DECORE - Distribuição e Escalonamento para o emulador CORE (*Common Open Research Emulator*)

Eduardo Maroñas Monks[1]

[1]PPGC – Universidade Federal de Pelotas (UFPEL)
Pelotas – RS – Brasil

emmonks@gmail.com

Abstract. *This article presents a automated distribution solution and scheduling for the network emulator CORE (Common Open Research Emulator). It is proposed 4 types of schedulers and realized comparative performance tests.*

Resumo. *Este artigo apresenta uma solução de distribuição automática e escalonamento para o emulador de redes CORE (Common Open Research Emulator). São propostos 4 tipos de escalonadores e realizados testes comparativos de desempenho entre cada um dos escalonadores.*

1. Introdução

O uso de emuladores e simuladores tornaram-se fundamentais para o ensino e a pesquisa na área de redes de computadores. Para representar a complexidade das rede atuais, é necessário uma grande capacidade computacional. Desta forma, o uso de agrupamento de recursos disponíveis dos hosts ociosos em rede tornou-se uma das opções mais comuns para aumentar o poder computacional. O aproveitamento de recursos ociosos, processamento, disco e memória, de forma cooperativa possibilitam que, com baixo investimento, seja possível alcançar escalabilidade para emulação de cenários complexos. Com o uso de virtualização, a disponbilização e configuração de aplicações distribuídas foram facilitadas para os administradores de redes e de sistemas. As máquinas virtuais possibilitam o isolamento de sistemas operacionais e aplicações e são recursos importantes para a emulação de hosts. O emulador CORE (*Common Open Research Emulator*) [Ahrenholz et al. 2008] é baseado em máquinas virtuais leves que representam os nós do tipo roteador, host, switch e hub. Com o uso de uma configuração distribuída, o emulador CORE pode funcionar em modo *cluster*, de forma a emular cenários complexos. Entretanto, a forma de distribuição dos nós é manual, sendo o papel do usuário escolher os servidores disponíveis e distribuir as máquinas virtuais. Desta forma manual, o critério para distribuir os nós fica a cargo do usuário, sem analisar as condições da rede e dos recursos dos servidores. Sendo assim, o objetivo deste artigo é criar um escalonador de máquinas virtuais para o emulador CORE, denominado DECORE (Distribuição e Escalonamento para o emulador CORE) baseado nas políticas de escalonamento mais comuns utilizadas em *cloud computing*. O artigo está dividido em trabalhos relacionados, emulador CORE, a descrição do sistema DECORE, uma seção com os testes e resultados e as considerações finais sobre o trabalho.

2. Trabalhos relacionados

Os trabalhos relacionados a escalonamento de máquinas virtuais são voltados a computação em nuvem (*cloud computing*) que possuem semelhanças com o objetivo deste

artigo. O trabalho de [Litvinski and Gherbi 2013] faz uma análise do escalonador do OpenStack, que possui três tipos de escalonadores, um aleatório (*chances*), um que exclui nós devido ao não atendimento de recursos necessários (*filtering*) e outro que utiliza um *score* para definir qual nó possui melhores recursos para receber as máquinas virtuais (*weighing*). Em [Lloydab et al. 2014], é proposto um *frontend* para os escalonadores dos gerenciadores de nuvem Eucalyptus [Eucalyptus 2015], Openstack [Openstack 2015] e Amazon EC2 [Amazon 2015]. No artigo [Prajapati 2013], é realizada uma análise comparativa sobre os algoritmos de escalonamento utilizados em *cloud computing*. Em [Zhan et al. 2015] , os autores exploram, por meio de um *survey*, os vários tipos de recursos e escalonamentos necessários em um ambiente de nuvem. Para o escalonamento de VMs, são classificados três tipos básicos de objetivos que são balanceamento de carga, custo e eficiência energética. No artigo [Mann 2015], os autores apresentam diversos trabalhos relacionados a consolidação de máquinas virtuais e as estratégias e objetivos relacionados. Em [Lago et al. 2012], os autores propõe um escalonador de máquinas virtuais para redução de consumo de energia. Nesta dissertação [Rego 2012], é proposta uma arquitetura para tornar a contabilização dos recursos de CPU homogêneos em ambientes de *datacenters* heterogêneos. Não foram encontrados trabalhos relacionados a escalonamento de máquinas virtuais para o emulador CORE ou algum outro emulador que possua as mesmas características e objetivos propostos neste trabalho.

