Avaliação de consumo de energia em aplicação peer-to-peer para dispositivos de borda

Lucas D. Fonseca, Cícero A. S. Camargo,
Maurício L. Pilla, Gerson Geraldo H. Cavalheiro

CDTec Computação – UFPEL
Campus Universitário, s/n – Caixa Postal 354 – 96010-900
Pelotas – Brasil

{ldfonseca,cadscamargo,pilla,gersonc}@inf.ufpel.edu.br

***Resumo*—Este artigo estuda o consumo de energia em protocolos peer-to-peer no contexto da distribuição de arquivos em redes de dispositivos de borda e sensores sem fio, osquais possuem limitações de poder de processamento e bateria. Os protocolos Gnutella, Chord, CAN, Pastry e Tapestry foram comparados com a abordagem cliente-servidor em simulações com as ferramentas Simgrid e Triva, estudando o custo das mensagens trocadas. Os protocolos peer-to-peer apresentaram menor consumo para distribuir um arquivo em uma rede com 10 dispositivos, sendo que os melhores resultados foram obtidos com o protocolo Chord. A arquitetura cliente-servidor apresentou os piores resultados, como esperado, devido à concentração da carga em um único servidor. O custo de execução do protocolo Gnutella foi comparado com o cliente-servidor, com resultados muito semelhantes em termos de consumo de energia.**

I. Introdução

A base da computação pervasiva é a integração entre os sensores ou dispositivos embarcados e o ambiente do usuário. Estes dispositivos apresentam graves limitações de recursos, como capacidade de processamento, capacidade de armazenamento e energia. Apesar da evolução natural da tecnologia, acredita-se que os dispositivos portáteis permanecerão apresentando limitações, principalmente se comparados ao ambiente de rede fixa [1]. Deve-se então otimizar a utilização desses recursos de modo a manter a funcionalidade e maximizar o desempenho do sistema. Para isto é necessário encontrar uma maneira de distribuir de modo uniforme a carga computacional da rede entre os vários nós desta de forma a minimizar a energia e o tempo de processamento gastos em cada nó.

A abordagem cliente-servidor pode não se mostrar satisfatória nesse objetivo devido à centralização e, consequentemente, sobrecarga dos nós servidores e subutilização dos nós clientes, principalmente quando não há nós servidores apropriados e conectados à alimentação de energia. Uma alternativa a essa abordagem é o uso de uma arquitetura *peer-to-peer* (P2P) para comunicação, onde todos os nós podem tanto realizar requisições quanto atendê-las.

Neste artigo, as abordagens cliente-servidor e peer-to-peer no nível de aplicação para a distribuição de arquivos em um cenário com nós uniformes são avaliadas em função da energia consumida. Os protocolos Gnutella, Chord, CAN, Pastry e Tapestry são avaliados usando as ferramentas SimGrid, Triva e Avrora e o sistema operacional TinyOS.

A divisão do artigo é como segue. Na Seção II são discutidas as principais diferenças entre os protocolos estudados. Na Seção III, as ferramentas usadas são apresentadas. O *workflow* de simulação e avaliação é detalhado na Seção IV. Na Seção V, os resultados das simulações são analisados. Finalmente, a Seção VI apresenta as conclusões e trabalhos futuros.

Trabalho realizado com recursos do projeto Green Grid PRONEX FAPERGS/CNPq e Edital Universal CNPq

II. Protocolos Peer-to-Peer

O Gnutella [2] define uma arquitetura onde cada nó, ao realizar uma pesquisa por um recurso da rede, envia uma mensagem para todos os seus vizinhos. Os vizinhos, por sua vez, também encaminham a mensagem para todos os seus contatos. Essa busca acaba quando o recurso for encontrado ou quando a mensagem passar por um certo número de nós, evitando que a busca fique indefinidamente na rede. Esse protocolo requer um grande poder de processamento e largura de banda, além de não garantir a acessibilidade dos recursos. Para resolver estas limitações existem os protocolos de arquitetura estruturada, onde a organização da rede ocorre através de um procedimento determinístico. O procedimento mais amplamente utilizado é organizar os recursos da rede em tabelas *hash* distribuídas.

