Modificação de DAGs de Aplicações para Execução em Nuvens Computacionais

Aluno: Marcelo A. G. dos Santos
Orientador: Prof. Dr. Nelson Luis Saldanha da Fonseca

e-mail: masantos22e@gmail.com, nfonseca@ic.unicamp.br

LRC - IC - Unicamp

CNPq/PIBIC

Vigência: 08/2012 – 07/2013

Resumo

O sucesso de nuvens computacionais deve-se principalmente ao avanço nas técnicas de virtualização. Para que aplicações funcionem em nuvens é necessário que recursos virtuais sejam instanciados nos melhores recursos físicos considerando os objetivos do usuário. Quando o usuário deseja executar uma aplicação distribuída com o objetivo de minimizar o tempo de execução, pode-se utilizar escalonadores clássicos desenvolvidos para grades. Porém, é necessário embutir os requisitos de recursos virtuais na descrição das aplicações. Este documento apresenta uma proposta de projeto de iniciação científica cujo objetivo é a validação de um código que implementa um algoritmo [1] proposto pelo orientador e que já foi publicado.

Introdução

Os bons resultados alcançados com grades computacionais levou à proposta de um novo paradigma chamado de nuvem computacional [3]. Neste paradigma, aplicações e serviços de TI, inicialmente operado e mantido por uma organização, passam a ser de responsabilidade de outras organizações, que disponibilizam o ambiente através da Internet de forma transparente para as organizações contratantes.

Assim como acontece em grades, as aplicações submetidas para execução em nuvens precisam ser escalonadas, ou seja, é necessário que o provedor dos serviços defina em quais recursos as aplicações dos usuários serão executadas. Uma solução imediata para lidar com o escalonamento de aplicações em nuvens seria utilizar escalonadores já existentes e que foram propostos para aplicações em grades. Entretanto, as aplicações em nuvens definem requisitos de software que são atendidos através da instanciação de máquinas virtuais (MVs). A instanciação das MVs precisa ser realizada antes da execução das tarefas das aplicações. Portanto, não basta escalonar as aplicações em nuvens sem considerar a necessidade de instanciação das MVs.

Uma solução para o problema seria implementar escalonadores específicos para nuvens, que levassem em consideração a instanciação das máquinas virtuais. Entretanto, a adição de mais variáveis ao problema, em relação ao inicialmente resolvido para grades, tende a deixá-lo mais complexo. Uma outra solução para o problema é a de modificar a descrição das aplicações de modo a considerar que as instanciações das máquinas virtuais façam parte da execução da aplicação, como se fossem tarefas que devem ser executadas antes das tarefas originais da aplicação. Dessa forma, a nova descrição poderia ser passada como entrada para escalonadores clássicos de grades, sem necessidade da proposta de um novo escalonador específico para nuvens.

Em [4], foi proposto um mecanismo para integrar a instanciação de máquinas virtuais no escalonamento de aplicações em grades. Embora a proposta seja voltada para grades, ela pode ser utilizada no escalonamento em nuvens já que a necessidade de instanciar máquinas virtuais antes da execução da aplicação é equivalente. Na proposta apresentada em [4], é utilizado um repositório de imagens de máquinas virtuais e as tarefas das aplicações devem descrever qual das imagens atende os seus requisitos. A descrição dos requisitos é representada com a criação de uma nova tarefa no grafo da aplicação. Cada tarefa da aplicação portanto passa a ter uma nova tarefa posicionada imediatamente antes dela. Após a adição das novas tarefas, o grafo resultante pode ser escalonado por qualquer escalonador de tarefas já existente, pois para esse escalonador não há diferenças entre as novas tarefas adicionadas e as tarefas originais da aplicação.

O problema da proposta apresentada em [4] é o fato dela causar uma grande utilização da rede já que cada tarefa requer a instanciação de uma máquina virtual. Dessa forma, uma aplicação com k tarefas exige que k imagens sejam transferidas do repositório para as máquinas da grade. O ideal seria reaproveitar algumas das imagens já transferidas de modo que com menos de k transferências a aplicação pudesse ser executada, diminuindo assim a probabilidade de congestionamento nos enlaces de rede. Porém, reaproveitar as instanciações de máquinas virtuais para mais de uma tarefa leva a um problema complexo, pois existem diversas formas de embutir as instanciações de máquinas virtuais na descrição de uma aplicação (Conforme detalhado em [2], listar todas essas formas é equivalente a encontrar todas as partições de um conjunto, uma tarefa custosa que leva a um algoritmo com complexidade exponencial). A Figura 1 apresenta um método proposto pelo orientador deste trabalho e que foi publicado em [2]. Neste método, é criada uma nova tarefa, que equivale à instanciação de 1 MV, para cada conjunto de tarefas por caminho que dependam de uma mesma MV. Desta forma evita-se um atraso desnecessário na execução da aplicação, já que tarefas dependentes executarão na mesma MV, diminuindo assim a quantidade de dados transferidos via rede.