3. Emulador CORE (*Common Open Research Emulator*)

O CORE (*Common Open Research Emulator*) é um emulador de redes em código-fonte aberto, desenvolvido em Python, criado em 2008 e derivado do projeto IMUNES [IMUNES 2015]. Foi desenvolvido na universidade de Zagreb (Croácia), patrocinado pelas empresas Boeing e Ericsson em 2004, sendo a sua principal função a construção de redes virtuais. Esta ferramenta trabalha a partir da camada de rede do modelo OSI, simulando as camadas física e de enlace de dados e está disponível para os sistemas operacionais Linux e FreeBSD. Atualmente, o emulador CORE está na versão 4.8.

3.1. Arquitetura do CORE

O emulador CORE é baseado em virtualização leve, com o uso de contêiners em Linux (*Linux Network Namespaces*) ou em Jails quando usado em FreeBSD. O uso de contêiners possibilita alta escalabilidade e desempenho [Felter et al. 2014] em comparação ao uso de máquinas virtuais convencionais. Por exemplo, cada nó roteador consome cerca de 2,2MB de memória RAM, com a configuração ativa dos protocolos OSPFv2 e OSPFv3 do serviço de roteamento Quagga [Quagga 2015]. As máquinas virtuais assumem diversos tipos, sendo os principais os tipos roteador, host, switch e hub. Os nós do tipo switch e do tipo hub fazem simulações de switches e hubs ethernet e funcionam com configuração de bridge em Linux. O hub funciona com uma bridge promíscua, passando todos os pacotes para todas as portas. Devido a estas características, os nós do tipo switch e hub são limitados e possuem restrições em relação a VLANs (*Virtual Local Area Network*) e protocolos tais como STP (*Spanning-Tree Protocol*). No CORE, o hospedeiro compartilha o sistema de arquivos com os nós virtualizados, sendo a pilha de rede isolada entre cada um dos nós e entre os nós e o hospedeiro. Com esta configuração, as aplicações e arquivos disponíveis no sistema hospedeiro ficam disponibilizadas também para todos os nós virtualizados.

Os nós do tipo roteador e host podem ser configurados com diversos serviços, praticamente, qualquer aplicação ou serviço que possa ser executado no hospedeiro. Portanto, o consumo de processamento e memória de cada nó virtualizado poderá variar de acordo com as configurações estabelecidas. No artigo original do CORE [Ahrenholz et al. 2008], em um hardware de um único núcleo, Intel Xeon 3.0 GHz com 2,5GB de RAM, na versão 3.1 do CORE rodando em FreeBSD, foram utilizados de 30 a 40 nós do tipo roteador com os protocolos OSPFv2 e OSPFv3 sem problemas. Com mais de 100 nós, o cenário de emulação se tornou instável, não sendo possível distinguir as atividades de cada nó. Nos testes realizados neste artigo, na versão 4.8 do emulador, foram emulados cenários com mais de 300 nós em um único hospedeiro, em configurações de hardware comuns, tais como processador Intel i5 e com 2GB de RAM, sem maiores problemas.
Os hospedeiros devem executar o serviço *core-daemon* para a possibilitar a emulação. A interface gráfica (*core-gui*) é opcional e tem como função facilitar a montagem dos cenários e a interação com usuário. Para a criação dos cenários pode ser utilizado diretamente a API em Python, sem o uso da interface gráfica.

