FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA – UNIFOR

ARETUSA MARIA ALMEIDA LOPES

PROCESSAMENTO ADAPTATIVO DE CONSULTAS EM REDES DE SENSORES SEM FIO

Fortaleza
2006

FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA – UNIFOR

ARETUSA MARIA ALMEIDA LOPES

PROCESSAMENTO ADAPTATIVO DE CONSULTAS EM REDES DE SENSORES SEM FIO

Dissertação apresentada ao Curso de Mestrado em Informática Aplicada da Universidade de Fortaleza como requisito parcial para a obtenção do Título de Mestre em Ciências da Computação.

Orientador: Prof. Dr. Angelo Roncalli Alencar Brayner

Fortaleza
2006

ARETUSA MARIA ALMEIDA LOPES

PROCESSAMENTO ADAPTATIVO DE CONSULTAS EM REDES DE SENSORES SEM FIO

Data de Aprovação: _____

Banca Examinadora:

_____

Prof. Angelo Roncalli Alencar Brayner, Dr.-ing.

(Presidente da Banca)

_____

Prof. Javam de Castro Machado, Dr.

(Universidade Federal do Ceará - UFC)

_____

Prof. André Luís Vasconcelos Coelho, Dr.

(Universidade de Fortaleza - UNIFOR)

LOPES, ARETUSA MARIA ALMEIDA

Processamento Adaptativo de Consultas em Redes de Sensores Sem Fio [Fortaleza] 2006

135p., 29,7 cm (MIA/UNIFOR, M. Sc, Ciência da Computação, 2006)

Dissertação de Mestrado – Universidade de Fortaleza, MIA

1. Banco de Dados
2. Processamento Adaptativo de Consultas
3. Redes de Sensores Sem Fio

I. MIA/UNIFOR II. TÍTULO (série)

Ao Magno, meu marido, e Rafael e Maristela, meus pais.

AGRADECIMENTOS

A Deus por me dar forças para concretizar meus objetivos e coragem para enfrentar os obstáculos.

Aos meus pais, Rafael e Maristela, que me ensinaram a buscar meus sonhos guardando valores essenciais como perseverança e honestidade.

Ao meu marido e amigo, Magno, por sua compreensão e incentivo na realização de meus objetivos.

Aos meus irmãos, Ranniere e Rafael, pelo apoio e presteza que sempre me dispensaram.

Ao professor Ângelo Brayner, pelo seu excelente trabalho de orientação que possibilitou a concretização desta dissertação.

Ao amigo Ricardo Vasconcelos, que trabalhou incansavelmente na implementação das propostas apresentadas neste trabalho.

Ao professor Ronaldo Menezes por suas importantes sugestões na revisão dos textos e idéias aqui incluídos.

Aos amigos, Marcos Câmara, Fábio Bezerra, Fernando Parente, Gabriel Júnior, Sergiana Freitas, Gabriela Teles, Nadja Helene e Karol Castro pelo apoio e ajuda.

Ao SERPRO (Serviço Federal de Processamento de Dados) pelo apoio durante o andamento do mestrado.

E a todas as pessoas que contribuíram, direta ou indiretamente, para a realização deste trabalho, meus sinceros agradecimentos.

Resumo da Dissertação apresentada ao MIA/UNIFOR como parte dos requisitos necessários para a obtenção do título de Mestre em Ciência da Computação.

PROCESSAMENTO ADAPTATIVO DE CONSULTAS EM REDES DE SENSORES SEM FIO

Aretusa Maria Almeida Lopes

Novembro / 2006

Orientador: Dr.-Ing. Ângelo Roncalli Alencar Brayner
Programa: Ciência da Computação

Uma Rede de Sensores Sem Fio (RSSF) consiste de grupos de nós-sensores que coletam dados do meio ambiente, como forma de prover informações a respeito de um ou mais fenômenos físicos. Em uma RSSF, os nós-sensores agem colaborativamente, de modo que os dados são passados de um nó para o outro da rede, até alcançarem uma estação-base, requisitante da informação. Estratégias de processamento de consultas destinadas a extrair dados de uma RSSF deveriam ser capacitadas a lidar com as sérias restrições de recursos dos nós da rede: (i) limitada capacidade de processamento, armazenamento e disponibilidade de energia dos nós-sensores; e (ii) limitada disponibilidade de memória principal das estações-base. Esta dissertação propõe um mecanismo para realizar o processamento adaptativo de consultas em RSSFs, a qual consiste em ajustar o processamento da consulta à disponibilidade de recursos dos nós da rede. Adicionalmente, também são investigados caminhos para a especificação de consultas declarativas e modelagens de dados, no contexto das RSSFs. Desta forma, esta dissertação explora o uso da tecnologia de banco de dados como instrumento para prover um uso mais eficiente dos recursos físicos das RSSFs, bem como tornar os detalhes físicos destas redes "transparentes" ao usuário.

Abstract of Thesis presented to MIA/UNIFOR as a partial fulfillment of the requirements for the degree of Master of Science Computer

ADAPTIVE QUERY PROCESSING IN WIRELESS SENSOR NETWORKS

Aretusa Maria Almeida Lopes

November / 2006

Advisor: Dr.-Ing. Ângelo Roncalli Alencar Brayner
Department: Computing Science

A Wireless Sensor Network (WSN) consists of groups of sensor nodes which collect data from the environment in order to provide information about one or more physical phenomena. In a WSN, sensor nodes act in a collaborative way. Collected data are passed from sensor to sensor until the base station is reached, which is responsible for the information request. Query processing strategies responsible for extracting data from a WSN should be able to deal with serious node resource constraints: (i) limited processing capability, memory and battery power of the sensor nodes; and (ii) limited main memory availability of the base stations. This dissertation proposes an approach to perform adaptive query processing in WSNs, which consists in adjusting query processing according to node resource availability. In addition, approaches to specify declarative queries and data models are also investigated in the context of WSNs. Thus, this dissertation exploits database technology as an instrument to provide a higher efficient use of WSN physical resources, as well as to make physical network details “transparent” to users.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1

INTRODUÇÃO

4.1 Motivação

A evolução dos sistemas eletrônicos, eletro-mecânicos e de comunicação sem fio tem levado a grandes avanços na área de Redes de Sensores Sem Fio (RSSFs) [62]. As novas gerações de RSSFs são dotadas de nós-sensores capazes de processar e disseminar dados coletados do ambiente para um ponto da rede onde a informação foi inicialmente requisitada.

A possibilidade de colaboração entre os nós-sensores viabilizou o desenvolvimento de poderosas aplicações nas mais diversas áreas do conhecimento como, por exemplo, de engenharia, médica, militar e industrial. Aplicações comuns de RSSFs incluem o monitoramento ambiental (por exemplo, retorno de informações sobre condições do habitat de vidas selvagens, detecção de incêndios em florestas e medição de níveis de poluição), o monitoramento de infra-estruturas (por exemplo, detecção de fissuras em pontes, medição dos níveis de açudes e monitoramento da distribuição de água), além do acompanhamento de operações de guerra (por exemplo, detecção da localização de soldados, mapeamento dos movimentos de tropas inimigas e detecção de ataques químicos). Entre os grandes desafios enfrentados por estas aplicações estão a suscetibilidade a falhas, a mobilidade e as fortes restrições de *hardware* dos nós sensores (limitada capacidade de processamento, armazenamento e disponibilidade de energia) [1]. Estas características requerem novas representações de modelos computacionais, algoritmos, protocolos, metodologias de projeto e ferramentas, especialmente adequados às especificidades das RSSFs.

Nos últimos anos, o número de pesquisas envolvendo a tecnologia de banco de dados e as RSSFs vem sendo crescente [39][52][57]. Algumas propostas, defendidas no escopo destas pesquisas, tratam uma rede de sensores como um grande banco de dados distribuído. Neste cenário, consultas são submetidas à rede através de um computador central (estação-base), que é um nó robusto quanto à capacidade de processamento e armazenamento. Em resposta, dados são

coletados do ambiente pelos nós-sensores e passados de um nó para o outro, até alcançarem a estação-base, onde o resultado da consulta é finalmente disponibilizado ao usuário requisitante da informação. Desta forma, observa-se que o nó-sensor, a infra-estrutura de comunicação e a estação-base são pontos críticos em uma RSSF.

A contribuição da tecnologia de banco de dados para as RSSFs está presente, principalmente, na aplicação de técnicas de processamento de consultas, as quais podem ser capacitadas a disponibilizar resultados de consultas incrementalmente, além de promover o uso eficiente dos recursos dos nós da rede. A produção incremental de resultados é importante, particularmente, quando se admitem consultas denominadas contínuas, as quais são permanentemente executadas e os seus resultados continuamente atualizados e entregues ao usuário. O uso eficiente dos recursos da rede pode ser obtido com a aplicação de operações sobre os dados, como agregações ou junções, as quais busquem reduzir o volume de dados enviados através da rede. O tempo de vida de um nó-sensor é extremamente dependente da disponibilidade de energia da sua bateria. Visto que o consumo de energia é proporcional ao volume de dados transmitido ou recebido pelo nó [1], o uso de estratégias que resultem em um menor volume destes dados transmitidos pode levar a economias significativas de energia, contribuindo para aumentar a sobrevida do nó-sensor.

A infra-estrutura de comunicação tem como foco tanto os protocolos de roteamento de pacotes como o uso eficiente de energia na rede. Por exemplo, protocolos de roteamento devem evitar a “inundação” da rede, provocada pela excessiva duplicação de pacotes. O tráfego de um grande número de pacotes na rede tem duas conseqüências negativas. A primeira é o aumento das chances de colisão, o que certamente resulta em maiores taxas de perdas de pacotes. A segunda é a elevação dos gastos de energia, o que impacta diretamente no tempo de vida da rede como um todo.

Uma RSSF comumente é composta por centenas ou até milhares de nós-sensores [1]. Logo, o volume de dados enviado à estação-base pode vir a ser muito grande. Embora não tendo as mesmas restrições de recursos dos nós-sensores, a estação-base pode representar um gargalo para a aplicação, visto que as operações

previstas nas consultas submetidas à rede, como agregações ou junções, podem vir a manipular um volume de dados maior do que a capacidade da memória principal, forçando com que parte dos dados seja enviada para disco, como forma de evitar perda de dados. Neste caso, torna-se necessária a adoção de políticas eficientes de alocação de dados entre disco e memória principal, as quais suportem o tratamento de um montante de dados que, geralmente, é indeterminado, devido à ausência de estatísticas.

Observa-se que a limitação de recursos, em menor ou maior proporção, encontra-se em todos os pontos críticos de uma RSSF. Desta forma, algoritmos e protocolos, projetados para estas redes, deveriam ser sensíveis às restrições de energia e memória, de maneira a serem capacitados a adaptarem-se a cada novo cenário apresentado à rede.

4.2 Objetivo

Neste trabalho é explorado o processamento adaptativo de consultas [4] como estratégia para alcançar maior eficiência na execução de consultas e no uso dos recursos da rede, no escopo das RSSFs. Pela adaptação, algoritmos são capacitados a ajustar seu comportamento em resposta à ocorrência de eventos específicos, como restrições de energia e memória, em um determinado momento do tempo. Neste contexto, são propostos dois algoritmos adaptativos: o ADAGA, que realiza a filtragem e agregação dos dados em nós-sensores; e o ADAPT, que processa consultas do tipo junção-agregação na estação-base. O objetivo do ADAGA é maximizar o tempo de vida do nó-sensor através do monitoramento pró-ativo do uso de energia e memória, ajustando as atividades do nó-sensor de acordo com a disponibilidade destes recursos. O ADAPT, por sua vez, tem o objetivo de permitir a produção incremental dos resultados de uma consulta, aspecto de grande relevância em uma RSSF, visto que comumente as consultas submetidas a estas redes são contínuas.

Adicionalmente, duas outras contribuições são obtidas neste trabalho. A primeira consiste em especificar um modelo de dados genérico, que captura as características de uma RSSF, com o objetivo de relacionar diferentes fenômenos físicos monitorados. A vantagem do uso de um modelo de dados é permitir ao usuário uma visão lógica do fluxo de dados manipulado pelo sistema (aplicação

executada em uma RSSF), sendo o fluxo de dados caracterizado como sendo uma cadeia não estruturada de dados que é ordenada temporalmente. A segunda consiste em uma linguagem de consultas para RSSFs, denominada SNQL (*Sensor Network Query Language*). A linguagem SNQL estende o padrão SQL pela adição de cláusulas específicas para as RSSFs, as quais fornecem os valores de parâmetros necessários à execução dos algoritmos propostos, ADAGA e ADAPT.

4.3 Estrutura do trabalho

Este trabalho está organizado conforme descrito a seguir. O Capítulo 2 discute aspectos fundamentais ligados à área de redes de sensores sem fio, bem como algoritmos e protocolos propostos para estas redes. O Capítulo 3 investiga os aspectos ligados ao processamento de dados em RSSFs. Primeiramente, são apresentadas algumas das propostas existentes para permitir o processamento adaptativo de fluxos de dados e a aproximação de resultados de consultas, conceitos aplicáveis a RSSFs. Em seguida, são abordadas linguagens de consultas para manipulação de fluxos de dados e as principais propostas de bancos de dados para redes de sensores. No Capítulo 4 é apresentado o cenário, considerado neste estudo, para o processamento de consultas em RSSFs, bem como o modelo de dados admitido e a especificação da linguagem de consultas SNQL. Nos Capítulos 5 e 6 são apresentadas as propostas para otimização do uso dos recursos físicos da rede, através da aplicação dos algoritmos de processamento de consultas ADAGA e ADAPT. Finalmente, o Capítulo 7 discute as conclusões e os trabalhos futuros.

Capítulo 2

REDES DE SENSORES SEM FIO

2.1 Introdução

Os sensores vêm sendo largamente utilizados em aplicações destinadas a detectar eventos específicos ou coletar dados sobre propriedades ambientais. Apesar de pequenos e com uma diminuta capacidade de processamento e armazenamento, um conjunto de sensores reunidos em uma rede sem fio (Rede de Sensores Sem Fio - RSSF) oferece suporte ao desenvolvimento de poderosas aplicações. Estas redes vêm sendo particularmente utilizadas por aplicações que requerem pouca ou nenhuma intervenção humana.

Basicamente, uma RSSF é responsável pela disseminação dos dados coletados pelos sensores, embutidos nos nós da rede (nós-sensores), fazendo com que estes dados cheguem ao usuário requisitante da informação. Algumas características comuns a estas redes incluem [27]:

(i) capacidade de auto-organização;
(ii) utilização de nós-sensores com limitações de energia, poder de comunicação, memória e capacidade de processamento;
(iii) comunicação *broadcast,* utilizando freqüências de rádio e protocolos de roteamento do tipo múltiplos saltos;
(iv) existência de um grande número de nós-sensores densamente organizados em uma área geográfica;
(v) realização de trabalho colaborativo entre os nós-sensores; e
(vi) alteração dinâmica da topologia da rede, devido a falhas e/ou mudanças na localização dos nós.

Neste capítulo, são investigadas as características físicas dos nós-sensores (Seção 4.2) e os aspectos a serem avaliados no projeto de RSSFs (Seção 2.3).

2.2 Nós-sensores

Um nó sensor é um dispositivo capaz de coletar dados através da detecção de sinais ou de eventos ocorridos no meio ambiente, atividades conhecidas como sensoriamento. Outras tarefas destes dispositivos em uma rede de sensores incluem a recepção e o envio de pacotes através da rede, com o objetivo de atender a requisições do usuário. Adicionalmente, nós-sensores inteligentes são dotados com processador e memória, sendo capacitados a avaliar e transformar os dados coletados. Além da restrita capacidade computacional e de armazenamento, estes equipamentos têm o seu tempo de vida limitado à disponibilidade de energia da sua bateria. Por este motivo, uma das principais preocupações dos algoritmos projetados para os nós-sensores é reduzir o consumo de energia da bateria, como forma de aumentar a sobrevida do nó. Nas próximas subseções são apresentados a arquitetura básica dos nós-sensores (Subseção 2.2.1) e alguns dos nós-sensores desenvolvidos no contextos de pesquisas acadêmicas e comerciais (Subseção 2.2.2).

2.2.1 Arquitetura do nó-sensor

Sensores mais simples são capazes apenas de captar informações do ambiente e enviá-las a outros nós da rede. Um nó-sensor é composto, basicamente, de um dispositivo de sensoriamento, para captar informações do ambiente; um transceptor, que possibilita a transmissão e recepção de dados; e uma fonte de energia, que alimenta os demais componentes do nó-sensor. Neste tipo de arquitetura, todos os dados coletados do ambiente pelo nó-sensor são enviados através da rede. Assim, se for considerada uma RSSF com um grande número de nós-sensores, o tráfego de pacotes na rede tende a ser muito grande.

Uma evolução dos dispositivos-sensores mais simples é o nó-sensor inteligente, que conta com um microprocessador embutido em sua estrutura. Estes nós são capacitados a processar os dados coletados, antes de enviá-los a outros nós [1].

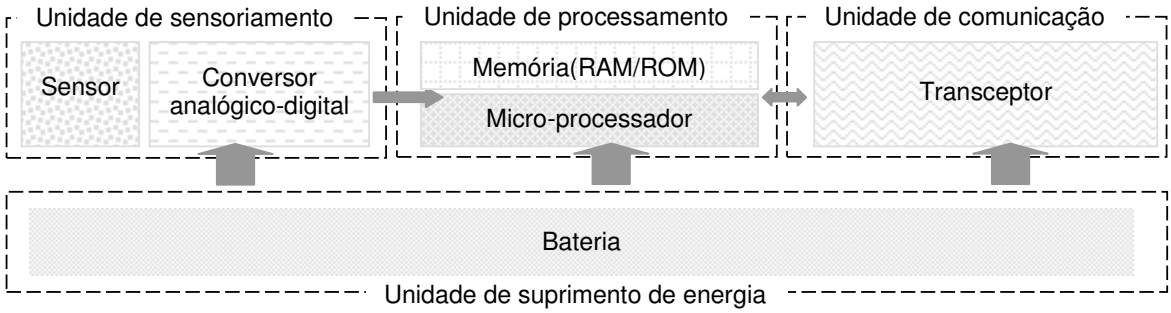


Figura 2.1 Componentes de um nó-sensor inteligente.

O processamento de dados pode incluir, por exemplo, a aplicação de técnicas de sumarização dos dados, como forma de reduzir o tamanho e o número dos pacotes enviados através da rede. A Figura 2.1 mostra os componentes físicos básicos de um nó-sensor inteligente [1]. Observa-se que a energia da bateria alimenta as unidades de sensoriamento, processamento e comunicação.

A unidade de sensoriamento é responsável por coletar as informações do ambiente, sendo constituída por um dispositivo-sensor e um conversor de sinal analógico-digital. Quando um evento específico é detectado, o conversor analógico-digital converte o sinal analógico, produzido pelo dispositivo-sensor, em um sinal digital, o qual alimenta a unidade de processamento [1].

A “inteligência” do nó sensor é proveniente da unidade de processamento. Esta unidade possui um microprocessador com memória RAM (volátil), que garante ao nó-sensor a capacidade de processar e armazenar dados temporariamente.

A unidade de comunicação conecta o nó-sensor à RSSF, permitindo a transmissão e recepção de pacotes de dados. O dispositivo que constitui esta unidade, denominado transceptor, pode realizar a comunicação sem fio do nó-sensor através de um meio ótico (emissão de raios infravermelho) ou utilizar freqüências de rádio. O infravermelho é um meio de comunicação barato e fácil de construir, porém, requer um sinal dedicado entre transmissor e receptor [59]. Transceptores baseados em freqüências de rádio requerem modulação, banda passante, filtragem, demodulação e multiplexação de circuitos, o que coloca este meio de comunicação entre os mais caros e complexos. Ainda assim, a freqüência de rádio é o meio mais utilizado em RSSFs, devido a algumas características comuns a estas redes que favorecem o uso da freqüência de rádio. Estas

características incluem o uso de pacotes pequenos, baixas taxas de transmissão e curtas distâncias entre nós-sensores que se comunicam, o que permite o reuso da freqüência do sinal na transmissão e recepção de pacotes [1].

A unidade de suprimento de energia é um dos componentes mais importantes do nó-sensor. Algumas baterias que compõem estas unidades de suprimento são recarregáveis (por exemplo, via uso de células solares), porém, a maior parte das arquiteturas de nós-sensores utiliza baterias de tempo de vida limitado.

Além dos componentes mostrados na Figura 2.1, também podem fazer parte da estrutura física dos nós-sensores outras subunidades como, por exemplo, subunidades responsáveis por fornecer a localização ou permitir a sincronização na recepção e envio de pacotes entre os nós-sensores. Muitas aplicações e técnicas de roteamento requerem o conhecimento preciso da localização do nó-sensor. Neste caso, o nó-sensor deve contar com um sistema de localização como o GPS (*Global Position System*). Todavia, equipar cada nó-sensor com um GPS pode não ser uma alternativa viável para RSSFs, devido aos custos de se implantar o GPS em um grande número de nós. Uma alternativa é utilizar o GPS apenas em alguns nós-sensores da rede, os quais ajudariam os seus nós-vizinhos a calcularem suas próprias localizações [1]. Um outro componente possível de um nó-sensor é a subunidade de sincronização, que sincroniza a recepção e envio de pacotes através da rede, o que requer o uso de relógios internos pelos nós da RSSF. Aspectos relacionados à sincronização em redes de sensores são explorados em [62].

2.2.2 Nós-sensores inteligentes

Nesta Subseção serão descritas as características dos principais dispositivos, desenvolvidos no contexto de pesquisas acadêmicas e comerciais, que podem se comportar como nós-sensores inteligentes.

2.2.2.1 PicoNodes

O PicoNode [44] é um dispositivo-sensor capaz de permitir a computação e comunicação ubíqua[1] em RSSFs. Proposto no contexto do projeto PicoRadio, da Universidade da Califórnia, este dispositivo tem apenas 1 cm^3 e consome menos do

[1] Comunicação ubíqua é a designação comumente utilizada para referenciar a integração da computação móvel e pervasiva com o espaço físico.

que 10 mW (Figura 2.2). O PicoNode foi projetado para ser um equipamento de baixo consumo de energia.

A arquitetura deste dispositivo apresenta quatro módulos básicos:

(i) uma unidade de processamento, dedicada principalmente a atender às requisições das aplicações;
(ii) uma unidade de processamento configurável, destinada à realização de tarefas que requeiram maiores recursos computacionais;
(iii) uma camada física configurável através de parâmetros; e
(iv) um módulo de conversão de freqüências de rádio.

Embora a implementação das unidades de processamento do PicoNode chegue a requerer até três vezes mais energia do que outras arquiteturas, a maior vantagem do PicoNode é a flexibilidade quanto a configurações de elementos físicos do nó-sensor via *software* (sistema operacional µ-OS). A configuração é realizada através de parâmetros que definem, por exemplo, modos de controle de energia e esquemas de modulação.

Visando reduzir o consumo de energia requerido pela unidade de comunicação, a camada física do PicoNode requer a ativação do transceptor apenas durante o período estritamente necessário para a aquisição de dados. Assim, enquanto não for requerida a recepção ou transmissão de dados, o transceptor permanece em um estado no qual apenas um mínimo gasto de energia é requerido para detectar a chegada de novos dados. Além disso, o PicoNode também foi projetado para extrair energia do ambiente, como alternativa para a reposição da energia gasta, o que pode ser feito, por exemplo, através do uso de baterias recarregáveis por energia solar.

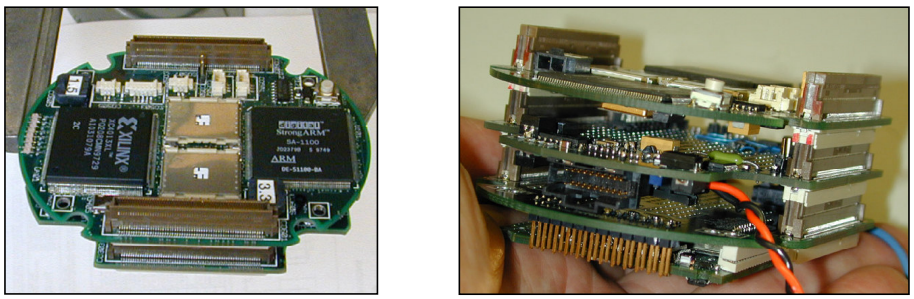


Figura 2.2 PicoNode, nó-sensor desenvolvido na Universidade da Califórnia.

As características físicas do nó-sensor PicoNode incluem:

(i) Atividade de sensoriamento: Temperatura, umidade, intensidade luminosa, ondas sonoras, aceleração, campos magnéticos e localização;
(ii) Memória: DRAM de 4MB, Memória flash de 4MB; e
(iii) Comunicação: Rádio transceptor utilizando tecnologia *Bluetooth*[2], com capacidade para transmissões de 100 Kbps a distâncias de até 100 m.

2.2.2.2 µamps

Os µAMPS (*micro-Adaptive Multi-domain Power-aware Sensors*) [49], desenvolvidos no Massachusetts Institute of Technology (MIT), são nós-sensores sem fio que relacionam seus aspectos físicos a parâmetros configuráveis na aplicação. O objetivo destes dispositivos é permitir que o consumo de energia da rede seja ajustado, em resposta às mudanças no meio ambiente, no comportamento da rede (por exemplo, devido a falhas ou alteração na localização dos nós da rede) ou nos valores dos parâmetros especificados na aplicação. Assim, algoritmos, protocolos de rede e sistema operacional podem adaptar-se às características físicas dos nós-sensores. O objetivo desta estratégia é alcançar a redução do consumo global de energia da RSSF e obter a conseqüente maximização do tempo de vida da rede.

Em Shih *et al.* [49], destaca-se a importância da redução dos intervalos de tempo em que os nós da rede permanecem ativos, para a conservação de energia em RSSFs. Além de buscar a maximização do tempo em que os nós-sensores permanecem inativos, os *µ*AMPS também avaliam a quantidade de energia gasta no acionamento do nó-sensor. A necessidade desta avaliação se deve ao fato de que a energia consumida durante o acionamento pode ser ainda maior do que aquela que seria gasta se o nó-sensor permanecesse, permanentemente, ativo. Esta situação pode ocorrer porque a quantidade de energia gasta durante a recepção e envio de pacotes varia com o volume de dados tratado, enquanto a energia gasta no acionamento do nó-sensor é uma constante inerente à implementação física deste equipamento. Assim, quando se têm poucos pacotes a enviar e a recepção/transmissão ocorre muito rapidamente, o consumo de energia para a

[2] A tecnologia *Bluetooth* é, basicamente, um padrão para comunicação sem fio de baixo custo e de curto alcance.

ativação do nó-sensor pode ser superior ao que seria obtido se o nó-sensor não tivesse sido desativado.

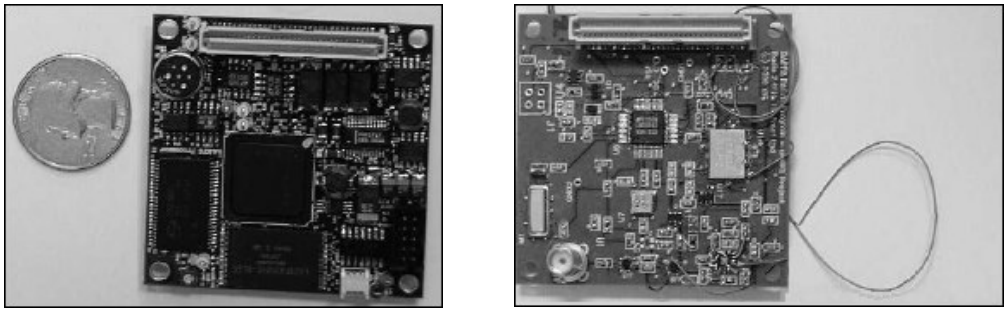


Figura 2.3 μAMPS, nó-sensor desenvolvido no MIT.

Na Figura 2.3 o tamanho do nó-sensor μAMPS é comparado ao tamanho de uma moeda. A figura à esquerda mostra a placa que contém o sensor e o processador. A figura à direita mostra a placa que contém o transceptor [49].

As características físicas do nó-sensor μAMPS incluem:

(i) Atividade de sensoriamento: medições sísmicas e de ondas sonoras;
(ii) Memória: 1MB de SRAM e 1MB de memória flash ROM; e
(iii) Comunicação: Compatível com *Bluetooth,* possibilitando taxas de transmissão inferiores a 10 Kbps a distâncias de até 100 m.

2.2.2.3 MicaMotes

Os MicaMotes [32] consistem em uma série de nós-sensores programáveis, desenvolvidos por pesquisadores da Universidade da Califórnia. Um dos representantes mais recentes desta série de sensores é o Mica2, que tem sua estrutura composta por uma unidade de sensoriamento, um processador, um transceptor e duas baterias (Figura 2.4).

Estes componentes são controlados pelo TinyOS (*Tiny Microthreading Operating System*) [32], um sistema operacional para RSSFs, escrito em NesC[3], que gerencia os recursos de *hardware* e *software* dos nós-sensores. O TinyOS é capaz de controlar a captação de dados do ambiente, a transmissão de pacotes entre nós da rede, o processamento de dados, além de fazer com que o nó-sensor fique

[3] NesC é uma extensão da linguagem de programação C, projetada para a construção de aplicações na plataforma do sistema operacional TinyOS.

inativo quando nenhum dado estiver sendo recebido ou enviado. O TinyOS trabalha com pacotes de dados de, aproximadamente, 36 bytes.

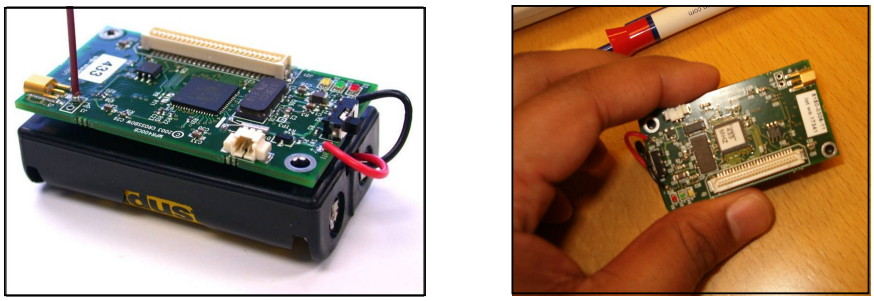


Figura 2.4 Mica2, nó-sensor desenvolvido na Universidade da Califórnia.

As características físicas do nó-sensor Mica2 incluem:

(i) Atividade de sensoriamento: Intensidade luminosa, aceleração, medição sísmica, acústica, campos magnéticos e localização;
(ii) Memória: 512 KB de armazenamento externo não-volátil, 4 KB de memória RAM e 128 KB de memória ROM; e
(iii) Comunicação: Rádio transceptor capaz de transmitir 40 Kbps a distâncias de até 20 metros.

2.2.2.4 WINS

Os WINS (*Wireless Integrated Network Sensors*) e os WINS NG(*Next Generation*) [43], desenvolvidos na Universidade da Califórnia, são nós-sensores inteligentes que oferecem suporte ao estabelecimento de redes auto-organizáveis, destinadas ao sensoriamento contínuo ou à detecção de eventos pré-determinados. Quando utilizados em aplicações que requerem detecção de eventos, a identificação de cada evento deve ser previamente armazenada em cada um dos nós-sensores da rede.

Os WINS NG integram múltiplas funções que incluem o sensoriamento, a comunicação, além da interoperabilidade entre diferentes redes e bancos de dados. A economia no consumo global de energia de uma rede de sensores WINS é obtida a partir de estratégias que envolvem:

(i) roteamento do tipo múltiplos saltos;
(ii) adoção de curtas distâncias entre os nós-sensores;

(iii) utilização de protocolos projetados para manter o rádio de transmissão/recepção desligado o maior tempo possível;

(iv) processamento local de dados, o que pode incluir a aplicação de operações de agregação sobre os dados, como forma de reduzir o volume de dados enviados através da rede; e

(v) reconfiguração de propriedades físicas dos sensores por protocolos e aplicações.

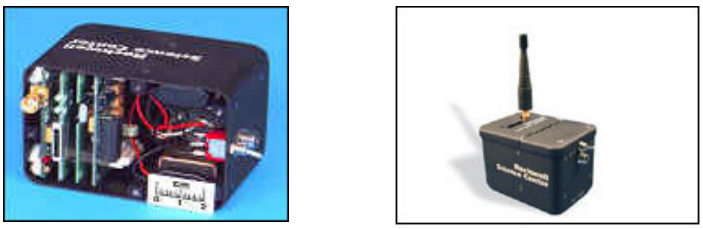


Figura 2.5 WINS, nó-sensor desenvolvido na Universidade da Califórnia.

As características físicas do nó-sensor WINS incluem:

(i) Atividade de sensoriamento: Temperatura, pressão, aceleração, medição sísmica, acústica, campos magnéticos e localização;

(ii) Memória**:** Memória flash de 4MB e 1MB de memória SRAM; e

(iii) Comunicação: Rádio transceptor de 900 MHz que transmite 100 Kbps a distâncias de até 100 metros.

2.3 Aspectos de redes de sensores sem fio (RSSFs)

Devido às características particulares das RSSFs, principalmente as restrições de recursos físicos, os algoritmos e protocolos projetados para redes convencionais não são aplicáveis a estas redes. O projeto de uma RSSF envolve a análise de aspectos relacionados aos nós-sensores, tais como consumo de energia, tolerância a falhas, sensoriamento e processamento de dados, e outros relacionados à estrutura da rede, tais como topologia, escalabilidade, comunicação e roteamento. Estes aspectos são descritos a seguir.

2.3.1 Consumo de energia

O aspecto mais importante a ser considerado no projeto de uma RSSF refere-se ao consumo de energia. Cada nó-sensor tem seu tempo de vida ditado pela disponibilidade de energia da sua bateria. Este fato leva a uma conseqüente

limitação do tempo de vida da rede como um todo, caso não seja adotada uma estratégia de reposição dos nós da rede ou de recarga da bateria dos nós-sensores. Vale salientar que a recarga ou substituição da bateria não é uma opção fácil para estas redes, visto que elas geralmente são compostas por centenas ou até milhares de nós [1]. Desta forma, algoritmos, protocolos e componentes físicos projetados para RSSFs devem ter como principal objetivo a minimização do consumo de energia da bateria dos nós-sensores.

