Serviço de Presença sobre uma Estrutura Gossip em Cloud

Peterson Wilges
UFRGS
pwilges@inf.ufrgs.br

Henrique Dalla Costa Lovison
UFRGS
henrique.lovison@inf.ufrgs.br

Sérgio Luis Cechin
UFRGS
cechin@inf.ufrgs.br

Taisy da Silva Weber
UFRGS
taisy@inf.ufrgs.br

Regina Moraes
UNICAMP
regina@ft.unicamp.br

***Resumo*— Este artigo traz uma implementação de um modelo de serviço de presença sobre uma estrutura *gossip* em *cloud*. Diferente da maioria dos serviços de presença existentes, este foi desenvolvido de forma totalmente distribuída. Através de disseminação epidêmica e a redundância inerente, o serviço é tolerante a falhas de comunicação e perda de mensagens. No cenário de avaliação desenvolvido, mostramos como acontece a propagação para diferentes números de nodos e como ajustes na estrutura *gossip* podem beneficiar a disseminação.**

I. INTRODUÇÃO

Um serviço de presença tem por objetivo gerenciar a presença de entidades em uma rede. Para ser eficiente e tolerante a falhas em um ambiente dinâmico com um grande número de nodos, é necessário que o serviço supere problemas de escalabilidade. Em nossa abordagem desenvolvemos um serviço de presença de nodos em uma rede através de uma estrutura *gossip*. Desta forma, criamos um modelo que segue uma abordagem diferente da sugerida nas normas da IETF através das RFCs 2778 e 2779[1][2]. Nosso serviço, diferente do modelo sugerido, é totalmente distribuído e facilmente escalável.

Os atuais serviços de presença existentes no mercado possuem uma abordagem centralizada. Em contrapartida, nossa aplicação é totalmente descentralizada, sendo cada nodo responsável por gerenciar quais as entidades presentes em uma determinada rede. Essa característica distribuída pode ser muito mais eficiente em termos de tempo de transmissão e de disponibilidade de serviço, como será demonstrado no cenário em que desenvolvemos.

O nosso modelo foi construído usando um protocolo de comunicação epidêmica entre seus participantes. Assim, cada notificação de presença enviada para o grupo interessado, que chamaremos de *cloud*, é retransmitida para outros membros do grupo, que por sua vez, retransmitem a mensagem para outros membros. O protocolo utilizado, chamado NeEM [3][4], provê mecanismos para a aplicação gerenciar o número ideal de nodos para transmissão (o qual chamaremos de *fanout*), que o mesmo deve possuir para atingir uma alta probabilidade de entrega de uma mensagem. É fácil notar, desta maneira, que o protocolo é inerentemente redundante: fazendo uma analogia podemos compará-lo a uma fofoca sendo disseminada em um grupo de pessoas. Uma pessoa do grupo conta uma história para outro integrante, que por sua vez repassa a mesma para outros integrantes. Cada pessoa que ouvir a história, contará novamente para pessoas do grupo, e assim observamos que um indivíduo deve, provavelmente, escutar a mesma história mais de uma vez. Entretanto, o NeEM possui mecanismos que visam amenizar essas transmissões desnecessárias [4]. Também precisamos observar que a mensagem não deve ficar eternamente sendo disseminada entre os participantes, sendo essa retransmissão limitada por um parâmetro do protocolo chamado *time-to-live (TTL)*. O TTL da mensagem determina o número de rodadas para a mensagem ser retransmitida.

Neste trabalho, além de demonstrar como estes mecanismos funcionam, aplicamos alguns testes que demonstram como a disseminação das informações acontece de maneira rápida. Desenvolvemos um cenário utilizando um único computador, já que a sincronização de eventos com base em um único relógio facilita o monitoramento do experimento de teste. Neste cenário, fizemos um nodo lançar uma notificação utilizando o NeEM, que por sua vez, faz a inundação para os outros nodos da rede. Desta maneira, analisamos o tempo de disseminação para um número diferente de nodos. Em uma segunda análise avaliamos o uso de diferentes *fanouts* que junto com o *time-to-live* e o número de nodos determinam a probabilidade de uma mensagem atingir toda a rede [5].