4. DECORE (Distribuição e Escalonamento para o emulador CORE)

O sistema de distribuição e escalonamento para o CORE (DECORE) é baseado nas políticas de escalonamento dos ambientes de computação em nuvem Openstack, Eucalyptus e OpenNebula [OpenNebula 2015]. Devido a natureza da aplicação, emulação de redes e o uso ser voltado para o ensino e pesquisa, as políticas de escalonamento foram adaptadas para este cenários de utilização. O sistema DECORE foi desenvolvido em Python, versão 2.7, utiliza o serviço de plugins NRPE do Nagios[Galstad 2013], para monitoramento dos recursos dos nós e utilitários em *shell script* para Linux. O sistema recebe como entrada um arquivo de cenário do CORE no formato **.imn** e qual tipo de escalonador a ser utilizado. O arquivo do cenário é analisado para extrair informações sobre a quantidade de nós, tipos de nós e número de ligações entre os nós, valores que serão os insumos para calcular o fator de complexidade do cenário. Após, será lido o arquivo que lista os servidores participantes na distribuição localizado em **/home/core/.core/servers.conf**, e realizada a coleta dos recursos em cada um dos nós ativos. Os dados coletados são armazenados em um banco de dados SQLite para obter o histórico de escalonamento. Para participarem do escalonamento, os servidores devem estar respondendo ping, com o serviço Nagios NRPE ativo e com o serviço *core-daemon* em execução. A latência é um fator excludente para o aproveitamento do servidor, sendo 100ms o limite máximo para o uso. Outro fator excludente para os servidores é o tempo de atividade (*uptime*) que deverá ser maior do que 60 minutos. O motivo para estes fatores serem excludentes é devido ao atraso prejudicar a troca de mensagens entre os nós distribuídos e o tempo de atividade ser um indicador que o servidor não está sendo reiniciado com frequência. A capacidade de CPU é medida pelo tempo necessário para calcular o número 32 de Fibonacci, sendo uma espécie de *benchmark* do servidor. Na Figura 1, está ilustrado o funcionamento em blocos do sistema DECORE.

4.1. Tipos de escalonadores

Os escalonadores disponíveis no sistema DECORE são Random, Igualitário, Pack policy e Igualitário por tipo de nó. O tipo **Random** sorteia um nó do cenário e associa a um dos

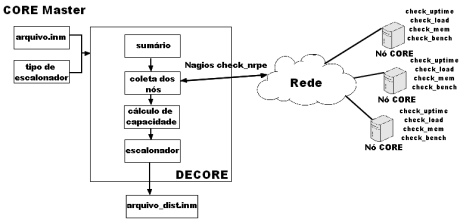

Figura 1. Arquitetura do DECORE

servidores disponíveis, até o limite da capacidade do servidor. Caso o número de nós no cenário seja maior do que os recursos dos servidores remotos, o excedente ficará alocado no servidor master. O tipo **Igualitário** associa os nós de acordo com a capacidade menor entre os servidores. O excedente será distribuído nos demais servidores até o limite da capacidade de cada um. Caso o número de nós no cenário seja maior do que os recursos dos servidores remotos, o excedente ficará alocado no servidor master. O tipo **Pack Policy** associa os nós para um servidor, até o limite da capacidade deste servidor, antes de usar recursos de outro servidor. Caso existam mais servidores disponíveis, a capacidade de cada um irá ser preenchida até terminarem os nós ou o limite de nós no servidor. Neste caso, se houver mais capacidade de nós servidores do que nós no cenário, será possível evitar o uso de recursos nos demais servidores disponíveis. O tipo **Igualitário por tipo de nó**, é similar ao escalonamento igualitário, mas a distribuição será feita por tipo de nó, na ordem de primeiro roteador, segundo hosts, terceiro switch e quarto hub. Assim, todos os roteadores serão distribuídos primeiro, depois o host, e após os switches e hubs.

4.2. Critérios de escalonamento

Os critérios utilizados para definição da quantidade de nós em cada servidor são memória livre, carga de processamento nos últimos 5 minutos, número de cores do processador, capacidade de CPU e o fator complexidade do cenário. O fator complexidade do cenário leva em consideração a interligação existente no cenário a ser distribuído, sendo feito uma ajuste no peso do nó de acordo com o valor do fator. O peso do nó foi configurado como sendo 3, baseado no valor médio de consumo de memória por um nó com aplicações e serviços comuns (Quagga, Apache e SSH) que chegam a 3 MB de RAM. O valor do *score* é determinado pelo valor da memória livre do servidor, menos o valor de carga nos últimos 5 minutos e a capacidade de CPU. O número de cores é um multiplicador do *score*. Para saber a capacidade de nós, o valor do *score* é divido pelo peso de cada nó multiplicado pelo valor do fator de complexidade. Ao final, o valor de quantidade de nós será utilizado pelos escalonadores para fazer a distribuição dos nós entre os servidores. Esta fórmula, listada no trecho de código em Listagem 1, foi criada a partir de simulações em planilhas de cálculo e testes com a capacidade de um servidor hospedeiro isolado.