A partir da Figura 2.1 (Subseção 2.2.1), é possível identificar três fatores a serem considerados no consumo de energia em nós-sensores: sensoriamento, que é o objetivo primário de um sensor; processamento, que torna os nós-sensores dispositivos inteligentes, capazes de aplicar operações como, por exemplo, agregações e filtragens, sobre os dados coletados do ambiente; e comunicação, responsável pelas operações de transmissão e recepção, essenciais para formar uma RSSF [1]. Embora todas estas atividades consumam energia, é importante notar que o custo de energia para a transmissão ou recepção de um único dado sobre um meio sem fio, a pequenas distâncias, é muito maior do que o requerido para processar este dado ou coletá-lo do ambiente (por exemplo, o processamento de 3000 instruções pode ser executado, aproximadamente, com o mesmo gasto advindo do envio de 1 bit a uma distância de 100m) [62]. Assim, o caminho mais eficiente para diminuir o consumo de energia em uma RSSF é através da redução dos custos com comunicação [39].

Em geral, o transceptor (rádio-transceptor) de um nó-sensor pode operar em quatro modos distintos: Transmissão, Recepção, Ocioso, e Inativo. Cada um destes modos é caracterizado por diferentes taxas de consumo de energia. O modo Ocioso consome quase a mesma quantidade de energia que o modo de Recepção. Logo, o transceptor deve ser completamente desligado, colocado em modo Inativo, quando nenhum dado estiver sendo transmitido ou recebido, ou quando houver longos intervalos de tempo entre coletas sucessivas de dados; pois, é apenas no modo Inativo que a energia da bateria pode ser temporariamente conservada. Todavia, é importante observar que, no modo de operação Inativo, o nó-sensor não pode ser contactado por outros nós, o que muda a topologia ativa da rede [45].

Algumas alternativas, como a apresentada em [49], defende o uso de nós-sensores com múltiplos rádio-transceptores. Neste caso, um rádio com baixo consumo de energia é usado, exclusivamente, para ativar o rádio de mais alta potência, quando for necessário realizar a transmissão e recepção de dados. Desta forma, o nó-sensor não fica inacessível, pois o rádio-transceptor de mais baixa potência permanece ligado continuamente.

Outro aspecto a ser considerado é que, na maior parte das arquiteturas de rádio-transceptores, a simples mudança entre modos de operação causa uma dissipação significativa de energia, por exemplo, devido à mudança do modo Inativo para o modo de Transmissão. Portanto, é benéfico operar com pacotes tão grandes quanto possível, objetivando amortizar tanto o número de mudanças no modo de operação do transceptor quanto o *overhead* fixo sobre a manipulação de uma quantidade maior de bits (por exemplo, o cabeçalho do pacote) [45]. Por outro lado, um longo intervalo de tempo sem transmissão aumenta a latência total na troca das informações, o que pode não ser aceitável para uma dada aplicação.

Alternativas para obter uma maior economia de energia em RSSFs requerem o uso de algoritmos “conscientes” da disponibilidade de energia, projetados para ajustar seu comportamento a eventos específicos, como, por exemplo, restrições de recursos. Estes algoritmos deveriam ser capacitados a fazer uma análise dinâmica entre o consumo de energia, desempenho do sistema e fidelidade dos dados produzidos para o usuário. Estratégias de otimização no uso de energia podem ser aplicadas aos nós, ao canal de comunicação ou à rede como um todo, tendo por objetivo o aumento do tempo de vida da RSSF [45].

2.3.2 Tolerância a falhas

Considerando as restrições de energia da bateria dos nós-sensores e as condições extremas a que, muitas vezes, estes nós são expostos (por exemplo, implantados no fundo de oceanos, no interior de vulcões ou presos a corpos de animais), observa-se que os nós-sensores são altamente susceptíveis a falhas. Porém, a falha de um ou alguns nós-sensores não deve ser suficiente para comprometer o funcionamento da rede [1] ou distorcer a precisão dos resultados apresentados ao usuário.

A falha de um sensor que faz parte da rota de um pacote não deve impedir que o pacote chegue ao seu destino, visto que a topologia da rede deve dar caminhos

alternativos para que os pacotes alcancem os seus destinos, seja através da adoção de mecanismos de retransmissão de pacotes ou apenas pela duplicação destes pacotes na rede. Para garantir a disponibilidade de rotas alternativas para os pacotes, os protocolos de roteamento precisam reorganizar a rede periodicamente, traçando novas rotas em substituição àquelas interrompidas pela falha de nós-sensores, buscando, assim, uma maior tolerância a falhas.

2.3.3 Sensoriamento

A função primária de uma aplicação de RSSF é a atividade de sensoriamento, provida pelos sensores integrantes dos nós da rede. O tipo de informação a ser coletada depende do dispositivo físico de sensoriamento utilizado, que, por sua vez, é escolhido com base no objetivo da aplicação, como, por exemplo, coletar medições de temperatura. No contexto das aplicações de RSSFs, a forma de obtenção dos dados pode ser classificada como [47]:

(i) Contínua, quando os dados são coletados continuamente;
(ii) Reativa, quando os dados são fornecidos em resposta a uma consulta do usuário ou a um evento específico do ambiente; e
(iii) Periódica, quando os dados são coletados segundo condições previamente configuradas na aplicação.

Algumas propostas, como a elaborada no Projeto TinyDB [39], da Universidade da Califórnia, suportam modelos híbridos, que consistem na coexistência de diferentes tipos de coleta de dados.

2.3.4 Processamento

O processamento de dados em nós-sensores é uma importante propriedade dos nós-sensores inteligentes. Conforme mencionado na Subseção 2.3.1, o consumo de energia com comunicação deve ser prioritariamente minimizado. Tendo em vista que este custo é proporcional ao volume total de dados transmitido através da rede, processamentos aplicados aos dados podem levar a reduções significativas do volume de dados enviado através da rede.

O processamento de dados em nós-sensores pode envolver operações de agregação, filtragem e outras técnicas de sumarização ou análise de dados, como as apresentadas na Subseção 3.2.2. Convém salientar que o processamento pode

envolver não apenas operações aplicadas aos dados coletados do ambiente, mas também a fusão destes dados com aqueles recebidos de outros nós-sensores, originando novos pacotes de dados [47].

A agregação em rede [15][41][50] vem sendo a estratégia mais explorada na redução do volume de dados transmitido dos nós-sensores para a estação-base. Esta estratégia consiste na aplicação progressiva de operações de agregação aos dados, à medida que os pacotes vão sendo passados de um nó para outro na rede. Desta forma, a agregação de dados é efetuada ao longo de cada rota, por onde passam os pacotes.

A Figura 2.6 ilustra um exemplo de como funciona a agregação em rede. Neste exemplo hipotético é mostrada a planta baixa de um *shopping center*. Considera-se que cada entrada de veículo no estacionamento do *shopping center* é monitorada por um dispositivo-sensor, que faz parte de uma RSSF; e que um usuário, operando a estação-base, requisita o número de veículos que estão no estacionamento do *shopping center* em um determinado instante. Em resposta, os valores da contagem de veículos, computados em cada nó-sensor, são progressivamente somados, à medida que vão sendo passados através da rede, até chegarem à estação-base, onde o resultado final para a consulta é produzido.

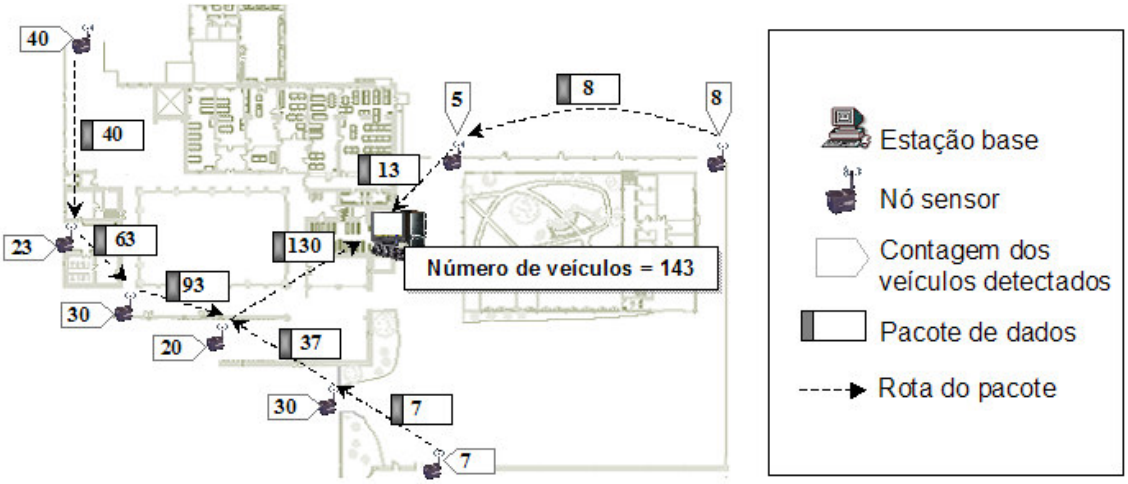


Figura 2.6 Exemplo de agregação em rede.

2.3.5 Topologia

O número de nós-sensores em uma RSSF pode vir a ser muito grande e a densidade destes nós, na região geográfica considerada, pode ser alta (por exemplo, superior a 20 nós/m^3). A distribuição dos nós-sensores no meio ambiente será mais ou menos aleatória, de acordo com a estratégia utilizada na implantação dos nós, por exemplo, de forma manual ou despejados a partir de aviões [1].

Após a primeira etapa de implantação dos nós de uma RSSF, alguns nós-sensores podem vir a mudar de localização, devido às condições do ambiente, por exemplo, devido a ventos ou deslizamentos, o que pode alterar a disposição física destes nós na rede. Além disso, seja por necessidade de reposição dos nós-sensores falhos ou por necessidade de crescimento da rede, novos nós poderão vir a ser implantados. Estas alterações de localização dos nós-sensores, bem como as manutenções realizadas na RSSF, requerem uma reorganização da estrutura da rede e, conseqüentemente, a atualização das rotas de transmissão dos pacotes por parte dos protocolos de roteamento.

2.3.6 Escalabilidade

Em uma RSSF, os protocolos da rede devem ser capazes de trabalhar não apenas com o número inicial de nós-sensores implantados no meio ambiente, mas também garantir a escalabilidade da rede; o que implica que o desempenho da aplicação não deverá ser fortemente afetado pelo aumento do tamanho da rede.

Adicionalmente, protocolos de roteamento podem ser escaláveis o suficiente para responder a eventos específicos do ambiente. Nesta estratégia, proposta em [2], a maior parte dos nós-sensores permanece no modo Inativo, sendo que apenas alguns nós-sensores da RSSF permanecem ativos para processar dados de pacotes remanescentes. Durante este período, os resultados fornecidos ao usuário serão menos precisos, visto que têm por base dados coletados apenas por alguns poucos nós-sensores ativos. Quando um determinado evento, configurado na aplicação, ocorre, todos os nós-sensores são ativados e a aplicação passa a dispor de resultados mais precisos. Esta estratégia favorece a economia de recursos do nó-sensor em detrimento da precisão dos resultados, porém, a produção de resultados mais precisos é garantida nos períodos estabelecidos pelos eventos configurados na aplicação.

2.3.7 Comunicação

A comunicação não apenas permite o processamento colaborativo entre os nós-sensores, mas também a interação entre os usuários da aplicação e a rede. Em Ruiz *et al.* [47] são apontados dois tipos de comunicação: de infra-estrutura e de aplicação. A comunicação requerida para manutenção da infra-estrutura da rede objetiva manter atualizadas as rotas de comunicação, independente das falhas dos nós-sensores, da mobilidade destes nós ou do crescimento da rede (inclusão de novos nós). Já a comunicação requerida pela aplicação refere-se à disseminação dos dados pela rede. Durante a disseminação dos dados, a energia gasta na recepção e transmissão de um pacote tem um custo fixo, relacionado ao *hardware*. Para a operação de transmissão, também está associado um custo variável, que depende da distância entre o nó-origem da informação e o nó-destino. Desta forma, quanto mais próximos estiverem dois nós sensores, menor será o consumo de energia necessário para realizar a comunicação entre eles, pois a potência dissipada será menor.

Além da distância entre os nós da rede, um outro fator crítico, relacionado à comunicação, é volume de dados que converge para o ponto no qual as informações são entregues à estação-base. Embora a estação-base seja robusta quanto à memória, disponibilidade de energia e capacidade de processamento, o tráfego de pacotes gerado por um número grande de nós-sensores para esta estação pode ser pesado. Portanto, a estação-base pode representar um gargalo que penaliza o *throughput* da rede. A perda de pacotes, seja por falha de alguns nós da rede ou pela incapacidade da estação-base em receber todos os pacotes a ela enviados, pode vir a gerar diferenças nos resultados apresentados ao usuário a cerca do fenômeno que está sendo considerado. Este aspecto é analisado em alguns trabalhos [27][47][58] através da garantia da qualidade de serviço (*QoS – Quality of Service*), referindo-se à capacidade da rede de produzir resultados corretos e em tempo aceitável.

A capacidade de esforço coletivo dos nós-sensores representou um grande avanço para as RSSFs. Os protocolos de comunicação provêem a infra-estrutura necessária para que o processamento colaborativo seja possível. Em Akyildiz *et al.*

[1] é proposto que, além das camadas convencionais de redes (física, enlace, rede, transporte e aplicação), também sejam considerados componentes destinados ao:

(i) **Gerenciamento de energia**: monitora o consumo de energia dos nós-sensores. A informação fornecida por este componente poderia, por exemplo, sinalizar para um nó-sensor enviar um aviso (por *broadcast*) aos seus vizinhos, informando que seu nível de energia esteja baixo e, portanto, não seja capaz de participar do roteamento de pacotes. Assim, a energia remanescente neste nó-sensor apenas passaria a ser utilizada em atividades de sensoriamento, processamento e envio de seus próprios dados;

(ii) **Gerenciamento de mobilidade**: monitora a movimentação dos nós-sensores. Este monitoramento consistiria em detectar e registrar o movimento de um dado nó-sensor. A partir desta informação, o nó-sensor poderia ser capacitado a identificar os seus novos nós-vizinhos e manter atualizadas as suas rotas de acesso à estação-base; e

(iii) **Gerenciamento de tarefas**: monitora a distribuição de tarefas entre os nós-sensores. Basicamente, este componente faria o balanceamento das tarefas de sensoriamento em uma dada região geográfica. Por exemplo, nem todos os nós-sensores de uma região precisam ser utilizados para responder a uma solicitação do usuário, o que pode ocorrer quando dois ou mais nós-sensores compartilham a mesma área de sensoriamento. Neste caso, os nós-sensores com maior disponibilidade de energia podem ser selecionados para coletar os dados do ambiente, poupando recursos dos demais nós-sensores [1].

Esses componentes de gerenciamento contribuem para que cada nó-sensor seja capaz de controlar sua atividade de sensoriamento, baixar seu consumo de energia, permitir o compartilhamento de seus recursos e o trabalho colaborativo no roteamento dos pacotes.

2.3.8 Roteamento

Em Al-karaki *et al.* [2] é sugerida uma divisão para os protocolos de roteamento de RSSFs em três classes:

(i) plana, na qual todos os nós-sensores da rede desempenham o mesmo papel e possuem as mesmas características físicas;

(ii) hierárquica, na qual grupos de nós da rede têm diferentes responsabilidades e disponibilidades de recursos e;

(iii) adaptativa, que permite ao protocolo adaptar-se às condições da rede e à disponibilidade dos níveis de energia dos nós-sensores, através da configuração de parâmetros que mapeiam características físicas dos nós da RSSF.

A classe de protocolos hierárquica vem sendo largamente explorada como estratégia capaz de poupar recursos dos nós-sensores. Estes protocolos definem que os dados sejam passados de um nó para outro, através de roteamento do tipo múltiplos saltos, até alcançarem a estação-base. Estações-base são a chave para uma comunicação do tipo *backbone*, servindo como *gateway* para outras redes, o que permite a integração entre diferentes redes [53]. O problema em utilizar uma estrutura hierárquica em RSSFs é que o tráfego torna-se cada vez mais intenso, à medida que os pacotes aproximam-se da estação-base. Assim, enquanto os nós-sensores mais distantes da estação-base apenas coletam dados do meio ambiente, os nós mais próximos têm seus recursos penalizados, devido ao fato de precisarem tanto coletar dados do ambiente como recepcionar pacotes de outros nós da rede.

Estratégias de endereçamento convencionais, como o uso de IPs (*Internet Protocols*), não são aplicáveis a RSSFs porque demandariam um alto custo para atribuição e manutenção de endereços, tendo em vista o grande número de nós destas redes. O roteamento de pacotes em RSSFs também deve oferecer suporte à mobilidade dos nós. Tarefa difícil caso o endereço do nó não forneça nenhuma informação sobre a direção em que o pacote deve ser roteado. Uma alternativa é manter um servidor central, capaz de manter atualizadas informações sobre a posição de todos os nós da rede. Outra alternativa é utilizar um agente de endereçamento, associado a cada nó, capaz de manipular e redirecionar todas as requisições destinadas a um dado nó para a sua localização corrente [26]. Na proposta do PicoRadio [44], os nós podem ser endereçados com base em suas posições geográficas. A localização dos nós é uma informação muito útil, pois permite que os protocolos de roteamento enviem os pacotes na direção correta.

As subseções a seguir detalham algoritmos que implementam protocolos de roteamento para RSSFs.

2.3.8.1 *Flooding*

Esta é uma estratégia clássica de roteamento que consiste em um nó-sensor enviar pacotes para todos os seus vizinhos, independente do fato destes vizinhos já terem ou não recebido os mesmos pacotes. Neste caso, são feitas cópias de cada novo pacote para todos os vizinhos do nó, exceto aquele do qual ele acaba de receber o pacote. A maior vantagem deste algoritmo está na simplicidade quanto ao gerenciamento das rotas e endereços. Todavia, três deficiências não tratadas pelo *flooding* e enumeradas em Heinzelman *et al.* [31] desencorajam o uso deste protocolo em RSSFs:

(i) Implosão: situação na qual pacotes duplicados são enviados para o mesmo nó, desperdiçando os recursos físicos da rede;
(ii) Sobreposição: fato que ocorre quando dois ou mais nós compartilham a mesma área geográfica e, portanto, coletam a mesma informação, gerando dados duplicados para a aplicação; e
(iii) Desconhecimento dos recursos físicos: neste caso o protocolo não é capacitado a adaptar seu comportamento à disponibilidade de recursos físicos do nó-sensor, como, por exemplo, energia.

2.3.8.2 *Gossiping*

Na proposta do *gossiping* [28], cada pacote p_1 de um nó n_1 é enviado apenas para um dos vizinhos de n_1, o qual é selecionado aleatoriamente. Desta forma, p_1 é propagado através da rede até alcançar a estação-base.

O uso do *gossiping* consome menos energia do que o *flooding,* pois permite reduzir o número de pacotes enviados por nó. Outra vantagem desta estratégia é a resolução do problema da implosão de pacotes, o que é conseguido graças ao fato de que apenas uma cópia de p_1 é mantida na rede, a cada intervalo de tempo. Todavia, o *gossiping* não trata alguns aspectos como:

(i) caso o objetivo seja disseminar o pacote para todos os nós da rede, o intervalo de tempo necessário para atender a este objetivo será muito maior do que em estratégias baseadas em *broadcast*;

(ii) não resolve o problema da duplicação de dados, resultante do compartilhamento de áreas geográficas entre nós-sensores vizinhos [31];

(iii) é possível que um nó n_1 envie o pacote p_1 para um vizinho n_2, que é o mesmo nó que havia enviado p_1 para n_1, anteriormente. Este fato pode gerar retardos ou, no pior caso, um ciclo que impede os demais nós da rede de receberem p_1; e

(iv) como os nós-sensores são suscetíveis a falhas, as chances de perda de um pacote devido à falha de um nó-sensor, localizado na rota deste pacote, aumentam muito, visto que apenas uma cópia do pacote é mantida na rede a cada intervalo de tempo.

2.3.8.3 SPIN (*Sensor Protocols for Information via Negotiation*)

Consiste em uma família de protocolos de roteamento, aplicáveis a RSSFs, que apenas disseminam pacotes na rede após uma prévia negociação com seus nós vizinhos. Esta negociação se dá através de trocas de mensagens, as quais evidenciam o interesse de cada nó em receber ou não o pacote que está sendo disseminado. As mensagens trocadas no SPIN são de três tipos: ADV, enviada por um nó n_1 aos seus vizinhos, advertindo sobre a existência de um novo pacote; REQ, enviada a n_1 por seus vizinhos, indicando o interesse de cada um em receber o novo pacote; e DATA, que é o próprio pacote de dados enviado por n_1 aos nós que retornaram uma mensagem do tipo REQ [31].

Para que haja sucesso na negociação, os nós devem ser capazes de descrever, sucintamente, os dados que eles coletam do ambiente; esta informação é representada por metadados, específicos de cada aplicação. Com base nas informações contidas nos metadados, é realizada uma negociação entre os nós. Esta negociação garante que somente os pacotes úteis para um dado nó serão enviados a ele. Assim, consegue-se uma maior conservação de energia pela redução do número de pacotes enviados.

O SPIN é projetado para evitar os três problemas do *flooding* clássico (Subseção 2.3.8.1). O problema de implosão é solucionado através da negociação que precede o envio dos pacotes, evitando a transmissão de dados redundantes; a sobreposição de área geográfica é tratada através da avaliação dos metadados; e o problema de desconhecimento de recursos físicos é resolvido pela verificação da

disponibilidade de energia do sensor. Porém, a adoção do SPIN leva a um maior *overhead*, provocado pela troca extra de mensagens, o que aumenta os tempos de disseminação dos pacotes. Além disso, cada nó deve, periodicamente, atualizar sua lista de nós-vizinhos, o que requer consumo de memória e energia [60].

2.3.8.4 SMECN (*Small Minimum Energy Communication Network*)

Este protocolo constrói sub-redes, a partir de uma RSSF, que possuem caminhos otimizados entre quaisquer pares de nós considerados. O caminho otimizado garante que a rota tomada por um pacote, a partir do nó-origem até o nó-destino, seja o que resulta em um menor consumo de energia. Vale salientar que o SMECN não encontra a rota que resulta no menor consumo de energia, mas apenas uma sub-rede que contém esta rota [2].

2.3.8.5 SAR (*Sequential Assignment Routing*)

Este protocolo [2] cria múltiplas árvores, sendo que o nó-raiz de cada árvore é um nó-vizinho da estação-base. O SAR é utilizado por cada nó-sensor da rede para escolher a rota de envio de um dado pacote para a estação-base. Esta escolha é feita com base em três critérios: recursos de energia, o nível de prioridade de cada pacote e o nível de QoS (*Quality of Service*) calculado para cada caminho. As métricas de QoS adotadas incluem valores de *throughput* e tempos de retardo.

Para cada pacote roteado através da rede, uma métrica de QoS é computada. Esta métrica é calculada como sendo o produto entre a adição das métricas de QoS e um coeficiente associado ao nível de prioridade do pacote. Este cálculo é usado para avaliação de desempenho da rede.

O objetivo do algoritmo SAR é minimizar a média da métrica de QoS calculada durante todo o tempo de vida da rede. Pelo uso do SAR, cada nó da rede possui múltiplas rotas de envio possíveis, o que é vantajoso, pois caso uma rota seja interrompida pela falha de um nó sensor, outras rotas podem ser utilizadas. Como forma de contemplar as mudanças na topologia da rede; devido a falhas em nós sensores, alterações da disponibilidade de energia destes nós ou mesmo alterações dos valores de QoS; o SAR, periodicamente, requer que a estação base dispare uma atualização (recálculo) das rotas da rede. Este reprocessamento das rotas da

rede demanda um custo de energia e processamento, particularmente caro para a estação base.

2.3.8.6 LEACH (*Low-Energy Adaptive Clustering Hierarchy*)

Em Heinzelman *et al.* [30] é proposto o LEACH como um protocolo de comunicação auto-organizável e adaptativo que adota uma estrutura hierárquica para os nós de uma RSSF. A hierarquia é baseada na segmentação dos nós da rede em grupos, sendo que cada grupo tem um de seus nós eleito como coordenador. O nó-coordenador recebe os pacotes dos demais nós do grupo, realiza processamentos sobre os dados destes pacotes (agrupamento de dados), e os envia à estação-base.

Devido às tarefas extras realizadas pelo coordenador, observa-se que o consumo de energia neste nó será maior do que nos demais nós do seu grupo. LEACH objetiva distribuir a carga de consumo de energia uniformemente entre os nós, o que é conseguido através do revezamento, entre os nós-sensores, do papel de coordenador de grupo.

2.3.8.7 *Directed diffusion*

A proposta *Directed diffusion* [34] consiste em um protocolo de comunicação orientado aos dados, o que significa que o roteamento dos pacotes é definido com base nas propriedades dos dados coletados, como, por exemplo, a região geográfica. Este protocolo especifica que a estação-base deve, inicialmente, disseminar uma notificação sobre os dados que tem interesse em obter, o que é realizado através do envio de mensagens, em *broadcast,* para todos os nós-vizinhos. Esta notificação de interesse descreve uma tarefa requerida aos nós-sensores, a qual consiste em pares de atributo e valor, por exemplo, a especificação de um intervalo entre coletas de dados = 20 segundos.

À medida que as notificações de interesse vão sendo passadas através da rede, gradientes vão sendo formados, indicando a direção em que os pacotes de dados deverão fluir de volta dos nós-origem de dados para a estação-base. Cada nova notificação de interesse recebida pelos nós sensores é armazenada localmente em sua memória cachê e interpretada para servir como parâmetro na coleta de dados. Finalmente, em cada nó-sensor, os dados coletados que atendem às especificações das notificações de interesse são enviados à estação-base.

Como benefícios desta proposta, tem-se uma adaptação mais rápida da rede às mudanças de topologia, provocadas pela falha ou mobilidade dos nós; e uma diminuição do consumo de energia dos nós-sensores, visto que menos pacotes são transmitidos. Além disso, é prevista a aplicação da técnica de agregação em rede, o que contribui para também reduzir o volume de dados transmitido.

2.4 Considerações finais

Este capítulo apresentou a arquitetura dos nós-sensores inteligentes, além de aspectos relevantes a serem considerados no projeto físico e lógico de uma RSSF. Observou-se que o consumo de energia em nós-sensores é afetado por todos os aspectos aqui considerados: aumento da tolerância a falhas, freqüência de sensoriamento, volume de processamento, tipo de topologia adotado, alteração do número de nós da rede, protocolos de roteamento e comunicação. Portanto, estes aspectos devem ser cuidadosamente avaliados para que não levem a uma rápida sobrevida dos nós-sensores da rede. Uma importante propriedade dos nós-sensores apresentados é a possibilidade de reconfiguração de suas características físicas por meio de *software*, o que permite que o nó seja reconfigurado de acordo com as características de funcionamento requeridas pela aplicação ou de acordo com a disponibilidade de seus recursos físicos. Os conceitos abordados neste capítulo serão utilizados como a base para apoiar os demais capítulos desta dissertação.

Capítulo 3
PROCESSAMENTO DE CONSULTAS EM REDES DE SENSORES SEM FIO

3.1 Introdução

Contrariamente aos sistemas de bancos de dados convencionais, que buscam tuplas em disco para atender às consultas submetidas pelos usuários, aplicações de RSSFs manipulam seqüências temporalmente ordenadas e não-estruturadas de dados coletados do ambiente, denominadas fluxos de dados [25]. Estas aplicações freqüentemente trabalham com consultas que permanecem ativas continuamente, denominadas consultas contínuas, podendo vir a produzir resultados potencialmente infinitos.

O objetivo deste capítulo é caracterizar o processamento de consultas em RSSFs. Inicialmente, características gerais dos sistemas de fluxos de dados são abordadas. A Seção 3.2 apresenta aspectos relacionados à adaptabilidade, aproximação de resultados de consultas e algumas pesquisas acadêmicas na área de sistemas de fluxos de dados. A Seção 3.3 apresenta aspectos relacionados aos sistemas de fluxos de dados voltados para aplicações de RSSFs. Nessa seção são apresentadas as características dos fluxos de dados, modelagem de dados, linguagens de consultas, estratégias de processamento de dados e projetos de pesquisa envolvendo os sistemas de processamento de fluxos de dados para RSSFs.

3.2 Sistemas de fluxos de dados

De uma maneira geral, fluxos de dados são caracterizados como sendo uma seqüência temporalmente ordenada e contínua de itens, gerados em tempo real, por uma determinada fonte de dados. Consultas aplicadas a estes fluxos de dados podem vir a ser executadas continuamente, de modo a retornar resultados incrementais, à medida que novos dados vão sendo recepcionados.

Muitas das estratégias previstas em Sistemas de Banco de Dados (SBDs) convencionais não são aplicáveis a fluxos contínuos de dados ou à execução de

consultas contínuas, visto que os SBDs trabalham com conjuntos finitos de tuplas. Por exemplo, operadores da álgebra relacional, como junções e agregações, bloqueiam o fornecimento dos resultados das consultas, até que pelo menos uma das relações envolvidas tenha sido completamente lida; o que é inviável quando se considera o processamento de fluxos de dados de tamanho indeterminado. Assim, Sistemas de Fluxos de Dados (SFDs) [5][37] vêm sendo propostos como alternativa para prover funcionalidades de um SBD convencional às aplicações que processam fluxos de dados. A Tabela 3.1 apresenta os principais aspectos que diferenciam SBDs e SFDs [37].

Tabela 3.1 Aspectos que diferenciam SBDs e SFDs.

Características	**Sistemas de Bancos de Dados (SBDs)**	**Sistemas de Fluxos de Dados (SFDs)**
Recursos físicos (e.g.,memória)	Razoável disponibilidade	Comumente restritos
Modelo de dados	Relações persistentes	Relações transitórias
Relação	Conjunto de tuplas	Seqüência de dados
Consulta	Transitória	Comumente persistente
Plano de consultas	Fixo	Adaptativo
Dados do resultado da consulta	Previamente materializados	Gerados em tempo-real
Processamento de consulta	Voltado à otimização do tempo de resposta	Voltado à otimização no uso dos recursos físicos
Ordenação dos dados para o processamento da consulta	O sistema detém o controle sobre a ordem em que os dados são recuperados para atender à consulta	Não há controle sobre a ordem em que os dados chegam para serem processados
Tipo de consulta	Direcionadas a um intervalo definido de dados	Comumente são do tipo contínua, direcionadas a um volume ilimitado de dados
Resultado das consultas	Preciso	Geralmente aproximado
Dados do resultado	Permanecem materializados	Descartados após atender à consulta
Atualização de dados	Inclusão, alteração e exclusão de valores	Novos valores anexados ao final do fluxo de dados

De um modo geral, SFDs exploram dois ingredientes-chave para o processamento de fluxos de dados: adaptabilidade e geração de resultados aproximados [5]. Aplicações que trabalham com fluxos de dados geralmente têm que lidar com um volume de dados que supera os limites da memória principal disponível. A adaptabilidade capacita algoritmos a ajustarem seu comportamento a eventos específicos como, por exemplo, recepções tempestivas de dados ou limitações do espaço de armazenamento. Ainda assim, “estouros” de memória podem ocorrer. A alternativa mais explorada para a prevenção de situações de “estouro” de memória é a geração de resultados aproximados. Estes resultados são obtidos com base na avaliação amostral dos dados ou na aplicação de outras

técnicas que permitam a condensação destes dados. Aspectos referentes à adaptabilidade e à geração de resultados aproximados são apresentados nas subseções a seguir.

3.2.1 Adaptabilidade

Algoritmos que adotam estratégias de adaptabilidade, denominados algoritmos adaptativos, são capazes de alterar seu comportamento em resposta a mudanças no seu ambiente de execução. O uso destes algoritmos possibilita a redução de perdas de desempenho causadas por problemas como, por exemplo, alterações nas taxas de fornecimento de dados, mudanças no ambiente de execução da consulta, "estouro" de memória principal, limitações de energia ou desconhecimento de estatísticas sobre os dados.

Algoritmos adaptáveis devem considerar três fatores [10]:

(i) a freqüência com que o sistema pode receber informações e adaptar seu comportamento ao novo cenário;
(ii) os efeitos provocados no comportamento do sistema diante de um determinado evento que alterou a configuração do ambiente; e
(iii) a extensão da adaptabilidade, que corresponde à repetição, no decorrer da execução da consulta, do processo de adaptação de comportamento em função das mudanças no ambiente.

Para muitas aplicações, longas esperas até ser obtido algum resultado são inviáveis. A alternativa encontrada pelos algoritmos adaptativos para amenizar este problema foi buscar a disponibilização de resultados simultaneamente ao processamento da consulta, tornando possível ao usuário tomar decisões baseadas nos dados até então liberados. Esta estratégia é particularmente interessante para aplicações executadas sobre RSSFs, visto que, comumente, as consultas submetidas a estas redes são muito longas ou mesmo de tempo indeterminado. Por exemplo, a execução de consultas contínuas requer a adoção de alguma estratégia que permita a produção incremental de resultados, caso contrário seus resultados nunca seriam produzidos.

3.2.2 Aproximação de resultados

A geração de fluxos de dados como resultado da execução de consultas contínuas pode levar a “estouros” de memória principal, tanto nos nós-sensores quanto na estação-base, tendo em vista o limitado tamanho desta memória. De modo que seria interessante que as estratégias utilizadas para realizar o processamento de fluxos de dados buscassem reduzir o volume de dados a ser processado. A adoção dessas estratégias tem como conseqüência o fornecimento de resultados baseados em amostragens dos dados, o que leva à geração de resultados apenas aproximados dos resultados reais, os quais seriam obtidos caso todo o fluxo de dados fosse considerado no atendimento da consulta. Limitações de memória são ainda mais fortes em nós-sensores, o que requer a adoção de mecanismos que evitem a perda de dados provocada por “estouros” de memória. Estratégias que levam ao fornecimento de resultados aproximados são descritas nas subseções a seguir.