Em sistemas baseados em Tabela *Hash* Distribuída (*Distributed Hash Table* - DHT), os nós e recursos da rede recebem uma chave identificadora, normalmente calculada através de uma função *hash* consistente [3]. Os protocolos mais conhecidos baseados em DHT são Chord, CAN, Pastry e Tapestry (agora Chimera). Em todos os citados, o processo de descoberta de um novo nó é externo ao protocolo.

No Chord [4], cada nó é mapeado em uma chave de m bits, proporcionando o máximo de 2^m nós. Estes são ordenados de forma crescente por identificador, em forma de anel. Cada nó conhece seu sucessor. Recursos também

possuem identificadores e são armazenados nos nós de acordo.

O protocolo CAN (*Content Addressable Network*) [5] define identificadores como pontos em um espaço cartesiano d-dimensional virtual. Cada nó é responsável por uma área, definida de acordo com sua localização virtual.

O protocolo Pastry [6] organiza nós e recursos em um espaço circular virtual ordenado pelas chaves identificadoras, sendo baseado no algoritmo apresentado em [7]. As chaves identificadoras têm 128 bits. Cada nó mantém três tabelas de roteamento. A primeira mantém um conjunto de folhas, com os identificadores mais próximos ao nó local. A segunda tabela mantém $\lceil \log_{2^b} N \rceil$ linhas e 2^b colunas, sendo N o número de nós na rede. Cada linha se refere ao nó cujo prefixo possui n bits em comum com o nó presente. A última tabela mantém os nós mais próximos de acordo com alguma métrica, tal como latência.

O protocolo Tapestry [8], agora Chimera, é similar ao protocolo Pastry. No Chimera cada nó possui uma tabela com informações de alguns nós da rede. Esta tabela é dividida em níveis, onde o i-ésimo nível da tabela possui a localização dos nós conhecidos com $i-1$ bits do prefixo do identificador iguais aos do prefixo do identificador do nó local. Assim, a busca por uma chave acontece gradativamente nos níveis a cada *hop*. Na publicação, o nó que possuir o identificador mais próximo numericamente do identificador do recurso será responsável pela localização deste. Um nó ao publicar um recurso envia uma mensagem de publicação para os nós de seu primeiro nível. Estes nós por sua vez encaminham a publicação para o seu segundo nível e assim, sucessivamente até encontrar o nó que ficará responsável pela localização do recurso. Durante a publicação, todos os nós que participaram desta também guardam a informação da localização do recurso publicado. A obtenção de um recurso é semelhante a publicação. Para se obter um recurso faz-se uma pesquisa baseada na identificação do recurso pretendido. Como todos os nós que participaram da publicação sabem onde está o recurso previamente publicado, basta achar um nó que faz parte da árvore de publicação do recurso procurado para obter sua localização.

III. Ferramentas de Simulação

TinyOS [9] é um sistema operacional minimalista desenvolvido para redes de sensores sem fio. Este sistema foi projetado com foco em operações que exijam o mínimo consumo de energia, devido às severas restrições dos microcontroladores dos sensores. O TinyOS fornece abstrações para serviços como sensoriamento, comunicação e armazenamento. A comunicação entre os componentes ocorre através das interfaces, logo uma hierarquia de componentes é estabelecida. A execução de aplicações se dá através de interações de componentes do usuário e do sistema operacional.

Avrora [10] é um conjunto de ferramentas de simulação e análise para programas escritos para microcontroladores AVR encontrados nos sensores ATMel e Mica2. A simulação é realizada em nível de instrução, o que garante grande precisão. É possível simular redes de sensores, adquirir o consumo de energia, em Joules, atingido em cada sensor, verificar o número de ciclos que cada instrução do programa consome, conferir as chamadas e retornos de funções, checar o uso de memória da aplicação, entre outras funcionalidades. A simulação gera um relatório, formatado em texto plano, com todas as informações requeridas pelo usuário.