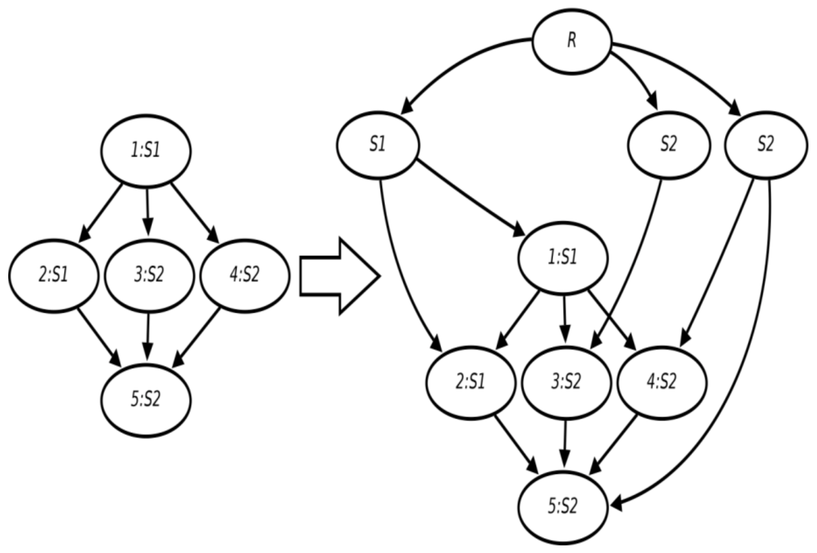


Figura 1: Exemplo de transformação: 1 MV para cada conjunto de tarefas por caminho.

Algoritmo

[1] **Algoritmo 1** Modificador de DAG (1 MV por caminho)

Entrada: DAG D com requisitos de software S não embutidos no DAG.
Saída: DAG modificado M com requisitos de software embutidos.
1: **for** cada caminho *h* € D **do**
2: **for** cada tarefa t € h **do**
3: Inclua t no conjunto Ph,s, onde Ss é a dependência de software de t
4: **end for**
5: **end for**
6: Crie uma nova tarefa de entrada R com peso 0 para representar o repositório
7: **while** houver pelo menos um conjunto P **do**
8: **for** cada software s € *S* IƎ pelo menos um conjunto Ph,s **do**
9: Faça Ps igual ao maior conjunto Ph,s {O peso de um conjunto Ph,s é calculado em bytes através da soma dos pesos dos arcos no caminho h}
10: Crie uma nova tarefa ps com peso equivalente ao tempo de boot da máquina virtual contendo o software Ss
11: Crie um novo arco da tarefa ps para cada uma das tarefas em Ps com peso ∞ {A atribuição de ∞, fará com que as tarefas executem no mesmo host onde as máquinas virtuais serão instanciadas}
12: Crie um novo arco da tarefa R para a tarefa ps com peso equivalente ao tamanho da máquina virtual que contém o software Ss
13: Remova cada uma das tarefas de Ps de todos os outros conjuntos P {Se um conjunto ficar vazio, remova ele da lista de conjuntos}
14: Apague Ps
15: **end for**
16: **end while**

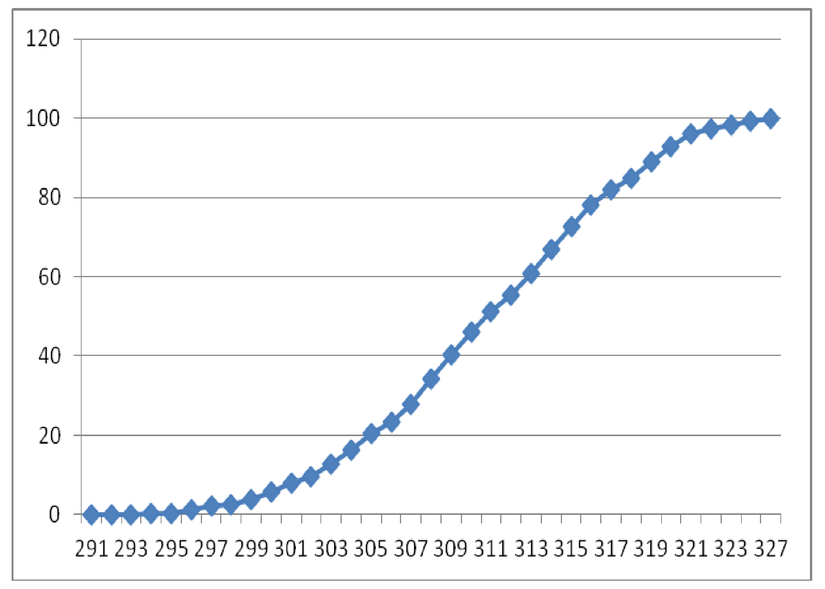
A implementação foi realizada por um aluno do curso de Ciência da Computação como trabalho da disciplina “Redes de Computadores”, ministrada pelo orientador, no último semestre.