3.2.2.1 Histogramas

Esta estratégia captura a distribuição dos diferentes valores presentes em um conjunto de dados. Valores distintos, associados as suas freqüências de ocorrência, são uniformemente distribuídos em partições, o que permite agilizar a atualização dos valores destas freqüências, à medida que cada novo dado é recebido. Os Histogramas vêm sendo empregados em aplicações de *data warehouse*, *data mining*, em otimizadores de consultas de bancos de dados convencionais (por exemplo, para estimar fator de seletividade de valores de atributos) e, mais recentemente, aproximação dos resultados de consultas aplicadas a fluxos de dados [5].

3.2.2.2 *Wavelets*

Consiste na aplicação progressiva de funções matemáticas aos dados gerados em resposta a uma consulta do usuário, objetivando produzir uma sinopse compacta do resultado [20].

Para a demonstração da sua versão mais simples (*Haar Wavelet*), pode-se tomar o exemplo a seguir. Seja um fluxo de dados s_1= [127,71,87,31] e uma função (*f*), que obtém a média aritmética entre os elementos de s_1 tomados aos pares. A nova seqüência de dados, s_2, obtida a partir de s_1 pela aplicação da função *f*, seria

[99,59], pois (127+71) /2 = 99 e (87+31)/2=59. Para se evitar a perda de informações quando for necessária a restauração de *s*, deve-se manter um coeficiente associado a cada elemento em s_2. Este coeficiente corresponde à diferença entre cada elemento de s_2 e o segundo elemento do par de valores utilizado pela função *f*. Assim, o coeficiente associado ao primeiro elemento de s_2 será 28, pois 99 – 71=28. A mesma função é aplicada sucessivamente, gerando seqüências de dados cada vez mais compactas, com seus conjuntos de coeficientes associados. A Tabela 3.2 apresenta o exemplo descrito.

Tabela 3. 2 Exemplo de aplicação da técnica de *Wavelets*.

Fluxo de dados	Coeficientes associados
127,71,80,30	-
99,55	(28,25)
77	(22)

Esta estratégia vem sendo particularmente aplicada em processamento de imagens, em ferramentas OLAP e para a obtenção de resultados aproximados em aplicações que trabalham com fluxos de dados. Observa-se que os resultados obtidos a partir do uso de *Wavelets* são gerados mais rapidamente e com maior precisão do que aqueles obtidos a partir do uso de Histogramas, considerando a mesma quantidade de memória em ambas as estratégias [20].

3.2.2.3 *Sliding Windows*

Esta estratégia define uma janela sobre os dados que permite limitar o volume de dados a ser considerado no processamento de uma dada consulta. Critérios comuns para limitar o volume de dados são baseados em critérios que envolvem a especificação de intervalos de tempo ou definição da quantidade de dados a ser extraída do fluxo dos dados. Por exemplo, poder-se-ia definir que apenas os dados coletados nas últimas duas horas serão considerados para responder a uma consulta.

Considerando aplicações que trabalham com fluxos de dados em RSSFs e que possuem recursos de memória limitados, as *Sliding Windows* constituem uma estratégia que permite delimitar um intervalo de dados a ser considerado na produção dos resultados da consulta [22].

3.2.2.4 *Punctuations*

Permite reduzir o volume de dados a ser processado através da avaliação semântica dos dados. Basicamente, cada item de dado é submetido a funções booleanas que irão identificar, com base em um predicado de seleção (*Punctuation*), se o dado é desnecessário ao resultado, podendo, portanto, ser eliminado.

Fisicamente, os *Punctuations* podem ser vistos como predicados inseridos no fluxo de dados, os quais marcam o fim de subconjuntos dos dados tratados para responder à consulta [54]. Este artifício permite tratar um fluxo de dados, potencialmente infinito, como uma composição de fluxos finitos, que vão incrementalmente compondo o resultado. Um exemplo de *Punctuation* poderia definir que apenas os dados com um determinado valor constante interessariam ao resultado da consulta. Avaliando informações como estas é possível inferir quais dados podem ser eliminados ou mantidos, como forma de reduzir a quantidade de dados materializados para a composição de resultados.

Embora os *Punctuations* objetivem conseguir resultados corretos incrementalmente, estouros de memória no ambiente de execução da consulta podem ocorrer, forçando a eliminação de dados que deveriam figurar no resultado. Neste caso, os resultados produzidos terão sua precisão reduzida, gerando resultados apenas aproximados.

3.2.2.5 *Sketches*

O propósito desta estratégia é rapidamente estimar o número de ocorrências de valores distintos para os campos dos fluxos de dados, como forma de reduzir a necessidade de espaço em memória. O problema da contagem de valores distintos de um conjunto de n elementos requer $\Omega(n)$ de espaço para obtenção de soluções exatas e $\Theta(\log n)$ de espaço para obtenção de soluções próximas às soluções exatas [15]. As estratégias mais comuns que adotam *Sketches* envolvem o uso de funções *hash*, as quais associam mapas de bits aos valores distintos dos campos de um fluxo de dados. Os *Sketches* são particularmente aplicáveis a RSSFs dada a sua capacidade de sumarização de dados e aproximação de resultados.

3.2.2.6 *Sampling*

Quando os dados chegam mais rápido do que o processador de consultas é capaz de computar, a consideração de todos os elementos de um fluxo de dados para a resolução de uma consulta pode tornar-se inviável. O *Sampling* [5] é uma alternativa de aproximação de resultados que consiste em desconsiderar partes de um fluxo de dados recebido tempestivamente. Como resultado, a amostragem produzida pode ser usada como uma estrutura de sumário, na qual uma pequena amostra é suficiente para capturar características essenciais dos conjuntos de dados que atendem à consulta.

É possível, ainda, que outras sinopses sejam obtidas a partir de amostras anteriores, porém, sucessivas aplicações desta estratégia reduzem progressivamente a precisão dos resultados. Nesta situação, torna-se difícil avaliar a precisão do resultado produzido, tendo em vista a dificuldade de mensuração dos limites de confiança associados ao grau de erro introduzido pela extração das amostras [5].

3.2.3 Pesquisas acadêmicas em processamento de fluxos de dados

A adaptabilidade e a adoção de alternativas para obter aproximações dos resultados produzidos a partir de fluxos de dados vêem sendo exploradas em inúmeras pesquisas acadêmicas. A seguir são apresentadas algumas das pesquisas voltadas para o processamento de fluxos de dados.

(i) NiagaraCQ[14]: Sistema destinado ao processamento de consultas contínuas em bancos de dados de aplicações Internet. Seu objetivo é obter escalabilidade na submissão de um grande volume de consultas através do agrupamento destas consultas. Este agrupamente consiste em fazer com que consultas com predicados similares compartilhem processamentos comuns, sendo executadas sobre uma mesma massa de dados. Desta forma, menos planos de execução precisam ser mantidos em memória e menos operações de E/S são necessárias;

(ii) Tukwila[35]: Promove a integração de dados provenientes de múltiplas fontes. O Tukwila alterna fases de planejamento e execução da consulta, produzindo planos completos ou parciais, os quais são progressivamente otimizados. Operadores adaptativos de consulta são adotados durante

cada fase de execução. Estes operadores permitem a produção incremental de resultados, além de serem capacitados a adaptar-se a eventos específicos, como, por exemplo, insuficiência de memória e imprevisibilidade na freqüência de recepção dos dados;

(iii) Aurora[12]: Voltado para aplicações de monitoramento do ambiente, o Aurora implementa operadores básicos da álgebra relacional como junções e agregações, que são aplicáveis a fluxos de dados. Permite reduzir o volume de dados considerado na execução da consulta através do uso de *Sliding Windows*, eliminação aleatória de dados e filtragem de dados. Suporta consultas contínuas, *ad-hoc*[4] e visões, sendo que fluxos de dados podem ser tanto utilizados por consultas contínuas como por outros tipos de consultas;

(iv) STREAM (*Stanford Stream Data Manager*)[42]: Seu objetivo principal é maximizar a precisão do resultado da consulta utilizando estratégias de aproximação de resultados, necessárias em situações de restrição de recursos. Para isso, utiliza *Sliding windows*, seleção de amostragens de dados ou eliminação de parte dos dados recebidos durante a execução da consulta. Estas estratégias são suportadas por uma linguagem de consultas declarativa, CQL (Subseção 3.3.5.2), e pelo compartilhamento de planos de consultas que possuem operadores comuns;

(v) Gigascope[16]: Sistema de banco de dados capaz de tratar fluxos de dados de aplicações voltadas ao gerenciamento de redes. Seu objetivo é utilizar consultas cujos resultados forneçam informações para a análise, por exemplo, do tráfego da rede, configurações de rotas, monitoramento de desempenho ou detecção de ataques. As consultas são expressas em uma linguagem declarativa, GSQL (GigaScope Query Language), que é capaz de suportar operadores básicos de seleção, projeção, junção e agregação. Mas não oferece suporte a consultas contínuas; e

(vi) TelegrathCQ[13]: Explora o processamento adaptativo de consultas sobre grandes fluxos de dados. Capacitou a arquitetura do PostgreeSQL a trabalhar com o processamento compartilhado de consultas contínuas. Utiliza *Sliding windows* e estratégias como o *Eddy* e o *Fjord*. *Eddy* é um

[4] Consultas ad-hoc são formuladas pelo usuário e submetidas ao sistema, sem que haja um conhecimento prévio do sistema a cerca da combinação de atributos.

mecanismo que reordena os operadores de um plano de consulta para alcançar ganhos de desempenho. Já o *FJord* permite que planos de consultas utilizem, simultaneamente, conexões de entrada e saída entre os vários módulos da arquitetura Telegraph.

3.3 Sistemas de fluxos de dados para RSSFs

As aplicações convencionais executadas em RSSFs geralmente utilizam nós-sensores apenas dedicados à coleta de dados do ambiente. Após a coleta nos nós-sensores, os dados são enviados à estação-base, onde serão utilizados para a resolução das consultas submetidas pelos usuários. Estender o uso da tecnologia de banco de dados até os nós-sensores, de forma a permitir que parte do processamento da consulta seja feita nestes nós, permite reduzir o volume de dados transmitido através da rede e, conseqüentemente, reduzir o consumo de energia requerido em operações de comunicação.

Observa-se que os dados coletados pelos nós-sensores estão mais sujeitos a imprecisões devido às fortes restrições de recursos e à susceptibilidade a falhas dos nós-sensores [18]. Por outro lado, considerando-se que as estações-base de RSSFs consistem em nós robustos, quanto à capacidade de processamento e armazenamento, estratégias de implementação de SFDs são principalmente aplicáveis a estes nós, tendo em vista o volume de dados que pode ser gerado por um grande número de nós-sensores.

3.3.1 Armazenamento temporário de dados em RSSFs

O processamento de consultas em RSSFs, assim como em outras arquiteturas de rede, requer o armazenamento, pelo menos temporário, dos dados coletados pelos sensores em um ou mais pontos da rede, denominados pontos de materialização. Este armazenamento pode se dar de três formas [18]:

(i) Local: quando o dado é materializado nos nós sensores;
(ii) Externo: a materialização ocorre em nós mais robustos da rede;e
(iii) Centrado no tipo de dado: quando o armazenamento se dá de acordo com a capacidade de um nó em responder a uma determinada consulta.

As técnicas de armazenamento de dados influenciam diretamente no consumo de energia da rede como um todo. Assim, a decisão de como os dados devem ser

armazenados deve ser resultado da avaliação das características da rede (por exemplo, topologia e número de nós) e da aplicação (por exemplo, o fenômeno monitorado e a natureza das consultas) [18].

O armazenamento local requer a materialização dos dados nos nós-sensores, que são os pontos de origem da informação. Esta estratégia de armazenamento pressupõe que a consulta seja conhecida por todos os nós da rede e que cada um destes nós seja capaz de processar a consulta localmente. Como conseqüência, reduz-se o consumo de energia necessário à transmissão/recepção de dados, pois apenas os dados realmente requisitados pelo usuário são enviados através da rede. Nesta estratégia, o custo de distribuição da consulta a todos os nós da rede é *O(n)*, enquanto os dados são enviados à estação-base com um custo de $O(\sqrt{n})$, tendo *n* como o número de nós da rede [46]. Por outro lado, além do custo relacionado ao consumo de energia necessário à distribuição da consulta para todos os nós da rede, há também um aumento significativo da energia consumida para o processamento dos dados nos nós-sensores. Em Ganesan *et al.* [19] é proposta a abordagem *Dimensions*, que explora as correlações de tempo e espaço entre os dados coletados pelos nós-sensores, utilizando técnicas de compressão de dados como, por exemplo, *Wavelet*s, visando reduzir o consumo de energia em uma estratégia de armazenamento temporário local.

No armazenamento externo, os dados coletados pelos nós-sensores são continuamente enviados para pontos da rede, estações-base, que possuam uma maior disponibilidade de recursos. Neste caso, embora não haja nenhum consumo de energia relacionado ao processamento das consultas nos nós-sensores, há um alto custo relacionado à comunicação, visto que todos os dados coletados devem ser enviados às estações-base, sem nenhuma análise prévia. Assim como o *Dimensions, o* PREMON (*PREdiction-based MONitoring*) [21] é uma proposta que também explora as correlações de tempo e espaço existentes entre os dados coletados pelos sensores, porém, utilizando armazenamento externo. A redução no custo de comunicação é buscada através da utilização de modelos de predição, os quais permitem a uma estação-base prever os dados mais prováveis de serem coletados por subconjuntos de nós-sensores. Estas predições são enviadas aos nós-sensores, os quais são instruídos a somente transmitirem dados cujos valores não estejam próximos àqueles previstos nas estações-base. Porém, o custo associado

ao cálculo e distribuição da predição pode gerar dificuldades quanto à escalabilidade da rede. Aplicações de detecção de eventos, por exemplo, requerem um custo $O(\sqrt{n})$ para cada evento detectado que é enviado para armazenamento externo, tendo n como o número de nós da rede [46].

No armazenamento centrado no tipo de dado, são atribuídos nomes a cada tipo de informação coletada pelos nós-sensores, sendo estes nomes utilizados para acessar os dados. Esta estratégia requer um consumo extra de energia para que os nós, onde são armazenados os dados, tenham o conhecimento das informações coletadas pelos outros nós da rede. Porém, o consumo de energia resultante do processamento da consulta na rede como um todo é reduzido, pois uma consulta apenas é direcionada aos nós capazes de atender aos seus predicados, não sendo necessário o seu envio a todos os nós na rede [18]. Em Ratnasamy *et al.* [46] é proposto um algoritmo de roteamento geográfico que explora o armazenamento centrado no tipo de dado. Nesta proposta, são utilizadas funcionalidades de uma tabela *hash* para prover um mecanismo de buscas ponto-a-ponto, sendo que os dados são nomeados e acessados através de seus tipos, por exemplo, tipo de medição = temperatura, de forma que todos os dados que possuam o mesmo tipo sejam armazenados em um mesmo nó da rede, e não necessariamente naquele em que o dado foi coletado.

O armazenamento centrado no tipo de dado se sobressai em relação a outras estratégias de armazenamento por requerer um custo de $O(\sqrt{n})$, tendo n como o número de nós da rede, tanto para a distribuição das consultas na rede quanto para o envio dos dados coletados pelos nós-sensores às estações-base [46].

3.3.2 Modelo de dados para fluxos de dados em RSSFs

A especificação de um modelo de dados é importante para dar aos desenvolvedores de aplicação uma visão lógica do banco de dados, fornecendo uma descrição da representação e da semântica dos dados. Considerando particularmente as RSSFs, o modelo de dados permite abstrair das aplicações aspectos físicos relacionados à localização dos nós-sensores, ao roteamento de pacotes (por exemplo, alterações no esquema de localização dos dados e/ou roteamento) e à coleta de dados do ambiente [23].

A versão mais popular de modelo de dados é a relacional [23], na qual um conjunto de dados com semântica semelhante é representado logicamente por relações e fisicamente por tabelas. Com o advento dos sistemas de bancos de dados propostos para atender às necessidades das RSSFs, a modelagem relacional também ganhou espaço no desenvolvimento de aplicações voltadas para estas redes. Propostas de modelagem de dados no contexto das aplicações executadas sobre RSSFs são apresentadas a seguir. Estas propostas têm em comum o fato de considerar que suas tabelas estão particionadas horizontalmente através dos nós-sensores da rede, de forma que cada subconjunto destes nós alimenta uma tabela correspondente na estação base.

Em Govindan *et al.* [23] é proposta uma modelagem na qual uma coleção de tuplas[5] de tipos similares (por exemplo, tipo de medição = temperatura), provenientes de um mesmo subconjunto dos nós-sensores, seja representada como uma tabela. Em um sistema de banco de dados relacional, as tabelas são tipicamente armazenadas em disco; porém, no contexto das RSSFs, todas as tabelas são virtuais, representando visões relacionais sobre os dados gerados pelos nós-sensores da rede. As consultas processadas sobre as tabelas virtuais são automaticamente traduzidas em operações de coleta de dados, por exemplo, “colete o valor corrente da temperatura”, sendo direcionadas apenas aos nós-sensores relevantes. Por exemplo, um nó-sensor de temperatura pode produzir uma estrutura na forma <*l*, *t*, *v*>, onde *l* é a localização do nó-sensor, *t* é o instante de tempo em que a coleta foi feita e *v* é o valor da temperatura sensoriada. Considerando esta estrutura, as tuplas geradas pelos nós-sensores de temperatura irão compor a tabela *temperatura*.

No sistema de banco de dados para redes de sensores TinyDB [39] (abordado na Subseção 3.3.6.3), da Universidade da Califórnia, os dados gerados a partir das coletas dos nós-sensores são todos representados por uma tabela denominada *sensors*. Nesta tabela, é gerada uma linha para cada nó-sensor, em cada instante de tempo, sendo que cada coluna na tabela *sensors* representa um tipo de informação coletada pelos nós-sensores da rede (por exemplo, medição de temperatura, umidade e pressão atmosférica). Nesta proposta, os dados coletados do ambiente

[5] Em Govidan *et al* é considerada como sendo uma tupla a estrutura de dados gerada a partir de uma coleta de dados do ambiente, realizada por um nó-sensor.

apenas são materializados na tabela *sensors* quando necessários à resolução de uma consulta. Neste caso, os resultados das operações de projeção e/ou transformação, aplicadas aos dados, são temporariamente armazenados nos nós-sensores, compondo pontos de materialização.

A arquitetura de gerenciamento de dados Cougar [57] (abordada na Subseeção 3.3.6.2) modela cada tipo de dispositivo-sensor, por exemplo, nó sensor de temperatura, como um TDA (Tipo de Dado Abstrato), que é um valor de atributo encapsulado em uma coleção de dados relacionados. Esta idéia advém do fato de que os bancos de dados objeto-relacional e orientados a objetos suportam TDAs. Nesta arquitetura, um objeto TDA em um banco de dados corresponde a um nó-sensor físico no mundo real. A interface do TDA corresponde às funções específicas suportadas pela arquitetura Cougar. Estas funções são utilizadas para requisitar informações aos nós-sensores. A arquitetura Cougar também utiliza o conceito de "relação virtual", como sendo uma representação tabular de uma função. Cada função contribui com uma porção à relação virtual na qual está associada. Um registro em uma relação virtual contém os argumentos de entrada e os argumentos de saída da função, com a qual ela está associada. Assim, seja uma relação qualquer definida por RFSensors(*Sensor*, *x*, *y*), onde *Sensor*, x e y correspondem aos atributos da relação; tem-se que o primeiro atributo é um TDA que representa um nó-sensor físico de temperatura, enquanto os outros dois atributos denotam a localização do nó-sensor [7]. As funções disponibilizadas em cada nó-sensor retornam, por exemplo, o valor corrente da temperatura, sendo que as coletas de dados são momentaneamente armazenadas nos próprios nós-sensores.

3.3.3 Processamento de consultas em RSSFs

O objetivo-chave do processamento de consultas é a extração eficiente dos dados de um banco de dados. Considerando aplicações de RSSFs, o processamento da consulta poderá ser distribuído entre a estação-base e os nós-sensores. Neste caso, o consumo de energia requerido para processar a consulta nos nós-sensores passa a ser um importante parâmetro a ser considerado na otimização do uso de recursos da rede.

Consultas submetidas a bancos de dados convencionais freqüentemente buscam tuplas materializadas em unidades de disco. Em aplicações de RSSFs não

existe uma materialização prévia, pois os dados que irão compor o resultado de uma consulta apenas serão produzidos após a recepção da consulta pelos nós-sensores. Além disto, a noção de tupla, como sendo um conjunto de atributos com uma estrutura bem definida, não é aplicável às RSSFs, visto que os nós sensores apenas trabalham com fluxos de dados.

As etapas seguidas pelos bancos de dados convencionais para o processamento de consultas são: análise, tradução, otimização e avaliação [50]. Durante a etapa de análise (*parsing*) a consulta escrita em uma linguagem declarativa de alto nível, como as linguagens derivadas do SQL, é analisada sintaticamente, por exemplo, verificando se os nomes de atributos e relações são válidos, gerando uma árvore de análise como resultado. Na etapa de tradução, a árvore de análise é traduzida para uma forma de representação interna do sistema, por exemplo, expressões da álgebra relacional. A etapa de otimização consiste em planejar a estratégia de execução da consulta. Nesta etapa o otimizador do banco de dados gera alguns planos equivalentes de consulta e escolhe o mais eficiente[6]. Finalmente, na etapa de avaliação, a consulta é executada segundo o plano escolhido.

Em uma RSSF, as etapas convencionais seguidas para o processamento de uma consulta devem estar associadas a estratégias de fragmentação e distribuição da consulta para os nós-sensores, os quais devem coletar dados do ambiente e processá-los, por exemplo, via aplicação de operações de filtragem e agregação. Os dados pré-processados são enviados através da rede até alcançarem a estação-base que, por sua vez, finaliza o processamento sobre os dados, produzindo os resultados da consulta para o usuário.

3.3.3.1 Agregação de dados em RSSFs

Em Yao *et al.* [56] são consideradas três diferentes estratégias para o tratamento de dados em nós-sensores de RSSFs. A primeira é a entrega direta, que é a estratégia mais simples, pois consiste apenas no envio e repasse de pacotes entre os nós-sensores, até que os dados cheguem à estação-base, onde serão finalmente processados. A segunda estratégia é a fusão entre pacotes, que une diversos

[6] O plano de consultas mais eficiente, em sistemas de bancos de dados convencionais, será geralmente aquele que resulta em um menor número de acessos a disco e, conseqüentemente, em um menor tempo de resposta da consulta.

pequenos pacotes em um único e grande pacote, o que consome menos recursos, pois são reduzidos os custos na reserva do canal de comunicação e os custos de se ter vários cabeçalhos de pacotes. A terceira estratégia á aplicável quando se pode processar a agregação em rede, sendo esta a estratégia que resulta no menor volume de dados enviado através da rede.

É importante ressaltar que a estratégia de agregação em rede desvia os algoritmos de roteamento, antes realizados com base no endereço dos nós, para um novo paradigma, que tem por base o roteamento centrado nos dados. Neste novo paradigma, o objetivo é encontrar as rotas de múltiplas fontes origem (nós-sensores) para um destino (nó-sensor ou estação-base) capazes de consolidar os dados através da aplicação de operações de agregação [36], o que é realizado a partir de avaliações sobre a semântica dos dados tratados. As propostas que fazem a comunicação do tipo múltiplos saldos, associadas à agregação em rede, têm recebido especial atenção das pesquisas voltadas para RSSFs [30][34][36][38].

O tipo de agregação especificado em uma consulta pode definir diferentes formas de computação de resultados parciais. Em Madden *et al.* [41] são descritas propriedades relacionadas a operações de agregação que são particularmente relevantes para aplicações de RSSFs:

(ii) Sensibilidade à duplicação: quando a duplicação de dados afeta a precisão do resultado da operação de agregação;
(iii) Sumarizável: quando é aceitável que a operação de agregação seja aplicada a apenas um subconjunto dos dados coletados, produzindo resultados apenas aproximados; e
(iii) Monotonicidade: quando a agregação aplicada à coleta de dois itens de dados tem como resultado apenas um item de dado.

Em Gray *et al.* [24] e Madden *et al.* [41], as operações de agregação são classificadas conforme a seguir:

(i) Distributiva: o tamanho de cada resultado parcial, obtido pela aplicação de uma função de agregação, tem o mesmo tamanho do resultado final. Por exemplo, as funções MAX, MIN, SUM e COUNT produzem um único valor para cada grupo de dados. Vale salientar que funções como SUM e

COUNT são sensíveis a duplicações, o que não ocorre com as funções MAX e MIN;

(ii) Algébrica: cada resultado parcial corresponde a um par de valores, obtidos a partir da aplicação das funções distributivas SUM e COUNT. É o caso da função AVERAGE, para obtenção de médias aritméticas;

(iii) Holística: nenhuma agregação parcial é útil, pois todos os dados devem ser enviados à estação-base, onde só então poderão ser agregados. É um tipo de agregação sensível a duplicações e o seu principal exemplo é a função MEDIAN, para a obtenção da mediana;

(iv) Única: a quantidade de resultados parciais propagados através da rede é proporcional ao número de valores distintos de grupos. Este tipo de agregação não é sensível a duplicações. É o caso de expressões como COUNT DISTINCT, que conta o número de ocorrências de cada valor distinto computado; e

(v) Sensível ao conteúdo: os resultados parciais obtidos nos nós-sensores são proporcionais, em tamanho, aos resultados obtidos a partir de avaliações estatísticas aplicadas aos dados coletados do ambiente. Este tipo de agregação é utilizado em muitas das estratégias de aproximação de resultados (Subseção 3.2.2). É o caso da função HISTOGRAM, que captura a distribuição dos diferentes valores em uma amostra de dados.

3.3.3.2 Aproximação de resultados em RSSFs

Devido à limitada disponibilidade de memória e energia dos nós-sensores, aplicações de RSSFs comandam a coleta de dados somente em intervalos de tempo predeterminados. Logo, os resultados destas aplicações baseiam-se apenas em amostras de dados, as quais fornecem resultados apenas próximos ao real. Para muitas aplicações, como as de monitoramento de temperatura, um resultado preciso só seria conseguido através da coleta e envio contínuo dos dados para a estação-base, o que demandaria um grande consumo de memória e energia. A suscetibilidade a falhas dos nós-sensores e do canal de comunicação também são fatores que contribuem para que as aplicações de RSSFs produzam resultados apenas aproximados.

Em Deshpande *et al.* [17] são apontados dois possíveis problemas relacionados aos resultados produzidos pelas aplicações de RSSFs:

(i) Falta de representatividade dos dados, que se deve ao fato de que os dados coletados do meio podem não ser uma representação confiável do mundo real, devido a fatores como, por exemplo, implantação não-uniforme dos nós sensores em uma região geográfica, falha dos nós-sensores ou altas taxas de perda de pacotes; e

(ii) Ineficácia na aproximação de resultados, que se refere ao fato de que os resultados produzidos a partir de uma RSSF são aproximados, no sentido de que eles somente representam o comportamento do ambiente monitorado em uma determinada localização, onde os nós-sensores estão posicionados, e no momento em que a coleta foi realizada pelos nós.

Considerando que os sensores têm sua energia consumida particularmente em operações de transmissão e recepção, o envio imediato de um dado, após a sua captação do ambiente, torna-se inviável, devido à quantidade de energia que seria gasta em comunicação. Por outro lado, a materialização dos dados coletados, durante longos períodos, poderia facilmente levar a "estouros" de memória.

Em Deshpande *et al.* [17] é considerado que as propostas de processamento de consultas para RSSFs devem tentar coletar a maior quantidade possível de dados do ambiente, objetivando alcançar a maior precisão possível dos resultados produzidos para as consultas. Para compensar a conseqüente penalização dos recursos da rede é proposto o uso de modelos de dados estatísticos, os quais prevêem o uso de valores estimados das leituras dos nós-sensores, em um dado período de tempo, de modo que os resultados das consultas sejam gerados a partir destas estimativas. Desta forma, quando as incertezas quanto à precisão dos resultados alcançam um certo limite, os nós-sensores podem ser consultados para que estas estimativas sejam refinadas. O modelo adota funções probabilísticas, $k(x_1, x_2,..., x_n)$, assinalando uma probabilidade correspondente a cada atributo, $x_1,..., x_n$. Nesse caso, x_i índice representa um atributo em um nó-sensor particular, por exemplo, nó-sensor de temperatura. Já o valor $k(x_i)$ denota a probabilidade de que o valor x_i seja aceitável na resolução de uma dada consulta e n seria o número de nós-sensores da rede.

3.3.4 Linguagens de consultas aplicáveis a RSSFs

A tecnologia de banco de dados proporciona ao usuário um meio transparente para acessar os dados de uma aplicação, o que é realizado através da submissão de consultas, comumente escritas em linguagens declarativas. Este aspecto é particularmente relevante quando estas consultas são submetidas a aplicações executadas sobre RSSFs, tendo em vista a necessidade de se dar transparência tanto para aquelas tarefas executadas pelo sistema de banco de dados, como geração de planos de consulta e otimização; quanto para aquelas realizadas pela rede, como distribuição de consultas, coleta de dados e roteamento de pacotes. Além disto, é essencial considerar as restrições de recursos dos nós-sensores, em todas as etapas do processamento da consulta.

No contexto das aplicações de RSSFs, as características dos processadores de consultas convencionais precisam ser ajustadas, de modo a oferecer suporte ao processamento de fluxos de dados e à execução de consultas contínuas. Consultas comuns submetidas a aplicações de RSSFs são mostradas na Tabela 3.3 [62].

Tabela 3.3 Tipos de consulta comuns em aplicações executadas em RSSFs.

Tipo de consulta	Funcionalidade	Exemplo
Histórica	As informações são coletadas pelos nós-sensores e armazenadas em bancos de dados na estação-base.	Qual foi a temperatura média em Fortaleza em Janeiro de 2006?
Pontual	Informações coletadas do ambiente em um momento específico.	Qual a temperatura média corrente em Fortaleza?
Baseada em intervalos de tempo	Informações coletadas do ambiente por um período de tempo, previamente especificado.	Retorne a temperatura média em Fortaleza a cada 1 hora, considerando medições efetuadas entre as 12:00hs e as 18:00hs de hoje.
Baseada em eventos	A coleta de dados do ambiente é realizada em resposta à ocorrência de um evento previamente especificado.	Se a temperatura média em Fortaleza exceder os 35°C envie um aviso de alerta à estação-base.
Multidimensional	Envolve mais do que um dos atributos de dados coletados pelos nós-sensores, podendo especificar limites de busca para estes atributos (filtros).	Liste a posição de todos os nós-sensores, em Fortaleza, que captaram valores de temperatura entre 30°C e 40°C e valores de umidade relativa do ar entre 60% e 75%.

A familiaridade dos usuários no uso de linguagens declarativas, derivadas do padrão SQL, faz destas linguagens uma forma mais natural para a elaboração de consultas submetidas às aplicações de RSSFs. Estas linguagens permitem a formulação de consultas *ad-hoc*, independentemente das mudanças na estrutura física da rede, por exemplo, devido a falhas em nós-sensores ou à mobilidade dos

nós da rede. Além disto, assim como ocorre em sistemas de bancos de dados convencionais, a implementação de interfaces de alto nível permite que usuários não-especialistas possam facilmente interagir com o banco de dados [56][61][62]. Nas subseções a seguir são apresentadas linguagens de consultas propostas no contexto de pesquisas acadêmicas na área de processamento de fluxos de dados e que são aplicáveis a RSSFs.

3.3.4.1 Linguagem aquisicional TinyDB

Implementada no contexto do projeto TinyDB [39] (abordado na Subseção 3.3.6.3), a linguagem de consultas aquisicional explora o fato de que os nós-sensores inteligentes têm o controle sobre onde, quando e com que freqüência os dados precisam ser coletados do ambiente para serem entregues aos operadores de processamento de consultas [40].

Tabela 3.4 Cláusulas definidas para a linguagem aquisicional proposta no TinyDB [39].

Clausula	Funcionalidade
SELECT- FROM[7]- WHERE- GROUP BY- HAVING	Assim como no padrão SQL, permitem especificar operações de projeção (SELECT), seleção (WHERE) e agregação (GROUP BY) sobre as relações especificadas na consulta.
ON EVENT	Provoca a iniciação da coleta de dados em resposta a algum evento externo (por exemplo, detecção da presença de um animal), uma outra consulta ou alguma ação realizada pelo sistema operacional.
STOP ON EVENT	Especifica uma condição de parada para uma consulta contínua, o que é feito quando a ocorrência do evento é detectada.
OUTPUT ACTION SIGNAL	Especifica um comando externo que poderia ser invocado em resposta a um dado coletado, que satisfaz a consulta, sem a necessidade de enviar uma mensagem de retorno à estação-base (por exemplo, “Ligar um ventilador caso a temperatura atinja um limite pré-configurado”).
INTO STORAGE POINT	Permite especificar pontos de materialização, comparáveis às visões materializadas dos bancos de dados convencionais.
LINEAR INTERPOLATE- COMBINE	Quando um ponto de materialização e uma sub-consulta entregam dados em diferentes taxas, duas situações são possíveis para a combinação dos dois fluxos de dados em um único fluxo de dados: utilizar a cláusula LINEAR INTERPOLATE se a sub-consulta for mais rápida ou utilizar a cláusula COMBINE, caso contrário.
SAMPLE PERIOD – FOR	Define o intervalo de tempo no qual uma consulta deverá ser executada e o intervalo de tempo entre coletas sucessivas de dados.
ONCE	Permite examinar o *status* de um nó em particular ou de um conjunto de nós em um determinado momento.
LIFETIME	Especifica o tempo de vida em que uma consulta permanecerá em execução, podendo o intervalo de tempo ser especificado em dias, semanas ou meses.

As consultas consistem basicamente de cláusulas SELECT-FROM-WHERE-GROUP BY, oferecendo suporte a seleções, junções, projeções e agregações.

