entre os *switches* tem $50Mbit/s$ de largura de banda. Os três clientes (`C1`, `C2`, `C3`) têm respetivamente larguras de banda de $50MBit/s$, $50Mbit/s$ e $10Mbit/s$, e RTTs de $50ms$, $40ms$, e $40ms$. A experiencia procede da seguinte forma. No inicio apenas `C1` tem um fluxo ativo e por isso usa toda a largura de banda disponível. Após 10 segundos, `C2` é iniciado passando, portanto, a competir pela largura de banda na ligação partilhada. Por esta altura, uma vez que `C2` tem um menor RTT que `C1`, obtém uma quota da largura de banda proporcionalmente maior, seguindo o modelo descrito na Secção 3. O tempo de reação, ou seja, quanto tempo o Kollaps demora a diminuir a largura de banda disponível ao `C1` por estar a competir com os fluxos do `C2` é apenas 1 segundo. Aos 20 segundos, `C3` é iniciado e chega ao limite de $10Mbit/s$. Sendo a largura de banda dos outros dois clientes é ajustada para lidar com este novo fluxo concorrente.

De seguida, alteramos as propriedades de rede do sistema. Aos 30 segundos, é duplicada a largura de banda da ligação entre os switches. Como `C3` já está a usar o máximo de largura de banda permitido por outra ligação, a largura de banda extra é partilhada entre `C1` e `C2`, novamente de forma inversamente proporcional aos respetivos RTTs. Aos 40 segundos, a ligação entre `C1` e `s1` passa a ter uma latência de 5ms em vez de 10ms. Como na topologia do lado do servidor todas as ligações são idênticas, as propriedades dos caminhos de `C1`

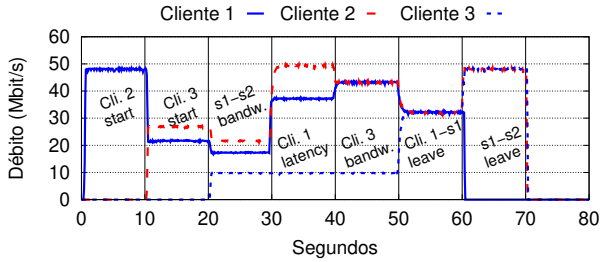

Figura 3: Regulação de largura de banda sob a) diferentes fluxos, b) diferentes propriedades de ligaçao, e c) diferente topologia.

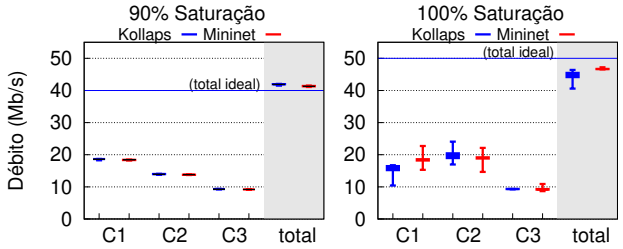

Figura 4: Precisão da emulação do KOLLAPS e Mininet com ligações partilhadas não-saturadas (90%, esquerda) e saturadas (direita)

e `C2` para qualquer servidor são idênticas, o que é refletido pelo facto de as suas larguras de banda se tornarem iguais. Aos 50 segundos, a ligação entre `C1` e `s1` passa também a ter uma largura de banda igual a $100Mbps$. Todas as ligações dos clientes para `s1` são agora idênticas, resultando em três fluxos idênticos. Aos 60 segundos, é removida a ligação entre `C1` e `s1` da topologia, ou seja, o grafo já não está ligado. `C1` já não é capaz de enviar nenhum fluxo para a rede. A quota de largura de banda libertada é partilhada entre `C2` e `C3`. Finalmente, aos 70 segundos, a ligação entre os switches é também removida, deixando de ser observado qualquer tráfego.

4.3 Comparação com Mininet e Maxinet

Nesta secção comparamos o KOLLAPS com o Mininet [26] e o Maxinet [41] em dois cenários distintos. **Pequenas topologias.** Primeiro, consideramos a topologia na Figura 3. Usamos iPerf [5] para estabelecer fluxos contínuos de TCP entre clientes e servidores. As experiências executam por 10 minutos. Os primeiros 60 segundos são descartados como período de aquecimento.