Listagem 1. Cálculo da capacidade de nós por servidor

```
# Peso por nó emulado
pno=3
# Calcula o score de acordo com os recursos coletados dos servidores
score=((memoria)-(carga*25)-(bench*50))*(ncores)
# Calcula a capacidade de nós
nos=(score/(pno*fcomplex))
```

Na Figura 2 estão mostradas as distribuições de nós por tipo de escalonador para cada cenário de teste. Nestes gráficos, os servidores disponíveis estavam ocisosos, com uso exclusivo para a emulação. Os escalonadores priorizam a concentração de nós nos servidores que possuem maior capacidade, e o restante dos nós não distribuídos são executados no servidor CORE Master.

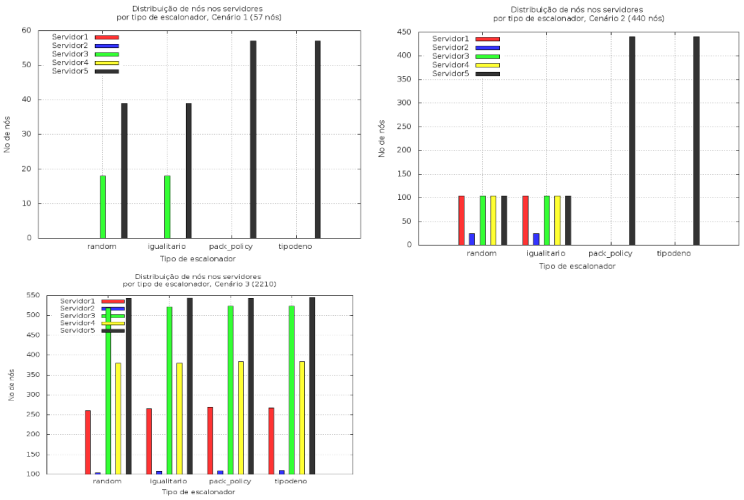

Figura 2. Distribuição de nós para cada cenário, em cada servidor e por tipo de escalonador

5. Testes e Resultados

Os testes foram realizados em três cenários. Os cenários foram montados com o objetivo de serem complexos e os mais próximos possíveis de aplicações para ensino e pesquisa na área de redes de computadores. O cenário 1 é composto por uma rede na topologia *full mesh* (totalmente conectada), com 57 nós, sendo 16 roteadores. O cenário 2, é composto por 440 nós, sendo todos roteadores interligados em uma topologia em estrela distribuída. No cenário 3, composto por 2210 nós, sendo 2000 hosts, 10 roteadores e 200 switches.
Os servidores participantes do escalonamento foram disponibilizados em máquinas virtuais com o virtualizador Vmware Player. As máquinas virtuais foram configuradas com a distribuição Linux Lubuntu, o serviço NRPE e plugins do Nagios e as demais aplicações necessárias para a emulação. As configurações dos recursos de hardware das máquinas virtuais estão listados na Tabela 2. O servidor denominado ”CORE Master”, possui o sistema DECORE e é o responsável por receber os nós excedentes do cenário de emulação. A configuração da máquina virtual ”CORE MASTER”foi definida com 2GB de RAM e 2 cores de CPU. A rede utilizada para os testes foi uma rede local em 1000Mbit/s onde estavam todos os servidores envolvidos.

5.1. Metodologia dos testes

Os tempos de início de emulação e de convergência do protocolo OSPF foram comparados entre os tipos de escalonamento disponíveis no sistema DECORE. A verificação dos tempos de início da emulação foram coletados do arquivo de logs do serviço *core-daemon* no

