Maestro - Sistema Operacional de Redes (Maio 2011)

Arthur Kazuo Tojo Costa
Universidade Federal de São Carlos, Campus Sorocaba
Sorocaba, São Paulo
Email: aktojo@hotmail.com

***Resumo* — Este artigo busca definir a plataforma Maestro, avaliar desempenho diante outro sistema operacional de redes (NOX) e destacá-lo em relação as funcionalidades de controle, eventos multi-thread, escalabilidade e paralelismo para cada sessão.**

***Index Terms* — gerenciamento, Maestro, multithread, OpenFlow, otimização, performance, redes, sistema operacional de redes**

I. INTRODUÇÃO

Sistema Operacional de Redes é um conjunto de módulos que ampliam os sistemas operacionais, complementando-os com um conjunto de funções básicas, e de uso geral, que tornam transparente o uso de recursos compartilhados da rede. Assim como como os sistemas operacionais convencionais, o Maestro possui gerenciador de recursos e fornece a abstração dos mesmos, facilitando a programação em cima da plataforma.

A característica fundamental de uma rede OpenFlow, é que o controlador é responsável em inicializar a conexão de cada fluxo entrando em contato diretamente com todos os switches relacionados. Desta maneira, a performance do controlador pode ser o gargalo. Este artigo mostra como este problema fundamental é solucionado com o paralelismo. O estado do controlador OpenFlow baseado, denominado NOX, possui um modelo de programação simples para desenvolvimento das funcionalidades de controle, devido aos seus eventos serem single-thread, sem a exploração do paralelismo definido no controlador que será abordado neste artigo. A proposta é que o Maestro disponibiliza aos programadores um modelo simples de programação e explora o paralelismo em cada sessão, além das técnicas adicionais de otimização de transferência.

Experimentos mostram que a taxa de transferência do Maestro pode alcançar o nível linear de escabilidade, caso executada em um servidor de oito núcleos, demonstrando um padrão de alto nível ao se comparer com outros modelos sem esta funcionalidade.

II. MAESTRO

Maestro é um sistema operacional para a orquestração de aplicações de controle de rede, fornecendo interfaces para implementação de aplicações de rede modular de controle de acesso, modificação do estado da rede e coordenação de suas interações. Pode ser definido também como uma plataforma para a realização de funções de rede automática e programáticas de controle usando esses aplicativos modularizados.

Embora o projeto envolvendo o Maestro centre-se na construção de um controlador de OpenFlow para seu uso, a plataforma não se limita apenas ao OpenFlow de redes, mas fornece interfaces que possibilitam:

- Manutenção do estado da rede em nome dos componentes de controle;
- Introdução de novas funções personalizadas de controle;
- Adicionando componentes de controle, especificando a sequencia de execução e o estado da rede compartilhada dos componentes.

Além disso, o Maestro explora paralelismo além de ser um sistema portátil e escalável (tais informações serão melhores detalhadas nos capítulos posteriors). Todas essas características com o intuito de melhora na performance do sistema final, em aspectos gerais.

Atualmente, o Maestro fornece os componentes de controle para realizar qualquer switch de rede de aprendizagem, ou uma rede roteada utilizando switches OpenFlow.

III. CONCEITOS E RECURSOS

O Sistema Operacional de Redes Maestro foi inspirado na arquitetura 4D, sendo que o sistema resultante baseado no OpenFlow opera as duas principais funções de um roteador ou switch convencionais: transmissão de dados através comutação de pacotes e plano de controle de encaminhamento de tomada de decisão.

A máquina do controlador OpenFlow encarrega-se da funcionalidade do plano de controle, instalando e removendo as entradas de fluxo nos dispositivos de switches. O OpenFlow permite aos usuários uma flexibilidade para o controle plano das funcionalidades no controlador, além da liberdade em se

controlar os dispositivos de switches.

O sucesso da técnica empregada no OpenFlow pode ser visualizada em um grande número de usos recentes, tais como Novos Protocolos de Roteamento, permissão de redes não-IP, sistemas de medição de redes, design de redes empresariais ou de datacenters. Uma característica fundamental é que o controlador é responsável por estabelecer cada fluxo na rede, sendo que o switch sempre vê o primeiro fluxo de pacote para correspondê-lo. visto que inicialmente não há nenhuma entrada de fluxo configurada na tabela de fluxo do switch (a instrução é geralmente implementado em TCAM).