Na primeira experiência, são atribuídas larguras de banda de $20Mb/s$, $15Mb/s$ e $10Mb/s$, aos clientes `C1`, `C2`, e `C3`, respectivamente. O objetivo é observar o comportamento do KOLLAPS e Mininet quando as ligações entre os switches não estão saturadas. Neste cenário, a largura de banda cumulativa usada pelos três clientes é $45Mb/s$ com uma máxima largura de banda de $50Mb/s$ permitida pela topologia. A Figura 4 (esquerda) mostra os resultados num diagrama de caixa e

Tabela 1: Erro quadrático médio exibido nos testes de latência com topologias scale-free com Kollaps, Mininet e Maxinet.

Topology size	Kollaps	Mininet	Maxinet
1000	0.0261	0.0079	28.0779
2000	0.0384	NA	347.5303

fios de bigode. A área cinzenta mostra a taxa de transferência total na ligação partilhada. Neste cenário de baixa contenção, Kollaps e Mininet comportam-se de forma semelhante. Por outro lado, quando existe elevada contenção e os clientes saturam a ligação, no lado direito da Figura 4, podemos ver que os dois sistemas comportam-se de forma diferente. Isto deve-se ao facto do Mininet emular o completo estado da topologia de rede. Neste cenário de elevada contenção, os buffers nos switches emulados enchem e começam a perder pacotes pela diferença entre as disponível e requerida larguras de banda. Como resultado, o TCP tipicamente ajusta a taxa de transferência para as conceções de forma independente, levando a picos e elevada variância. Isto é observável nos fios de bigode dos clientes `C1` e `C2` com o Mininet na Figura 4 (direita) que mostra elevada variância.

Topologias de grande escala. Consideramos também topologias de grande escala gerados através do método de conexão preferencial [10]. Este método gera redes *scale-free* que são representativas das características das topologias da Internet. A experiência consiste em ter os nós a enviar ICMP echo requests (`ping`) para outros nós aleatórios durante 10 minutos e comparamos os RTTs observados com os valores teóricos calculados estaticamente a partir da topologia. Os resultados são apresentados na Tabela 1 como o erro quadrático médio entre esses dois valores. Consideramos duas topologias, respetivamente com 1000 elementos (666 serviços e 334 bridges) e 2000 elementos (1344 serviços e 656 bridges).

O Kollaps e o Maxinet são executados sobre quatro máquinas enquanto que o Mininet é executado sobre apenas uma máquina dado que execução sobre um cluster não é suportada. Observamos que o Mininet produz erros inferiores aos do Kollaps. No entanto, o maior desvio do `RTT` teórico observado com o Kollaps foi de apenas 0,4ms em ambas as topologias de 1000 e 2000 elementos. Como referência, o menor `RTT` teórico é de 10ms e 22ms para as topologias de 1000 e 2000 respectivamente. Devido às atuais limitações do Mininet, não é possível obter resultados para maiores topologias. O erro observado com o Maxinet é significativamente superior ao do Kollaps e Mininet, sendo o maior desvio de 11ms e 40ms para as topologias de 1000 e 2000 respetivamente, maiores que a latência mínima teórica para cada topologia.

Nós atribuímos isto à existência de um controlador externo, e ao tipo de controlador que afecta os resultados obtidos. Em adição a isto, mas com menor importância, o Maxinet sofre também dos pequenos mas mensuráveis atrasos quando pacotes necessitam de atravessar as ligações físicas.

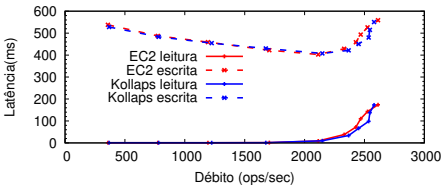

Figura 5: Débito/latência do Cassandra replicado geograficamente em ambos Amazon EC2 e KOLLAPS.

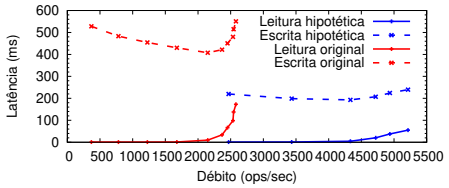

Figura 6: Débito/latência do Cassandra replicado geograficamente sob KOLLAPS usando uma topologia hipotética, respondendo à questão: "o que aconteceria se nós fossem movidos de `ap-south` para `ap-northeast`?"

4.4 Sistemas Replicados Geograficamente

Viramos agora a nossa atenção para macro-benchmarks para verificar e motivar o comportamento do KOLLAPS em cenários realistas.