Triva [11] é uma ferramenta usada para analisar arquivos de traço no formato *paje*, registrados durante a execução de aplicações paralelas. Em conjunto com a biblioteca GraphViz [12], Triva apresenta informações visuais sobre o comportamento da aplicação monitorada. É possível gerar o grafo da rede simulada de modo a expor sua topologia ou somente observar o gasto de cada nó e as conexões de rede.

O SimGrid [13] é um conjunto estruturado de serviços de código aberto implementado em linguagem C para a simulação orientada a eventos. O módulo MSG tem como objetivo facilitar a prototipação de aplicações distribuídas, abstraindo detalhes como o modo de comunicação. *Traces* são gerados e avaliados, apresentando as mensagens trocadas entre os nós da rede.

IV. Metodologia de Simulação

O primeiro passo no ambiente SimGrid é gerar a plataforma sobre a qual a simulação será executada. Esta plataforma será definida em um arquivo XML, gerado com o auxílio da ferramenta Simulacrum. Após, deve-se definir a aplicação a ser simulada e os processos presentes nesta em um arquivo em linguagem C. No momento seguinte, define-se o arquivo XML com a descrição da distribuição dos processos entre os nós. Uma vez definidos estes arquivos, é possível executar a simulação no SimGrid.

A simulação gera um arquivo de traço de execução, o qual é passado como entrada para o software Triva. O Triva, por sua vez, analisa o arquivo e gera o gráfico desejado, apresentando características específicas da simulação.

No ambiente TinyOS, primeiramente define-se um arquivo fonte nesC com a aplicação a ser executada. Este arquivo é compilado gerando um arquivo objeto. Este arquivo é processado pelo aplicativo *avr-objdump*, presente em sistemas GNU/Linux, gerando um novo arquivo, este executável em arquiteturas passíveis de simulação no Avrora. A simulação disponibiliza as informações de gasto de energia obtidas.

V. Resultados

Para realizar comparações entre todos os protocolos desenvolvidos foi criado um cenário onde há uma rede com dez nós e um arquivo a ser distribuído entre eles. Na abordagem cliente-servidor há dez clientes e um servidor, enquanto nas abordagens P2P há somente dez *peers*. Há basicamente quatro tipos de mensagens que serão utilizadas: *(i)* mensagens de requisição, utilizadas por um nó que deseja receber alguma informação ou recurso; *(ii)* respostas a estas requisições; *(iii)* mensagens que

encapsulam arquivos; e *(iv)* mensagens de confirmação de recebimento do arquivo.

Mensagens de requisição possuem tamanho muito pequeno e, portanto, não são contabilizadas. Respostas às requisições apresentam tamanho pequeno e constante, empiricamente escolhido como cinco *kilobytes*. Mensagens de confirmação e a mensagem que encapsula o arquivo foram definidas como 1 MiB. O tamanho do arquivo foi escolhido com o objetivo de produzir uma carga significativa na rede, sem sobrecarga. O processamento de cada mensagem é proporcional ao seu tamanho.

Os experimentos foram divididos em medições do custo das mensagens na rede e medições do custo dos algoritmos de roteamento nos dispositivos. Medições da rede foram simuladas usando SimGrid, enquanto que as demais foram executadas no TinyOS.

No SimGrid foram desenvolvidas seis simulações de protocolos de comunicação diferentes. Uma das simulações usa a abordagem cliente-servidor e as restantes utilizam as abordagens P2P discutidas na Seção II: Gnutella, Chord, CAN, Pastry e Tapestry.

Foram criadas duas plataformas: uma para o cenário de cliente-servidor e outra para os cenários de P2P. Ambas plataformas são bastante semelhantes, sendo a principal diferença a presença de um servidor central na plataforma para cliente-servidor. Estas plataformas foram criadas utilizando a ferramenta Simulacrum [14].

As simulações são determinísticas no SimGrid, portanto não é necessário executar o mesmo experimento múltiplas vezes. No Avrora, cada experimento foi executado 30 vezes e a média calculada.