O primeiro pacote será enviado para o controlador e chamamos esse primeiro pacote de um pedido de fluxo (flow request). Inicialmente, esse fluxo será verificado pelo controlador para determiner se há permissão diante às políticas de segurança. Caso esta permissão seja aceita, o controlador calculará um caminho para esse fluxo, e instalar fluxos de entradas em cada switch ao longo do caminho escolhido. Finalmente, o próprio pacote será enviado de volta ao interruptor do switch de origem.

Conforme a expansão do tamanho da rede, cresce também o número de fluxos que devem ser estabelecidos pelo mesmo processo e, caso o controlador não tenha a capacidade para lidar com todos esses pedidos de criação de fluxo, ele se tornará um indesejável gargalo na rede.

Por enquanto, torna-se um desafio essencial o aumento da demanda dos switches OpenFLow nas aplicações, a ponto de conectar milhares de servidores com o intuito de melhorar o desempenho do sistema de controle, visto que esta estratégia já está sendo utilizada para projetar redes de datacenters de dimensões amplas.

As medições de tráfego a partir de datacenters com tamanhos e finalidades diferentes mostraram que o número de concorrentes fluxos ativos é pequena, o que implica que os switches OpenFlow podem ser bem adequados para serem aplicados em construção de redes de datacenter. Porém, autores citam que para um datacenter com 100 switches de borda, o comando centralizado pode ter que esperar cerca de 10 milhões de pedidos de fluxo por segundo. Isso cria um desafio fundamental para ocontrolador OpenFlow centralizado para que este possa ser implantado em um ambiente similar de larga escala.

Felizmente, o processamento da requisição de tal fluxo do controlador é potencialmente paralelizável, pois não existem dependências de dados complexos e toda a computação é realizada sobre a verificação de um fluxo contra as políticas de segurança, encontrando um caminho para ela, e enviando mensagens de configuração do fluxo de entrada. Através da adoção generalizada e da facilidade no acesso a processadores multi-core (até mesmo em processadores com fácil acesso à população), considera-se então que é uma oportunidade e um momento ótimo para que tais tecnologias sejam analisadas e abordadas para que gargalos sejam evitados.

O NOX, outro sistema operacional de redes que também utiliza o sistema controlador de OpenFlow, é um sistema centralizado e single-threaded. A vantagem de que tal projeto tenha um modelo de programação mais simples para desenvolvimento da função de controle por se basear em um mecanismo single-threaded e um event-loop pode ser um argumento insuficiente diante a vantagem que se pode adquirir dos avanços atuais na tecnologia, com o crescente avanço e popularização de sistemas multi-core.

Uma das propostas dos criadores do Maestro foi, inclusive, curar uma ineficiência do NOX, que consiste no processamento individual a cada solicitação de fluxo, além de que todos os pacotes gerados nesse sentido seriam enviados individualmente na saída. Se analisado o perfil do NOX pode-se detalhar que cerca de 80% do tempo de processamento do fluxo de pedido é gasto no envio destas mensagens.

IV. Linguagem

Os autores do Maestro decidiram implementar o sistema operacional na linguagem Java a partir de muitos argumentos que poderiam ser, futuramente, decisivos para a aceitação e escolha dos programadores e usuários em geral.

Primeiramente, pelo fato dos programas em Java serem considerados fáceis de realizar a escrita ou a manutenção, com um nível de segurança para inclusive facilitar sua depuração. A linguagem ainda suporta carregamento dinâmico de aplicações e estruturas de dados sem efetuar a recompilação, não necessitando reiniciar todo o sistema, gerando um ambiente mais flexível para expansão. Pela facilidade em migrar códigos da linguagem para outras diferentes plataformas, com a premissa de que exista suporte a máquina Virtual Java, visto que o código precisa ser muito pouco ou não mesmo sem nenhuma modificação quando se quer efetuar tal migração.

E ainda que a linguagem possa ser considerada menos eficiente do que outras, como por exemplo, o C ou o C++, também foi avaliado no quesito de otimização, pois se pode conseguir um bom desempenho a partir da possibel escalabilidade.