Este artigo está organizado como segue: Na seção II apresentamos uma descrição do protocolo utilizado bem como seu funcionamento. Na seção III descrevemos como funciona nossa aplicação. Na seção IV, descrevemos algumas arquiteturas de serviço de presença existentes. As simulações feitas foram descritas na seção V. E finalmente na seção VI foram apresentadas conclusões sobre os resultados obtidos e possíveis trabalhos futuros.

II. Disseminação epidêmica e protocolo NeEM

Como já mencionado, o nosso aplicativo faz uso da biblioteca Network-friendly Epidemic Multicast (NeEM), responsável pela disseminação epidêmica. O protocolo, também chamado de *gossip-based* ou protocolo probabilístico, faz uso de conexões TCP para gerenciar o conjunto de nodos com o qual cada ponto está conectado. Esse conjunto chama-se *overlay*, e é gerenciado automaticamente pelo NeEM.

A disseminação epidêmica funciona da seguinte maneira: cada nodo se conecta a um número de nodos k, formando o *overlay*. A cada mensagem recebida, o nodo retransmite para f (*fanout*) nodos a mensagem sendo f<k, ou seja, retransmite para um número menor do que as conexões estabelecidas em *overlay*. O número maior de conexões em overlay do que de *fanout* deve-se a uma redundância que permite maior tolerância a falhas bem como melhor gestão dos nodos. Caso um nodo falhar ou cair, outro pode receber a mensagem. Também em toda retransmissão nodos diferentes entre os k nodos são escolhidos (f) e de períodos em períodos conexões novas são estabelecidas mudando a camada de overlay. Isso permite que usuários novos, bem como usuários que deixaram o *cloud,* sejam efetivamente ligados ou desligados do grupo.

O protocolo usado também possui mecanismos que visam amenizar altos usos da banda com transmissões desnecessárias, que é inerente a este tipo de protocolo *gossip*. Para tanto, o protocolo funciona sobre dois modos de operação. O primeiro chamado *eager* para quando a mensagem possui um *payload* pequeno, o modo prevê a transmissão da mensagem tão logo se dê seu recebimento. Já em contrapartida, quando houver um grande *payload*, o modo *lazy* entra em operação, mandando uma mensagem pequena aos nodos conectados, e esperando que aqueles que desejarem a mensagem a solicitem [4].

Protocolos *multicast* epidêmicos possuem um alto rendimento independentemente do número de nodos e falhas na rede. Também são escaláveis para um grande número de participantes. Entretanto, esses protocolos geram um grande volume de tráfego na rede devido à redundância.

III. Serviço de Presença PingCloud

Para fazer a gerência de quais nodos estão presentes no *cloud*, criamos um serviço de presença chamado PingCloud. Geralmente, junto a aplicativos de mensagens instantâneas, os serviços de presença têm por objetivo gerenciar entidades presentes da rede, que podem ser humanos, dispositivos ou simplesmente um aplicativo, como em nosso caso.

Para realizar essa gestão, fizemos uso de notificações, onde cada nodo que deseja informar sua presença manda uma mensagem para o *cloud*, informando sua presença. Essa notificação deve ser enviada em intervalos de tempo determinados por um parâmetro. Cada nodo mantém uma lista com os nodos presentes em determinado momento. O período que um nodo pode ficar na lista sem enviar notificação é determinado por um segundo parâmetro. O período que a lista é verificada, no nodo, para remoção de nodos ausentes é determinado por um terceiro parâmetro. Se um dado nodo presente na lista não enviar uma nova notificação dentro do intervalo de atualização da lista, o mesmo deve ser retirada da lista de presença. O ajuste destes três parâmetros é essencial para o funcionamento do aplicativo. Contudo, para se conectar ao serviço, o nodo precisa conhecer ao menos um nodo presente no grupo.

No PingCloud, não há diferenciação entre tipo de disponibilidade tal como existem nos aplicativos de mensagem instantânea onde o usuário pode estar presente, ausente, ocupado entre outros. Por se tratar da presença ou não de um computador na rede, existem apenas duas possibilidades: estar presente ou não. Para informar a presença, um nodo simplesmente manda uma mensagem notificando a presença. Caso ele não estiver presente ou desejar sair, basta simplesmente não mandar notificações. Junto com a informação também é enviado o tempo em que a mensagem foi gerada para ordenação dos eventos.