A. Resultados sobre Rede

O consumo médio nas simulações do cliente-servidor, de 16528 u.e., e o respectivo desvio-padrão, 24700 u.e. A média não segue a distribuição de Poisson, com a maior parte da energia sendo gasta no servidor (90909 u.e.), como esperado. Como cada cliente apenas busca o arquivo no servidor, a energia gasta é de 1/10, ou seja, 9090 u.e.

A Figura 1 apresenta o consumo médio e o desvio-padrão para os protocolos P2P. Um desvio-padrão é representado como o erro sobre cada coluna. O consumo médio do protocolo Chord é o menor, com 627 u.e. Em segundo lugar, o protocolo CAN tem consumo médio de 678 u.e. O protocolo Gnutella apresentou o maior consumo médio, com 1557 u.e., mais que o dobro do Chord e do CAN. Em termos de desvio-padrão, o melhor resultado foi obtido com o protocolo Tapestry, com 119 u.e., apesar do consumo ligeiramente superior ao Chord e ao CAN. O protocolo Chord apresenta um desvio-padrão de 347 u.e. O Gnutella novamente apresenta os piores resultados, com desvio-padrão de 981 u.e.

Os resultados piores para o Gnutella, em relação aos outros protocolos P2P, deve-se principalmente ao fato de não ser um protocolo estruturado, como os de tabela *hash* distribuída. As operações de busca por recursos envolvem enviar mensagens para todos os vizinhos, as quais são propagadas para todos os vizinhos do vizinho. Este *flooding* gera um grande número de mensagens e, em consequência, maior consumo. Ainda assim, o Gnutella apresentou consumo médio 10 vezes menor que o cliente-servidor.

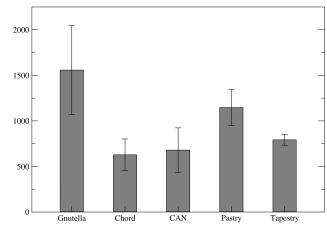

Figura 1. Consumo médio e desvio padrão (u.e.) para protocolos peer-to-peer

A Figura 2 mostra a distribuição do consumo entre os peers no protocolo Chord. Cada coluna corresponde a um peer, sendo que a parte superior é relacionada às mensagens usadas para organizar a rede e a parte inferior ao consumo na transferência de arquivos propriamente dita. Note-se que não é necessário realizar uma etapa de organização para cada transferência de arquivos. A distribuição do consumo com grande variação deve-se principalmente ao fato que cada nó distribui o arquivo para um número de nós variável, ao contrário do protocolo cliente-servidor, onde a carga fica concentrada em um só nó.

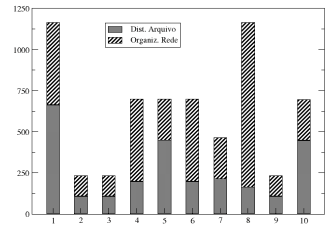

Figura 2. Distribuição de consumo entre os peers no protocolo Chord

B. Resultados sobre Processamento

No TinyOS foram desenvolvidas duas aplicações: uma no modelo cliente-servidor e uma P2P, utilizando o protocolo Gnutella explicado anteriormente. O protocolo Gnutella foi escolhido para comparar o pior caso dentre os algoritmos P2P com o cliente-servidor. Ambas utilizam o cenário descrito no início desta seção.

A ferramenta Avrora foi utilizada para simular a rede de sensores e analisar o gasto de energia de cada nó.

Cada simulação foi executada 30 vezes, sendo que os resultados exibidos aqui expressam a média das execuções. A unidade de energia usada pelo Avrora para medir o gasto de energia é o Joule.

A Tabela I apresenta o gasto médio de energia registrado em cada nó da rede durante a distribuição de um arquivo utilizando a abordagem Cliente-Servidor e o protocolo Gnutella. Para ambos a diferença entre clientes (ou peers, no caso do Gnutella), em termos de consumo no dispositivo, são muito pequenas. Como esperado, o Gnutella consome mais nos dispositivos, visto que além de transmitir o arquivo ainda deve encontrá-lo primeiro. A diferença de consumo dos peers para os clientes é muito pequena, de 7,2% apenas. No entanto, mesmo contando a energia gasta pelo servidor, o cliente-servidor ainda é mais econômico em média, necessitando de 6,5% menos energia que o Gnutella. Mas como há um nodo servidor a mais, o gasto final é maior, com 3,5332 J no cliente-servidor e 3,4350 J no Gnutella.