Avaliação NoSQL. Comparamos agora os resultados das experiências com um sistema geograficamente replicado Apache Cassandra [3,24] em Amazon EC2 e KOLLAPS. A experiência consiste em 4 réplicas em Frankfurt, 4 em Sydney e 4 clientes YCSB [14] em Frankfurt. O Cassandra está configurado para replicação ativa e com um fator de replicação de 2. As operações YCSB estão configuradas para necessitar de um quórum para escritas e apenas uma resposta para leituras, com uma repartição de 50/50 para leituras e escritas. Os clientes YCSB vão dirigir a maior parte das escritas para as réplicas em Frankfurt que estão mais próximo. No entanto, tem de existir sempre pelo menos uma resposta de uma das réplicas em Sydney para que um quórum de escrita seja bem sucedido. Para modelar a topologia de rede no KOLLAPS, registamos a latência média e o jitter entre todas as instâncias de Amazon EC2 usadas, antes de executar a experiência nos servidores da Amazon.

A Figura 5 mostra a relação entre o débito e a latência obtidos em ambas as execuções. As curvas para ambas leituras e escritas são semelhantes, mostrando apenas pequenas diferenças após o ponto de inflexão onde a latência das respostas cresce rapidamente visto que as réplicas se encontram sob stress. Apesar de ser inesperado as latências para as escritas diminuir ligeiramente com o aumento do débito (antes do ponto de inflexão), este comportamente ocorre em ambas as execuções real e com KOLLAPS. Esta experiência mostra como tais problemas podem ser identificados, depurados e corrigidos com o KOLLAPS antes de um *deployment* real e dispendioso.

Avaliação de situações hipotéticas. Por fim, apresentamos um possível caso de uso para o KOLLAPS, avaliar aplicações em cenários hipotéticos. Por exemplo, pode ser útil saber o comportamento do Cassandra se as latências entre regiões EC2 passarem a metade. Na prática, isto corresponderia a mover 4 nós de Cassandra de Sydney (`ap-south`) para Seoul (`ap-northeast`). A Figura 6 mostra os resultados obtidos. Por uma questão de legibilidade e facilidade

de comparação, incluímos as latências reais do EC2 com a hipotética latência cortada em metade. Neste caso, o Cassandra comporta-se como o esperado: a latência dos pedidos passa para perto de metade e são observados maiores débitos.

5 Conclusão

O trabalho presente provem de uma necessidade de simplificar a avaliação de aplicações de grande escala, replicadas geograficamente. Em vez de emular todo o estado da rede, nós defendemos que as métricas ao nível da aplicação são maioritariamente afetadas pelas macro propriedades da rede, como latência, largura de banda, perda de pacotes e *jitter*. Avaliámos a viabilidade desta ideia desenhando, implementando e avaliando o Kollaps, um emulador de topologias de rede descentralizado. As nossas experiências, em topologias de pequena e grande escala, em ambos ambientes estáticos e dinâmicos mostram que o Kollaps é capaz de reproduzir com precisão execuções reais de sistemas polulares, como o Cassandra. Na nossa comunidade, reprodutibilidade é um tópico cada vez mais importante e acreditamos que o Kollaps é uma ferramenta importante para alcançar este objetivo. O Kollaps pode também ser usado por engenheiros para prever o desempenho e correção de aplicações sob diferentes condições de rede hipotéticas, controladas na sua totalidade.

Agradecimentos Este trabalho é parcialmente financiado pelo Fundo Europeu de Desenvolvimento Regional (FEDER) através do Programa Operacional Regional de Lisboa e por Fundos Nacionais através da FCT - Fundação para a Ciência e a Tecnologia no âmbito do projeto Lisboa-01-0145-FEDER-031456 (Angainor) e UID/CEC/50021/2019.

Referências

1. *Aeron*. https://github.com/real-logic/aeron, 2019.
2. *Amazon EC2 - T1 micro instances*. https://aws.amazon.com/, 2019.
3. *Apache Cassandra*. https://cassandra.apache.org/, 2019.
4. *Docker Swarm*. https://docs.docker.com/engine/swarm/, 2019.
5. *iPerf*. https://github.com/esnet/iperf, 2019.
6. *Kubernetes*. https://kubernetes.io/, 2019.
7. *Linux LXC*. https://linuxcontainers.org/, 2019.
8. *Linux Traffic Control*. https://linux.die.net/man/8/tc, 2019.
9. J. Banks et al., *Discrete-event System Simulation*, Prentice Hall, 2010.
10. A. L. Barabási et al., *Emergence of scaling in random networks*, Science, 286 (1999), pp. 509–512.
11. R. F. Boisvert, *Incentivizing Reproducibility*, Commun. ACM, 59 (2016), pp. 5–5.
12. T. D. Chandra et al., *Paxos made live: An engineering perspective*, PODC '07.
13. B. Chun et al., *Planetlab: an overlay testbed for broad-coverage services*, SIGCOMM, (2003).
14. B. F. Cooper et al., *Benchmarking cloud serving systems with ycsb*, SoCC '10.
15. M. Devera, *HTB - Hierarchy Token Bucket*, 2002.