É na manutenção da lista de presença que o nosso serviço possui sua grande diferença em relação aos outros. Neste sentido, a arquitetura utilizada no mercado hoje, possui uma abordagem centralizada, sendo um servidor central responsável por gerenciar a rede. Já em nossa arquitetura, cada nodo possui sua lista de presença. Essa lista começa vazia, e a cada mensagem recebida ou a cada período estabelecido, ela é atualizada, retirando os nodos que não mais mandaram atualização e inserindo novos nodos que desejam entrar na rede.

IV. Trabalhos Relacionados

Descreveremos alguns exemplos de serviços de presença de código aberto. Como veremos, todos aqui descritos seguem uma abordagem centralizada.

A. Extensive Message and Presence Protocol (XMPP)

O XMPP [6] é um protocolo aberto para troca de mensagem e informação de presença. Sua arquitetura segue uma abordagem centralizada, já que a lista de presença dos usuários fica em servidores que intermediam toda a comunicação entre os usuários. Todos os nodos (aplicações) são endereçados através de um identificador único. Dois usuários que querem notificar a presença e se comunicar devem cada um fazer uma conexão a um servidor XMPP. Os servidores se comunicam passando as devidas informações de presença e trocando mensagens entre esses dois usuários. Toda a comunicação é feita com trocas de mensagens em formato XML.

B. SIMPLE

O SIMPLE [7] é um protocolo de serviço de presença e mensagem instantânea baseado no protocolo SIP (*Session Initiation Protocol*), que é um protocolo para iniciar, gerenciar e finalizar sessões multimídia como, por exemplo, aplicações de voz sobre IP. O *SIMPLE* possui três métodos adicionais ao SIP para troca de mensagem e informação de presença:

- *Subscribe:* método invocado quando um nodo quer receber alguma informação de presença.

- *Notify:* invocado para enviar informação de presença.
- *Message:* invocado para enviar uma mensagem.

A comunicação entre dois usuários pode ser feita por P2P, desde que os nodos se conheçam antecipadamente. Caso os usuários não se conheçam, eles fazem uso de um servidor central que contém a lista de presença e os respectivos endereços.

C. Wireless Village(WV)

Fundado pela Ericson, Motorola e Nokia o WV [8] foi desenvolvido para prover um conjunto de especificações universais para mensagem instantânea e serviço de presença em dispositivos móveis. Usa uma arquitetura cliente-servidor, onde o cliente é qualquer terminal móvel e o servidor é o Wireless Village Server. O WV Server é responsável pelo gerenciamento de presença bem como outras funcionalidades, como troca de mensagens, gerenciamento de grupos e compartilhamento de arquivos.

V. Análise Sobre o Serviço de presença PingCloud

Para analisar o funcionamento do modelo de serviço de presença desenvolvido, construímos um cenário para demonstrar seu funcionamento. O cenário é construído através de um único computador que simula vários nodos. Um nodo envia uma notificação que se propaga para os outros nodos. Desta maneira, fizemos duas avaliações. Na primeira, desconsideramos o atraso de propagação e simplesmente fizemos a simulação para diferentes números de nodos. Na segunda, simulamos um atraso de propagação de 100ms para todos os pacotes através de uma ferramenta de injeção de falhas [9]. A ferramenta permite introduzir falhas de comunicação como perda e corrupção de mensagens e também atrasos em algumas ou todas as mensagens e tem sido usada para avaliação de protocolos de comunicação [10].

Simulamos uma rede de tamanho fixo de nodos e diferentes configurações de *fanout*.

De acordo com estes cenários foram feitas algumas simulações e seus resultados serão descritos nesta seção.

A. Avaliação do tempo de disseminação de uma notificação de presença

A primeira avaliação foi feita para verificar em quanto tempo uma dada notificação de presença demora a chegar a todos os nodos presentes no *cloud,* desconsiderando o tempo de propagação. Para esta avaliação, criamos diferentes números de nodos presentes na nuvem para avaliar o serviço. Enviamos uma mensagem de um dos nodos e coletamos, através dos *logs* gerados, o tempo em que a mensagem chegou aos outros nodos. Os parâmetros do NeEM foram ajustados para *fanout* igual a 5 e com *time-to-live* igual a 6.