[7] A cláusula FROM pode se referir tanto a uma relação (tabela *sensors* para o TinyDB) quanto a pontos de materialização, que são também tratados como relações.

Como forma de suportar características específicas das RSSFs, a linguagem de consultas aquisicional TinyDB permite a definição de sub-consultas através de pontos de materialização[8], além de utilizar cláusulas adicionais, descritas na Tabela **3.4**. Estas cláusulas objetivam atender a consultas baseadas em pontos de materialização, coletas de amostragens, coletas realizadas em resposta a eventos ou coletas feitas em intervalos de tempo predeterminados, que são implementações dos tipos de consulta apresentados na Tabela 3.3.

3.3.4.2 CQL

A linguagem *Continuous Query Language* (CQL) [3] foi proposta no contexto do projeto STREAM (Subseção 3.2.3). É uma linguagem de consultas declarativa que estende o padrão SQL, tendo sido especialmente projetada para suportar consultas contínuas aplicáveis tanto a fluxos de dados como relações.

CQL define três classes de operadores que fazem o mapeamento entre fluxos de dados e relações. Estas classes de operadores são descritas a seguir:

(i) Fluxo-relação: na qual o operador produz uma relação a partir de um fluxo de dados, utilizando, por exemplo, *Sliding windows*;

(ii) Relação-relação: na qual o operador produz uma relação a partir de uma ou mais outras relações, utilizando, por exemplo, operadores de junção relacional; e

(iii) Relação-fluxo: na qual o operador produz um fluxo de dados a partir de uma relação. CQL define os operadores *Istream*, *Dstream* e *Rstream* (Tabela 3.5), para processar tuplas de uma relação e produzir um fluxo de dados como resultado.

Um exemplo de consulta CQL poderia ser definido como a seguir. Seja s um fluxo de dados recebido no instante t_1; e s_1,s_2 e s_3 subconjuntos de s, de forma que $s(t_1) = \{<s_1,t_1>,<s_2,t_1>,<s_3,t_1>\}$. A partir de s é obtida a relação $R(t_1)$ em que $R(t_1) = \{s_1,s_2,s_3\}$.

[8] Pontos de materialização acumulam uma pequena área de armazenamento temporário de dados que pode ser compartilhada entre consultas [14].

Tabela 3.5 Operadores adotados na linguagem de consultas CQL.

Operador	Funcionalidade	Consulta-exemplo
[RANGE *t*]	Expressa o uso de uma *Sliding window* de tempo *t*, significando que apenas os dados recebidos durante as últimas *t* unidades de tempo serão consideradas na resolução da consulta.	SELECT valorUmidade, Count(*) FROM Umidade [RANGE 60] GROUP BY valorUmidade
[NOW]	Representa o uso de uma *Sliding window* de tempo *t*, com $t = 0$.	SELECT valorUmidade FROM Umidade [NOW]
[RANGE UNBOUNDED]	Expressa o uso de uma *Sliding window* de tempo *t*, com $t = \infty$.	SELECT valorUmidade FROM Umidade [RANGE UNBOUNDED] WHERE valorUmidade > 0,5
[ROWS *u*]	Define que a saída da consulta será uma relação gerada a partir das últimas *u* tuplas do fluxo de dados de entrada.	SELECT valorUmidade, Count(*) FROM Umidade [ROWS 1000] GROUP BY valorUmidade
[PARTITION BY x_1,..., x_k ROWS *u*]	Particiona um fluxo de dados *s* de acordo com um sub-conjunto $\{x_1, ..., x_k\}$ dos atributos de *s* (similar à cláusula Group By do SQL padrão). Para cada partição, são consideradas apenas as *u* últimas ocorrências do atributo em *s*.	SELECT AVG(valorUmidade) FROM Umidade [PARTITION BY RegiaoGeografica ROWS 10]
ISTREAM	Operador que aplicado a uma relação *R* gera um fluxo de dados correspondente apenas às novas tuplas incluídas em *R*.	SELECT Istream(*) FROM Umidade [RANGE UNBOUNDED]
DSTREAM	Operador aplicado a uma relação *R* para se obter um fluxo de dados das tuplas existentes em *R*, antes da execução da consulta.	SELECT Dstream(*) FROM Umidade [RANGE UNBOUNDED]
RSTREAM	Operador que é aplicado a uma relação *R* para gerar um fluxo de dados correspondente a todos os dados da relação *R*.	SELECT Rstream(*) FROM Umidade [NOW]

3.3.4.3 GSQL

GSQL (*GigaScope Query Language*) foi desenvolvida no contexto do projeto GigaScope (abordado na Subseção 3.2.3), como uma linguagem declarativa de consultas projetada para o tratamento de fluxos de dados provenientes de aplicações de monitoramento de redes.

GSQL analisa propriedades definidas através de marcas de tempo[9] e predicados da consulta, associados aos fluxos de dados de entrada, visando principalmente permitir o processamento de operadores com bloqueio (por exemplo, junções e agregações). Esta linguagem suporta operadores de seleção, junção e agregação; além de estender o padrão SQL com a proposta do operador *merge*. O operador *merge* é similar a um operador de união, exceto pelo fato de que ele une dois fluxos de dados de entrada, ordenados por marcas de tempo. Adicionalmente, como forma de prover maior flexibilidade às aplicações de análise de redes, GSQL

[9] Marcas de tempo ou *timestamps* identificam o momento em que um determinado evento, relacionado a um dado, ocorre.

também oferece suporte a funções definidas pelo usuário. Esta linguagem considera que a consulta recebe fluxos de dados de entrada, os quais estarão associados à cláusula FROM da consulta (por exemplo, SELECT fluxo1.atributo2, fluxo2.atributo3 FROM fluxo1, fluxo2 WHERE fluxo1.atributo1 = fluxo2.atributo1). Após o processamento da consulta, um único fluxo de dados de saída será produzido como resultado da consulta [16].

Considerando consultas contínuas compostas, nas quais a saída de uma consulta q_1 é entrada para uma outra consulta q_2, todas as novas informações geradas como resultados de q_1 devem ser freqüentemente reinterpretadas em q_2, à medida que os dados fluem. A dificuldade de implementação e resolução de consultas com estas características desencorajou os autores da proposta de GSQL a implementarem o suporte a consultas contínuas, pois os mesmos consideraram inapropriado o uso destas consultas em aplicações de monitoramento de redes [16].

3.3.4.4 SQTL

SQTL (*Sensor Query and Tasking Language*) é uma linguagem procedimental[10] de consultas para aplicações executadas sobre RSSFs, que permite a delegação de tarefas aos nós-sensores [52]. Esta linguagem age no papel de uma interface de programação entre aplicações de RSSFs e a arquitetura do *middleware*[11] SINA (abordada na Subseção 3.3.6.1). SQTL oferece suporte a três tipos de eventos:

(i) *Receive:* eventos gerados quando uma mensagem é recebida por um nó-sensor;
(ii) *Every:* para especificar eventos disparados periodicamente por um marcador de tempo; e
(iii) *Expire:* que representa eventos causados pela expiração de um marcador de tempo.

SQTL apresenta operadores que permitem acessar o *hardware* do nó-sensor, de modo a requisitar certas informações, tais como os dados coletados pelos nós-sensores, a localização destes nós ou informações sobre os nós-sensores vizinhos.

[10] Uma linguagem procedimental é aquela na qual o programa é executado seqüencialmente conforme a seqüência de chamadas de instruções e funções no programa.
[11] Um *middleware* é um *software* que conecta componentes de *software* ou aplicações.

Também é possível disparar uma ação como, por exemplo, ligar ou desligar algum componente do nó-sensor.

Uma consulta escrita em SQTL pode ser interpretada e executada em qualquer nó da rede. Para que esta consulta chegue a seu destino, um receptor específico ou um grupo de receptores, a mensagem deve ser empacotada, de modo que o pacote gerado receba um cabeçalho, com a indicação do transmissor e de seus nós-destinatários. O empacotador definido em SQTL adota a sintaxe de XML (*Extensible Markup Language*), para definir um cabeçalho que especifica o endereçamento [52].

3.3.5 Projetos de pesquisa em processamento de dados em RSSFs

A seguir são apresentados sistemas que oferecem suporte ao processamento de fluxos de dados em aplicações de RSSFs, desenvolvidos no contexto de pesquisas acadêmicas.

3.3.5.1 SINA

SINA (*Sensor Information Networking Architecture and Applications*) é uma arquitetura de RSSF que modela a rede como uma coleção de objetos maciçamente distribuídos [52]. SINA atua no papel de *middleware*, permitindo que aplicações submetam consultas à RSSF, requisitem a execução de tarefas pela rede, coletem respostas e produzam resultados. As funcionalidades do SINA incluem:

(i) Organização hierárquica de conjuntos de nós-sensores: adota como critério para a formação de um conjunto de nós a semelhança quanto à localidade e a disponibilidade de energia dos nós-sensores;

(ii) Descrição de atributos: necessária para aplicações centradas em dados. Estas aplicações admitem que os usuários tenham interesse na informação obtida por um conjunto de nós-sensores, e não apenas por um único nó-sensor individual; e

(iii) Conhecimento quanto à localização do nó: os nós-sensores devem operar de forma cooperativa e, portanto, o conhecimento da localização é importante para que os nós possam se comunicar.

3.3.5.2 Cougar

Projetado como uma arquitetura para gerenciamento de dados em RSSFs, Cougar [57] propõe o uso de uma nova camada de consultas em cada nó-sensor, a qual interage tanto com a camada de roteamento como com a camada de aplicação da rede.

Na arquitetura Cougar, a estação-base possui um otimizador de consultas, destinado a gerar planos de execução para o processamento distribuído das consultas. Um plano de execução especifica como será tratado o fluxo de dados entre os nós-sensores e como será realizada a computação dos dados em cada nó-sensor. Cada plano de consulta é criado tendo por base as informações de um catálogo[12] e as especificações constantes na própria consulta. Após a sua definição, o plano de execução das consultas é disseminado para todos os nós-sensores relevantes, segundo os critérios especificados nos predicados da consulta. O processamento realizado nos nós-sensores consiste, basicamente, na aplicação da técnica de agregação em rede.

3.3.5.3 TinyDB

TinyDB [39] é um processador de consultas distribuído que é executado em nós-sensores inteligentes, como os MicaMotes (abordados na Subseção 2.2.2.3). A proposta básica do TinyDB é realizar o processamento aquisicional de consultas, com o objetivo de minimizar o consumo de energia, maximizando a precisão dos resultados finais da consulta.

TinyDB é executado no topo do sistema operacional TinyOS, que consiste em um conjunto de componentes para o gerenciamento e acesso ao *hardware* do nó-sensor MicaMote. Também utiliza uma versão simplificada da linguagem de programação C, denominada nesC. Estes componentes provêem abstrações de *software*, incluindo componentes para modular pacotes sobre o rádio, leituras de valores de sensores para diferentes *hardwares*, sincronização de relógios entre transmissor e receptor e alteração de estados do *hardware* para permitir baixos consumos de energia [14].

[12] O catálogo do sistema é o local onde os sistemas gerenciadores de banco de dados relacionais armazenam os metadados do esquema, tais como informações sobre nomes de objetos e relacionamentos entre eles.

Como forma de aproximar o processamento de consultas convencional daquele realizado em uma RSSF, TinyDB objetiva oferecer suporte a operações de junção, projeção e agregação, a serem realizadas nos nós-sensores da rede. TinyDB oferece suporte a consultas baseadas em operações de agregação pela adoção da técnica de agregação em rede. Consultas aninhadas não são previstas, devido às dificuldades que seriam adicionadas ao tratamento dos fluxos de dados gerados por consultas contínuas.

Também é proposta uma estratégia de roteamento de pacotes que consiste no uso de uma árvore semântica de roteamento (ASR). Esta árvore permite à estação-base disseminar uma consulta através da rede e coletar dados enviados pelos nós-sensores em resposta à execução da consulta. A estratégia de uma ASR é permitir que um nó n_1, com base na avaliação dos valores dos atributos coletados, identifique quando ele e seus nós-filhos não estejam capacitados a responder à consulta. Neste caso, a consulta não é repassada à sub-árvore que tem n_1 como raiz, reduzindo os custos com a disseminação das consultas. Nesta estratégia, cada nó da rede mantém uma pequena lista dos nós com os quais comunicou-se mais recentemente. Desta forma é possível ao nó tomar conhecimento de qual é o seu nó-pai e quais são seus nós-filhos na árvore de roteamento a que pertence.

Na proposta do TinyDB, a adaptabilidade é estudada quanto aos seguintes aspectos:

(i) Contenção do canal de comunicação da rede: este aspecto é tratado através da monitoração quanto ao uso e contenção do canal de comunicação, de forma que o número de pacotes transmitidos seja reduzido, progressivamente, à medida que a contenção do canal aumenta. Porém, o uso desta estratégia poderá resultar no aumento das filas de pacotes armazenados nos nós-sensores, forçando o descarte dos pacotes menos relevantes. A relevância do pacote é avaliada com base na semântica dos valores nele contidos; e

(ii) Consumo de energia dos nós-sensores: este aspecto considera que a previsão de consumo de energia em um nó-sensor seja conhecida, para uma dada consulta e em um determinado intervalo de tempo, de forma que seja possível a comparação freqüente entre a energia gasta prevista

e a energia disponível. Caso a diferença se acentue, a quantidade de energia prevista para ser gasta é recalculada e uma nova estimativa de tempo de vida do nó-sensor é gerada.

3.4 Considerações finais

Este capítulo apresentou uma visão geral sobre o processamento de consultas em RSSFs. Primeiramente, foram descritas as características dos SFDs, bem como estratégias de adaptabilidade e aproximação de resultados, propostas no contexto de pesquisas acadêmicas voltadas para o processamento de fluxos de dados. Também foram abordados aspectos dos SFDs voltados para atender às necessidades específicas das aplicações de RSSFs como: formas de armazenamento temporário de dados, propostas de construção de modelos de dados, estratégias de processamento de consultas e algumas das linguagens de consultas declarativas desenvolvidas no contexto de importantes pesquisas acadêmicas. Os conceitos abordados neste capítulo serão utilizados como base para a especificação da estratégia de processamento distribuído de consultas em RSSFs, que é a contribuição do Capítulo 4 para esta dissertação.

Capítulo 4

UMA ESTRATÉGIA PARA O PROCESSAMENTO DISTRIBUÍDO DE CONSULTAS EM REDES DE SENSORES SEM FIO

4.1 Introdução

Conforme visto no Capítulo 2, uma arquitetura de RSSF pode assumir diferentes configurações, as quais influenciam diretamente no desempenho da rede e nos tipos de aplicações suportadas. Neste capítulo é apresentada, primeiramente, a topologia de rede considerada no escopo deste trabalho (Seção 4.2). A Seção 4.3 sugere a construção de um modelo genérico de dados, com o objetivo de prover uma visão lógica das aplicações executadas sobre RSSFs. Este modelo lógico de dados é explorado como base para a formulação de consultas declarativas escritas na linguagem SNQL (*Sensor Network Query Language*), proposta como uma extensão do padrão SQL, sendo especialmente voltada para atender às necessidades das aplicações de RSSFs [9]. Uma aplicação-exemplo é apresentada na Seção 4.4 e um detalhamento da SNQL é apresentado na Seção 4.5. Finalmente, a Seção 4.6 descreve o modelo de processamento de consultas considerado para o cenário de RSSF admitido neste trabalho.

4.2 RSSFs com topologia *Scale-free*

Uma arquitetura comum proposta para RSSFs é baseada na distribuição de nós-sensores em áreas geográficas, de maneira que os nós-sensores utilizem protocolos de roteamento do tipo múltiplos saltos para enviar os dados coletados a uma estação-base. Geralmente, estas propostas organizam os nós-sensores em árvores de roteamento [33][41][51].

O cenário de rede considerado neste trabalho consiste em uma RSSF de topologia *scale-free* [6]. Redes *scale-free* tendem a conter um subconjunto de seus nós, denominados *hubs*, conectados a um grande número de outros nós da rede. Observe que estes *hubs* são nós críticos, por influenciarem fortemente na forma como a rede opera. Por exemplo, a falha de um *hub* poderá prejudicar a

comunicação de todos os nós conectados a ele. Em Barabasi *et al*. [6] é mapeada a conectividade da *Web*, classificando-a como sendo um exemplo de topologia *scale-free* ao considerar cada página como um nó (*hub*) que leva a várias outras páginas, através de *hyper links*. A própria organização da sociedade humana ou mesmo a interação de proteínas em um organismo vivos são também considerados exemplos de uma configuração scale-free. Todos estes exemplos podem ser vistos como redes de nós que interagem uns com os outros através de alguns poucos *hubs* que conectam um vasto número de nós, os quais, por sua vez, possuem poucas conexões.

No caso de uma RSSF de configuração *scale-free*, alguns nós-sensores agem como nós-pais de um conjunto de outros nós-sensores. Desta forma, dados coletados são passados de um nó-sensor para o outro até eles alcançarem a estação-base, onde um processamento final é aplicado aos dados, e os resultados da consulta são produzidos para o usuário.

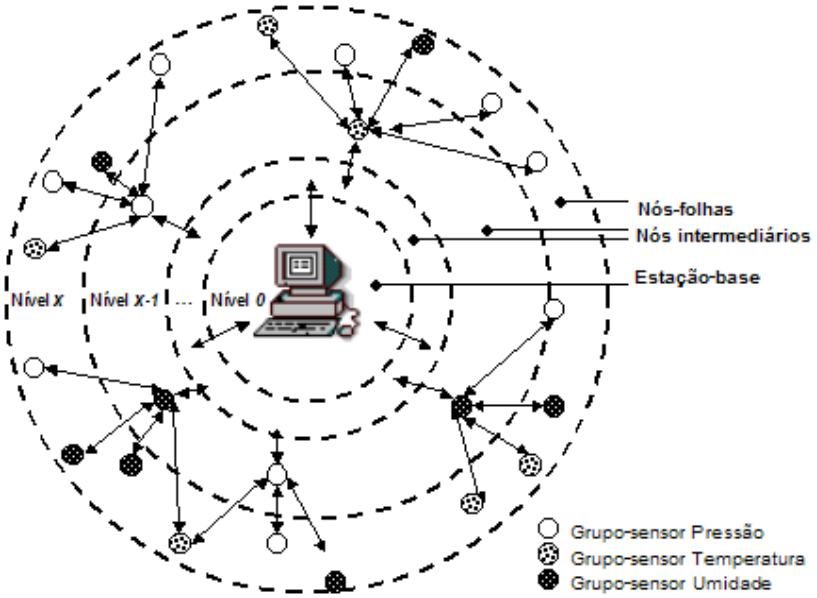


Figura 4.1 Arquitetura simplificada de uma RSSF de configuração *scale-free*.

A partir da rede *scale-free* considerada neste trabalho é formulada uma visão lógica da rede, a qual categoriza os nós-sensores por sua atividade de sensoriamento, por exemplo, coleta de valores de temperatura. Assim, um conjunto de nós-sensores com a mesma atividade de sensoriamento é denominado como sendo um grupo-sensor, por exemplo, grupo-sensor Temperatura [9]. Desta forma,

cada grupo-sensor é responsável por prover informações sobre algum fenômeno físico, tendo por base os dados coletados em determinadas regiões geográficas. Vale salientar que existem nós-sensores capazes de realizar *j* diferentes atividades de sensoriamento [32][49]. Neste caso, estes nós estarão categorizados em *j* diferentes grupos-sensores. Por simplicidade, a RSSF admitida neste trabalho considera uma única estação-base que se comunica com um conjunto de nós-sensores (Figura 4.1).

Observe que a RSSF apresentada na Figura 4.1 pode conter *x* níveis, dentro dos quais os nós da rede estão distribuídos: o nível 0, onde se encontra a estação-base; os níveis de 1 a *x*-1, onde se encontram os nós intermediários; e o nível *x*, onde estão os nós-folhas. A funcionalidade de cada um destes três tipos de nós é descrita a seguir [8]:

(i) Estação-base**:** Nó robusto quanto à capacidade de processamento, armazenamento e disponibilidade de memória. Situado no nível 0, este nó é responsável por organizar a hierárquia entre os nós da rede, distribuir consultas para os nós-sensores, receber pacotes de dados enviados pelos nós-sensores e retornar resultados para os usuários;

(ii) Nós intermediários: Distribuídos em uma área geográfica em níveis intermediários da rede (níveis 1 a *x*-1). Estes nós coletam dados do ambiente, recebem pacotes advindos de outros nós-sensores, além de processar, empacotar e enviar os dados para outros nós da rede; e

(iii) Nós-folhas: São os nós situados no último nível da rede (nível *x*). Por esta razão, suas funcionalidades incluem apenas a coleta de dados do ambiente, o processamento e o empacotamento destes dados para posterior envio aos seus nós mais próximos (seus nós-pais).

A escalabilidade da rede apresentada na Figura 4.1 pode ser conseguida através da comunicação entre diversas estações-base ou entre nós ligados a estações-base distintas, utilizando o meio de comunicação (Figura 4.2). Assim, uma RSSF pode ter vários pontos da rede (estações-base) através dos quais os usuários podem requisitar informações da rede. Convém salientar que quando um usuário escolhe um ponto da rede para submeter uma consulta q_1 ao sistema (aplicação de RSSF), este será o mesmo ponto que irá produzir o resultado final para q_1.

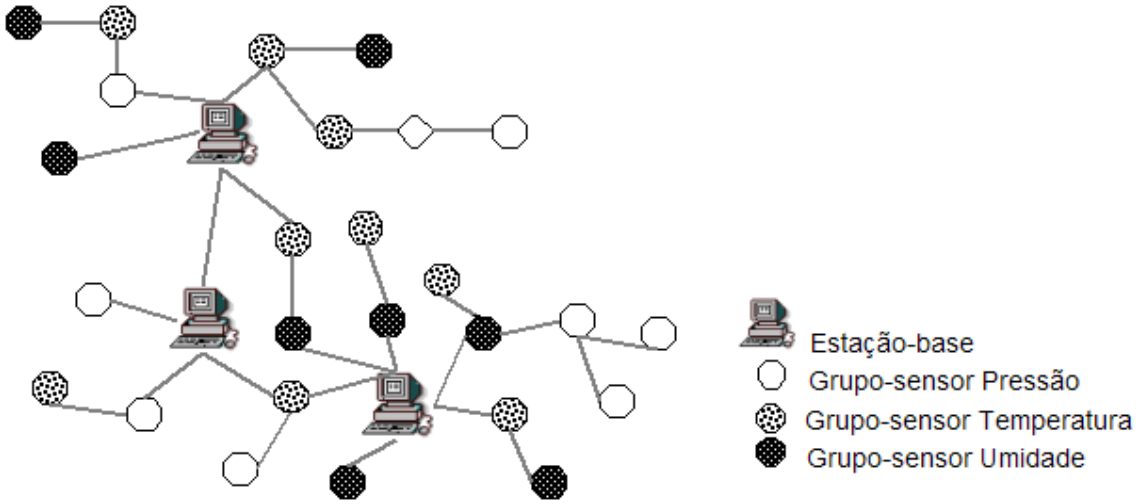


Figura 4.2 Generalização de uma RSSF com configuração *scale-free*.

Considera-se que as estações-base são conscientes da hierarquia da rede, o que significa que as informações sobre esta hierarquia devem ser regularmente atualizadas, objetivando refletir as possíveis mudanças na localização dos nós. Nesta abordagem, cada nó-sensor é capacitado a processar agregação em rede. Os dados processados pelos nós-sensores podem ser provenientes da coleta local, via atividade de sensoriamento, ou de pacotes de dados enviados por outros nós-sensores da rede. Considera-se, ainda, que dados locais e pacotes de dados que contenham informações a cerca de um mesmo fenômeno físico podem ser agregados, para compor um único pacote a ser enviado através da rede.

Observe que os nós intermediários de uma RSSF serão os nós-sensores mais penalizados quanto ao consumo de energia, pois estes nós não apenas coletam dados do ambiente como também recebem dados advindos de outros nós-sensores. Assim, o volume de dados a ser processado e enviado por estes nós tende a ser maior do que aquele manipulado pelos nós-folhas da rede. Como conseqüência, a bateria destes nós tem um desgaste mais rápido, o que influencia diretamente no tempo de vida da rede como um todo, visto que, no caso de falhas em nós intermediários, os nós-folhas provavelmente não terão um sinal suficientemente forte para alcançar a estação-base. Por esta razão, é muito importante que os algoritmos projetados para o processamento de dados em RSSFs sejam otimizados para reduzir o consumo de energia, particularmente, o custo com comunicação [39].

4.3 Modelo de dados proposto

Nesta seção é apresentado um modelo de dados genérico para aplicações de RSSFs. O objetivo, com o uso deste modelo, é permitir uma visão lógica sobre os fluxos de dados manipulados pelo sistema, de forma que os dados que fluem através da rede sejam “vistos” como tuplas de relações (virtuais). A vantagem no uso do modelo de dados proposto está em abstrair o usuário de detalhes físicos como [56]:

(i) identificar quais nós-sensores são relevantes para responder a uma dada consulta (por exemplo, uma consulta que envolve apenas os dados coletados em uma região geográfica específica);
(ii) determinar quais operadores de consulta podem ser aplicados aos dados nos nós-sensores ou apenas na estação-base; e
(iii) aplicar estratégias que visem reduzir o volume de dados transmitido através da rede (por exemplo, agregação em rede).

Além disso, usuários podem definir consultas declarativas baseadas no modelo de dados, da mesma forma como ocorre em aplicações de bancos de dados convencionais.

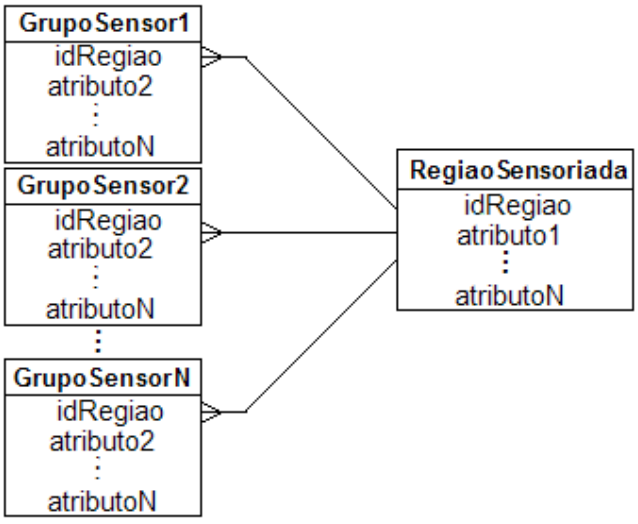


Figura 4.3 Modelo de dados para aplicações executadas sobre RSSFs.

O modelo de dados proposto captura as características de uma RSSF, de forma que cada fenômeno monitorado, considerado como sendo um grupo-sensor

(*GrupoSensor1*, *GrupoSensor2*, ..., *GrupoSensorN*), é representado por uma entidade específica, que tem atributos relacionados ao fenômeno monitorado. A Figura 4.3 ilustra o modelo de dados proposto.

As entidades representantes dos grupos-sensores são relacionadas através de uma entidade, denominada *RegiaoSensoriada*, a qual representa áreas geográficas consideradas para a coleta de dados. A entidade *RegiaoSensoriada* possui um atributo denominado *idRegiao*, que identifica uma área geográfica onde um subconjunto dos nós-sensores da rede coleta dados, bem como atributos que dão as coordenadas destas regiões. O atributo *idRegiao* é comum a todas as entidades, assim, ele permite que dados gerados por diferentes grupos-sensores sejam relacionados através da área geográfica (região) onde foram coletados. Vale salientar que cada região pode conter nós-sensores pertencentes a diferentes grupos-sensores.

4.4 Aplicação-exemplo

A aplicação considerada neste trabalho consiste no mapeamento da biocomplexidade ambiental de uma dada região geográfica. O objetivo desta aplicação é relacionar informações sobre temperatura, pressão e umidade, advindas de nós-sensores espacialmente distribuídos no ambiente. Estas informações são relacionadas através da região geográfica onde cada nó-sensor se encontra.

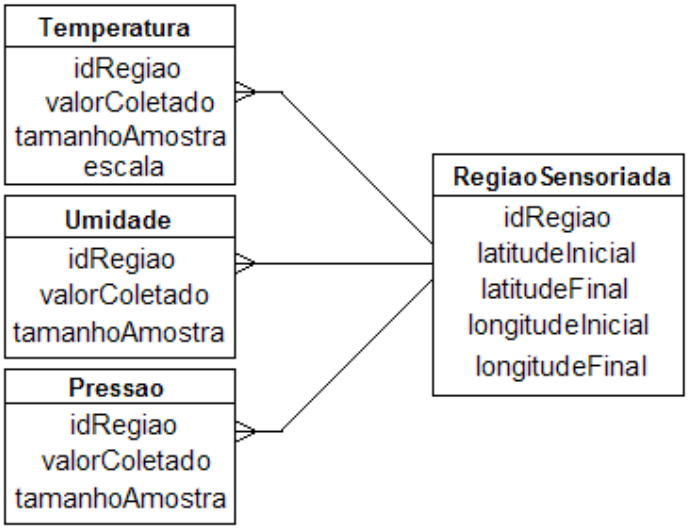


Figura 4.4 Modelo de dados - aplicação em biocomplexidade ambiental.

A Figura 4.4 ilustra a instanciação do modelo de dados proposto para a aplicação de mapeamento da biocomplexidade ambiental. Vale salientar que Temperatura, Umidade e Pressão são, de fato, tabelas virtuais, as quais representam os dados que fluem através da RSSF. Observe que cada entidade possui informações relacionadas ao fenômeno físico representado. Por exemplo, a relação Temperatura tem, como atributos, a região geográfica onde os dados são coletados (atributo *idRegiao*), o valor da temperatura mensurada (atributo *valorColetado*), o número de coletas realizadas (atributo *tamanhoAmostra*) e a escala de temperatura (Celsius ou Fahreneit) considerada (atributo *escala*). De forma similar, a relação *RegiaoSensoriada* tem atributos para representar as propriedades específicas de cada região.

Um exemplo clássico da necessidade de relacionar informações em uma aplicação de mapeamento da biocomplexidade ambiental é o índice de calor (índice de Temperatura-Umidade), que fornece a temperatura aparente ou sensação térmica do ambiente. O índice de calor relaciona valores de temperatura e umidade, com o propósito de estudar os riscos desta combinação de valores para a saúde, visto que, quanto maior este índice, maior será o risco. Por exemplo, se a temperatura registrada pelos sensores for superior a 30ºC e, ao mesmo tempo, um alto valor de umidade é medido, o valor do índice de calor poderá ser muito superior à temperatura registrada pelos nós-sensores de temperatura.

Para esta aplicação-exemplo, admite-se que todos os nós-sensores da rede têm disponibilidade inicial de recursos semelhante (capacidade de processamento, quantidade de memória e energia); e que a RSSF está organizada como uma rede de topologia *scale-free*, como mostrado na Figura 4.1.

4.5 SNQL – Uma linguagem de consultas para RSSFs

O uso de nós-sensores inteligentes representou uma grande evolução para as RSSFs. Dois fatores contribuíram particularmente para esta evolução. O primeiro foi a distribuição do processamento de dados entre os nós da rede. Este fator fez com que a rede deixasse de ser passiva, por utilizar os nós-sensores apenas para coleta e envio dos dados, passando a dividir o processamento de dados entre todos os nós da rede, inclusive os nós-sensores. O segundo fator foi a possibilidade de alterar, dinamicamente, configurações físicas do nó-sensor. Este último fator permitiu a

parametrização de variáveis capazes de informar ao nó-sensor, por exemplo, quando realizar a coleta de dados do ambiente. Esta técnica de passagem de parâmetros para os nós-sensores evoluiu para o uso das linguagens de consultas, especialmente projetadas para as aplicações de RSSFs. Algumas destas linguagens foram abordadas na Subseção 3.3.5.

Neste contexto, a linguagem de consultas SNQL (*Sensor Network Query Language*) é proposta neste trabalho como uma alternativa para a especificação de consultas declarativas em aplicações de RSSFs. As propriedades da linguagem SNQL são descritas a seguir [9]:

(i) Permite expressar consultas declarativas: partindo de um modelo de dados, que utiliza a noção de uma RSSF segmentada em grupos de nós-sensores, SNQL permite aos usuários a especificação de consultas declarativas (*ad-hoc*), construídas com base nas informações disponibilizadas na rede pelos nós-sensores;

(ii) Controle da precisão dos resultados e do volume de dados a ser coletado: SNQL possui cláusulas que permitem ao usuário controlar o tamanho da amostra a ser coletada em resposta a uma consulta, bem como o quão preciso será o resultado obtido a partir desta amostra;

(iii) Suporte a consultas contínuas: as consultas contínuas são especialmente aplicáveis a ambientes que trabalham com fluxos de dados. Quando uma consulta contínua é submetida à RSSF, a consulta é executada por um período de tempo indeterminado, sendo o resultado continuamente atualizado na estação-base e incrementalmente disponibilizado. Desta forma, a cada instante de tempo, uma nova posição sobre o resultado da consulta é fornecida ao usuário;

(iv) Suporte a consultas pré-definidas: consiste em re-submeter uma mesma consulta ao sistema por um número indeterminado de vezes. Considera-se que cada submissão resulta em uma execução, de tempo limitado, da consulta; o que é considerado como uma nova instância desta. Ao final do tempo-limite de execução de cada uma destas instâncias, os dados, até então materializados na estação-base, são descartados; e uma nova materialização de dados é iniciada, a partir da execução de uma nova instância da consulta. Observa-se que uma consulta contínua resulta em

uma única instância da consulta executada por tempo indeterminado, enquanto uma consulta pré-definida resulta em um número indeterminado de instâncias da consulta, cada uma sendo executada por um tempo limitado; e

(v) Trabalha com fragmentos de consultas: SNQL permite especificar fragmentos de código destinados a ajustar os valores das cláusulas de uma consulta, correntemente em execução, de acordo com a necessidade do usuário. Por exemplo, o usuário pode decidir que o resultado da consulta, até então disponibilizado, já é suficiente para se chegar a alguma conclusão. Desta forma, a consulta pode ser finalizada através do envio de um fragmento de consulta, que informa aos nós-sensores que a consulta, correntemente em execução, deve ser interrompida.

