Adaptação do Middleware de Grids POP-C++ para Redes de Sensores sem Fio

Augusto Born de Oliveira, Lucas F. Wanner e Augusto A. Fröhlich

[1]Laboratório de Integração de Software e Hardware
Universidade Federal de Santa Catarina
CP 476 – 88049-900 – Florianópolis, SC, Brasil

{augusto, lucas, guto}@lisha.ufsc.br

Resumo. *O tópico de interação entre Redes de Sensores sem Fio (RSSFs) e outros sistemas computacionais tem recebido relativamente baixa atenção científica, e a interface entre a fonte de dados e as aplicações que usam esses dados continua um problema para o programador da aplicação. Este trabalho estende o POP-C++, uma linguagem de programação e sistema de suporte de tempo de execução para programação do Grid para permitir que aplicações do Grid utilizem as capacidades de sensoriamento e processamento das RSSFs de forma transparente e concorrente.*

1. Introdução

Apesar das Redes de Sensores sem Fio (RSSFs) terem sido o foco de muitos esforços de pesquisa nos anos recentes, o tópico de interação das RSSFs com outros sistemas computacionais tem recebido relativamente baixa atenção. Os esforços de pesquisa que dão atenção a isso, como o TinyDB e o Cougar, abstraem os nodos de sensoriamento individuais e dão acesso à RSSF como um todo, permitindo que aplicações façam consultas como fariam em uma database; enquanto esse nível de abstração permite a otimização de consultas, minimizando a quantidade de mensagens a serem enviadas pelo meio sem fio, ele tira controle do programador de aplicação, já que a exploração da capacidade de processamento dos nodos da RSSF se torna difícil. Além disso, uma solução deste tipo não esconde a fronteira entre a RSSF e o resto do sistema computacional.

Em contraste com essas estratégias, este trabalho integra RSSFs e o Grid transparentemente sem remover controle do programador da aplicação através da extensão do POP-C++ [Nguyen and Kuonen 2007]. POP-C++ é uma linguagem de programação e sistema de suporte de tempo de execução orientada a objetos, capaz de suportar objetos paralelos distribuídos em uma rede. Os objetivos específicos desta extensão são permitir que:

- Aplicações no Grid comuniquem-se com RSSFs transparentemente: Ao esconder toda interação com a rede em uma interface de chamada remota de métodos, detalhes da pilha de rede e do meio físico se tornam transparentes para a aplicação;
- Uso concorrente das capacidades de sensoriamento das RSSFs por múltiplas aplicações do Grid: Ao permitir que múltiplos objetos sejam executados em cada nodo e que cada objeto seja utilizado por múltiplas interfaces, uso concorrente da RSSF por múltiplas aplicações se torna possível;

- Software independente de aplicação nos nodos de sensoriamento: ao permitir que aplicações usem um conjunto comum de métodos, espera-se minimizar a necessidade de reprogramações custosas da memória de progrma dos nodos e permitir aplicações adicionais a serem iniciadas após a instalação da RSSF.

2. POP-C++

POP-C++ é uma extensão de C++ criado para suportar objetos distribuídos, paralelos e orientados a requisitos. No modelo de objetos do POP-C++, objetos paralelos têm a habilidade de descrever suas necessidades de recurso em tempo de execução e serem alocados em qualquer um dos nodos do Grid que pode suportar sua execução. O processo de procura por um nodo adequado e transmissão do código a ser executado é transparente para o programador. O POP-C++ também tem semânticas especiais de invocação de métodos, porém a sintaxe de invocação de métodos não difere entre objetos remotos ou locais. Além disso, objetos paralelos são compartilháveis, isto é, referências para um objeto podem ser passadas em qualquer método, local ou remoto.

A arquitetura de tempo de execução do POP-C++ consiste em três objetos reais para cada classe paralela que o usuário implementa: a Interface, o Broker e o Objeto real. A Interface é um objeto por si só, instanciado no lado do invocador; ela compartilha a interface de métodos com o Objeto real, provendo transparência de interação para a aplicação.

O Broker é o correspondente da interface no lado do invocado; ele recebe chamadas de método da rede, desempacota os dados, invoca o método no Objeto real e então re-empacota os valores de retorno e os envia de volta à Interface. O Objeto real é a implementação do usuário, com o código que deve ser distribuído.