Para mandar uma única mensagem do nodo, ajustamos o parâmetro de intervalo de envio de notificação de maneira que o nodo não envie duas mensagens ou mais durante o experimento. Assim determinamos um alto intervalo de envio.

Como nosso aplicativo faz uso de um protocolo de disseminação epidêmica, o tempo para disseminar a informação de presença não cresce linearmente com o aumento da rede. No primeiro momento, quando poucos nodos possuem a mensagem, há pouca redundância de transmissão, a assim, a transmissão para f nodos é total, ou praticamente total. Em um segundo momento, quando uma grande quantidade de nodos já possui a notificação, há muita redundância, sendo os últimos *hops* menos eficientes, já que poucas transmissões serão de fato aproveitadas e a grande maioria, redundante, será descartada. Apesar de neste caso não se fazer necessária, a redundância é uma das características importantes em protocolos epidêmicos, pois o torna tolerante a falhas.

Para demonstrar a disseminação da mensagem no cenário descrito, realizamos a simulação para 15, 25 e 40 nodos, que foi o limite encontrado para um único computador, no qual o resultado não fosse prejudicado pelo grande número de *threads*.

Como observado anteriormente, é possível visualizar no gráfico mostrado na figura 1 que a notificação é disseminada rapidamente em sua fase inicial, e vai perdendo desempenho com o passar do tempo.

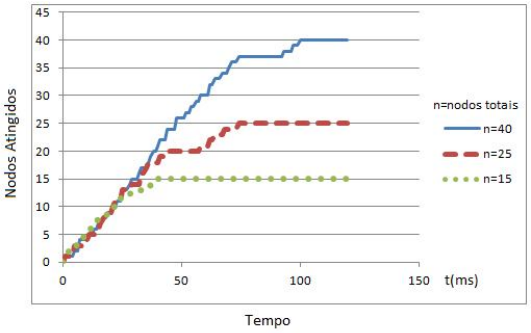

Figura 1 - Análise gráfica para diferentes números de nodos

B. Avaliação do tempo de disseminação de uma notificação para diferentes fanouts

Nesta avaliação, foram realizados duas simulações, ambas com 30 nodos na rede. Na primeira, foi usado *fanout* igual a 3, e na segunda *fanout* igual a 7. Nesta simulação, utilizamos uma ferramenta chamada FIRMAMENT [9], para colocar um atraso de 100ms em todos os pacotes transmitidos.

Desta maneira, com os atrasos fixos, é possível analisar como acontece cada *round* de transmissão. Primeiro o nodo que deseja transmitir envia uma notificação para f (*fanout*) outros nodos na rede. Como nenhum destes possui esta notificação todas as transmissões são efetivadas. No segundo *round,* cada um dos f nodos que receberam a mensagem, retransmitirão a mesma para f nodos, que desta vez podem ser

redundantes. No terceiro e seguintes *rounds*, acontece o mesmo que no passo dois, com a diferença que quanto maior o *round* maior será a o número de transmissões redundante.

Agora, analisando a simulação para *fanout* 3, é possível observar que no primeiro *round* 3 notificações serão enviadas. No segundo, serão enviadas 9 notificações, havendo uma probabilidade de algumas mensagens serem redundantes. No terceiro, serão transmitidas 3 vezes o número de nodos que receberam a mensagem no *round* anterior, descartando aqueles que só receberam mensagens redundantes, já que estes nodos simplesmente descartarão esta mensagem. Desta maneira, é importante observar que se em uma rodada todas as mensagens transmitidas forem redundantes, todas serão descartadas e a propagação é encerrada.

Para *fanout* igual a 7, acontece analogamente a *fanout* igual a 3 com a diferença que cada nodo deve transmitir a mensagem para 7 outros nodos. É fácil observar assim, que a disseminação é mais rápida e menos rodadas serão necessárias.

De acordo com este entendimento, é possível observar na figura 2, os diferentes *rounds* de transmissão, separados por intervalos de aproximadamente 100ms. Vemos também, que para *fanouts* maiores um número menor de *rounds é* necessários para todos os nodos serem atingidos por uma notificação.