V. Sistemática e Implementação

A. *Estrutura Geral do Maestro*

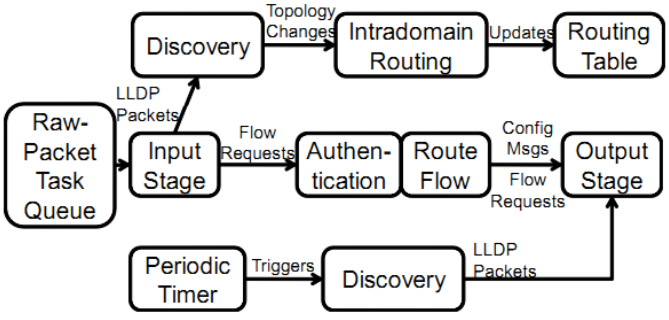

Figura 1 - Estrutura Geral do Maestro

Estrutura Funcional

A figura 1 esquematiza a estrutura de funcionalidade do Maestro. Qualquer controlador OpenFlow irá compartilhar essas mesmas funcionalidades, embora o modo em que isso é feito possa ser implementado de diversas maneiras.

O Maestro envia e recebe mensagens OpenFlow de e para os switches de rede através de conexões TCP. O "Input Stage" (Estágio de Entrada) e "Output Stage" (Estágio de Saída) lidam com detalhes de baixo nível de leitura e escrita a partir de buffers de sockets, traduzindo as mensagens OpenFlow raízes e dos dados contidos em estruturas de alto nível.

Estas funcionalidades de baixo nível ficam fixas com uma versão particular do protocolo OpenFlow. Outras funcionalidades de alto nível podem variar e são implementadas em módulos denominados “aplicações” no Maestro. Programadores podem flexivelmente modificar o comportamento dessas aplicações, ou adicionar novos aplicativos para atender os diferentes objetivos desejados, no esquema da Figura 1, essas aplicações são “Discovery”, “Intradomain Routing”, “Authentication” e “RouteFlow”.

Quando o switch se junta a rede através da criação de conexões TCP com o Maestro, a aplicação “Discovery” envia periodicamente mensagens de sondagem aos vizinhos de cada switch as quais estão em conformidade com o LLDP (Link Layer Discovery Protocol). Este é representado pelo caminho de execução referente a parte inferior da figura.

Então, quando "Discovery" recupera a mensagem de sondagem exibida pelo LLDP, sendo que ele sabe que na mensagem contém informações do switch do pacote que está sendo trafegado, desta maneira é possível determinar a topologia da rede.

Quando o "Discovery" encontra mudanças na topologia, inicia a "Intradomain Routing", uma aplicação para atualizar a estrutura "Routing Table" em conformidade. Este é representado pelo caminho de execução do aplicativo no topo da figura. Esta tabela de roteamento contém a listagem dos caminhos mais curtos dos pontos da rede, e será utilizada pelo aplicativo "Route Flow" encontrar caminhos para os pedidos de fluxo.

Quando o Maestro recebe uma solicitação de fluxo a partir de um switch, a requisição será primeiramente analisada contra as políticas de segurança através da “Authentication”. Somente quando permitido, a aplicação “Route Flow” irá tentar encontrar o caminho para a requisição e irá gerar uma mensagem de configuração de fluxo para cada um dos switches ao longo do caminho escolhido. Estes dois aplicativos unidos são denominados “fase de fluxo de processo". Depois disso, todas as mensagens de configuração do fluxo serão enviadas para seu switch de destino, e o pacote de solicitação de fluxo será enviado de volta para o switch de origem, por si próprio. Este é representado pelo caminho de execução do aplicativo no meio da figura, denominada "flow request execution path" (caminho de execução do fluxo solicitado).

Todos os três caminhos de execução de aplicativos são executados em paralelo. O Maestro garante que caso a tabela de roteamento (“Routing Table”) seja utilizada enquanto o aplicativo de fluxo de caminho (“Route Flow”) já está em execução, este continua com a versão mais antiga do "RoutingTable" para gerar resultados consistentes.