4.5.1 Especificação

A Tabela 4.1 apresenta a descrição das funcionalidades das cláusulas da SNQL.

Tabela 4.1 Especificação das cláusulas admitidas em SNQL.

Cláusulas[13]	Especificação
SELECT {<expr>}	<expr>: Especifica um subconjunto de atributos das relações consideradas.
FROM {<sensor group>}	<sensor group>: Define os grupos-sensores considerados.
[*WHERE* {<pred>}]	<pred>: Especifica um conjunto de predicados que filtram tuplas processadas.
[*GROUP BY* {<exprgroup>} [*HAVING* {<predhaving>}]]	<exprgroup>: Define um subconjunto de atributos nos quais funções de agrupamento devem ser aplicadas. <predhaving>: Especifica predicados, baseados nas funções de agrupamento, para filtrar resultados agregados.
TIME WINDOW <twseconds> | CONTINUOUS	<twseconds>: Define o intervalo de tempo durante o qual a consulta é válida para o sistema (por exemplo, 3600 s). O valor pré-definido *CONTINUOUS* especifica que a consulta tem um tempo de validade indeterminado.
[*DATA WINDOW* <dwnumrows>]	<dwnumrows>: Especifica a quantidade máxima de dados a ser coletada por cada nó-sensor (por exemplo, 50.000 coletas).
SEND INTERVAL <sndseconds>	<sndseconds>: Define o intervalo de tempo entre dois envios sucessivos de pacotes de dados (por exemplo, 120 s).
SENSE INTERVAL <snsseconds>	<snsseconds>: Especifica o intervalo de tempo entre duas coletas consecutivas de dados do ambiente pelos nós-sensores (por exemplo, 10 s).
[*SCHEDULE* <numexecutions> [{datetime}] | CONTINUOUS]	<numexecutions>: Define o número de vezes em que uma consulta deve ser submetida à rede e quando cada submissão deve ocorrer, por exemplo, <2 '10-oct-05 14:00:00', '15-oct-2005 14:00:00'>. O valor pré-definido *CONTINUOUS* especifica que a consulta tem que ser submetida à rede por um número indeterminado de vezes. A cada vez, uma nova instância da consulta é executada.

[13] Considera-se "{}" para denotar conjunto, "[]" para denotar cláusulas opcionais, "<>" para uma expressão, e "|" para denotar que um ou o outro *token* deve aparecer, mas não ambos.

Observe que as cláusulas *SELECT*, *FROM*, *WHERE*, *GROUP BY* e *HAVING* são definidas da mesma forma como no padrão SQL. Estas cláusulas, juntamente com as demais especificadas na SNQL, permitem que os usuários definam consultas declarativas. As cláusulas de uso específico em consultas para aplicações de RSSFs serão discutidas a seguir.

A cláusula *TIME WINDOW* especifica o período de tempo de validade da consulta. Em outras palavras, o intervalo de tempo em que os nós-sensores permanecerão coletando dados para responder à consulta, e também o intervalo de tempo em que a estação-base permanecerá recebendo pacotes de dados enviados pelos nós-sensores. Note que o valor pré-definido *CONTINUOUS* permite que o usuário defina uma consulta como sendo contínua. Já a cláusula *DATA WINDOW* especifica o número de coletas do ambiente a ser realizada por cada nó-sensor, para responder à consulta. Tanto a cláusula *TIME WINDOW* como a cláusula *DATA WINDOW* são úteis para definir o tamanho da amostra que será produzida em resposta a uma determinada consulta, porém, caso ambas sejam informadas na consulta, a cláusula *TIME WINDOW* terá prioridade.

A cláusula *SEND INTERVAL* define o intervalo de tempo entre envios sucessivos de pacotes de dados para outros nós da rede. O valor desta cláusula deve ser cuidadosamente definido pois, se o seu valor for muito alto, mais dados serão acumulados nos nós-sensores, o que aumenta as chances de situações de "estouro" de memória; porém, valores baixos podem produzir maiores quantidades de pequenos pacotes de dados, o que contribui para aumentar o tráfego da rede e também leva a um maior *overhead*, visto que mais cabeçalhos de pacotes serão necessários.

O intervalo de tempo entre coletas de dados sucessivas do ambiente é especificado na cláusula *SENSE INTERVAL*. Esta cláusula também deve ser cuidadosamente definida, visto que ela influencia diretamente na precisão dos resultados da consulta. Assim, se o seu valor é alto, menos dados serão coletados e os resultados serão menos precisos, pois diversos valores diferentes de medições podem ter ocorrido entre duas coletas de dados consecutivas. Por outro lado, se o seu valor for muito baixo, mais coletas de dados serão realizadas e, dependendo da função de agregação especificada na consulta, pode-se ter um maior acúmulo de

dados na memória do nó-sensor. Como conseqüência, tem-se que mais dados terão que ser processados e enviados através da rede, gerando um maior tráfego na rede. Desta forma, observa-se que os valores das cláusulas *SEND INTERVAL* e *SENSE INTERVAL* influenciam diretamente no desempenho da rede e no tempo de vida dos nós-sensores. Uma proposta para a realizar uma "calibração" dos valores destas cláusulas é explorada pelo algoritmo ADAGA, descrito em detalhes no Capítulo 5.

A cláusula *SCHEDULE* permite ao usuário definir o número de vezes e a periodicidade em que a consulta deve ser submetida ao sistema. Por exemplo, <2 '10-jan-06 14:00:00', '15-jan-2006 14:00:00'> significa que a consulta será executada duas vezes, na data e hora especificadas, o que resulta em duas instâncias da consulta. Caso o valor pré-definido *CONTINUOUS* seja associado a esta cláusula, assume-se que a consulta é do tipo pré-definida.

SNQL suporta a especificação de fragmentos de consultas para redefinir os valores das seguintes cláusulas (em tempo de execução): *TIME WINDOW, DATA WINDOW, SENSE INTERVAL*, *SEND INTERVAL* e *SCHEDULE*. Assim, o usuário pode submeter um fragmento de consulta que redefina, por exemplo, um intervalo de tempo menor entre captações sucessivas de dados do ambiente, objetivando aumentar a precisão dos resultados fornecidos para uma consulta correntemente em execução.

```
SELECT          r.IdRegiao AS RegiaoGeografica,
                MAX(t.valorColetado),  MIN(h.valorColetado)
FROM            Umidade h, Temperatura t, RegiaoSensoriada r
WHERE           r. IdRegiao = t. IdRegiao    AND r. IdRegiao = h. IdRegiao
AND             r.latitudeInicial    > 034308  AND r.latitudeFinal < 034316
AND             r.longitudeInicial > 383151  AND r.longitudeFinal < 383114
AND             t.valorColetado   > 20
AND             h.valorColetado  < 0.7
GROUP BY        r.IdRegiao
TIME WINDOW     3600
SEND INTERVAL   120
SENSE INTERVAL  10
SCHEDULE            2   '01-jan-06 14:00:00', '15-jan-06  14:00:00'
```

Figura 4.5 Exemplo de consulta escrita em SNQL.

Considere o exemplo de consulta descrito a seguir: verificar o maior valor de temperatura, associado ao menor valor de umidade, na área geográfica delimitada pelas coordenadas 3^o 43' 08" S - 38^o 31' 51" W e 3^o 43' 16" S - 38^o 31' 14" W. Além

disso, deve-se considerar apenas valores de temperatura superiores a 20^{o}C e valores de umidade inferiores a 70%. A consulta deve ser submetida ao sistema 2 vezes (em 01-jan-06 às 14:00:00 e em 15-jan-06 às 14:00:00), sendo que cada instância deve permanecer em execução por 3600 segundos. Os dados devem ser coletados do ambiente a cada 10 segundos e enviados através da rede a cada 120 segundos. A Figura 4.5 ilustra a representação desta consulta utilizando a notação da linguagem SNQL.

4.5.2 Suporte à agregação de dados

É muito importante que as linguagens de consulta para aplicações de RSSFs ofereçam suporte às funções de agregação. Quanto mais funções de agregação forem consideradas na especificação da linguagem, maiores serão as possibilidades de que a agregação em rede seja aplicável à resolução de uma consulta do usuário. A Tabela 4.2 mostra as funções de agregação previstas na especificação da SNQL. Observe que a contrapartida em se considerar uma maior diversidade destas funções é que a camada de aplicação da RSSF precisa tornar-se mais robusta, pois o suporte a estas funções envolve a implementação de cada uma delas.

Tabela 4.2 Funções de agregação admitidas na especificação da linguagem SNQL.

Funções	Especificação
COUNT()	Número de ocorrências dos valores de entrada não-nulos.
SUM()	Somatório de todos os valores de entrada.
MAX()	Maior valor entre todos os valores de entrada.
MIN()	Menor valor entre todos os valores de entrada.
AVERAGE()	Média aritmética de todos os valores de entrada.
VARIANCE()	Medida da dispersão estatística dos valores de entrada, indica o quão longe estes valores encontram-se do valor esperado.
STDDEV()	Corresponde à raiz quadrada da variância.
MEDIAN()[14]	É a medida de localização do centro da distribuição dos dados de entrada.
MODE()	Valor de entrada que ocorre com maior freqüência.

Para mostrar a relação entre o tamanho dos resultados parciais (produzidos em nós-sensores) e resultados finais (produzidos na estação-base) para diferentes funções de agregação, considere o cenário a seguir. Dois nós-sensores, n_1 e n_2, pertencem ao grupo-sensor *Temperatura*. Os nós n_1 e n_2 são capacitados a

[14] Para determinar a mediana deve-se ordenar os *y* elementos de entrada (amostra). Se *y* é ímpar, a mediana é o elemento localizado na posição central. Se *y* é par, a mediana é a semi-soma dos dois elementos centrais.

processar a agregação em rede, de acordo com a função de agregação especificada na consulta. Suponha que n_1 colete os valores de temperatura 20°C – 21°C – 22°C e gere o pacote p_1 com o resultado da função de agregação aplicada a estes valores, e que n_2 colete os valores de temperatura 21°C – 21°C – 22°C e gere o pacote p_2. Em seguida, os dados de p_1 e p_2 são também agregados na estação-base, onde o resultado será, finalmente, disponibilizado ao usuário. A Tabela 4.3 mostra os resultados das funções de agregação nos nós-sensores e na estação-base, conforme classificação sugerida em [24][41]. Observa-se que o tamanho dos pacotes gerados em n_1 e n_2 dependem da função de agregação utilizada.

Tabela 4.3 Resultados parciais obtidos para diferentes funções de agregação.

		Nó n_1	**Nó n_2**	**Estação Base**
Categoria ↓	Dados coletados →	20°C - 21°C - 21°C	21°C - 21°C – 22°C	*dados em p_1 e p_2*
Agregação distributiva	*COUNT()*	3	3	6
	SUM()	62	64	125
	MAX()	21°C	22°	22°C
	MIN()	20°C	21°	20°C
Agregação Algébrica	*AVERAGE()*	62 – 3	64 – 3	20,83 °C
	VARIANCE()	62 – 3	64 – 3	1
	STDDEV()	62 – 3	64 – 3	1
Agregação holística	*MEDIAN()*	20°C - 21°C - 21°C	21°C - 21°C - 22°C	21°C
	MODE()	20°C - 21°C - 21°C	21°C - 21°C - 22°C	21°C
		funções de agregação	*resultados parciais*	*resultado final*

No contexto das aplicações de RSSFs, para as funções de agregação distributiva, o tamanho do resultado parcial produzido em cada nó-sensor será o mesmo do resultado final produzido na estação-base. Por exemplo, a função *MAX()* irá produzir um resultado de mesmo tamanho, independente do local da rede onde for aplicada. Já as funções de agregação algébricas requerem que cada nó-sensor envie dois valores por pacote, correspondentes aos resultados parciais produzidos pelas funções *SUM()* e *COUNT()*, sendo que a função de agregação especificada na consulta (por exemplo, *AVERAGE())* apenas será computada na estação-base. Assim, o volume de dados transmitido através da rede, na agregação algébrica, será duas vezes maior que aquele gerado por agregações distributivas. As funções de agregação holística, por sua vez, não viabilizam a utilização da agregação em rede, visto que todos os dados coletados pelos nós-sensores devem ser enviados à estação-base, onde só então o resultado final poderá ser computado (por exemplo, a função *MEDIAN()*).

As estratégias de aproximação de resultados Histogramas, *Wavelets*, *Sliding Windows*, *Punctuations*, *Sketches* e *Sampling* (abordadas na Subseção 3.2.2) são alternativas para a sumarização de fluxos de dados, assim como as funções de agregação. SNQL já explora o conceito de *Sliding Windows* baseadas em tempo, através da cláusula *TIME WINDOW*.

4.5.3 Análise comparativa com outras linguagens

Embora as linguagens CQL (Subseção 3.3.5.2) e GSQL (Subseção 3.3.5.3) sejam extensões do padrão SQL aplicáveis à manipulação de fluxos de dados, estas linguagens não foram especialmente desenvolvidas para atender às necessidades das RSSFs. A Tabela 4.4 e a Tabela 4.5 comparam as linguagens de consulta aquisicional (LCA), SQTL e SNQL, as quais foram projetadas especificamente para RSSFs.

Tabela 4.4 Análise das propriedades suportadas em LCA, SQTL e SNQL.

Critério	LCA	SQTL	SNQL
Suporte a consultas contínuas	Sim	Sim	Sim
Suporte à agregação em rede	Sim	Parcialmente	Sim
Suporte ao processamento de junções	Sim	Parcialmente	Parcialmente
Reação a eventos[15]	Sim	Parcialmente	Sim
Utiliza pontos de materialização em todos os nós da rede	Sim	Parcialmente	Parcialmente
Ajuste dos valores das cláusulas da consulta (específicas para RSSFs) em tempo de execução	Parcialmente	Parcialmente	Sim

As consultas em LCA e SQTL são executadas continuamente até que o usuário ordene a sua parada, enquanto a linguagem SNQL fornece o valor pré-definido *CONTINUOUS* que, associado à cláusula *TIME WINDOW*, define uma consulta como sendo contínua.

Embora as três linguagens suportem agregação em rede, para a SQTL apenas alguns nós da rede são capacitados a agregar dados.

LCA oferece suporte ao processamento de junções em nós-sensores, enquanto SQTL e SNQL apenas prevêem o processamento de junções nas estações-base da rede, visto que as operações de junção podem vir a aumentar o volume dos

[15] A capacidade de reação a eventos precisa ser apoiada pelo sistema operacional utilizado no nó-sensor, visto que envolve a programação do hardware do equipamento.

resultados parciais, produzidos pelos nós-sensores, o que pode penalizar os recursos destes dispositivos.

As três linguagens suportam a reação a eventos externos ao ambiente. LCA define as cláusulas *ON EVENT* e *OUTPUT ACTION SIGNAL* com esta finalidade. SQTL é capaz de reagir a três eventos: recepção de mensagem (cláusula *RECEIVE)*, eventos disparados por um temporizador (cláusula *EVERY*) e eventos causados por expiração de tempo (cláusula *EXPIRE*). Já SNQL suporta o tratamento de eventos através da definição de predicados, especificados na cláusula *WHERE* da consulta.

LCA oferece suporte a pontos de materialização em nós-sensores e na estação-base, como forma de permitir o compartilhamento de dados entre consultas. SQTL considera que materializações, feitas por longos períodos de tempo, apenas devem ser realizadas na estação-base, tendo em vista as restrições de recursos dos nós-sensores. Embora SNQL permita que o usuário defina intervalos longos entre envios sucessivos de pacotes, o que resultaria em um volume maior de dados materializados no nó-sensor, ela não prevê o compartilhamento destes dados entre mais de uma consulta.

LCA e SQTL definem uma instrução que pode ser submetida à rede com a finalidade de interromper uma consulta em execução. Já SNQL permite alterar os valores das cláusulas, que são específicas para RSSFs (*TIME WINDOW*, *DATA WINDOW*, *SENSE INTERVAL*, *SEND INTERVAL* e *SCHEDULE*), através da submissão de fragmentos de consultas à rede. A Tabela 4.5. mostra um resumo das cláusulas suportadas pelas três linguagens aqui analisadas.

Tabela 4.5 Análise das cláusulas suportadas em LCA, SQTL e SNQL.

Funcionalidade das cláusulas	**Cláusula LCA**	**Cláusula SQTL**	**Cláusula SNQL**
Tempo de validade da consulta	Cláusula *FOR*	Não	*TIME WINDOW*
Volume máximo de dados coletados	Não	Não	*DATA WINDOW*
Intervalo entre coletas de dados	*SAMPLE PERIOD*	Não	*SENSE INTERVAL*
Intervalo entre envios de pacotes	Não	Não	*SEND INTERVAL*
Agendamento de consultas	Não	Não	*SCHEDULE*
Tratamento de eventos	ON EVENT e OUTPUT ACTION SIGNAL	*RECEIVE, EVERY e EXPIRE*	*WHERE*

4.6 Processamento de consultas

Geralmente, o processamento de consultas em RSSFs consiste em utilizar nós-sensores simples, que apenas coletam dados do ambiente e os enviam a uma estação-base, onde um banco de dados convencional processa os dados e os entrega ao usuário. Esta proposta traz como desvantagens o alto tráfego na rede e o alto consumo de energia requerido dos nós-sensores. A distribuição do processamento da consulta em RSSFs é uma alternativa para superar estes problemas.

O processamento de uma consulta pode ser visto como o conjunto de atividades envolvidas na extração de dados de um banco de dados. Neste trabalho, admite-se que o processamento de uma consulta ocorre de maneira distribuída na RSSF. Desta forma, cada nó-sensor é considerado tanto uma fonte remota de dados, como também um dispositivo capacitado a pré-processar dados. Considera-se que as atividades envolvidas no processamento de uma consulta estão distribuídas em quatro etapas distintas, análise, decomposição, processamento de fragmentos e pós-processamento, que têm suas execuções divididas entre nós-sensores e estação-base.

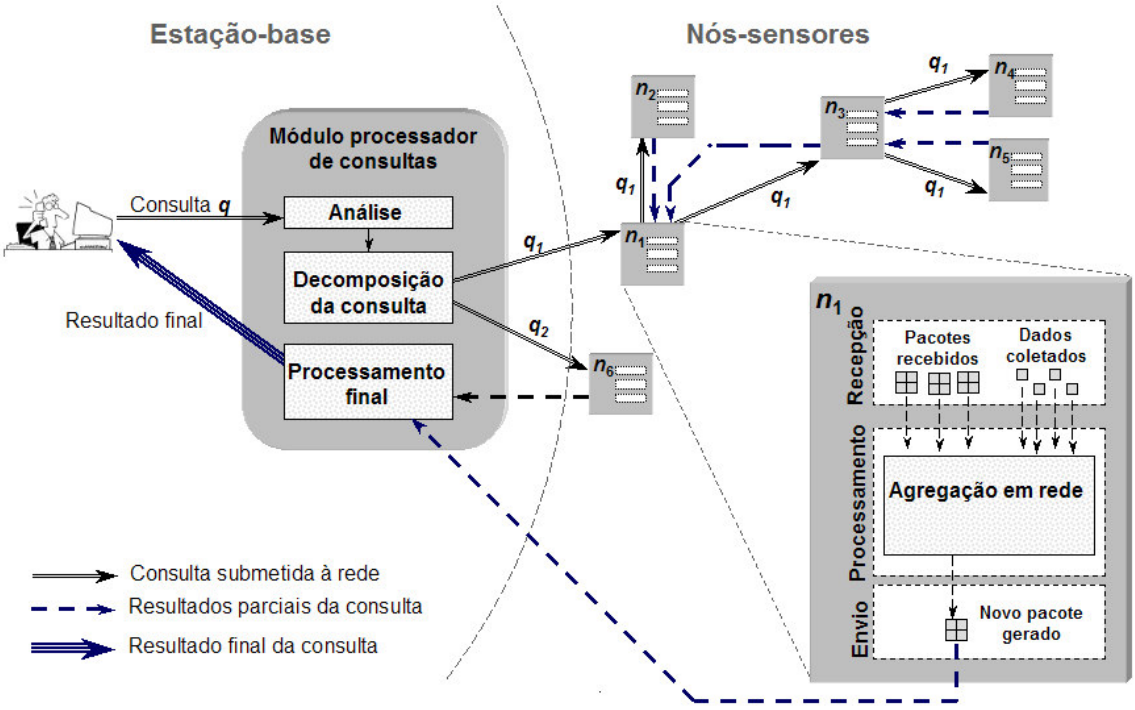


Figura 4.6 Estratégia admitida para o processamento de consultas em RSSFs.

A Figura 4.6 mostra um exemplo de como se dá o processamento de uma consulta no cenário de RSSF considerado. Quando o usuário submete uma consulta ao sistema (*q*), a consulta é analisada e decomposta nas sub-consultas q_1 e q_2, uma para cada grupo-sensor referenciado na cláusula FROM. Em seguida, as sub-consultas são enviadas aos nós-sensores n_1, n_2, n_3, n_4, n_5 e n_6. Após a recepção de uma consulta, as seguintes tarefas são executadas no nó-sensor: recebimento de pacotes de outros nós-sensores, coleta e processamento de dados, e envio de pacotes. A execução da consulta é finalizada no nó-sensor quando seu tempo de validade é alcançado. Na estação-base, a finalização da consulta envolve a suspensão do recebimento de pacotes de dados, a aplicação de algumas operações finais aos dados (por exemplo, filtragens e agregações de dados) e o fornecimento do resultado final ao usuário.

4.6.1 Análise

Nesta etapa, a consulta escrita em linguagem declarativa de alto nível, como SNQL, é analisada, objetivando verificar se a sintaxe utilizada está de acordo com as regras gramaticais da linguagem de consulta utilizada. Também é verificado se os nomes de atributos e relações, especificados na consulta, são válidos para o esquema do banco de dados, construído com base no modelo de dados (Seção 4.3). Finalmente, a consulta é convertida em uma árvore de análise, a qual é geralmente traduzida, no caso de sistemas de bancos de dados relacionais, para expressões da álgebra relacional. Como resultado desta etapa, um plano de execução da consulta é gerado. Observe que esta etapa é somente aplicada à estação-base, onde a consulta foi submetida pelo usuário.

4.6.2 Decomposição

Nesta etapa, o plano de execução, gerado na análise, é decomposto na estação-base, de acordo com os grupos-sensores especificados na consulta. Note que será gerada uma sub-consulta para cada grupo-sensor referenciado. Assim, uma consulta *q*, a qual relaciona dois grupos-sensores distintos g_1 e g_2, será decomposta em dois fragmentos, sub-consultas q_1 e q_2, correspondentes, respectivamente, aos grupos de sensores g_1 e g_2. A decomposição da consulta simplifica o trabalho dos nós-sensores, visto que um dado nó-sensor n_1 apenas processará o fragmento da consulta relacionado às informações que ele está capacitado a fornecer, poupando o

trabalho de interpretar cláusulas da consulta que não lhes dizem respeito. Para que isto seja possível, é necessário que o nó-sensor detenha o conhecimento sobre a parte do esquema da aplicação que modela os fenômenos tratados neste nó-sensor.

A decomposição é aplicada de forma diferenciada para os operadores de projeção, junção, seleção e agregação. Os atributos definidos na projeção serão decompostos em diferentes sub-consultas, de acordo com o grupo-sensor a que os atributos pertencem. Operações de junção não são previstas em uma sub-consulta, visto que esta apenas refere-se a um grupo-sensor. Operações de seleção também podem influenciar na decomposição da consulta. Por exemplo, uma consulta que envolva apenas um sub-conjunto das regiões geográficas cobertas pela RSSF pode ser dividida em sub-consultas direcionadas apenas a estas regiões. Tanto as operações de seleção quanto agregação são aplicáveis nos nós-sensores da rede, o que é conseguido com o uso de algoritmos baseados na técnica de agregação em rede.

Após a consulta inicial ser decomposta, cada fragmento é encapsulado em um pacote, cujo cabeçalho possui a informação necessária sobre o critério de decomposição, por exemplo, grupo-sensor ou localização do nó-sensor. Assim, quando um nó-sensor recebe um pacote, ele avalia o conteúdo do cabeçalho, como forma de identificar se deve aceitar ou simplesmente repassar a outros nós da rede a consulta ou o fragmento de consulta recebido.

4.6.3 Processamento de fragmentos

A execução desta etapa é distribuída entre os diversos nós-sensores da RSSF (veja Figura 4.6). Considerando uma consulta definida em SNQL, quando um nó-sensor recebe e aceita um fragmento de consulta, um temporizador *t* é iniciado e, em seguida, a coleta de dados tem início, obedecendo aos intervalos definidos na cláusula *SENSE INTERVAL*. O valor da cláusula *SEND INTERVAL* especifica quando as atividades do nó-sensor devem ser suspensas para que o envio e recepção de pacotes ocorra. Quando o nó-sensor não está coletando dados, ou recebendo e enviando pacotes, sua atividade consiste em processar os dados localmente armazenados, o que é feito através da agregação em rede. Finalmente, quando o temporizador *t* alcança o valor definido na cláusula *TIME WINDOW*, a coleta de dados e a recepção de pacotes é interrompida, e é feito o empacotamento

e envio dos dados correntemente armazenados no nó-sensor; o que encerra a execução da consulta neste nó. Uma estratégia para a execução dos passos da etapa de processamento de fragmentos é proposta no algoritmo ADAGA, descrito em detalhes no Capítulo 5.

4.6.4 Processamento final

Esta etapa é executada com a finalidade de preparar os dados para serem apresentados ao usuário. O processamento final consiste em executar:

(i) Operações de projeção, para reunir os atributos, especificados na consulta submetida pelo usuário, provenientes de diferentes grupos-sensores;
(ii) Operações de junção e/ou união, para relacionar os dados enviados por nós de diferentes grupos-sensores;
(iii) Operações de agregação, para reunir os dados advindos de diferentes nós-sensores; e
(iv) Outras operações, não processadas nos nós-sensores por apenas terem sentido no contexto da estação-base. Um exemplo clássico é a aplicação da cláusula *HAVING*, que especifica predicados aplicáveis ao resultado final, obtido após a agregação de todos os resultados parciais gerados para a consulta. Neste caso, apenas a estação-base é capaz de processar a operação definida na cláusula *HAVING*.

O último passo desta etapa é disponibilizar o resultado final da consulta para o usuário. Uma alternativa proposta para a junção de dados provenientes de diferentes grupos-sensores é proposta pelo algoritmo ADAPT, descrito em maiores detalhes no Capítulo 6.

4.7 Considerações finais

Este capítulo descreveu a estratégia de processamento de consultas para RSSFs admitida neste trabalho, incluindo a topologia de rede e as etapas do processamento de consultas previstas. Também foi proposto um modelo de dados genérico que captura as características de uma RSSF para representar os fenômenos físicos tratados na aplicação e suportados pela rede. Adicionalmente, foi proposta a linguagem SNQL como alternativa para a especificação de consultas declarativas

para aplicações de RSSFs, a qual especifica cláusulas que dão suporte aos algoritmos ADAGA e ADAPT, descritos em detalhes no Capítulo 5 e no Capítulo 6.

Capítulo 5

ADAGA - AGREGAÇÃO DE DADOS EM RSSFs

5.1 Introdução

O objetivo deste capítulo é descrever a proposta de ADAGA [9], um algoritmo adaptativo para o processamento de agregação em rede nos nós-sensores inteligentes de uma RSSF. Este algoritmo é caracterizado pela adoção de duas estratégias. A primeira é a adaptabilidade, que permite ao algoritmo ajustar seu comportamento à ocorrência de eventos como, por exemplo, limitações de recursos. A segunda é a capacidade de processar a agregação em rede, o que contribui para reduzir o volume de dados enviado através da rede.

Este capítulo está dividido conforme descrito a seguir. A Seção 5.2 descreve as propriedades de ADAGA e a Seção 5.3 descreve o funcionamento do algoritmo. A Seção 5.4 apresenta a estratégia para o monitoramento de recursos do nó-sensor. A Seção 5.5 descreve a estratégia de roteamento de pacotes admitida e a Seção 5.6 avalia os resultados produzidos pelo ADAGA.

5.2 Propriedades do algoritmo ADAGA

O algoritmo ADAGA (*ADaptive AGgregation Algorithm for wireless sensor networks*) é projetado para processar os dados coletados do ambiente por nós-sensores inteligentes de uma RSSF. Este processamento consiste em aplicar operações de agregação em rede, como forma de reduzir o volume de dados armazenados nos nós-sensores e o número de pacotes enviados através da rede. ADAGA também explora técnicas de adaptabilidade objetivando ajustar seu comportamento de acordo com a disponibilidade de energia e memória. O objetivo é maximizar a precisão dos resultados da consulta em situações de restrição de recursos. Em outras palavras, busca-se um resultado próximo ao resultado exato, o qual seria obtido se não houvesse limitações de memória e energia.

O algoritmo ADAGA apresenta as seguintes propriedades:

(i) Processamento de agregação em rede: Conforme descrito no Capítulo 2, esta técnica reduz a quantidade de dados transmitida pelos nós-sensores,

o que, conseqüentemente, reduz o consumo de energia e o tráfego de dados na RSSF. O objetivo é reduzir o número e o volume dos pacotes a serem enviados a outros nós da rede, o que também leva a uma menor ocupação da memória do nó-sensor;

(ii) Monitoramento do uso de energia e memória: Uma estratégia comum em aplicações de RSSFs consiste em interromper a coleta de dados quando o espaço livre em memória é esgotado. Uma situação pior ocorre quando um nó-sensor não tem mais energia disponível para o seu funcionamento. Neste caso, todos os dados coletados, e correntemente armazenados na memória do nó-sensor, são perdidos. O algoritmo ADAGA monitora o uso de energia e memória objetivando ajustar seu comportamento à disponibilidade destes recursos, através da redução das atividades internas realizadas pelo nó-sensor. Em outras palavras, a estratégia adotada consiste em escalar progressivamente os valores das cláusulas *SENSE INTERVAL* e *SEND INTERVAL,* especificadas em uma consulta SNQL, de acordo com a disponibilidade de energia e memória;

(iii) Melhor aproximação de resultados: O aumento do valor das cláusulas *SENSE INTERVAL* e *SEND INTERVAL* reduz as atividades de sensoriamento e processamento do nó-sensor. Conseqüentemente, menos energia é gasta e a sobrevida dos nós-sensores aumenta. Outro benefício é que a coleta de um menor volume de dados pode também levar a uma ocupação mais lenta da memória do nó-sensor. Por outro lado, realizar a consulta sobre uma amostragem menor dos dados reduz a precisão dos resultados finais. ADAGA estima o valor das coletas de dados não realizadas com base nos resultados até então coletados, como forma de produzir resultados tão aproximados quanto possível dos resultados exatos; e

(iv) Tolerância a falhas: Visto que os nós-sensores são suscetíveis a falhas, é importante que os pacotes enviados através da rede tenham mais de uma opção de caminho para chegar até a estação-base. O algoritmo ADAGA permite a reprodução de pacotes (geração de cópias) para aumentar a tolerância a falhas do sistema. Porém, um problema advindo desta estratégia é a repetição de dados, provenientes de cópias de um mesmo pacote, no resultado. Para evitar este efeito colateral da reprodução de

pacotes, ADAGA elimina progressivamente as cópias dos pacotes, à medida em que elas se aproximam da estação-base.

5.3 Estágios do algoritmo ADAGA

Basicamente, nós-sensores inteligentes realizam a coleta e processamento de dados, e a recepção e envio de pacotes. Para dar suporte a estas tarefas, ADAGA é projetado para ser executado em cinco estágios. Estes estágios precisam ser executados seqüencialmente, pois a maior parte dos nós-sensores não é capaz de realizar tarefas em paralelo (por exemplo, Mica Motes – Subseção 2.2.2.3). Outros dispositivos-sensores, como o μAMPS (Subseção 2.2.2.2), são capazes apenas de receber e enviar dados simultaneamente, porém, não suportam a execução de outras operações em paralelo.

O algoritmo proposto usa três estruturas lógicas (listas encadeadas) para armazenar os dados temporariamente:

(i) uma área de recepção, que armazena pacotes recebidos;
(ii) uma área de processamento, onde os dados a serem processados são armazenados; e
(iii) uma área de envio, onde são colocados os pacotes a serem transmitidos.

Observe que estas estruturas não penalizam a memória, visto que cada dado apenas estará armazenado em uma destas estruturas lógicas por vez. A forma como estas estruturas lógicas são exploradas pode ser melhor entendida através da descrição dos cinco estágios do ADAGA, apresentada a seguir (Figura 5.1).