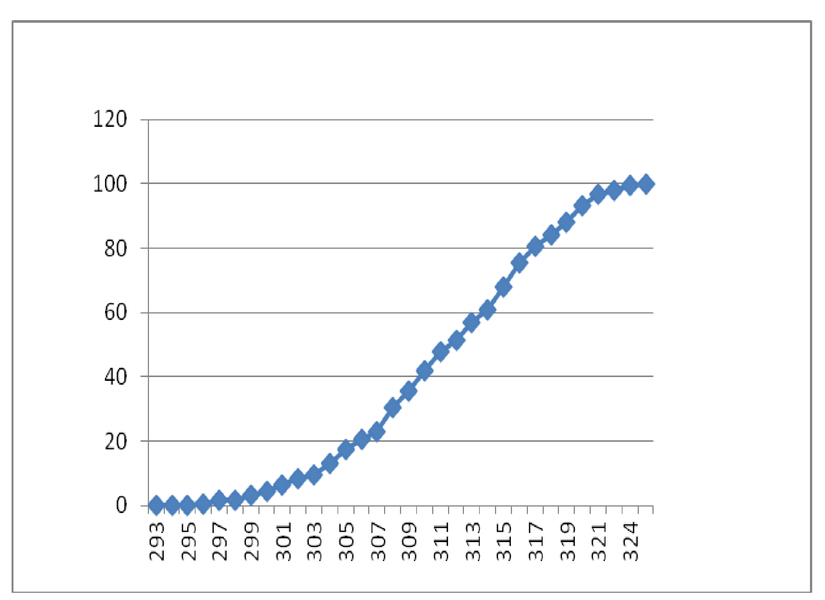
Metodologia

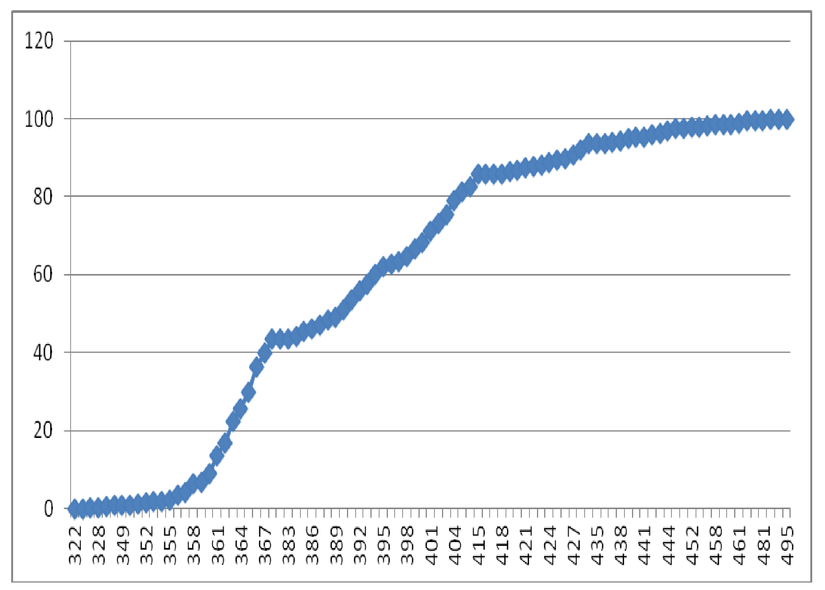
Para a validação do Algoritmo 1 [1], implementado em C, foi necessário criar três gráficos e comparar o resultado obtido com o resultado esperado pela teoria . Em seguida, foram criados diversos DAGs com dependências de softwares distintas e comparou-se a eficiência do Algoritmo 1 [4] em relação a outros dois algoritmos que foram citados no artigo [1].

Resultados

Para validar o Algoritmo 1 [1], foram feitos três gráficos com todos os DAGs gerados por um dos códigos feitos junto com o artigo [2] mas com dependências de tarefas distintas para cada gráfico. Com todos os DAGs criados, rodamos o escalonador HEFT que gera o makespans de cada uma das DAGs e ordenamos a lista dos makespans. Em seguida, plotamos os gráficos da função de distribuição cumulativa, como mostrado abaixo:







Conclusão

Comparando os três gráficos com a figura 4 do artigo [2] conseguimos observar que em todas as figuras quando temos a função de distribuição cumulativa maior que 50% das DAGs temos um makespans maior ou bem próximo de 330s, assim como ocorre na figura 4 do artigo [2] o que comprova o algoritmo em relação ao esperado pela teoria.
Por fim, com o ambiente de teste pronto, criamos diversas DAGs da mesma topologia, mas com diferentes dependências de software. Executamos o algoritmo proposto juntamente com os outros dois algoritmos mencionados no artigo [1] e, para todas as DAGs, o Algoritmo 1 [1] teve desempenho superior em comparação aos outros dois, comprovando assim sua eficiência.

Referência

[2] D. M. Batista, C. G. Chaves, and N. L. S. Fonseca. Embedding Software Requirements in Grid Scheduling. In Proceedings of the ICC 2011: IEEE International Conference on Communications (ICC2011), pages 1–6, 2011.

[3] I. T. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. In CCE’08, pages 1–10, Nov 2008.

[4] Ruben S. Montero, Eduardo Huedo, and Ignacio M. Llorente. Dynamic Deployment of Custom Execution Environments in Grids. In ADV- COMP ’08, pages 33–38, Washington, DC, USA, 2008. IEEE Computer Society.