O POP-C++ introduz duas extensões de sintaxe no C++ em adição às declarações de classes paralelas: Descrições de Requisitos e Semânticas de Métodos.

- Descrições de requisitos: Usando um descritor de objetos associado, o desenvolvedor pode expressar requisitos de recursos na forma de um endereço de rede, o número de MFlops necessário, a quantidade de memória necessária e a largura de banca necessária para comunicação entre o Objeto e suas Interfaces.
- Semânticas de invocação de métodos: As opções de semânticas de invocação são definidas em tempo de compilação pelo programador de aplicação e podem ser classificadas em dois tipos, no lado da Interface ou no lado do Objeto.
  - A semântica do lado da interface pode ser Síncrona ou Assíncrona; ela controla em que momento o método do lado da Interface retorna. No modo síncrono, o invocador espera até que o método no Objeto retorne, de forma análoga à invocação tradicional. Métodos Assíncronos retornam imediatamente, permitindo que o Invocador continue a execução.
  - Semânticas do lado do Objeto podem ser Mutex, Sequencial ou Concorrente. Semântica Mutex garante que não exista concorrência no Objeto, Sequencial garante que não haja concorrência entre os métodos com essa semântica e a semântica Concorrente permite execução completamente multi-thread.

```
parclass SensorNode
{
    public:
        SensorNode(int node, string machine) @{ od.url(machine);};

        async seq void setLEDs(char val);
        sync conc int getTemperature();

};
```

Figura 1. Classe POP-C++ SensorNode Básica

3. Estendendo POP-C++ para RSSFs

Para dar um modelo uniforme com o qual o programador de aplicação possa criar aplicações do Grid que usem RSSFs, o modelo POP-C++ foi estendido para RSSFs. Isto significa que não só o programador deve ser capaz de instanciar Interfaces nos nodos de sensoriamento para Objetos em execução em outros nodos, mas também instanciar Interfaces para esses Objetos de dentro do Grid. Não deve existir diferença entre chamdas de métodos "normais" (Grid-para-Grid), e chamadas feitas do Grid para as RSSFs.

A figura 1 ilustra a implementação e instanciação de um Objeto que é executado na RSSF e recebe chamadas de função do Grid. A implementação tem métodos que lêem valores do sensor de temperatura e atribui um valor a ser mostrado nos LEDs do nodo. Qualquer nodo no Grid pode instanciar uma Interface a esse Objeto e transparentemente fazer chamadas de método a ele.

3.1. Diferenças fundamentais entre o Grid e as RSSFs

Devido à natureza de poucos recursos dos nodos das RSSFs, a implementação do sistema de suporte de tempo de execução POP-C++ para RSSFs teve de ocupar a menor quantidade de memória principal e de programa quanto possível. Isto obrigou a realização de alguma concessões na implementação:

- Protocolos de comunicação constantes: Num ambiente como o Grid, Quanto num ambiente de Grid, é não só interessante mas necessário suportar protocolos de comunicação múltiplos, intercambiáveis em tempo de execução. Apesar disso, nas RSSFs o protocolo de comunicação tem grande chance de ser constante e global. Portanto, a implementação não suporta múltiplos protocolos de comunicação em tempo de execução.
- Alocação estática de recursos: No Grid o custo de alocação dinâmica de recursos é apenas fazer a carga e execução de um binário da rede, re-escrever a memória de programa nos nodos de uma RSSF é um procedimento custoso em termos de energia[Dunkels et al. 2006]. Para diminuir a necessidade de reprogramação, os programadores são encorajados a utilizar funções independentes de aplicação para possibilitar que novas aplicações sejam exeuctadas utilizando a RSSF após sua instalação.
- Paralelismo limitado: Em virtude da quantidade muito limitada de memória encontrada nos nodos de sensoriamento, a quantidade de threads que podem ser executadas concorrentemente num nodo também é bastante pequena. Isto significa