(i) Estágio 1*:* Administra a execução dos outros quatro estágios do algoritmo, agindo como a rotina principal. Basicamente, existe uma seqüência de laços aninhados. O primeiro laço (linha 1) é executado tantas vezes quanto forem definidas no valor da cláusula *SCHEDULE* da consulta. O segundo laço (linha 3) especifica que cada consulta deve ser executada enquanto esta for válida (considerando o tempo de validade da consulta definido na cláusula *TIME WINDOW*). O segundo laço ainda define quando devem ser realizados a recepção e o envio de dados, de forma que a cada nova iteração deste laço, pacotes são recebidos (linha 15) e armazenados na área de recepção; enquanto outros, na área de envio,

são transmitidos (linha 16). O terceiro laço (linha 6) envolve as ações de sensoriamento e processamento de dados. O intervalo de tempo para cada nova iteração do terceiro laço é definido pelo valor da cláusula *SEND INTERVAL*;

(ii) Estágio 2: Este estágio é responsável por processar a agregação em rede dos dados temporariamente armazenados na área de processamento. Estes dados podem ser provenientes da coleta de dados do meio ou da recepção de pacotes, enviados por outros nós-sensores. Após recuperar os dados da área de processamento (linha 1), os dados são submetidos aos filtros (definidos na cláusula *WHERE*) e às funções de agregação (definidas na cláusula *GROUP BY*) especificadas na consultas pela aplicação da função agg(ds_1) (linha 2), sendo o resultado ds_2 novamente armazenado na área de processamento (linha 3);

(iii) Estágio 3: responsável pelo monitoramento do uso de energia e memória no nó-sensor. Primeiramente, este estágio dispara a execução do procedimento *adaptSendInterval* (linha 6), que ajusta o valor da cláusula *SEND INTERVAL*, de acordo com a disponibilidade de energia, visto ser este o recurso mais crítico para o nó-sensor. Em seguida, o procedimento *adaptSenseInterval* (linha 7) é executado para adaptar o valor da cláusula *SENSE INTERVAL* à disponibilidade de memória. Este estágio é disparado tanto pelo Estágio 1 (Figura 5.1- linha 13) como pelo Estágio 4 (Figura 5.1- linha 2);

(iv) Estágio 4: Seu objetivo é recepcionar pacotes enviados por outros nós da rede. Neste estágio, os pacotes de dados recebidos são temporariamente armazenados na área de recepção (linha 3). Para cada pacote p_1, na área de recepção de um dado nó n_1, se p_1 é originário de nós pertencentes ao mesmo grupo-sensor de n_1 (linha 4), p_1 é analisado e seus dados são armazenados na área de processamento (linha 6). Caso contrário, p_1 é armazenado na área de envio (linha 9); e

(v) Estágio 5*:* Responsável por enviar pacotes a outros nós da rede. Cada vez que o intervalo de tempo de envio é alcançado, as demais atividades do nó-sensor são suspensas e o Estágio 5 é chamado. Neste estágio, os pacotes armazenados na área de envio são examinados e, em seguida, transmitidos a outros nós da rede. Para cada pacote p_1 na área de envio

(linha 1), considere os seguintes valores: *c*, o número de cópias de p_1 (registrado no cabeçalho de p_1); *l*, o número de cópias locais de p_1, correntemente armazenadas em um dado nó n_1 (linha 4); e $nc = (c–l+1)$ (linha 6). Se $nc > 1$, p_1 é enviado apenas para um dos nós-pais de n_1, escolhido aleatoriamente (linha 9), da mesma forma como acontece na proposta *Gossiping* [28]. Porém, se $nc = 1$, p_1 é enviado para todos os nós-pai de n_1 (linha 12).

Estágio 1: Adaga (Sensor n_1, Consulta *q*)

```
1:  Para i = 1 até nº. de execuções da consulta faça
2:    inicialize o temporizador tempoTimeWindow;
3:    Enquanto tempoTimeWindow < q.TimeWindow
4:      inicialize o temporizador timerSendInterval;
5:      n1.SenseInterval ← q.SenseInterval
6:      Enquanto tempoSendInterval < q. SendInterval do
7:        inicialize o temporizador timerSenseInterval;
8:        Se tempoSenseInterval < q. SenseInterval então
9:          d = ColeteDadoAmbiente();
10:         inclua d em um pacote da área de processamento;
11:       Fim se
12:       processarDado;
13:       monitorarRecursos(n1,q);
14:     Fim enquanto
15:     ReceberPacotes(Sensor n1);
16:     EnviarPacotes(Sensor n1);
17:   Fim enquanto
18: Fim para
```

Estágio 2: ProcessarDado()

```
1: ds1 ← carregar dados da área de processamento;
2: ds2 ← agg(ds1);//agregar dados do conjunto de dados ds1
3: armazenar ds2 na área de processamento;
```

Estágio 3: MonitorarRecursos(Sensor n_1, Consulta *q*)

```
1: x1 ← s.carregueEnergiaDisponivel ();
2: x2 ← s.carregueMemóriaDisponivel();
3: v  ← q.SendInterval;
4: t  ← q.TimeWindow;
5: n1 ← q.SenseInterval;
6: n1.SendInterval  ← adapteSendInterval(x1,v,t);
7: n1.SenseInterval ← adapteSenseInterval(x2,s,v);
```

Estágio 4: ReceberPacotes(Sensor n_1, *Consulta q*)

```
1:  Para cada pacote na área de recepção faça
2:    monitorarRecursos(n1,q);
3:    p1 ←  carregar pacote da área de recepção;
4:    Se grupo sensor de p1 = n1.grupoSensor então
5:      Se número de cópias no cabeçalho de p1 =1 então
6:        área de processamento ← p1
7:      fim se
8:    senão
9:      área de envio ← p1;
10:   fim se
11: fim para
```

Estágio 5: EnviarPacotes(Sensor n_1)

```
1:  Para cada pacote na área de envio faça
2:    p1 ← carregar pacote da área de envio;
3:    c ← número de cópias de p1 no seu cabeçalho;
4:    l ← carregar o número de cópias locais de p1;
5:    descartar l –1 cópias de p1;
6:    nc ← c - l;
7:    registrar nc no cabeçalho da cópia de p1 mantida;
8:    Se nc > 1 então
9:      n2 ← nó pai de n1 escolhido aleatoriamente;
10:     envie p1 para s´;
11:   senão
12:     envie p1 para todos os nós pai de n1;
13:   fim se
14:   descarte p1 da área de envio;
15: fim para
```

Figura 5.1 Estágios do algoritmo ADAGA.

A estratégia para tratamento das cópias dos pacotes, brevemente descrita no Estágio 5, será explorada em maiores detalhes na Seção 5.5.

5.4 Monitoramento de recursos dos nós-sensores

A materialização de dados em nós-sensores por longos períodos de tempo pode causar dois efeitos negativos. Primeiro, usuários podem ter que esperar muito tempo até que os resultados parciais de uma consulta sejam produzidos. Segundo, as chances de ocorrerem estouros de memória aumentam, levando a perdas de dados.

Por outro lado, curtos períodos de tempo entre dois envios sucessivos de pacotes podem resultar em gasto desnecessário de energia. Este fato ocorre porque um número maior de pacotes tende a ser enviado, sendo conseqüentemente requerida uma maior quantidade de cabeçalhos de pacotes. Além disso, através da diminuição dos intervalos de tempo entre envios de pacotes, os benefícios da agregação em rede são reduzidos, visto que menos dados são agregados antes do envio dos pacotes.

A Figura 5.2 ilustra um exemplo no qual uma consulta *q,* cujo objetivo é detectar medições distintas de temperatura, especifica que um nó-sensor, n_1*,* deve coletar dados a cada 5 segundos, por 80 segundos, e enviar pacotes para outros nós a cada 40 segundos. Neste caso, n_1 envia dois pacotes, p_1*,* com 4 valores inteiros, e p_2*,* com 2 valores inteiros. Um outro nó-sensor n_2 recebe uma consulta semelhante, porém com o valor do intervalo entre envios igual a 80 segundos. n_2 gera apenas um pacote, p_3*,* com 4 valores inteiros. Observa-se, nesta situação, que o consumo de energia e memória em n_2 foi menor do que em n_1, visto que um menor volume de dados foi transmitido por n_2*.*

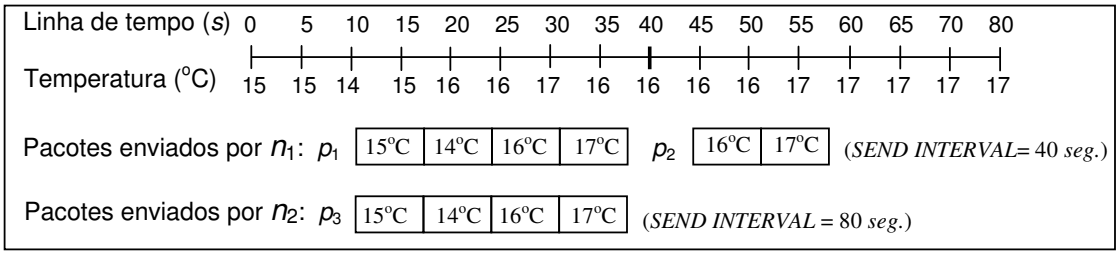


Figura 5.2 Agregação em rede utilizando dois diferentes valores de *SEND INTERVAL*.

Diferentes nós-sensores podem ter diferentes disponibilidades de recursos, o que é causado por fatores como:

(i) As consultas submetidas à rede podem ser enviadas para apenas um subconjunto dos seus nós, de modo que alguns nós-sensores trabalham mais do que outros;
(ii) Diferentes volumes de dados podem ser coletados em cada nó-sensor;
(iii) O papel de um nó-sensor na rede, definido pela sua localização, influencia em quanto ele trabalha (por exemplo, nós intermediários trabalham mais do que nós-folha); e
(iv) A mobilidade e a suscetibilidade a falhas dos nós-sensores podem mudar o papel de um nó-sensor na rede (por exemplo, um nó intermediário pode assumir o papel de nó-folha e vice-versa).

Geralmente, RSSFs têm um grande número de nós-sensores, o que torna muito difícil para os usuários determinarem os melhores valores para os intervalos de tempo entre coletas e entre envios de dados, a serem aplicados em todos os nós-sensores envolvidos em uma consulta. Desta forma, para atender às particularidades de cada nó-sensor, é importante que os algoritmos desenvolvidos para estes nós sejam auto-configuráveis.

Com a finalidade de tornar os nós-sensores auto-configuráveis, ADAGA age de maneira pró-ativa, monitorando a disponibilidade de recursos e dinamicamente ajustando a freqüência das atividades realizadas pelos nós-sensores. O objetivo é aumentar o tempo de vida da rede, bem como a precisão dos resultados da consulta, obtidos em situações de restrição de energia e memória. Duas estratégias são utilizadas em ADAGA: (*i*) ajustes do intervalo de tempo entre envios (*SEND INTERVAL*) e (*ii*) ajustes do intervalo de tempo entre coletas de dados (*SENSE INTERVAL*).

A primeira estratégia tem como foco a adaptação (ajuste) do intervalo de tempo entre envios e recepções de pacotes à disponibilidade de energia do nó-sensor. Aproximadamente 50% da energia gasta em um nó-sensor é devido às operações de comunicação (envio e recepção) [39]. Tendo em vista que a energia é um recurso crítico, ADAGA propõe retardar o envio dos pacotes, em situações de restrição de energia. Em outras palavras, ADAGA incrementa o valor do intervalo de tempo entre envios buscando reduzir a freqüência de transmissão dos pacotes, como forma de

poupar a bateria do nó-sensor. Esta estratégia ajusta o valor atribuído à cláusula *SEND INTERVAL* da consulta, de acordo com a função *f(v)* expressa a seguir.

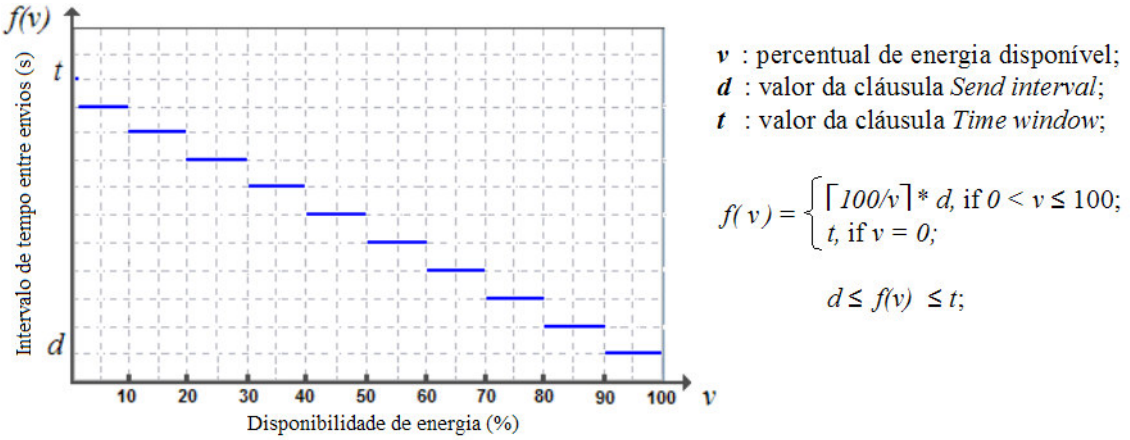


Figura 5.3 Função de ajuste do valor de *SEND INTERVAL* à disponibilidade de energia.

A Figura 5.3 mostra como o valor da cláusula *SEND INTERVAL* é incrementado. A função *f(v)* varia de um mínimo, que corresponde ao valor *d*, definido na cláusula *SEND INTERVAL*, até um máximo, que é o valor especificado para a cláusula *TIME WINDOW* (*t*). Note que se a disponibilidade de energia estiver próxima a 100%, *f(v)* retorna o valor *d*. Todavia, à medida que a energia é consumida, o valor resultante de *f(v)* é progressivamente aumentado. Assim, o envio de pacotes é adiado por $\lceil 100/v \rceil$ unidades de tempo, onde $\lceil 100/v \rceil \in N^*$, com $1 \leq \lceil 100/v \rceil \leq$ t/d. Finalmente, se a disponibilidade de energia estiver próxima a 0%, *f(v)* assume o valor *t*, o que significa que o nó-sensor não irá receber ou enviar mais pacotes.

Observe que *f(v)* não pode ser uma função contínua, visto que se supõe que os nós-sensores enviem e recebam pacotes em períodos de tempo pré-definidos (períodos conhecidos por todos os nós-sensores participantes da execução da consulta), o que garante um mínimo de sincronismo entre os nós. Este sincronismo refere-se ao fato de que é importante que um dado nó-sensor n_1 esteja escutando o meio (canal de comunicação), quando um outro nó n_2 estiver enviando pacotes para n_1. O sincronismo reduz as chances de perdas de pacotes. Por outro lado, é importante notar que, como os nós-sensores operam de maneira independente, seus relógios internos podem não estar ou permanecer sincronizados com os demais nós.

Neste trabalho, os atrasos relacionados à sincronização entre os relógios dos nós são desconsiderados. Em [62] é apresentada uma discussão aprofundada sobre os aspectos referentes a tempos de sincronização em RSSFs.

A segunda estratégia consiste em ajustar o intervalo de tempo entre coletas de dados do ambiente (*SENSE INTERVAL*) com base na avaliação da disponibilidade de memória. O objetivo é reduzir progressivamente a atividade do nó-sensor, através da diminuição do número de coletas de dados do ambiente, à medida que a disponibilidade de memória torna-se menor. Como conseqüência, menos processamento de dados é requerido, o que possivelmente levará a uma menor taxa de ocupação da memória. Além disso, o consumo de energia associado à atividade de sensoriamento também é reduzido. Assim, esta estratégia ajusta o valor da cláusula *SENSE INTERVAL* da consulta conforme a função a seguir.

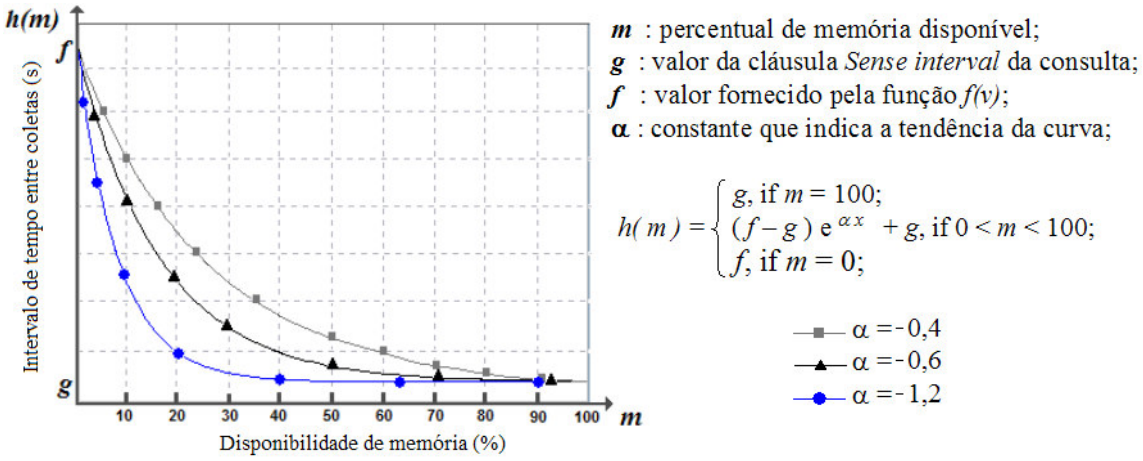


Figura 5.4 Função de ajuste do valor de *SENSE INTERVAL* à disponibilidade de memória.

A função *h(m)* varia de um mínimo, o valor da cláusula *SENSE INTERVAL*, até o seu máximo, que é o valor do intervalo de tempo entre envios sucessivos de pacotes (inicialmente definido na cláusula *SEND INTERVAL*), que é resultante da função *f(v)* apresentada anteriormente.

A Figura 5.4 mostra como o valor da cláusula *SENSE INTERVAL* escala. O gráfico mostra o comportamento da função *h(m)* utilizando três diferentes valores para o fator de ajuste α (constante que define a tendência da curva). Observa-se que para um valor de α = -0,4, o valor, inicialmente especificado pelo usuário, para a cláusula *SENSE INTERVAL,* é rapidamente alterado pelo algoritmo; o que pode não

ser desejável, visto que o tamanho da amostra poderá ser bem diferente daquele que o usuário pretendia obter. Por outro lado, para um valor de α = -1,2, o valor da cláusula *SENSE INTERVAL* apenas começa a ser ajustado quando já quase 80% da memória foi consumida, o que já pode ser uma disponibilidade crítica de memória. Pelos testes realizados, foi encontrado um equilíbrio maior com um valor de α = -0,6, de forma que os ajustes do intervalo de tempo entre coletas sucessivas de dados só começam a ser feitos após 50% da memória ter sido ocupada.

Observe que, para um valor de α = -0,6, a função *h(m)* permanece relativamente constante entre 50% e 100% de memória disponível. Neste caso, o nó-sensor coleta dados segundo o intervalo de tempo definido na cláusula *SENSE INTERVAL* da consulta. Todavia, à medida que a memória vai sendo consumida, o valor do intervalo de tempo entre coletas de dados sucessivas, adotado pelo nó-sensor, é progressivamente aumentado. Quando a disponibilidade de memória atinge 0%, o valor do intervalo de tempo entre coletas assume o mesmo valor do intervalo de tempo entre envios sucessivos de pacotes, o que significa que nenhuma nova coleta será realizada até que o próximo envio de pacotes ocorra.

É importante observar também que quanto maior o valor de *h(m)*, menor será o tamanho da amostra produzida para atender à consulta, o que, conseqüentemente, pode levar a uma menor precisão dos resultados da consulta. Para reduzir o impacto deste efeito colateral, ADAGA estima o valor das coletas não realizadas (devido ao aumento do valor do intervalo de tempo entre coletas sucessivas de dados) com base em valores de coletas feitas anteriormente, objetivando reduzir a distância entre os resultados gerados para a consulta e os resultados exatos.

Vale salientar que, quando os pacotes de um nó-sensor são enviados através da rede e a memória fica com espaço livre novamente, o intervalo de tempo de coleta de dados retorna ao seu valor inicial (valor definido na cláusula *SENSE INTERVAL* da consulta).

Quando o intervalo de tempo entre coletas de dados é ajustado para um valor superior ao dobro do valor da cláusula *SENSE INTERVAL* definido na consulta, o algoritmo ADAGA começa a inferir os valores das detecções não realizadas. Uma aproximação simples pode ser feita. Primeiramente, calcula-se, para cada intervalo entre coletas de dados, o número de detecções não realizadas (*u*) devido ao ajuste

do valor da cláusula *SENSE INTERVAL.* O próximo passo consiste em computar *u* detecções de um valor estimado de coleta. Este valor estimado pode consistir apenas em assumir o mesmo valor da última coleta realizada do ambiente. Outras estratégias mais complexas podem ser exploradas como forma de melhorar a aproximação dos resultados estimados como, por exemplo, média ponderada ou média dos valores coletados.

O algoritmo ADAGA é pró-ativo no sentido de que ele tenta evitar que problemas de *overflow* de memória aconteçam, bem como tenta aumentar a expectativa de vida dos nós-sensores, à medida que a energia é consumida. Propostas comuns de agregação em rede agem de forma reativa, não sendo capazes de anteciparem-se aos problemas dos nós-sensores. Estas propostas apenas interrompem as atividades do nó-sensor quando ocorrem *overflows* de memória ou esgotamento de energia. Em situações de restrições de recursos, ADAGA pode produzir resultados mais próximos dos resultados exatos do que as propostas que agem de forma reativa, conforme é mostrado na Seção 5.6.

5.5 Estratégia de roteamento de pacotes

Os nós intermediários de uma RSSF, geralmente, armazenam mais dados empacotados do que os nós-folhas. Desta forma, falhas nestes nós são mais críticas para os resultados da consulta. Objetivando reduzir o impacto da perda de pacotes no caso de falhas em nós-sensores (principalmente, nós intermediários), deveriam ser dadas rotas alternativas aos pacotes. Em outras palavras, permitir que um nó-sensor envie um mesmo pacote para alguns dos nós-sensores próximos a ele (considerados seus nós-pais), aumentando, assim, as chances de um pacote alcançar a estação-base. Todavia, esta reprodução deve ser limitada, para que não resulte no problema conhecido como implosão [31], enfrentado por protocolos como o *Flooding* (Subseção 2.3.8.1). Claramente, a implosão resulta em um maior consumo de energia devido a maior quantidade de pacotes transmitidos através da rede. Além disso, ela pode vir a afetar a precisão dos resultados, se as inúmeras cópias de um mesmo pacote não forem eliminadas, antes de ser computado o resultado final da consulta.

A eliminação de dados duplicados devido à reprodução de pacotes torna-se ainda mais difícil de resolver quando se considera o uso da agregação em rede. A

Figura 5.5 ilustra o exemplo de uma consulta cujo objetivo é receber dos nós-sensores valores distintos das temperaturas coletadas e a quantidade de ocorrências de cada um destes valores.

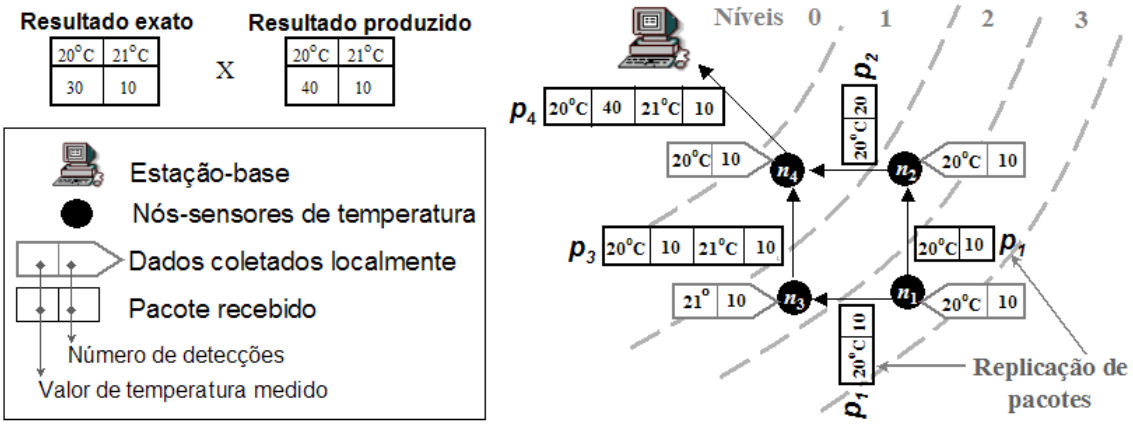


Figura 5.5 Distorções nos resultados da consulta, devido à reprodução de pacotes e uso da agregação em rede

Na Figura 5.5, considera-se um conjunto de nós-sensores de temperatura, representado pelos nós n_1, n_2, n_3 e n_4, distribuídos em três níveis (três diferentes distâncias a partir da estação-base) de uma RSSF. Admite-se também que um pacote transmitido através da rede é dividido em 2 partes: um cabeçalho, que contém a identificação do pacote; e um corpo, que armazena um conjunto de pares de valores (valor distinto de temperatura e número de ocorrências deste valor). Cada nó-sensor realiza 10 coletas de dados do ambiente, antes de compor um pacote e enviá-lo a seus nós-pais.

O nó n_1 envia o pacote p_1 tanto para o nó n_2 quanto para o nó n_3, provocando a reprodução do pacote p_1. n_2 agrega seus dados coletados localmente (20°C - 10) com os dados advindos de n_1, gerando um novo pacote p_2. n_3 agrega seus dados coletados localmente (21°C - 10) com os dados advindos de n_1, gerando um novo pacote p_3. n_4 agrega seus dados coletados localmente (20°C - 10) com os dados advindos dos pacotes enviados por n_2 e n_3 (p_2 e p_3), porém, n_4 não é capaz de detectar duplicações nos dados, porque a identificação inicial do pacote p_1 (gerada em n_1) foi perdida. Finalmente, a estação-base produz o resultado final da consulta, que é diferente daquele que seria gerado caso a reprodução do pacote p_1 não tivesse ocorrido.

Objetivando evitar o problema citado acima, o algoritmo ADAGA admite uma estratégia de roteamento como a descrita a seguir. Quando um pacote p_1 encontra o primeiro nó n_1, que possui mais de um nó-pai, uma cópia de p_1 é gerada para cada um dos nós-pais de n_1. Porém, após esta primeira reprodução de p_1, nenhuma nova cópia de p_1 poderá ser produzida. Logo, quando uma das cópias de p_1 (p_2) encontrar um novo nó n_2, que possua mais de um nó-pai, apenas um destes nós-pais será escolhido para receber p_2. ADAGA implementa esta estratégia de modo que, quando um pacote p_1 é gerado, o número de cópias geradas de p_1 (c) seja registrado no cabeçalho de p_1. Há duas possibilidades quando um nó n_1, que tem mais de um nó-pai, recebe o pacote p_1: (i) Se $c = 1$, p_1 é replicado para cada nó-pai de n_1; e (ii) Se $c > 1$, p_1 é enviado para apenas um dos nós-pais de n_1 (escolhido aleatoriamente).

Após a primeira reprodução de um pacote p_1, o número de cópias de p_1 é progressivamente decrementado, à medida que as cópias vão sendo transmitidas através da rede. Caso um nó n_1 receba l cópias de p_1, a cópia p_2 de p_1 que tiver o menor valor para c será mantida, enquanto as demais, $l - 1$ cópias, serão descartadas. Neste caso, o valor de c em p_2 é atualizado para $c - l - 1$. Observe que quando $c = 1$, o pacote pode ter seus dados agregados, visto que não há mais riscos de que duplicações de dados ocorram no resultado da consulta. Desta forma, as cópias dos pacotes vão sendo progressivamente eliminadas, à medida que vão se aproximando da estação-base.

Objetivando ilustrar como ADAGA processa a agregação em rede, considere o mesmo exemplo apresentado na Figura 5.5. A nova estrutura adotada para o pacote utiliza um cabeçalho composto por um identificador e o número de cópias reproduzidas do pacote (c). Observa-se na Figura 5.6 que o nó n_1 coleta dados e gera o pacote p_1 que é enviado para os seus nós pai, n_2 e n_3. Como p_1 possui $c = 2$, as cópias de p_1 não têm seus dados agregados aos dados coletados em n_2 e n_3. A seguir, como o nó n_4 recebe 2 cópias de p_1, n_4 descarta uma destas cópias e identifica que agora $c = 1$. Logo, n_4 agrega os dados da cópia mantida de p_1 com os dados localmente coletados e com os dados de p_2 e p_3. Finalmente, o pacote resultante p_4 é gerado por n_4 e enviado à estação-base.

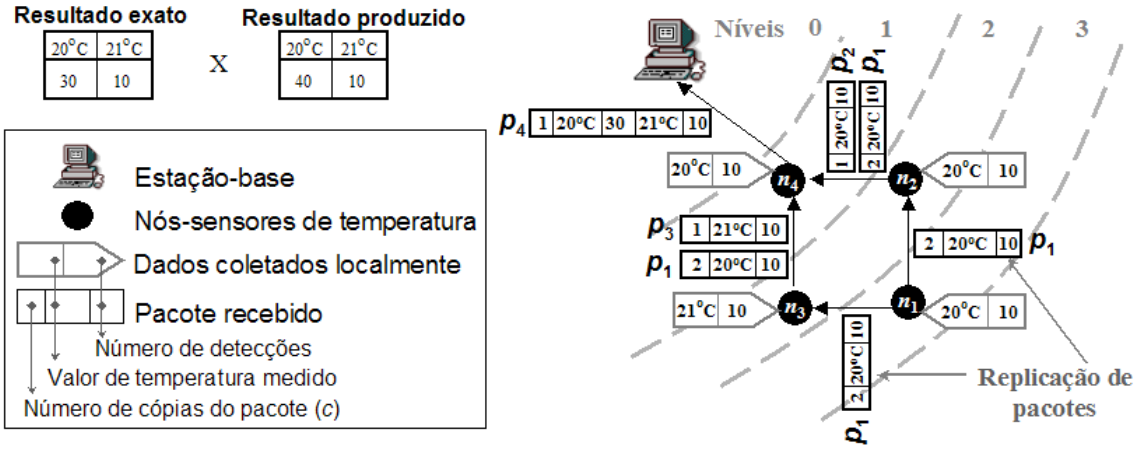


Figura 5.6 Agregação em rede aplicada pelo algoritmo ADAGA.

Claramente, alguns pacotes recebidos pela estação-base podem ainda ter cópias trafegando na rede. Similarmente ao que acontece nos nós-sensores, o conteúdo de um pacote p_1 não pode ser processado enquanto houver outras cópias de p_1 na rede. Todavia, se um pacote p_1, recepcionado pela estação-base, perder alguma de suas cópias (por exemplo, devido a falhas em algum nó-sensor), o pacote p_1 nunca terá seus dados agregados (pois $c > 1$). Há duas alternativas para superar este problema. A primeira consiste no uso de marcas de tempo para indicar quando um pacote pode ser analisado pela estação-base, mesmo tendo $c > 1$. A segunda é esperar que o tempo de validade da consulta seja alcançado (valor definido na cláusula *TIME WINDOW*), visto que se tem a garantia de que outras cópias do pacote que chegarem serão desconsideradas pela estação-base. Porém, caso a consulta seja definida como contínua, a segunda alternativa não é viável, pois, como o tempo de validade da consulta é indefinido, os pacotes com cópias perdidas na rede nunca teriam seus dados agregados na estação-base.

Outro aspecto que pode ser explorado nesta estratégia de roteamento consiste na avaliação da função de agregação especificada na consulta. Por exemplo, funções de agregação não-sensíveis a duplicações (abordadas no Capítulo 3), como *MIN* e *MAX*, dispensam a análise do número de cópias de um pacote, requerida por esta estratégia de roteamento.

As propostas que usam a agregação em rede são principalmente diferenciadas por seus protocolos de roteamento de dados. A proposta *Directed diffusion* (Subseção 2.3.8.7) requer um maior *overhead*, o que é causado pelo uso de uma

cashe extra. Uma outra proposta, GIT [33], defende o uso de uma arquitetura de rede rígida, não prevendo a mobilidade dos nós da rede. Já a proposta TAG [41], desenvolvida no contexto do projeto TinyDB, não admite o uso de rotas alternativas na transmissão dos pacotes, tornando-se, assim, mais suscetível às perdas de pacotes, no caso de falhas em nós-sensores. Conforme apresentado, ADAGA admite replicações de pacotes, ao mesmo tempo em que trata para que os dados não sejam duplicados no resultado, o que é feito através da eliminação progressiva das cópias dos pacotes, à medida que estas cópias aproximam-se da estação-base.

5.6 Análise

É importante observar que estatísticas são usadas para realizar inferências sobre uma população a partir de uma amostra. Neste trabalho, considera-se como a população todos os valores que seriam obtidos se houvesse uma atividade contínua de sensoriamento. Observa-se que, quando o intervalo de sensoriamento é informado em uma consulta, ele define uma amostra. De forma que, quando o intervalo de sensoriamento é aumentado, uma amostra menor é obtida e menos precisos serão os resultados obtidos a partir desta amostra.

O algoritmo ADAGA foi avaliado através do processamento de uma consulta cujo objetivo consiste em fazer com que os nós-sensores enviem para a estação-base pacotes compostos por: valores distintos de temperatura e o número de detecções de cada valor de temperatura. Esta consulta requer um agrupamento pelos valores distintos coletados, sendo bem mais crítica para os nós-sensores do que consultas que apenas utilizam funções de agregação, como *MAX*, *MIN*, *SUM* ou *COUNT*, as quais requerem um espaço mínimo de armazenamento. Para obter o resultado final da consulta, expressa na Figura 5.7, a estação-base deverá receber os dados provenientes dos nós-sensores e calcular o valor da média da temperatura do ambiente.

Para a realização dos testes foi utilizado um simulador de rede de sensores, desenvolvido em C++, e executado em uma máquina Pentium IV. Este simulador permite a configuração da disponibilidade de energia e memória para um conjunto de nós-sensores. Assim, é possível simular como o algoritmo ADAGA responderia em situações de restrição de recursos. Vale salientar que foi considerada uma

distribuição uniforme dos valores de temperatura, no intervalo entre -10^{o}C e 40^{o}C, para a simulação realizada.

Descubra quais são os valores distintos de temperatura e o número de detecções de cada valor coletado na área geográfica delimitada por: 3º43'08"S-38º31'51"W e 3º43'16"S-38º31'14"W. Submeta a consulta uma vez, em 10-Oct-06 às 14:00:00. A execução deve durar 9x10^{4} segundos. Dados devem ser coletados a cada 1x10^{2} segundos e enviados à estação-base a cada 3x10^{4} segundos.

```
SELECT t.valorColetado, count(t.valorColetado)
FROM   Temperatura t,   RegiaoSensoriada r
WHERE  r. idRegiao = t. idRegiao
AND    r.latitudeInicial   > 034308
AND    r.latitudeFinal  < 034316
AND    r.longitudeInicial > 383151
AND    r.longitudeFinal< 383114
GROUP BY        t.valorColetado
TIME WINDOW     90000
SEND INTERVAL   30000
SENSE INTERVAL  100
SCHEDULE        1 '10-Oct-06 14:00:00'
```

Figura 5.7 Consulta utilizada para avaliação dos resultados produzidos por ADAGA.