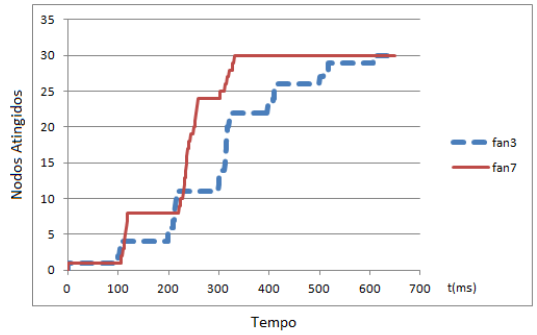

Figura 2 – Análise gráfica para diferentes valores de *fanout*

Convém enfatizar que a ferramenta de injeção de falhas foi usada neste cenário apenas para facilitar a avaliação. Neste caso o atraso inserido apenas emula uma situação de rede com atrasos naturais, não sendo considerada como uma situação de falha.

VI. Conclusões e Trabalhos Futuros

Podemos concluir que o modelo distribuído implementado neste trabalho mostra ser possível usar uma abordagem descentralizada para um serviço de presença. Nossa aplicação foi construída como um protótipo para uma futura aplicação mais elaborada deste serviço. Os resultados demonstraram que o uso do protocolo NeEM foi uma escolha oportuna para a disseminação multicast. Também é possível observar que o ajuste dos parâmetros do protocolo é fundamental para seu funcionamento correto e podem ser modificados pela aplicação de acordo com as necessidades da rede. Por exemplo, numa rede com maior índice de perdas de pacotes podemos aumentar a redundância de mensagens e manter assim uma alta probabilidade de entrega.

Algumas limitações não foram trabalhadas nesta primeira versão da aplicação do serviço de presença, como mecanismos de segurança para um nodo entrar no *cloud*. Outro ponto importante a ser explorado é aprimorar o aplicativo, de modo que ele gerencie os parâmetros do protocolo NeEM em tempo de execução. Parâmetros como *fanout* e *time-to-live*, são relativos ao número de nodos presentes na rede, e desta forma devem ser gerenciados pela aplicação.

Para futuros experimentos, estamos planejando testes com um número maior de computadores para simulação em maior escala. Estamos planejando também que os testes sejam complementados com injeção de falhas de comunicação para permitir, além da eficiência do serviço, extrair medidas como desempenho sob falhas e disponibilidade de serviço.

Agradecimentos

Esse trabalho está sendo desenvolvido no escopo do projeto JitCloud com o apoio da RNP.

Referências

[1] J. Rosenberg, M. Day, e others, “A model for Presence and Instant Messaging”, 2000.

[2] M. Day, S. Aggarwal, G. Mohr, e J. Vincent, “Instant messaging/presence protocol requirements”, Request for Comments, vol. 2779, 2000.

[3] J. Pereira, L.Rodrigues, M.J. Monteiro, R. Oliveira, A. –M Kermarrec Microsoft Resourch “NEEM: Network-friendly Epidemic Multicast”, 2003.

[4] J. Leitao, J. Pereira, e L. Rodrigues, “HyParView: A membership protocol for reliable gossip-based broadcast”, in Dependable Systems and Networks, 2007. DSN’07. 37th Annual IEEE/IFIP International Conference on, 2007, p. 419–429.

[5] P. T. Eugster, R. Guerraoui, A. M. Kermarrec, e L. Massoulié, “Epidemic information dissemination in distributed systems”, *Computer*, vol. 37, nº. 5, p. 60–67, 2004.

[6] P. Saint-Andre. “Extensive Messaging e Presence Protocol (XMPP)”, 2011.

[7] A. Niemi, “Session initiation protocol (SIP) extension for event state publication”, 2004.

[8] Ericsson, Motorola and Nokia. “The Wireless Village initiative: System Architecture Model”, 2001-2002.

[9] R. Drebes, G. Jacques-Silva, J. da Trindade, e T. Weber, “A kernel-based communication fault injector for dependability testing of distributed systems”, *Hardware and Software, Verification and Testing*, p. 177–190, 2006.

[10] T. Siqueira, B. Fiss, R. Weber, S. Cechin, e T. Weber. 2009. “Applying FIRMAMENT to test the SCTP communication protocol under network faults”. In Test Workshop, 2009. LATW ’09. 10th Latin American, 1 -6.