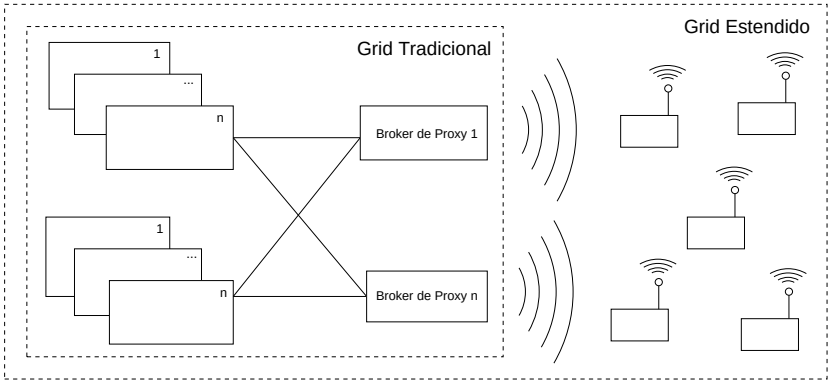

Figura 2. Brokers de Proxy integram o Grid com fio com os nodos da RSSF

que um número limitado de invocações, independentemente de suas semânticas, vão poder ser executadas em concorrência real, e as chamadas de métodos recebidas posteriormente deverão ser enfileiradas.

3.2. Endereçamento

Para permitir acesso direto para cada nodo de sensoriamento indiviual, o método de endereçamento de Objetos do POP-C++ teve de ser estendido. Existe a possibilidade de Interfaces seres instanciadas com um parâmetro de endereço de rede, forçando o Objeto a ser alocado no nodo com esse endereço. Este método foi estendido para RSSFs, onde dois endereços são usados:

- Endereço 1 - Ponto de contato entre o Grid e a RSSF: Toda RSSF necessita pelo menos um ponto de contato com o Grid. O ponto de contato precisa ser capaz de comunicar-se em ambos os protocolos utilizados pelo Grid e pela RSSF, portanto ele vai provavelmente precisar de hardware específico como o transceptor de rádio encontrado nos nodos de sensoriamento. Ao tomar esse endereço como parâmetro, é possível instanciar um Broker de Proxy no nodo apropriado do Grid, criando a ponte lógica que encaminha as invocações de métodos direcionadas à RSSF;
- Endereço 2 - Nodo da RSSF: Esse endereço permite que o Broker de Proxy direcione as chamadas de método para o nodo de sensoriamento correto. O formato dele é deixado em aberto pois métodos diferentes de endereçamento podem ser usados em diferentes RSSFs.

3.3. Broker de Proxy

Para permitir que chamadas de método sejam direcionadas para dentro da RSSF, um objeto Broker especial foi criado. Este é um Broker genérico que simplesmente recebe invocações de métodos do Grid como se fosse o Broker real do Objeto, e então as encaminha para o nodo da RSSF. Uma vez que o nodo de sensoriamento retorna da chamada de método, este Broker encaminha o valor de retorno ao invocador original. Isto cria o efeito de transparência para as Interfaces deste Objeto; para elas, as invocações nunca estão deixando o Grid.

A figura 2 mostra o Grid conectado à RSSF através de Brokers de Proxy; vale notar que podem existir mais de um ponto de contato entre o Grid e cada RSSF.

4. Avaliação do POP-C++ em RSSFs

Nesta seção é avaliado o sobrecusto que a implementação do sistema de suporte de tempo de execução do POP-C++ deste trabalho introduz através da comparação de duas implementações da seguinte aplicação: atribuir e consultar um valor de 8 bits, a ser mostrado nos LEDs do nodo. Uma versão da implementação foi feita em POP-C++ e a outra diretamente sobre o sistema operacional. No caso da implementação sobre o POP-C++, o cliente instancia uma Interface para um Objeto "SensorNode" que é executado em outro nodo, e invoca os métodos `get()` e `set()` para consultar e atribuir o dado. Na implementação nativa, o cliente envia pacotes pré-formatados que são abertos pelo servidor e respondidos com o dado na carga útil.

4.1. Plataforma de Teste: Hardware

Os testes foram realizados usando dois nodos de sensoriamento Mica2 desenvolvidos em Berkeley; eles utilizam um rádio CC1000 de um canal, um microcontrolador Atmel Atmega128 de 8 bits a 8MHz, 4KB de memória principal e 128KB de memória flash para programa.