O primeiro critério considerado na avaliação de ADAGA foi a precisão dos resultados obtidos. Primeiramente foram gerados os resultados para a consulta, apresentada na Figura 5.7, desconsiderando restrições de recursos (resultado exato). A mesma consulta foi executada utilizando duas estratégias que consideram o uso da agregação em rede, a proposta reativa e a proposta pró-ativa, considerando diferentes situações de restrição de energia e memória. A proposta reativa apenas interrompe suas atividades em um dado nó-sensor quando seus recursos se esgotam, enquanto que a proposta pró-ativa é a explorada no algoritmo ADAGA. Finalmente, uma terceira estratégia, que não admite o uso da agregação em rede, foi avaliada. Esta última estratégia é a mais simples, quanto ao *hardware* e ao *software* empregados no nó-sensor, todavia resulta em um maior volume de dados trafegando na rede.

A Figura 5.8 (a) mostra como os resultados da consulta em estudo assumem diferentes aproximações em relação ao resultado exato, com base nas três estratégias descritas anteriormente: reativa, pró-ativa e sem uso de agregação em rede. Já a Figura 5.8 (b) mostra a quantidade de detecções realizada em cada uma destas três estratégias.

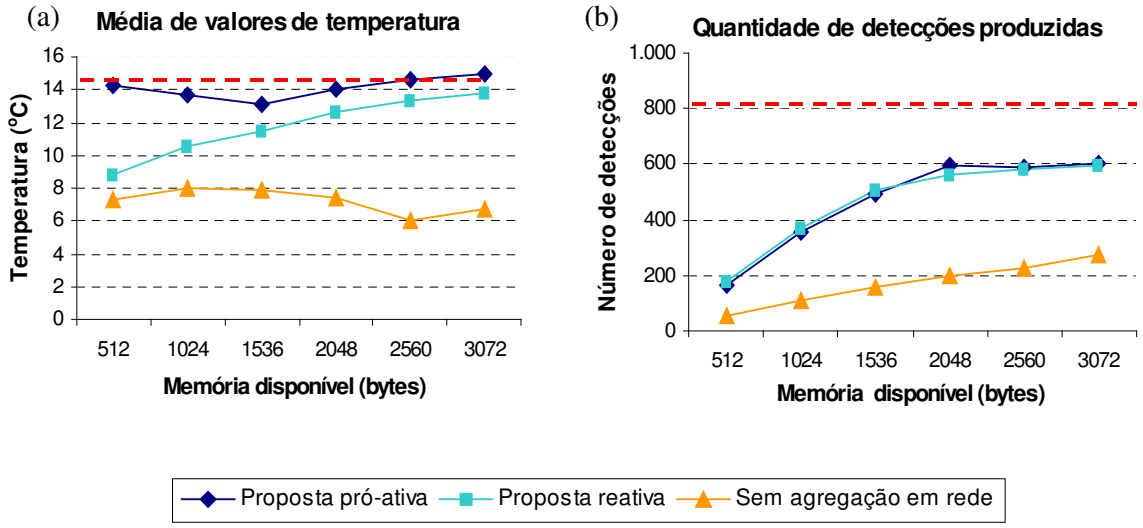


Figura 5.8 (a) Resultados obtidos para o cálculo da média de temperaturas coletadas. (b) Quantidade de detecções computadas para o cálculo da média de temperaturas.

Note que, em 90.000 segundos, 900 coletas teriam de ser realizadas, caso não houvesse restrições de recursos. Nesta situação, o valor obtido como resultado da média dos valores recebidos na estação-base seria 15,03^{o}C (linha **---**). Na Figura 5.8 (a), observa-se que ADAGA produz resultados mais próximos dos resultados exatos da consulta, considerando diferentes níveis de restrição de energia e memória, do que quando comparado aos resultados obtidos com a proposta reativa ou a proposta que não utiliza a agregação em rede. A Figura 5.8 (b) mostra um gráfico comparativo da quantidade de coletas realizadas. Neste caso, a proposta pró-ativa e a proposta reativa realizam quase o mesmo número de coletas, visto que ambas adotam a agregação em rede. Por outro lado, quando a agregação em rede não é utilizada, o número de coletas cai drasticamente, visto que a memória disponível é rapidamente consumida.

A Figura 5.9 compara o consumo de energia usando as três estratégias apresentadas acima. Observe que o consumo de energia aumenta com o consumo de memória, visto que mais dados são enviados para outros nós da rede. Desta forma, um maior consumo de energia é requerido. Todavia, a proposta pró-ativa alcança um consumo mais otimizado da energia quando comparado com as outras propostas apresentadas. Porém, quando a disponibilidade de recursos é suficiente

para computar o resultado exato da consulta, as três estratégias requerem o mesmo consumo de energia.

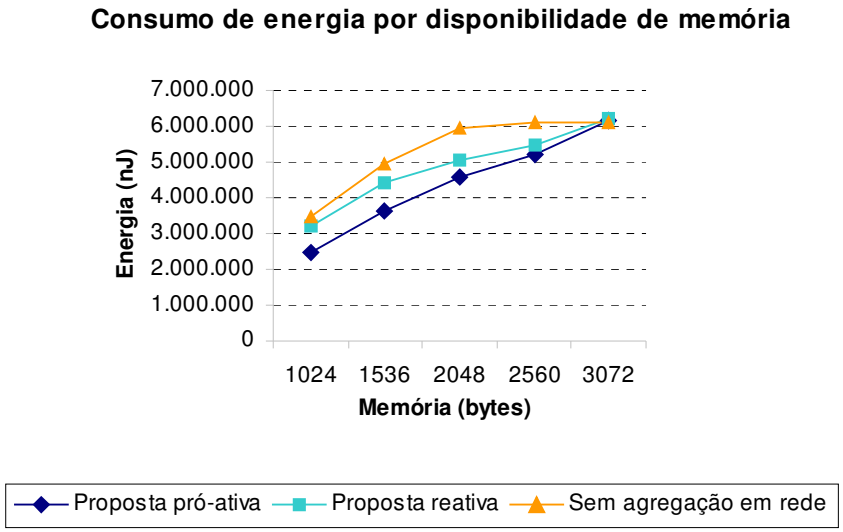


Figura 5.9. Consumo de energia para cálculo da média de valores de temperatura.

A Figura 5.10 mostra a quantidade estimada de dados que teria que ser enviada à estação-base por um dado nó-sensor, considerando a execução de diferentes funções de agregação. A amostra utilizada consiste em 16 coletas de valores de temperatura. Os valores distintos das temperaturas medidas são 20°C, 21°C , 22°C, 23°C e 24°C. Admite-se que para armazenar cada valor de temperatura é necessário um espaço de 4 bytes. Observa-se que, com o uso da agregação em rede (propostas pró-ativa e reativa) é necessário um espaço de armazenamento de apenas 4 bytes para computar o resultado das operações *MAX*, *MIN*, *COUNT* e *SUM*. O cálculo do resultado parcial, obtido em cada nó-sensor, para a função *AVERAGE* requer o dobro de espaço (8 bytes), visto que dois valores precisam ser computados, os resultados das funções *COUNT* e *SUM*. O tamanho do resultado parcial produzido pela função *DISTINCT* é proporcional ao número de diferentes valores de temperatura coletadas. Já a função *MEDIAN* força os nós-sensores a enviarem todos os dados coletados para a estação base, sem qualquer pré-processamento, produzindo um volume de dados semelhante ao obtido quando a agregação em rede não é adotada.

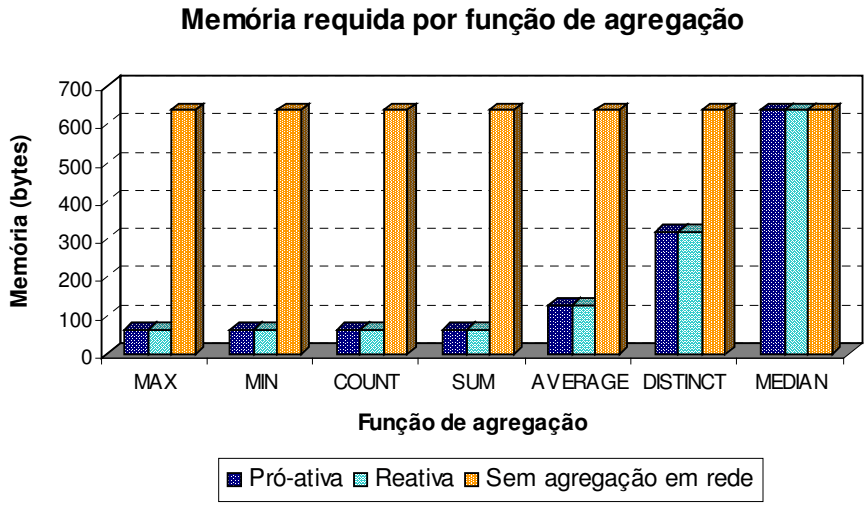


Figura 5.10. Volume de dados produzidos, considerando diferentes funções de agregação.

5.7 Considerações finais

Neste capítulo foi apresentada a proposta do algoritmo ADAGA, destinado a realizar o processamento adaptativo de operações de agregação em RSSFs. Foi visto que o foco de ADAGA é a realização do monitoramento pró-ativo dos recursos de energia e memória de cada nó-sensor individual. Além disso, ADAGA é capacitado a ajustar as atividades de coleta de dados e envio de pacotes dos nós-sensores, de acordo com a disponibilidade de recursos físicos destes dispositivos. Os resultados apresentados mostraram que ADAGA consegue produzir resultados mais próximos dos resultados exatos de uma consulta, quando comparado às estratégias reativa e a que não adota a agregação em rede.

Capítulo 6

ADAPT – OPERADOR ADAPTATIVO DE JUNÇÃO

6.1 Introdução

Este capítulo apresenta a proposta ADAPT, um algoritmo que implementa um operador adaptativo de junção a ser executado na estação-base de RSSFs. Tendo em vista o grande volume de dados que pode vir a ser gerado por uma RSSF com um grande número de nós, a adoção de uma estratégia adaptativa pode permitir que um algoritmo seja capacitado a reagir a situações de restrição de memória principal, por exemplo, durante o processamento de uma operação junção. ADAPT é caracterizado por permitir a disponibilização incremental dos resultados de uma consulta, o que é muito importante para aplicações de RSSF, visto que, muitas vezes, as consultas submetidas a estas redes são contínuas.

Este capítulo está dividido conforme descrito a seguir. A Seção 6.2 apresenta as propriedades de ADAPT e a Seção 6.3 descreve os estágios previstos neste algoritmo. A Seção 6.4 apresenta a estratégia adotada por ADAPT para evitar duplicações de dados no resultado da consulta e também reduzir o número de acessos a disco. Finalmente, na Seção 6.5 é realizada uma avaliação empírica do algoritmo.

6.2 Propriedades do algoritmo ADAPT

A arquitetura das RSSFs considerada neste trabalho admite que uma rede pode ser composta de uma ou mais sub-redes, cada uma possuindo sua própria estação-base. Considera-se que cada estação-base é um ponto potencial para a submissão de consultas pelos usuários (ponto de acesso). Mesmo sendo as estações-base nós robustos quanto à disponibilidade de recursos físicos, restrições de memória podem vir a ocorrer. Tento em vista que um dado ponto de acesso poderá recepcionar dados advindos de um grande número de nós da rede, é possível que um grande volume de dados tenha que ser manipulado por este ponto de acesso.

Objetivando uma melhor utilização dos recursos da estação-base, é proposto o algoritmo ADAPT: um operador adaptativo de junção que é capaz de reagir a

situações de restrição de memória e a eventos que podem aumentar consideravelmente o tempo de resposta do processamento de consultas em RSSFs.

O objetivo principal de ADAPT é realizar o processamento de junções envolvendo dados advindos de grupos de nós-sensores distintos, adaptando a sua execução à disponibilidade de memória principal em uma determinada estação-base. Este algoritmo é baseado nas seguintes propriedades:

(i) Produção incremental de resultados: operadores de junção convencionais produzem resultados quando pelo menos uma das relações envolvidas já foi completamente avaliada (lida). Por este motivo diz-se que junções são operações que requerem bloqueio. Em aplicações de RSSFs, as consultas submetidas ao sistema podem demorar muito até que sua execução seja finalizada. Desta forma, um longo tempo seria demandado até que algum resultado fosse apresentado ao usuário.

O algoritmo ADAPT explora a técnica de *pipelining*[16] objetivando produzir resultados incrementalmente, à medida que eles vão sendo processados. Considerando que, em aplicações de RSSFs, mesmo o resultado final é baseado na coleta de amostras de dados do ambiente, é possível que os usuários, que observam progressos no processamento de suas consultas, decidam parar a execução por considerar que os resultados obtidos até o momento já são suficientes para tomar uma determinada decisão;

(ii) Processamento de junções mesmo quando as fontes de dados experimentam atrasos: considerando que o envio de dados obedece a intervalos de tempo pré-definidos, os nós-sensores não estarão enviando dados a todo momento. Desta forma, é possível que as estações-base experimentem períodos de tempo durante os quais nenhum dado seja recepcionado. Objetivando manter a continuidade da produção de resultados e não desperdiçar tempo da CPU, ADAPT continua a execução da consulta usando dados já recepcionados, porém ainda não completamente processados;

[16] Operadores de consulta que suportam a técnica de *pipelining* geram tuplas de resultado tão logo tuplas de entrada do operador são recebidas (lidas).

(iii) Política de gerenciamento de dados para lidar com as restrições de memória: a despeito do fato de que a estação-base é robusta, quando comparada à capacidade de processamento e armazenamento dos nós-sensores, é possível que restrições de memória aconteçam, visto que a estação-base pode receber dados de um grande número de nós-sensores. Para lidar com a limitação de memória na estação-base, o algoritmo ADAPT utiliza uma política de gerenciamento que consiste em enviar parte dos dados para disco, quando a memória não tem disponibilidade de espaço para alocar os novos dados recebidos; e

(iv) Previne acessos desnecessários a disco: caso a memória principal não seja suficiente para abrigar todos os dados necessários a uma operação de junção, parte destes dados podem vir a ser descarregados em disco. Assim, durante o processamento da operação de junção, pode ser necessário ler dados descarregados anteriormente em disco, devido à escassez de memória, como forma de dar continuidade à execução da operação de junção e garantir a completude do resultado final da consulta. ADAPT é projetado para realizar a otimização na execução do processamento de consultas através da redução do número de acessos a disco.

6.3 Estágios do algoritmo ADAPT

Quando um pacote é recepcionado na estação-base, seu conteúdo é processado, de forma que os dados contidos no corpo do pacote são armazenados em uma área temporária (*buffer*). Esta área temporária é dividida entre os grupos-sensores referenciados pela consulta. Assim, quando um pacote chega à estação-base, a informação sobre o seu grupo-sensor, constante no cabeçalho do pacote, é extraída, como forma de identificar o tipo do dado a ser processado, por exemplo, dado de temperatura.

Observe que a estação-base deve ter conhecimento sobre o *layout* do pacote, para que o corpo deste pacote alimente corretamente as tuplas de uma dada tabela. A Figura 6.1 mostra a conversão dos pacotes de dados em tabelas temporárias na estação-base.

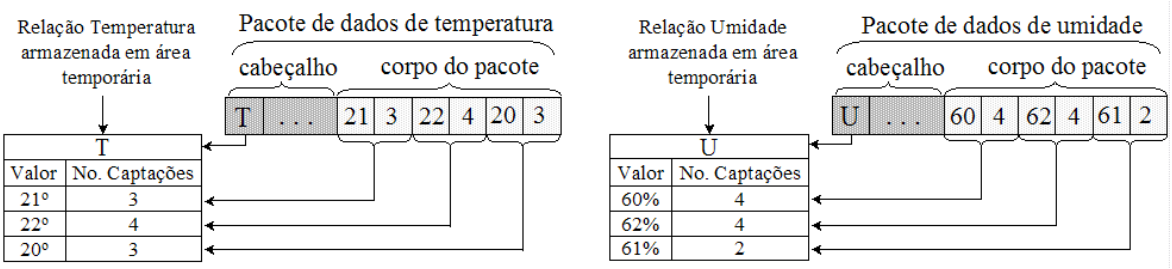


Figura 6.11 Conversão de pacotes de dados em tabelas armazenadas em área temporária na estação-base.

ADAPT é implementado como um operador de junção baseado nas técnicas de *hashing* e *pipelining*. Um algoritmo de *hashing* clássico consiste na utilização de uma função *hash* para particionar as tuplas das duas relações envolvidas em uma operação de junção, compondo uma tabela *hash* para cada uma das relações. Cada entrada na tabela *hash* corresponde a um valor de função *hash* compartilhado por um conjunto de tuplas. A etapa seguinte consiste na comparação das tuplas das duas relações, as quais têm o mesmo endereço na tabela *hash*. Uma propriedade importante desta técnica é que ela deve produzir uma distribuição equilibrada das tuplas nas entradas das tabelas *hash*, sob pena de degradar o desempenho do algoritmo. Outra técnica adotada em ADAPT é o uso de *pipelining*, que possibilita a geração de resultados tão logo as tuplas processadas pelo operador de junção sejam recebidas. Estas técnicas são também adotadas pelos operadores de junção SHJ (*Symmetric Hash Join*) [61], Xjoin [55], MobiJoin [11] e *Ripple Join* [29].

ADAPT estende o algoritmo MobiJoin pela exploração de algumas estratégias que resultam na redução do número de acessos a disco. Assim como em [55] e [11], ADAPT é projetado como um algoritmo adaptativo baseado em três estágios, executados como *threads*, cada um sendo destinado a responder a um evento específico. O primeiro estágio executa a operação de junção com base na técnica de *pipelining*, utilizando a estratégia proposta em [61]. Este estágio garante a produção de resultados incrementais, à medida que as tuplas recepcionadas vão sendo analisadas. O primeiro estágio também é capacitado a reagir a situações de "estouro" de memória, através da alocação de parte dos dados recepcionados em disco. Quando a recepção de tuplas de ambas as fontes de dados é temporariamente interrompida, o segundo estágio é iniciado para dar continuidade à produção de resultados, o que é feito através da avaliação de tuplas já

recepcionadas, mas não completamente processadas. Quando a recepção de tuplas é definitivamente interrompida, o terceiro estágio é executado para garantir que o resultado final da operação de junção esteja completo. Estes três estágios são detalhados a seguir.

Primeiro estágio

Neste estágio, o objetivo principal é processar a junção com a maior quantidade possível de tuplas residentes em memória. O primeiro estágio tem a sua execução iniciada quando as tuplas começam a ser recepcionadas e tem continuidade enquanto, pelo menos, uma das duas fontes de dados, envolvidas na operação de junção, estiver fornecendo tuplas.

Quando o primeiro estágio é iniciado, o primeiro passo é criar, em memória principal, as tabelas *hash* H_T e H_U correspondentes, respectivamente, às relações T e U, envolvidas na operação de junção. Assim, H_T é criada para armazenar as tuplas da relação T e H_U para armazenar as tuplas da relação U (Figura 6.12). As tabelas *hash* são divididas em áreas de mesmo tamanho, denominadas partições. Por exemplo, M_{T1} e M_{T2} são partições de H_T, e M_{U1} e M_{U2} são partições de H_U. Cada tupla recepcionada é submetida à função *hash*; o resultado desta operação é um endereço (identificador da partição) que indica a partição da tabela *hash* em que a tupla deve ser alocada. Além disso, a nova tupla é associada a um identificador único para a partição, denominado BTID. Vale salientar que BTID é um número seqüencial, incrementado a cada nova tupla alocada na partição, de forma que o último BTID indica o número de tuplas da sua partição.

Duas partições de tabelas *hash* diferentes são ditas correspondentes quando elas têm o mesmo identificador da partição, por exemplo, M_{T1} e M_{U1}. Em outras palavras, quando a função *hash* é aplicada aos atributos de junção de duas tuplas que pertencem a partições correspondentes, o mesmo identificador de partição é obtido.

A Figura 6.12 ilustra a execução do primeiro estágio. Observe que as tuplas t_{T1}, t_{T2} e t_{T3} são alocadas na partição M_{T1} e as tuplas t_{T4}, t_{T5} são alocadas na partição M_{T2} da tabela *hash* H_T. De forma semelhante, as tuplas t_{U1} e t_{U2} são alocadas na partição M_{U1} e t_{U3} na partição M_{U2} da tabela *hash* H_U. Quando uma tupla é alocada em uma partição (etapa de construção), ela é comparada com todas as tuplas da partição

correspondente da outra relação envolvida na operação de junção (etapa de teste). Caso alguma comparação seja bem sucedida (valores dos atributos de junção sejam coincidentes para as duas tuplas comparadas), resultados da operação de junção já podem ser produzidos (por exemplo, as tuplas com atributos de junção *A* e *E* na Figura 6.12). Observa-se que as etapas de construção e teste podem ser executadas em paralelo, segundo a estratégia de *pipelining* adotada no algoritmo proposto.

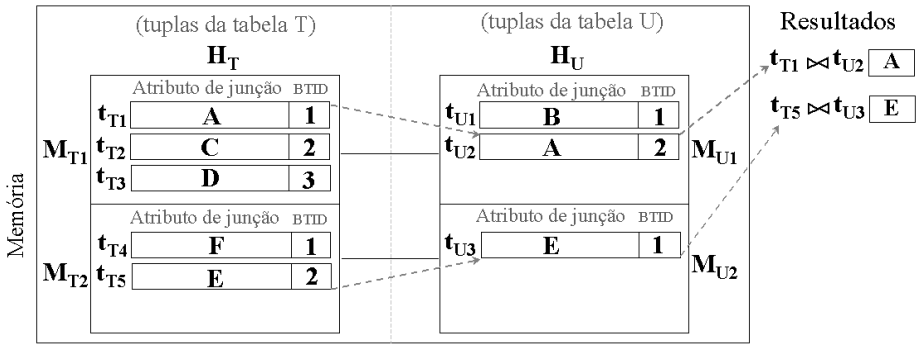


Figura 6.12 Primeiro estágio: comparação entre tuplas de partições correspondentes, alocadas em memória, pertencentes às duas relações envolvidas na operação de junção.

Durante a execução do primeiro estágio, há duas situações na qual *overflows* podem ocorrer: (i) *overflow* de partição, visto que cada partição tem um tamanho limitado; e (ii) *overflow* de memória, visto que o volume de dados enviado pelos nós-sensores da rede pode superar o limite de memória principal disponível para a operação de junção. De forma semelhante ao que é proposto em [55], ADAPT envia parte dos dados alocados em memória para disco, conforme explicado a seguir.

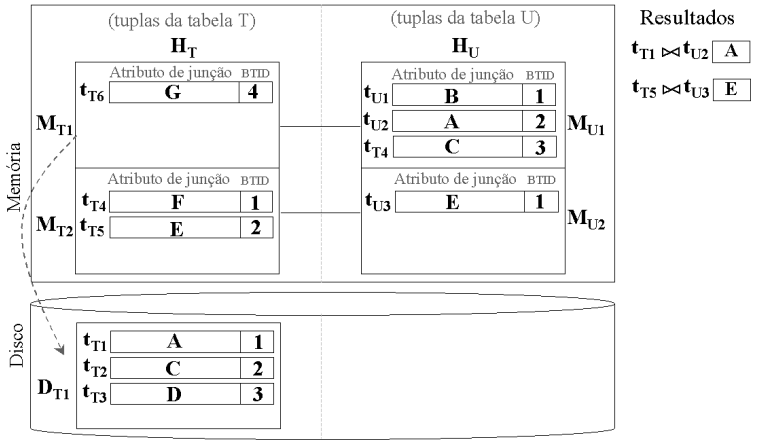


Figura 6.13 Situação da alocação de tuplas após o envio de dados para disco.

Quando é necessário alocar uma nova tupla em uma partição e a memória disponível para a junção ou o tamanho da partição estão no limite, por exemplo, t_{T6} na Figura 6.13, uma ou mais partições em memória têm seu conteúdo descarregado para uma partição correspondente em disco, por exemplo, a partição em disco D_{T1} corresponde à partição M_{T1}.

Observa-se que, com memória livre, o algoritmo pode novamente alocar novas tuplas nas tabelas *hash* residentes em memória; logo, a partição M_T pode receber tuplas novamente. Vale salientar que, sempre que uma partição M_T residente em memória é descarregada para disco, seu conteúdo é de fato anexado à partição correspondente, residente em disco, D_T.

Quando um *overflow* ocorre em uma determinada partição, esta será, necessariamente, a partição a ter seu conteúdo enviado para disco (partição-vítima). Porém, quando o *overflow* que ocorre é de memória, uma partição deve ser eleita como vítima para ser descarregada em disco. ADAPT adota a política sugerida em [48] para a escolha da partição-vítima. Esta política consiste em tentar eleger partições que tenham dados em memória e já tenham sido enviadas para o disco anteriormente. O objetivo é buscar uma menor fragmentação das partições entre disco e memória, tentando manter o maior número possível de partições inteiramente alocadas em memória e um pequeno número de partições que, provavelmente, estarão quase que inteiramente alocadas em disco. Os benefícios dessa política são evidenciados, particularmente, quando se considera que cada leitura do disco carrega uma página de dados.

A Figura 6.14 ilustra uma situação possível, comparando duas diferentes políticas de escolha de partições-vítimas. A primeira, adotada no algoritmo XJoin, escolhe como vítima a maior partição em memória (Figura 6.14-a); e a segunda, adotada em ADAPT (Figura 6.14-b), envia um número menor de partições para disco.

Como exemplo, considere que uma página de disco pode alocar até 6 tuplas e que cada partição em disco ocupa, pelo menos, uma página de dados. Neste caso, observe que a política demonstrada na Figura 6.14-a resultaria na *layout* de quatro páginas, para a recuperação de todos os dados do disco. Já na política apresentada na Figura 6.14-b, apenas duas páginas precisariam ser recuperadas do disco.

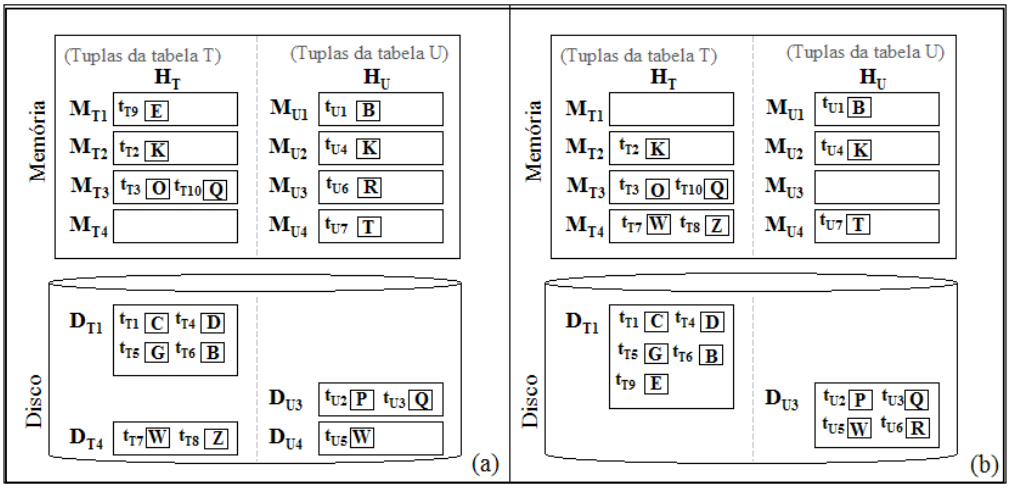


Figura 6.14 Políticas de escolha de partições-vítimas. (a) Política adotada no algoritmo XJoin. (b) Política adotada no algoritmo ADAPT.

Quando a estação-base experimenta intervalos de tempo sem receber dados, este estágio é interrompido e o segundo estágio é iniciado. Se o tempo de validade da consulta (especificado na cláusula *TIME WINDOW*) é alcançado, o primeiro estágio termina definitivamente e o terceiro estágio tem a sua execução iniciada. Se a consulta submetida ao sistema for contínua, é necessário adotar um delimitador de tempo para determinar o limite de tempo aceitável sem recepção de dados.

Segundo estágio

Este estágio tem sua execução iniciada quando o fornecimento de tuplas de ambas as relações envolvidas na operação de junção, é interrompido (bloqueado). Observe que tuplas de partições correspondentes das duas relações podem ainda não ter sido comparadas, visto que uma dada partição em memória M_T pode já ter sido descarregada em disco quando sua correspondente M_U recebeu novas tuplas.

Visando realizar a junção das tuplas ainda não comparadas no primeiro estágio, o segundo estágio escolhe uma partição residente em disco e a usa para comparar com as tuplas residentes em memória da partição correspondente da outra relação (relação oposta). As tuplas que satisfizerem a condição de junção são incluídas no resultado. A Figura 6.15 ilustra um exemplo de execução do segundo estágio no qual as tuplas de M_{T1}, descarregadas para disco, são comparadas com as novas tuplas de partição M_{U1} em memória.

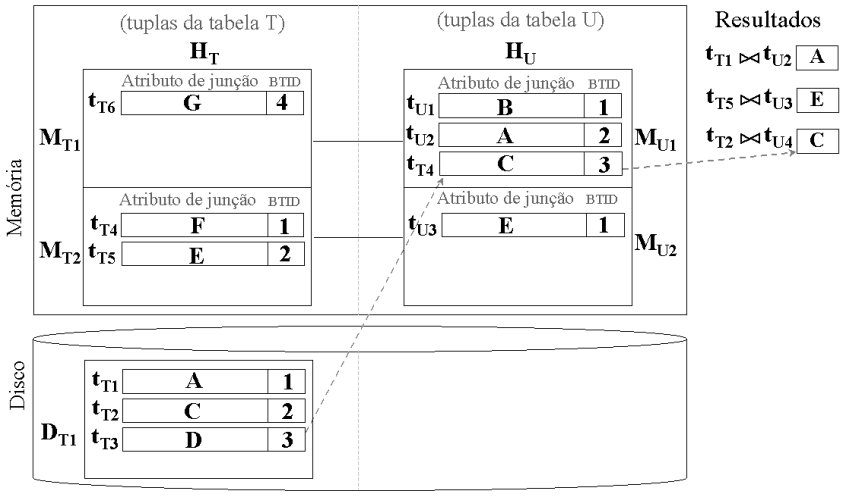


Figura 6.15 Segundo estágio: comparação entre tuplas em disco de uma partição com as tuplas em memória da partição correspondente da outra relação.

Caso a estação-base volte a receber tuplas de, pelo menos, uma das duas relações envolvidas na operação de junção, o segundo estágio é interrompido e o primeiro estágio volta a ser executado. Caso contrário, uma outra partição residente em disco é escolhida e comparada com a partição em memória da relação oposta.

Observe que a política de escolha de partições-vítimas, adotada por ADAPT no primeiro estágio, tende a gerar um pequeno número de grandes partições em disco. Um problema advindo desta estratégia ocorre quando uma partição em disco está sendo lida pelo segundo estágio e o fornecimento das tuplas é retomado. A leitura de grandes partições do disco, provavelmente, atrasará o retorno ao primeiro estágio do algoritmo. Uma alternativa para amenizar este problema consiste em verificar se novas tuplas foram recepcionadas, após cada página lida do disco, ou após um número configurável de páginas lidas do disco. No caso dos algoritmos Xjoin e MobiJoin, a verificação de chegada de novas tuplas no segundo estágio é feita ao final da leitura de cada partição em disco.

Assim, a execução do segundo estágio evita que o processador fique ocioso e adianta um trabalho que apenas seria feito no terceiro estágio do algoritmo; o que possibilita a continuidade da produção de resultados através da avaliação das tuplas já recebidas.

Terceiro estágio

Este estágio é iniciado quando todas as tuplas de ambas as relações envolvidas na operação de junção já foram recepcionadas. Para aplicações executadas sobre RSSFs, isto significa que as tuplas que chegarem após o tempo de validade da consulta (*TIME WINDOW*) devem ser desconsideradas. Observa-se que este estágio não é aplicável ao caso das consultas contínuas, visto que, para estas consultas, não há uma definição de quando a estação-base cessará a recepção de dados.

O objetivo principal do terceiro estágio é evitar a produção de resultados incompletos. No primeiro estágio, apenas as tuplas alocadas em memória durante intervalos de tempo coincidentes são comparadas. O que significa que este estágio não é capaz de realizar a junção de todas as tuplas pertencentes às partições correspondentes quando, pelo menos, uma delas já foi descarregada para disco. O segundo estágio não é capaz de realizar a junção das tuplas de uma partição residente em disco com aquelas que ainda não haviam chegado na partição correspondente em memória da relação oposta. Portanto, pares de tuplas não comparados nos dois primeiros estágios do algoritmo, devem, necessariamente, ser comparados no terceiro estágio, como forma de garantir a completude[17] do resultado final da consulta.

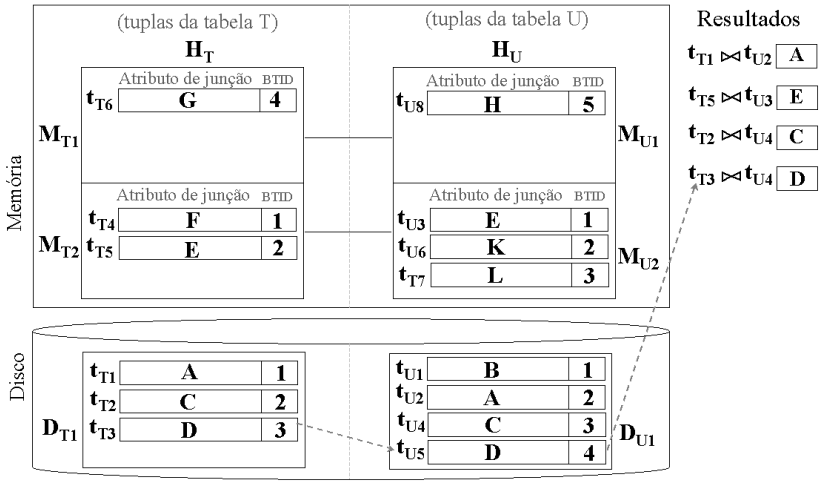


Figura 6.16 Terceiro estágio: comparação entre tuplas em disco de uma partição com as tuplas, em memória e em disco, da partição correspondente da outra relação.