Estrutura Multi-Thread

O Maestro possui um gerenciador de tarefas que provê uma estrutura unificada de interface para o gerenciamento de computações correntes/pendentes. Qualquer computação pode ser envolvida em uma tarefa em uma classe java para que seja enviada ao gerenciador, e este avalia o número real de threads correntes, sendo que os processos para execução são escolhidos mediante o número de núcleos do processador da máquina do controlador.

Os pacotes OpenFlow recebidos dos soquetes são envolvidos em tarefas juntamente com o código estágio de entrada e colocados na fila de tarefas de pacotes.Qualquer thread em execução disponível irá puxar uma tarefa desta fila, de modo que execute o código de entrada para processar o pacote em questão. No final do estágio de entrada, haverá pedidos de fluxo gerados para serem processados a partir da fase de solicitação de fluxo (relativas às aplicações "Authentication" e "RouteFlow"), e estas requisições são envolvidas em tarefas juntamente com o código da aplicação, colocando-as na fila de tarefa específica deste segmento de trabalho.

Ao final da fase de solicitação de fluxo, as mensagens de configurações geradas e as próprias solicitações de fluxo são envolvidas em tarefas juntamente com o código do estágio de saída, adicionados novamente na fila de tarefas específicas. Por fim, as threads operantes irão executar essas tarefas de saída com o intuito de enviar as mensagens para seus respectivos switches destino. O Maestro permite múltiplas instâncias simultâneas por diferentes segmentos das threads, considerando que cada aplicação é uma aplicação simples e single-threaded.

Também é uma opção deixar a carga de multi-threading aos programadores de aplicações em alto nível, como por exemplo, também nas aplicações da fase de solicitação de fluxo, os programadores podem dividir o fluxo de pedidos em várias partes, e criar posteriormente uma sequencia de tópicos java para que seja possível manipular as partes simultaneamente.

O Maestro mantém, portanto, um simples modelo de programação, sendo que os programadores de aplicativos não precisam lidar com multi-threading em aplicações, pois podem simplesmente escrever as aplicações em single-threaded. Após realizado e configuradas as aplicações, o sistema Maestro terá capacidade de executar várias instâncias concorrentemente de uma ou mais aplicações através do paralelismo gerado pela abstração do sistema operacional.

B. *Modelo Multi-Threading*

Quando se trata de um modelo criado multi-threading, devem ser definidas maneiras para que o sistema execute de maneira mais otimizada, as metas envolvem a distribuição do trabalho igualmente entre as threads e os núcleos disponíveis, de maneira que não sejam considerados o desequilíbrio a partir da sobrecarga ou o ócio em alguns deles. Através da minimização da sobrecarga que acontece entre núcleos e sincronização do cache, além da minimização do consumo de memória do sistema, com o objetivo de alcançar uma maior eficiência.

Estes casos serão descritos mais detalhadamente a seguir:

Distribuir o trabalho uniformemente

Para maximizar o rendimento do sistema e tentar operar sempre em qualidade máxima, todo trabalho desempenhado pelas threads e núcleos devem estar bem distribuídos, de modo que não haja núcleos ociosos enquanto houver pendência por trabalho em outro núcleo.

A problemática poderia ser facilmente resolvida através da concepção em que os pedidos de fluxo de entrada seriam igualmente distribuídos entre as filas de tarefas específicas das threads operantes, na esperança de conseguir a carga uniforme, porém, no caso do OpenFlow, esta solução não é, necessariamente, sempre verdadeira, pois o determinado pedido de fluxo pode exigir um número arbitrário de ciclos de CPU para processamento. Além disso, a autenticação de segurança realizada sobre os pedidos de fluxo também pode exigir um número variável de ciclos de CPU, dependendo da determinação do nível da política de segurança que é operada pelo fluxo.

Para solucionar, o Maestro possui um gerenciador que define que todas as threads operantes compartilham a mesma fila, de modo que quando há uma tarefa pendente, qualquer thread pode selecionar a tarefa para execução. Para que este método seja mantido, ocorre a sincronização constante entre as threads e a fila de tarefas.

Minimizar a sobrecarga entre núcleos

Quando o código em execução ou pacotes de dados são movidos de um núcleo para outro, ocorre uma sobrecarga no sistema entre os estados do núcleo e o cache, a qual se intensifica ainda mais quando o dado é proveniente de outro processador, pois estes não possuem a mesma memória e essa não é compartilhada entre eles.