Tabela 1: Configurações das máquinas virtuais

Servidor	Configuração
Servidor1	1GB de RAM, 1 core
Servidor2	512MB, 1 core
Servidor3	1GB de RAM, 2 cores
Servidor4	768 de RAM, 2 cores
Servidor5	1GB de RAM, 2 cores

servidor CORE Master. O arquivo de logs, por padrão, fica disponível em **/var/log/core-daemon.log**. Para a coleta do tempo de convergência do protocolo OSPF, foi utilizado um script em Python. Este script é executado na inicialização de nós pré-determinados em cada cenário e fica monitorando a quantidade de rotas na tabela de roteamento, a cada 100ms. Quando for atingido o número de rotas, passado como parâmetro que representa a convergência do cenário, o script encerra a execução e registra o tempo coletado. Devido aos caminhos alternativos variáveis nos cenários com o protocolo OSPF, o script contabiliza o parâmetro como o mínimo de rotas necessárias para considerar o cenário convergido.
Os cenários foram submetidos ao sistema DECORE com 5 servidores disponíveis. Os servidores forma executados em dois hospedeiros distintos, no primeiro hospedeiro foram disponibilizados 4 servidores e no segundo hospedeiro 1 servidor. Durante os testes, tanto os hospedeiros quanto os servidores ficaram com o uso exclusivo para a execução dos cenários de emulação. Foram realizados três repetições para cada cenário, com cada tipo de escalonador.
Para os testes realizados a respeito da distribuição de nós por servidor, foi executado o sistema DECORE com os 5 servidores disponíveis. O primeiro teste foi feito com os servidores ociosos e o segundo teste com os servidores com carga. Para gerar a carga nos servidores foi utilizada a ferramenta **Stress** [Stress 2015], com a configuração "'stress –cpu 2 -d 2 –io 4 –vm 2 –vm-bytes 32M –timeout 600s"'. Nesta linha de comando, serão gerados 2 processos com uso intensivo de CPU, 2 processos que vão criar e remover arquivos de 1GB, 4 processos com uso intensivo de entrada e saída e 2 processos com atividades de alocação de memória (32MB em cada), com duração de 10 minutos (600s).

5.2. Resultados

Ao realizar o escalonamento dos cenários com os 5 servidores disponíveis, a distribuição dos nós de acordo com cada tipo de escalonador e carga no servidores está disponível na Tabela 3. Percebe-se nos dados coletados que ao existir carga e consumo de memória nos servidores, a capacidade de emulação de nós diminui consideravelmente. Os valores negativos de nós indicam que o servidor não tem capacidade de participar do escalonamento e desta forma será excluído do conjunto de servidores aptos aos escalonadores.

Na Figura 3a, um gráfico do tempo de início da emulação, medido em cada um dos tipos de escalonadores e sem o uso de distribuição (local). Estes tempos levaram em consideração o tempo de término de instanciamento de todos os nós, sem considerar o tempo de convergência ou o início dos serviços habilitados em cada nó. Na Figura 3b, um gráfico com o tempo de convergência do cenário de emulação, medido em cada um dos tipos de escalonadores e sem o uso de distribuição (local). Estes tempos levaram em consideração o tempo de convergência do protocolo OSPF, onde todos os nós passam a

Tabela 2: Capacidade de nós com servidores ociosos e com carga

Servidor	Cenário 1		Cenário 2		Cenário 3	
	Ocioso(nós)	Carga(nós)	Ocioso(nós)	Carga(nós)	Ocioso(nós)	Carga(nós)
Servidor1	88	26	237	121	244	124
Servidor2	33	-14	88	-1	90	-1
Servidor3	161	43	430	240	441	243
Servidor4	120	34	318	121	331	128
Servidor5	172	63	458	254	473	262

ter conectividade uns com os outros. Nestas coletas, os cenários 2 e 3 não convergiram quando usado somente no servidor CORE Master. O processamento ficou em 100% e a máquina virtual congelou. No cenário 2, nos escalonadores random e Tipo de nó, não houve convergência devido a algum problema relacionado a criação de interfaces nos servidores remotos. Estas interfaces representam os enlaces entre os nós e são criadas no sistema operacional tal como interfaces virtuais.