16. M. a. Eriksen, *Trickle: A Userland Bandwidth Shaper for Unix-like Systems*, Usenix, (2005), pp. 61–70.
17. H. P. Fabio Ludovici, *NETEM(8)*, 2011.
18. S. Floyd et al., *Difficulties in simulating the internet*, IEEE/ACM Transactions on Networking (ToN), 9 (2001), pp. 392–403.
19. S. Floyd et al., *Internet research needs better models*, SIGCOMM, (2003).
20. M. Hibler et al., *Large-scale virtualization in the emulab network testbed*, in USENIX Annual Technical Conference, 2008.
21. B. Hubert, *tc - show / manipulate traffic control settings*, 2001.
22. F. Kelly, *Charging and rate control for elastic traffic*, European transactions on Telecommunications, 8 (1997), pp. 33–37.
23. W. Kim et al., *Understanding and Characterizing PlanetLab Resource Usage for Federated Network Testbeds*, in Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference, IMC '11, ACM, 2011.
24. A. Lakshman et al., *Cassandra: A decentralized structured storage system*, SIGOPS Oper. Syst. Rev., 44 (2010), pp. 35–40.
25. B. Lantz et al., *A network in a laptop*, Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets '10, (2010), pp. 1–6.
26. B. Lantz et al., *A network in a laptop: rapid prototyping for software-defined networks*, in Ninth Workshop on Hot Topics in Networks, HotNets, ACM, 2010.
27. H. H. Liu et al., *Crystalnet: Faithfully emulating large production networks*, 2017.
28. L. Massoulié et al., *Bandwidth sharing: Objectives and algorithms*, IEEE/ACM Transactions on Networking, 10 (2002), pp. 320–328.
29. D. Merkel, *Docker: lightweight Linux containers for consistent development and deployment*, 2014.
30. A. Montresor et al., *PeerSim: A scalable P2P simulator*, in Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on, IEEE, 2009.
31. J. Padhye et al., *Modeling tcp reno performance: a simple model and its empirical validation*, IEEE/ACM transactions on Networking, 8 (2000), pp. 133–145.
32. V. Paxson et al., *Why we don't know how to simulate the internet*, in In Proceedings of the 1997 Winter Simulation Conference, 1997, pp. 1037–1044.
33. R. D. Peng, *Reproducible research in computational science*, Science, 334 (2011).
34. M. Peuster et al., *Medicine: Rapid prototyping of production-ready network services in multi-pop environments*, in NFV-SDN, IEEE, 2016.
35. M. Peuster et al., *ContainerNet 2.0: A Rapid Prototyping Platform for Hybrid Service Function Chains*, in NetSoft, IEEE, 2018.
36. R. Ricci et al., *Introducing cloudlab: Scientific infrastructure for advancing cloud architectures and applications*, login:: the magazine of USENIX & SAGE, (2014).
37. G. F. Riley et al., *The ns-3 network simulator*, in Modeling and tools for network simulation, Springer, 2010, pp. 15–34.
38. L. Rizzo, *Dummynet: a simple approach to the evaluation of network protocols*, ACM Computer Communication Review, 27 (1997), pp. 31–41.
39. V. Schiavoni et al., *SplayNet: Distributed User-Space Topology Emulation*, in ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, Springer, 2013, pp. 62–81.
40. A. Vahdat et al., *Scalability and accuracy in a large-scale network emulator*, ACM SIGOPS Operating Systems Review, 36 (2002), pp. 271–284.
41. P. Wette et al., *MaxiNet: Distributed emulation of software-defined networks*, 2014 IFIP Networking Conference, IFIP Networking 2014, (2014).
42. B. White et al., *An integrated experimental environment for distributed systems and networks*, ACM SIGOPS Operating Systems Review, 36 (2002), pp. 255–270.