Tabela I
CONSUMO MÉDIO DE ENERGIA NOS DISPOSITIVOS

Protocolo	Consumo (J)	Diferença
Gnutella	0,3435	-
Cliente-Servidor (só clientes)	0,3190	7,2%
Cliente-Servidor	0,3212	6,5%

VI. CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, foram comparados dois cenários para a comunicação de um arquivo para dispositivos de borda. O primeiro cenário consistiu de uma arquitetura cliente/servidor, enquanto que, no segundo cenário, cinco algoritmos P2P foram testados.

As simulações com SimGrid e TinyOS demonstraram que a abordagem cliente-servidor não distribuiu nem minimizou o gasto de energia de forma eficiente. Os protocolos P2P apresentaram melhores resultados, sendo que o Chord mostrou-se o mais econômico em termos energéticos. O protocolo Tapestry apresentou a distribuição mais uniforme no consumo, seguido pelo protocolo Chord. Durante a fase de organização da rede, o menor consumo foi observado para o protocolo CAN. Na fase de roteamento e distribuição de arquivos, novamente o protocolo Chord apresentou melhores resultados. Dentre todos os protocolos P2P estudados, os piores resultados foram observados para o Gnutella. Desta forma, conclui-se que para cenários com poucos dispositivos, o Chord é a melhor escolha.

Como trabalhos futuros, espera-se simular e avaliar diferentes ambientes, com ênfase para redes com grande número de dispositivos. Também sugere-se implementar os protocolos baseados em DHT no TinyOS, para obter resultados mais apurados em termos de medidas de eficiência energética..

REFERÊNCIAS

[1] M. Satyanarayanan, "Pervasive computing: Vision and challenges," *IEEE Personal Communications*, vol. 8, pp. 10–17, 2001.

[2] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *Proc. of First Intl. Conf. on Peer-to-Peer Computing (P2P 2001)*, Linkoping, Suecia, Agosto 2001, pp. 99–100.

[3] A. R. Silva, H. M. P. Almeida, T. Macambira, D. O. Guedes, W. Meira, and R. A. C. Ferreira, "Hash consistente como uma ferramenta para distribuição de tarefas em sistemas distribuídos reconfiguráveis," in *Anais do WSCAD 2005*, Rio de Janeiro, 2005, pp. 169–176.

[4] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," in *Proc. of IEEE/ACM Trans. on Networking*, 2002.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. of ACM Sigcomm*, 2001.

[6] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM Intl. Conf. on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, Nov. 2001, pp. 329–350.

[7] C. G. Plaxton, R. Rajaraman, A. W. Richa, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," 1997, pp. 311–320.

[8] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, pp. 41–53, 2004.

[9] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "Tinyos: An operating system for sensor networks," in *Ambient Intelligence*. Springer Berlin Heidelberg, 2005, pp. 115–148.

[10] B. Titzer, D. K. Lee, and J. Palsberg, "Avrora: Scalable sensor network simulation with precise timing," in *Proceedings of IPSN 04*, 2004.

[11] L. M. Schnorr, G. Huard, and P. O. Navaux, "Triva: Interactive 3d visualization for performance analysis of parallel applications," *Future Generation Computer Systems*, vol. 26, no. 3, pp. 348–358, 2010.

[12] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull, "Graphviz - Open Source Graph Drawing Tools," *Graph Drawing*, pp. 483–484, 2001.

[13] H. Casanova, A. Legrand, and M. Quinson, "Simgrid: a generic framework for large-scale distributed experiments," in *10th IEEE Intl. Conf. on Computer Modeling and Simulation*, Mar. 2008.

[14] M. Quinson, L. Bobelin, and F. Suter, "Synthesizing Generic Experimental Environments for Simulation," in *V Intl. Conf. on P2P, Parallel, Grid, Cloud and Internet Comp.*, Fukuoka, Japão, 11 2010.