4.2. Plataforma de Teste: Software

Para prover o suporte para comunicação, gerência de memória e concorrência que ambas aplicações necessitam, foi utilizado o Embedded Parallel Operating System[Fröhlich and Schröder-Preikschat 1999] (EPOS). Ele consiste num framework baseado em componentes para a geração de suporte de tempo de execução para aplicações de computação dedicada. Para este teste, o protocolo de controle de acesso ao meio do EPOS foi configurado para confiabilidade, assegurando a entrega de pacotes através de ACKs constantes e minimizando o atraso através de um ciclo de atividade "sempre-ligado".

4.3. Medições de Performance

- Tamanho de Pacote: O tamanho dos pacotes do POP-C++ são maiores por duas razões:
  - Campo objeto: Para permitir que mais de um Objeto seja executado em cada nodo, os pacotes são individualmente endereçados;
  - Campo de Valor de Semântica: Os valores semânticos do método a ser invocado também são enviados no cabeçalho.

  A adição de funcionalidade equivalente na implementação nativa resultaria num pacote de tamanho similar, que é justificado pela informação adicional que precisa ser transferida ao se suportar um conjunto complexo de aplicações.
- Pedidos por Segundo: Para avaliar o sobrecusto de execução que o sistema POP-C++ adiciona nesta aplicação, foi conduzido um teste de performance que mediu quantos pedidos por segundo puderam ser feitas do nodo cliente para o nodo servidor. A implementação POP-C++ foi capaz de realizar 6,875 chamadas remotas de método por segundo, enquanto a implementação nativa fez 7,046 pedidos por segundo. Esta diferença de 2,42% se deve ao processamento adicional feito pelo sistema de suporte de tempo de execução POP-C++ quando chamadas de método chegam da rede, e também aos bytes adicionais que são transmitidos na implementação POP-C++ da aplicação.

5. Trabalhos Relacionados

A TinyDB [Madden et al. 2003], o Cougar [Yao and Gehrke 2002] e outros esforços de pesquisa [Madden et al. 2002][Bonnet et al. 2001] implementam processadores distribuídos de consultas, colocando grande esforço em otimização de consultas e roteamento eficiente. Usando estas técnicas, considerável redução em consumo de energia foi atingido, além da externalização de uma interface similar a SQL mais amigável ao programador da aplicação. A extensão do POP-C++ apresentada neste trabalho compartilha estes objetivos, mas ao invés de ativamente otimizar a comunicação, ela sai do caminho do programador de aplicação permitindo acesso total ao hardware dos nodos de sensoriamento.

6. Conclusão

Neste artigo descrevemos uma maneira de usar objetos remotos paralelos para integrar o Grid e as RSSFs, estendendo o sistema de suporte de tempo de execução do POP-C++ para as redes de sensores. Acreditamos que usando POP-C++ para realizar esta integração, permitimos ao programador da aplicação que use a RSSF para múltiplas aplicações de maneira transparente, usando interfaces instanciadas localmente para objetos que executam nos nodos de sensor.

Quando comparamos aplicações de funcionalidade equivalentes implementadas com e sem o POP-C++, nosso sistema de suporte de tempo de execução apresentou apenas um pequeno sobrecusto, que foi justificado pela habilidade de suportar múltiplas aplicações concorrentemente.

Referências

Bonnet, P., Gehrke, J., and Seshadri, P. (2001). Towards sensor database systems. In *MDM '01: Proceedings of the Second International Conference on Mobile Data Management*, pages 3–14, London, UK. Springer-Verlag.

Dunkels, A., Finne, N., Eriksson, J., and Voigt, T. (2006). Run-time dynamic linking for reprogramming wireless sensor networks. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 15–28, New York, NY, USA. ACM Press.

Fröhlich, A. A. and Schröder-Preikschat, W. (1999). EPOS: an Object-Oriented Operating System. In *2nd ECOOP Workshop on Object-Orientation and Operating Systems*, volume CSR-99-04 of *Chemnitzer Informatik-Berichte*, pages 38–43, Lisbon, Portugal.

Madden, S., Franklin, M. J., Hellerstein, J. M., and Hong, W. (2002). Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146.

Madden, S., Franklin, M. J., Hellerstein, J. M., and Hong, W. (2003). The design of an acquisitional query processor for sensor networks. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 491–502, New York, NY, USA. ACM Press.

Nguyen, T.-A. and Kuonen, P. (2007). Programming the grid with pop-c++. In *Future Generation Computer Systems (FGCS)*, volume 23. N.H. Elsevier.

Yao, Y. and Gehrke, J. (2002). The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18.