No sistema Maestro, ocorre a minimização da sobrecarga entre núcleos através da do "taskset" que realiza o vínculo de cada thread em operação para um núcleo em específico. Com isso, pode-se evitar que o código em execução seja "continuado" em outro núcleo do processador (processo chamado núcleo vinculativo).

Ainda basenado neste problema, uma forma de minimizar a sincronização de cache é assegurando de que todas as computações internas a uma requisição de fluxo são realizadas por um único núcleo de processador, fazendo com que todos os dados gerados e utilizados durante o processo não necessite migrar para de um núcleo para outro.

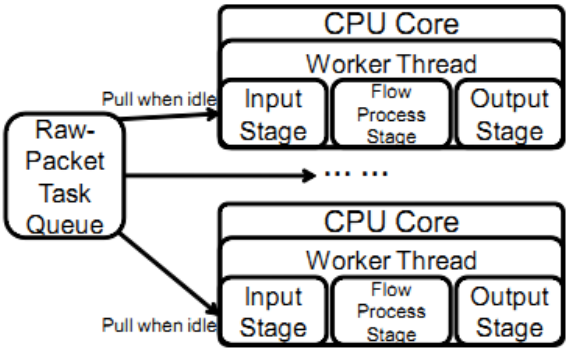

Figura 2 - Thread e núcleos vinculativos

O quadro gerenciador de tarefas fica, portanto, como descrito e ilustrado na figura 2.

Minimizar o consumo de memória

Quando a taxa de solicitações de fluxo de entrada excede a capacidade do sistema, dados como pacotes em estado bruto, requisições de fluxo geradas pelo estágio de entrada e mensagens de configuração geradas pelo estado de saída, esses dados irão acumular.

Se muitos dados estão pendentes nas filas, aguardando que sejam processados, eles estarão consumindo memória e possivelmente prejudicando o desempenho do sistema. Além de que, com o sistema todo implementado em Java, ocorre uma sobrecarga na coleta de lixo, tornando ainda mais elevada a taxa de utilização da memória, fazendo com que, inclusive, a alocação de memória se torne mais lenta.

Visando minimizar o consumo de memória e assegurar um buffer para que os dados possam ser processados, deve-se configurar cuidadosamente um limiar para que mantenha um tamanho razoável de fila para que seja o suficiente para atender as requisições e alocações.

Alternativamente, pode-se empregar prioridade restrita no agendamento de tarefas para os trabalhos das threads, dando prioridades diferentes para as tarefas e para as diferentes fases: Estabelecer prioridade alta para tarefas do estágio de saída, prioridade média para processos de fluxo e prioridade mais baixa para o estágio de entrada, visto que estes estão na fila compartilhada de pacotes.

VI. CONCLUSÃO

Avaliando os termos deste artigo, conclui-se que existe um espaço muito amplo que tende a ser ocupado por sistemas OpenFlow, gerenciados a partir do suporte da abstração do sistema operacional. As funcionalidades e conveniências

contidas no sistema Maestro permitem que a programação seja em um nível mais alto comparada aos demais, além de que o ajuste de alta performance para este sistema o identifica como um padrão viável a ser seguido.

REFERÊNCIAS

[1] T. S. Eugene Ng - Rice University – Computer Science
http://www.cs.rice.edu/~eugeneng/papers/TR10-11.pdf

[2] Robert Rice Brandt - Universidade Federal de Campina Grande
http://dee.ufcg.edu.br/~rrbrandt/cursos/redes/sor.shtml

[3] Tanenbaum, Andrew. Distribuited Systems, Principles and Paradigms. New Jersey : Prentice Hall, 2007.

[4] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martn Casado, Nick McKeown, and Scott Shenker. Nox: Towards an operating system for networks. ACM Computer Communication Review, July 2008. Short technical Notes

[5] Kurose, James F. e Ross, Keith W. Computer networking, a top-down approach featuring the Internetm, 2006.

[6] Maestro Platform - A scalable control platform writen in Java which supports OpenFlow switches
http://code.google.com/p/maestro-platform/

[7] Gustavo M. D. Vieira – Sistemas Operacionais
Universidade Federal de São Carlos, 2010