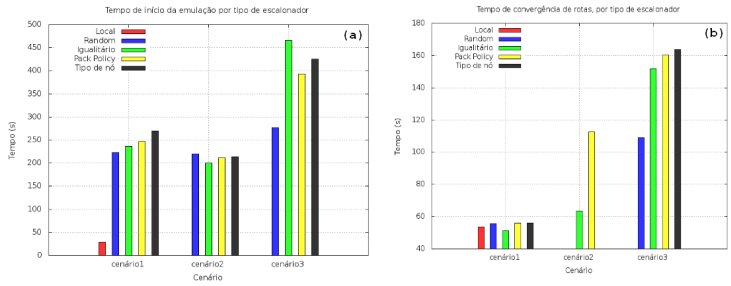

Figura 3. Tempos de início de emulação e convergência do protocolo OSPF, em cada cenário, por tipo de escalonador

6. Considerações finais

As ferramentas de emulação são essenciais para o ensino e pesquisa na área de redes de computadores. O emulador CORE é uma das opções existentes e possui algumas características interessantes para uso de soluções em Linux. O objetivo deste trabalho foi melhorar uma das funcionalidades da ferramenta CORE que é a distribuição dos nós nos cenários a serem emulados. A forma original é estática e manual, o que torna difícil de realizar em cenários mais complexos. O sistema proposto, DECORE, possibilitou o uso de cenários complexos de forma simplificada, tornando mais simples a distribuição do nós entre os hospedeiros.

Os escalonadores implementados no sistema DECORE possuem equivalência de desempenho, por exemplo, o escalonador **random** se mostrou tão bom quanto qualquer outro, o que comprova esta afirmação. Nos casos onde não foram possíveis as coletas dos tempos de emulação ou convergência, devem-se ao fato de serem necessários ajustes a fórmula de criação do *score* e do fator de complexidade do sistema DECORE. Estes ajustes e melhorias ao sistema DECORE são trabalhos futuros, além do uso de cenários mais complexos e a integração do sistema ao emulador.

Referências

Ahrenholz, J., Danilov, C., Henderson, T. R., and Kim, J. H. (2008). Core: A real-time network emulator. In *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1–7. IEEE.

Amazon (2015). Amazon elastic compute cloud (amazon ec2). Disponível em: <https://aws.amazon.com/ec2/>. Acesso em: nov 2015.

Eucalyptus (2015). Hp helion eucalyptus - open source hybrid cloud software for aws users. Disponível em: <http://www.eucalyptus.com/>. Acesso em: nov 2015.

Felter, W., Ferreira, A., Rajamony, R., and Rubio, J. (2014). An updated performance comparison of virtual machines and linux containers. *technology*, 28:32.

Galstad, E. (2013). Nagios: Nrpe documentation. *Sourceforge. net (May 2007)*.

IMUNES (2015). Imunes (integrated multiprotocol network emulator/simulator). Disponível em: <http://imunes.net/>. Acesso em: nov 2015.

Lago, D. G., Madeira, E. R., and Bittencourt, L. F. (2012). Escalonamento com prioridade na alocaçao ciente de energia de máquinas virtuais em nuvens. *XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 508–521.

Litvinski, O. and Gherbi, A. (2013). Openstack scheduler evaluation using design of experiment approach. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2013 IEEE 16th International Symposium on*, pages 1–7.

Lloydab, W. J., Davidab, O., Arabib, M., IIc, J. C. A., Greenc, T. R., Carlsonb, J. R., and Rojasd, K. W. (2014). The virtual machine (vm) scaler: an infrastructure manager supporting environmental modeling on iaas clouds.

Mann, Z. A. (2015). Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms. *ACM Comput. Surv.*, 48(1):11:1–11:34.

OpenNebula (2015). Opennebula. Disponível em: <http://opennebula.org/>. Acesso em: nov 2015.

Openstack (2015). Openstack open source cloud computing software. Disponível em: <http://www.openstack.org/>. Acesso em: nov 2015.

Prajapati, K. D. (2013). Comparison of virtual machine scheduling algorithms in cloud computing.

Quagga (2015). Quagga routing suite. Disponível em: <http://www.nongnu.org/quagga/>. Acesso em: nov 2015.

Rego, P. A. L. (2012). Faircpu: Uma arquitetura para provisionamento de máquinas virtuais utilizando características de processamento. Dissertação de mestrado em ciência da computação, Universidade Federal do Ceará, Fortaleza¸Ceará-Brasil.

Stress (2015). Stress. Disponível em: <http://people.seas.harvard.edu/ apw/stress/>. Acesso em: nov 2015.

Zhan, Z.-H., Liu, X.-F., Gong, Y.-J., Zhang, J., Chung, H. S.-H., and Li, Y. (2015). Cloud computing resource scheduling and a survey of its evolutionary approaches. *ACM Comput. Surv.*, 47(4):63:1–63:33.