[17] Embora os resultados de uma consulta em RSSFs sejam aproximados, devido ao fato de consultas serem aplicadas apenas a amostras coletadas do ambiente, admite-se que, após a chegada dos dados à estação base, não ocorrerá mais perdas de dados no processamento final da consulta.

No terceiro estágio, o algoritmo lê cada uma das partições residentes em disco e compara suas tuplas com aquelas das partições correspondentes, em memória e em disco, da relação oposta. Por exemplo, a Figura 6.16 ilustra a execução do terceiro estágio quando a partição D_{T1} é comparada com as partições M_{U1} e D_{U1}, produzindo a tupla de atributo de junção igual a *D*.

Observe que o segundo estágio avalia muitas das tuplas já comparadas no primeiro estágio, da mesma forma que o terceiro estágio repete muito do trabalho já realizado durante as várias execuções do segundo estágio de ADAPT. Assim, duplicações de dados no resultado estão sujeitas a acontecer.

6.4 Estratégia para evitar duplicações de resultados

Duplicações no resultado podem ser produzidas no segundo e terceiro estágios de ADAPT. Isto ocorre devido ao fato de que um mesmo par de tuplas pode ter seus elementos comparados mais de uma vez durante a execução do segundo e do terceiro estágios do algoritmo. Para lidar com este problema o algoritmo ADAPT gera uma tabela de controle para cada partição descarregada em disco. Desta forma, sempre que uma partição M_T é enviada para disco, uma tabela de controle é gerada para o par de partições composto por M_T e sua partição correspondente M_U. A tabela de controle permite que o algoritmo identifique quais os pares de tuplas que já tiveram seus elementos comparados.

A tabela de controle pode ser vista como uma matriz bidimensional, na qual as linhas representam as tuplas de uma partição e as colunas representam as tuplas da partição correspondente na relação oposta. Como a operação de junção envolve duas relações, o número de linhas da matriz é definido pelo número de tuplas pertencentes a uma relação e o número de colunas representa o número de tuplas pertencentes à outra relação. Os BTIDs associados a cada tupla correspondem a índices da matriz, os quais permitem localizar informações a respeito de uma dada tupla ou de um par de tuplas. Observe que é possível definir o número de linhas e colunas pela verificação do maior BTID de cada uma das partições representadas na tabela de controle.

Cada célula na tabela de controle pode assumir dois valores: “1”, indicando que os elementos de um par de tuplas já foram comparados; e “0”, significando que a comparação entre os elementos de um par de tuplas ainda não foi realizada. Esta

estrutura de dados é sugerida em [11], como alternativa para evitar a duplicação de resultados.

A Figura 6.17 ilustra um exemplo no qual uma nova tupla recepcionada, t_{T6}, deve ser alocada em M_{T1}. Considere que a alocação de t_{T6} em M_{T1} irá provocar um *overflow* de partição. Nesta situação, ADAPT descarrega o conteúdo da partição M_{T1} para disco (D_{T1}) e libera área em memória na partição M_{T1}. A tabela de controle *C*, com 4 linhas e 2 colunas, é gerada para o par de partições M_{T1} e M_{U1}. A Figura 6.17 mostra a distribuição de tuplas entre disco e memória, após a partição M_{T1} ter sido descarregada em disco e a tabela de controle *C* ter sido gerada para o par de partições M_{T1} e M_{U1}.

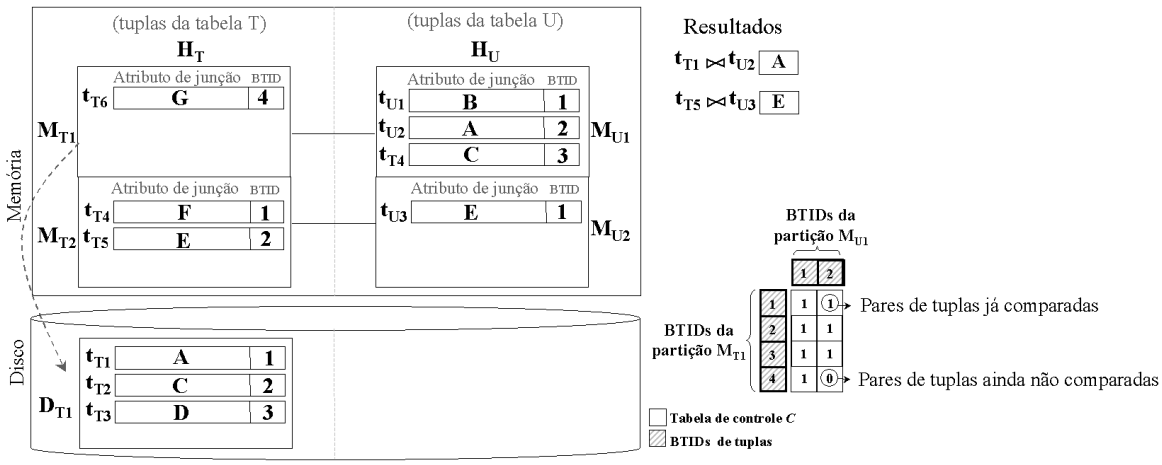


Figura 6.17 Envio de partição para disco e geração da tabela de controle.

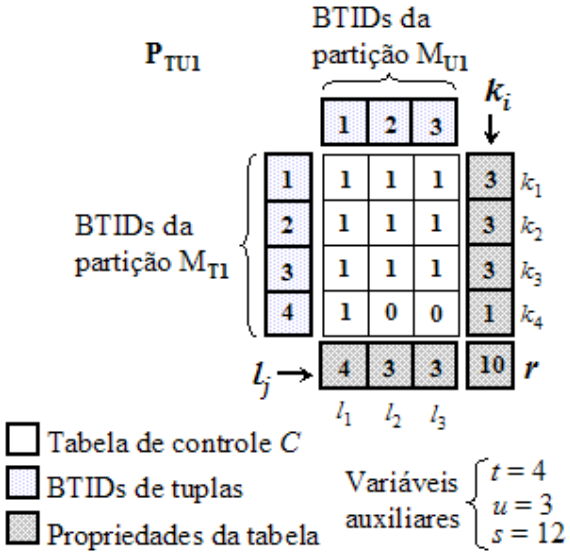
Quando a tabela de controle é criada, todas as suas células recebem o valor 1, visto que todas as tuplas representadas nesta tabela, até este momento, já devem ter sido comparadas em memória, ainda no primeiro estágio do algoritmo. À medida que novas tuplas vão sendo alocadas em uma partição M_T ou em sua partição correspondente da relação oposta M_U, a tabela de controle para este par de partições vai sendo redimensionada. Em outras palavras, uma nova linha ou uma nova coluna deve ser adicionada à tabela de controle sempre que uma nova tupla tiver de ser inserida em M_T ou em M_U. Embora a tabela de controle possa vir a ter um grande número de células, cada célula requer apenas um bit da memória, o que representa um *overhead* mínimo.

O algoritmo ADAPT explora algumas propriedades da tabela de controle objetivando obter um melhor tempo de resposta. Considere a tabela de controle *C*, mostrada na Figura 6.18, que relaciona duas partições: M_{T1} da relação T (temperatura) e a M_{U1} da relação U (umidade). Observe que os valores para *t* e *u* já estão disponíveis, visto que os maiores valores de BTIDs das partições M_{T1} e M_{U1} já são conhecidos. As seguintes propriedades da tabela de controle são exploradas, conforme a Figura 6.18:

(i) Se $k_i = u$, a tupla em M_{T1} que tem BTID igual a *i*, com $0 \leq i \leq u$, já foi comparada com todas as tuplas de M_{U1};

(ii) Se $l_j = t$, a tupla em M_{U1} que tem BTID igual a *j*, com $0 \leq j \leq t$, já foi comparada com todas as tuplas de M_{T1}; e

(iii) Se $r = s$, todas as tuplas de M_{T1} já foram comparadas com todas as tuplas de M_{U1}. Esta propriedade evita leituras desnecessárias de partições do disco para a memória.



t	Número de tuplas em M_{T1}
u	Número de tuplas em M_{U1}
s	$t \times u$
M_{T1i}	BTID de tuplas de M_{T1} na linha i, $1 \leq i \leq t$
M_{U1j}	BTID de tuplas de M_{U1} na coluna j, $1 \leq j \leq u$
k_i	$\sum_{j=1}^{u} C_{ij}$, $1 \leq i \leq t$
l_j	$\sum_{i=1}^{t} C_{ji}$, $1 \leq j \leq u$
r	$\sum_{i=1}^{t} k_i = \sum_{j=1}^{u} l_j$

Figura 6.18 Propriedades exploradas na tabela de controle adotada por ADAPT.

Analisando as propriedades da tabela de controle exemplificada na Figura 6.18, pode-se observar que todas as tuplas da partição M_{T1}, cujos BTIDs são 1, 2 ou 3, já foram comparadas com todas as tuplas da partição M_{U1}, visto que $k_1 = k_2 = k_3 = u = 3$ (propriedade (*i*)). Da mesma forma, a tupla da partição M_{U1} cujo BTID é 1 já foi comparada com todas as tuplas da partição M_{T1}, visto que $l_1 = t = 4$ (propriedade (*ii*)). A última propriedade, expressada em (*iii*), não é observada no exemplo dado, mas

ela é a propriedade mais importante a ser avaliada, visto que ela evita que acessos desnecessários a disco sejam feitos. Assim, se no exemplo dado na Figura 6.18 o valor para *r* fosse igual a *s* (*r* = *s* = 12), todas as tuplas de ambas as partições (M_{T1} e M_{U1}) já teriam sido comparadas e, portanto, não seria necessário ler porções destas partições alocadas em disco.

Se for considerado que a maior parte das porções de partições em disco já devem ter tido suas tuplas comparadas com as tuplas da partição oposta em estágios anteriores, os ganhos, advindos da leitura apenas das porções de partições em disco que ainda não tiveram todas as suas tuplas comparadas, representam um importante aspecto de otimização explorado pelo ADAPT, visto que a redução do número de acessos a disco é um dos principais objetivos em otimização de processamento de consultas.

Caso nenhuma das três propriedades seja verificada em uma determinada tabela de controle, o mecanismo para evitar duplicações funciona como descrito a seguir. Antes de ler as tuplas residentes em disco, ADAPT lê a tabela de controle e identifica os BTIDs das tuplas ainda não processadas (células da tabela de controle com valor 0). As tuplas residentes em disco associadas a células com valor 0 são comparadas com todas as tuplas da partição oposta correspondente. No caso do segundo estágio, a partição correspondente considerada é apenas a porção em memória. Já no terceiro estágio, todas as tabelas de controle são lidas, sendo que as porções residentes em disco que precisarem ser lidas são comparadas com as porções de partição correspondentes, tanto em disco quanto em memória.

6.5 Análise

Como visto na seção anterior, ADAPT reage a situações de bloqueio no fornecimento de tuplas provenientes das relações envolvidas na operação de junção. Para isto, o algoritmo proposto utiliza-se da estratégia, também adotada em MobiJoin e XJoin, de comparar as tuplas residentes em disco (que ainda não foram completamente processadas) com as tuplas da partição correspondente em memória da relação oposta. Porém, esta estratégia pode levar à produção de resultados duplicados.

Para evitar o problema de duplicações de resultados, XJoin implementa um mecanismo baseado em marcas de tempo. XJoin associa duas marcas de tempo a

cada nova tupla recepcionada: uma marca de tempo indicando o momento em que a tupla foi alocada na tabela *hash* em memória, e uma outra registrando o momento em que a tupla é descarregada em disco. Desta forma, Xjoin é capaz de distinguir pares de tuplas que estiveram em memória ao mesmo tempo e, portanto, já devem ter sido comparadas no primeiro estágio. Porém, este mecanismo não é suficiente para evitar duplicações quando uma mesma partição é avaliada por inúmeras execuções do segundo estágio do algoritmo. Para resolver este problema, XJoin associa a cada partição descarregada em disco uma lista encadeada. Cada elemento da lista encadeada armazena outras duas marcas de tempo, uma que registra o momento em que a última tupla da partição foi descarregada em disco e a outra que registra quando o segundo estágio leu a partição residente em disco. Desta forma, é possível identificar as tuplas já comparadas em alguma das execuções anteriores do segundo estágio. Todos estes controles representam um *overhead* extra para o processamento da operação de junção.

Além de XJoin requerer muitas estruturas de controle (um par de marcadores de tempo para cada tupla e uma lista encadeada de marcadores de tempo para cada partição descarregada em disco), os mecanismos utilizados para evitar duplicações podem ainda não ser suficientes em situações de chegada tempestiva de dados. Nestas situações pode ocorrer de várias tuplas, recepcionadas no mesmo instante, receberem os mesmos valores de marcadores de tempo, o que deixa o algoritmo sujeito a gerar duplicações. Este fato dependerá da precisão do relógio da máquina. MobiJoin e ADAPT não sofrem tal problema, visto que eles não trabalham com marcas de tempo.

MobiJoin associa um BTID (seqüencial) para cada tupla recepcionada em cada partição em memória. Quando uma partição é descarregada em disco, uma tabela de controle é gerada, relacionando pares de tuplas de partições correspondentes. Da mesma forma como em ADAPT, embora o BTID e a tabela de controle representem um *overhead*, o espaço requerido para o processamento da operação de junção é menor que aquele requerido em XJoin.

ADAPT estende o algoritmo MobiJoin, adequando-o às necessidades de um algoritmo de processamento de junções em estações-base de RSSFs através de duas estratégias. A primeira tem por objetivo a otimização do número de acessos a

disco, o que é conseguido através da exploração das propriedades da tabela de controle. Esta estratégia também resulta na otimização do processamento da consulta, particularmente, na execução do terceiro estágio do algoritmo. A segunda consiste em utilizar o número de nós-sensores e os valores das cláusulas *TIME WINDOW*, *DATA WINDOW*, *SENSE INTERVAL* e *SEND INTERVAL* para calcular, previamente, o volume de dados estimado a ser processado até o final da execução da consulta. Assim, pode-se permitir ao usuário observar os progressos de sua consulta (por exemplo, percentual de dados até então processados em relação ao volume total estimado) e, portanto, controlar o processamento da consulta em tempo de execução.

Foram realizados experimentos objetivando investigar o comportamento dos algoritmos ADAPT e XJoin em situações de restrição de memória (induzindo *overflows* de partição e de memória) e de ocorrência de interrupções no fornecimento de dados provenientes das duas relações (o que induz à execução da segunda fase do algoritmo). Além disso, durante as simulações, foi variado o tamanho da memória principal disponível e o número de tuplas envolvidas na operação de junção. Para o experimento, com resultados apresentados na Figura 6.19, foram consideradas tuplas de 12 bytes. Observa-se que o *overhead* para cada tupla de ADAPT corresponde ao BTID associado a cada tupla, que é de 4 bytes. Já XJoin associa dois marcadores de tempo a cada tupla; assim, o tamanho final da tupla em XJoin é de 20 bytes.

Os resultados expressos nos gráficos da Figura 6.19 demonstram que o comportamento, tanto de ADAPT quanto de XJOIN é semelhante quando a operação de junção envolve um volume de tuplas pequeno para a disponibilidade de memória, por exemplo, 20.000 tuplas. Neste caso, provavelmente, as tuplas estarão quase que inteiramente em memória, o que não requer acessos a disco. Porém, à medida que o volume de tuplas aumenta, mais partições têm que ser enviadas para disco, o que requer um maior número de acessos a disco. Os gráficos mostram que o número de acessos a disco realizados por ADAPT é menor que em XJOIN, para um maior volume de tuplas. Assim, a política de gerenciamento de dados entre disco e memória de ADAPT mostra-se mais eficiente quanto à redução do número de acessos necessários a disco para a realização da junção.

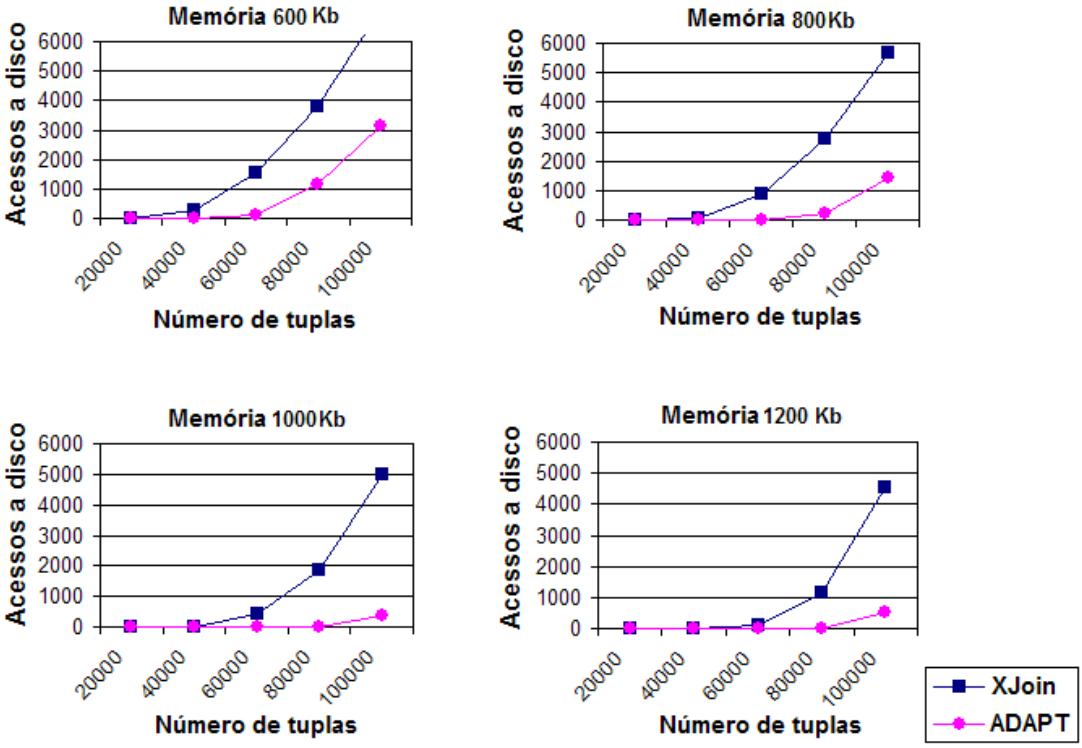


Figura 6.19 Comparação do número de acessos a disco em ADAPT e em XJoin.

Embora testes comparativos não tenham sido realizados com MobiJoin, os números de acessos a disco para este algoritmo provavelmente estarão representados por uma curva intermediária, abaixo da curva de XJOIN, tendo em vista que MobiJoin tende a ocupar menos memória, o que permite que menos tuplas sejam enviadas para disco; e acima da curva de ADAPT, tendo em vista que MobiJoin não adota uma estratégia para evitar acessos desnecessários a disco.

A maioria das consultas, mesmo sem cláusulas de agregação, provavelmente precisará de alguma agregação na estação-base, com o objetivo de agregar dados advindos de um mesmo grupo-sensor. A ordenação de dados pelos campos de agregação é uma técnica tradicional de agrupamento, porém ela requer que *(i)* nenhum resultado seja produzido até que todos os dados tenham sido ordenados; e *(ii)* cada grupo tem que ser completamente computado antes que um outro grupo seja considerado. Desta forma, o uso de estratégias de agregação, na estação-base, com estas características, impede a produção de resultados incrementais de consultas para o usuário, o que não é desejável para muitas aplicações de RSSFs.

Uma alternativa proposta para a realização de agregação *on-line*, a qual permite fornecer resultados incrementalmente, é sugerida em [29]. Esta proposta consiste na utilização de algoritmos de agrupamento baseados em *hash*. Estes algoritmos particionam uma relação com base nas colunas de agrupamento, definidas na consulta. Uma vantagem desta proposta é que, tão logo uma tupla seja recebida, uma estimativa atualizada da agregação para seu grupo pode ser produzida. Uma desvantagem desta estratégia de particionamento é que ela não escala devidamente com o número de valores de agrupamento, o que acontece quando a tabela *hash* excede o tamanho do espaço de armazenamento disponível em memória principal. Em Hellerstein *et al* [29] é proposto o uso do algoritmo *Hybrid Hashing* objetivando amenizar este problema. O algoritmo ADAPT pode ser estendido, de forma a oferecer suporte à agregação *on-line*, visto que ele adota uma política de gerenciamento de memória próxima àquela implementada no *Hybrid Hash Join* [48].

6.5 Considerações finais

Neste capítulo foi apresentada a proposta do algoritmo ADAPT, destinado à realização de operações de junção sobre dados provenientes de diferentes grupos-sensores de uma RSSF. Como foi visto, operações de junções executadas em estações-base estão sujeitas a restrições de memória principal. Mostrou-se que ADAPT, a exemplo de outros algoritmos adaptativos de junção, como o XJOIN e o MobiJoin, preocupa-se com o fornecimento incremental dos resultados da operação de junção, o que é particularmente importante quando se considera que muitas consultas submetidas a aplicações de RSSFs são contínuas. Adicionalmente à política de gerenciamento de dados entre disco e memória, que permite o fornecimento incremental de resultados, ADAPT também implementa um mecanismo para evitar acessos desnecessários a disco, como forma de obter um melhor desempenho da operação de junção como um todo.

Capítulo 7

CONCLUSÕES E TRABALHOS FUTUROS

Nesta dissertação foram abortados alguns dos aspectos mais relevantes a serem considerados no projeto de uma rede de sensores sem fio (RSSF). Também foi realizado um estudo das principais pesquisas relacionadas aos sistemas de fluxos de dados, bem como extensões destes sistemas propostas para atender às necessidades específicas das aplicações de RSSFs. Este embasamento teórico foi utilizado como base para introduzir conceitos explorados nos demais capítulos deste trabalho.

Como contribuição, esta dissertação propõe a adoção de uma metodologia para o processamento adaptativo de consultas em aplicações de RSSFs. O objetivo é o de realizar o processamento de consultas visando à disponibilização incremental de resultados de consultas e também otimizar o uso dos recursos físicos da rede, particularmente, dos nós-sensores. A metodologia proposta consiste em dois algoritmos, a serem executados nos nós de uma rede de sensores sem fio: ADAGA, para processamento de dados nos nós-sensores, e ADAPT, destinado ao processamento de junções nas estações-base destas redes.

O algoritmo ADAGA tem como objetivo contribuir para que o nó-sensor torne-se um dispositivo autoconfigurável, de forma que suas atividades sejam ajustadas à disponibilidade individual de seus recursos. As propriedades deste algoritmo são:

(i) Uso da técnica de agregação em rede como forma de otimizar o consumo de energia;
(ii) Monitoramento pró-ativo dos recursos de energia e memória, informação utilizada pelo algoritmo para ajustar as atividades dos nós-sensores;
(iii) Ajuste do intervalo de tempo entre coletas sucessivas de dados à disponibilidade de memória do nó-sensor;
(iv) Ajuste do intervalo de tempo entre envios sucessivos de pacotes de dados à disponibilidade de energia do nó-sensor;
(v) Aplicação de técnicas para estimar o valor das coletas não realizadas com base nos valores de coletas feitas anteriormente; e

(vi) Maior tolerância a falhas, visto que a duplicação de pacotes é permitida com a finalidade de evitar perdas de dados. Também é adotada uma estratégia para evitar que estas duplicações interfiram na precisão dos resultados das consultas.

Primeiramente, os testes realizados consideraram a execução de consultas em situações de disponibilidade ilimitada de recursos. Neste caso as consultas fornecem resultados exatos. As mesmas consultas foram consideradas para a execução de três estratégias de processamento em situações de restrição de recursos: (i) não-adoção de agregação em rede; (ii) proposta reativa; e (iii) proposta pró-ativa. Estas propostas tiveram seus resultados comparados ao resultado exato da consulta. A proposta pró-ativa, adotada pelo algoritmo ADAGA, foi a que forneceu resultados mais próximos dos resultados exatos, mesmo em situações de forte restrição de recursos.

ADAPT é um operador adaptativo que realiza a junção de dados, provenientes de grupos de nós-sensores distintos, sendo capacitado a reagir a situações de restrição de memória principal. O objetivo é que ADAPT seja executado na estação-base de uma rede de sensores sem fio, de forma que os dados recebidos dos inúmeros nós-sensores da rede sejam processados e incrementalmente disponibilizados para o usuário. As propriedades previstas para este algoritmo são:

(i) Produção incremental de resultados, de forma que os resultados sejam progressivamente disponibilizados no decorrer da execução da consulta;
(ii) Continuidade da execução da operação de junção, mesmo quando o fornecimento de tuplas das fontes de dados é suspenso;
(iii) Adoção de uma política de gerenciamento de dados para lidar com as restrições de memória principal, a qual consiste em enviar parte dos dados para disco; e
(iv) Prevenção de acessos desnecessários a disco, o que é uma das principais preocupações das estratégias de otimização de processamento de consultas.

Para a avaliação de ADAPT, foram consideradas duas fontes de dados, que forneceram dados gerados, dinamicamente, para a máquina responsável pelo processamento da junção. Operações de junção semelhantes foram processadas

com o algoritmo proposto ADAPT e com o algoritmo Xjoin [55], de forma a comparar o número de acessos a disco realizados em cada uma das propostas, sendo que ADAPT resultou em uma redução significativa deste número de acessos a disco, como resultado da sua política de gerenciamento de dados.

Adicionalmente, foi também proposto neste trabalho um modelo de dados genérico e uma linguagem de consultas declarativas, SNQL, para aplicações de redes de sensores sem fio.

O modelo de dados é baseado no conceito de grupo-sensor, que consiste em representar cada fenômeno monitorado como uma entidade. O conjunto das entidades do modelo é relacionado através da entidade RegiaoSensoriada, a qual representa a região geográfica onde os dados foram coletados.

A linguagem SNQL define cláusulas que especificam o tempo total de execução da consulta, o intervalo de tempo entre coletas sucessivas de dados e o intervalo de tempo entre envios sucessivos de pacotes através da rede. Também é especificada uma cláusula que age como um agendador, permitindo que o usuário especifique quantas vezes, e em que momento, uma dada consulta deve ser submetida à rede. As propriedades de SNQL são:

(i) Habilita o usuário a influenciar no tamanho da amostra produzida para a consulta;
(ii) Oferece suporte a consultas contínuas;
(iii) Oferece suporte a consultas pré-definidas; e
(iv) Permite a submissão de fragmentos de consultas, com o objetivo de alterar o valor de cláusulas SNQL, de uma consulta correntemente em execução.

Como trabalhos futuros planeja-se realizar as seguintes pesquisas:

(i) Capacitar ADAGA e SNQL a suportar a utilização de estratégias como, por exemplo, *Histograms*, *Wavelets*, *Sketches* e *Samplings*; como forma de obter uma maior sumarização de dados;
(ii) Permitir que ADAGA possa agregar dados provenientes de nós-sensores pertencentes a diferentes grupos-sensores, de maneira a reduzir o volume de dados trafegando na rede como um todo; e

(iii) Fornecer ao usuário um nível de confiança associado ao resultado da consulta, tendo em vista que a precisão do resultado pode ter sido afetada pela redução da disponibilidade de recursos físicos.

REFERÊNCIAS BIBLIOGRAFICAS

[1] Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E. *A survey on sensor networks*. IEEE Communications, pg. 102-114, Agosto de 2002.

[2] Al-Karaki, J., Kamal, A. *A taxonomy of routing techniques in wireless sensor networks*. Handbook of sensor networks: compact wireless and wired sensing systems. Editado por Mohammad Ilyas e Imad Mahgoub, CRC Press, pg. 117 – 140, Julho de 2004.

[3] Arasu, A., Babu S., Widom, J. *The CQL Continuous Query Language: semantic foundations and query execution*. Technical report, Stanford University, Outubro de 2003.

[4] Babu, S., Bizarro, P. *Adaptive query processing in the looking glass*. Conference on Innovative Data Systems Research - CIDR, Janeiro de 2005.

[5] Babu, S., Motwani, R., Babcock, B., Datar, M. *Models and issues in data stream systems*. ACM SIGMOD/PODS, Junho de 2002.

[6] Barabasi, A., Albert, R. *Emergence of scaling in random networks.* Science 286, pg. 509-512, Outubro de 1999.

[7] Bonnet, P., Gehrke, J., Seshadri, P. *Querying the physical world*. IEEE Personal Communications, Vol. 7, Nº 5, pg. 10-15, Outubro de 2000.

[8] Brayner, A., Lopes, A., Menezes, R., Vasconcelos, R. *Balancing energy consumption and memory usage in sensor data processing*. ACM Symposium on Applied Computing – SAC. Março de 2007.

[9] Brayner, A., Lopes, A., Meira, D., Vasconcelos, R., Menezes, R. *ADAGA: ADaptive AGgregation Algorithm for Sensor Networks*. XXI Simpósio Brasileiro de Banco de Dados – SBBD, pg. 191-205, Outubro de 2006.

[10] Brayner, A., Paulino, T. *Pré-Cálculo de junções para o processamento de consultas em ambientes com recursos computacionais limitados*. Dissertação de mestrado da Universidade Federal do Ceará - UFC, Agosto de 2004.

[11] Brayner, A., Vasconcelos, E. *MobiJoin: a join operator for mobile databases*. VI Workshop de Comunicação Sem Fio e Computação Móvel - WCSF, pg. 153-160, Outubro de 2004.

[12] Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N., Zdonik, S. B. *Monitoring streams - a new class of data management applications*. Very Large Data Bases – VLDB, pg. 215-226, Agosto de 2002.

[13] Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M. J., Hellerstein, J. M., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., Shah, M. A. *TelegraphCQ: continuous dataflow processing for an uncertain world*. Conference on Innovative Data Systems Research – CIDR, pg. 269-280, Janeiro de 2003.

[14] Chen, J., DeWitt, D. J., Tian, F., Wang, Y. *NiagaraCQ: a scalable continuous query system for internet databases*. ACM SIGMOD International Conference on Management of Data - COMAD, pg. 379–390, Maio de 2000.

[15] Considine, J., Li, F., Kollios, G., Byers, J. *Approximate aggregation techniques for sensor databases*. International Conference on Data Engineering – ICDE, pg. 449-460, Março de 2004.

[16] Cranor, C. D., Johnson, T., Spatscheck, O., Shkapenyuk, V. *Gigascope: a stream database for network applications*. ACM SIGMOD Conference, pg. 647-651, Junho de 2003.

[17] Deshpande, A., Guestrin, C., Madden, S. R., Hellerstein, J. M., Hong, W. *Model-driven data acquisition in sensor networks*. Very Large Data Bases – VLDB, pg. 588-599, Setembro de 2004.

[18] Elnahrawy, E. *Research directions in sensor data streams: solutions and challenges*. Technical report, Rutgers University, Maio de 2003.

[19] Ganesan D., Estrin D. *DIMENSIONS: why do we need a new data handling architecture for sensor networks?* Workshop on Hot Topics in Networks (Hotnets-I), pg. 143-148, Outubro de 2002.

[20] Garofalakis, M., Gibbons, P. B. *Wavelet synopses with error guarantees*. ACM SIGMOD International Conference on Management of Data – COMAD, pg. 476–487, Maio de 2002.

[21] Goel, S., Imielinski, T. *Prediction-based monitoring in sensor networks: taking lessons from MPEG*. ACM SIGCOMM Computer Communication - CCR, pg. 82-98, Outubro de 2002.

[22] Golab, L. *Querying sliding windows over on-line data streams*. International Conference on Data Engineering ICDE/EDBT Ph.D. Workshop, pg. 1-10, Março de 2004.

[23] Govindan, R., Hellerstein, J., Hong, W., Madden, S., Franklin, M., Shenker, S. *The sensor network as a database*. Technical Report, University of Southern California, Setembro de 2002.

[24] Gray, J., Bosworth, A., Layman, A., Pirahesh, H. *Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-total*. International Conference on Data Engineering - ICDE, pages 152-159, Fevereiro de 1996.

[25] Gurgen, L., Labbé, C., Olive, V., Roncancio, C. *A scalable architecture for heterogeneous sensor management*. IEEE International Workshop on Database ans Expert Systems Applications - DEXA, pg. 1108-1112, Agosto de 2005.

[26] Hac, A. *Wireless sensor network designs*. John Wiley & Sons press. University of Hawaii at Manoa, Honolulu, EUA, Dezembro de 2003.

[27] Haenggi, M. *Opportunities and challenges in wireless sensor networks*. Handbook of sensor networks: compact wireless and wired sensing systems. Editado por Mohammad Ilyas e Imad Mahgoub, CRC Press, pg. 22 – 35, Julho de 2004.

[28] Hedetniemi, S., Hedetniemi, S., Liestman, A. *A survey of gossiping and broadcasting in communication networks*. *Networks*, 18, pg. 129–134, 1988.

[29] Hellerstein, J. M., Haas, P. J. *Ripple Joins for Online Aggregation*. ACM SIGMOD Conference, pg. 287-298, Junho de 1999.

[30] Heinzelman, W., Chandrakasan, R., Balakrishnan, A. *Energy-efficient communication in wireless sensor networks.* Hawaii International Conference on Systems Sciences **-** HICSS, pg. 1-10, Janeiro de 2000.

[31] Heinzelman, W., Kulik , J., Balakrishnan, H. *Adaptive protocols for information dissemination in wireless sensor networks*. ACM/IEEE International Conference in Mobile Computing and Network – MobiCom, pg. 174–185, Agosto de 1999.

[32] Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K. *System architecture directions for networked sensors.* International Conference on Architectural Support Programming Languages Operating Systems - ASPLOS, pg. 93–104, Novembro de 2000.

[33] Intanagonwiwat, C., Estrin, D., Govindan, R., Heidemann, J. *Impact of network density on data aggregation in wireless sensor networks*. International Conference on Distributed Computing Systems - ICDCS, pg. 457, Julho de 2002.

[34] Intanagonwiwat, C., Govindan, R., Estrin D. *Directed difusion: a scalable and robust communication paradigm for sensor networks*. ACM International Conference in Mobile Computing and Network - MobiCom, pg. 56-67, Agosto de 2000.

[35] Ives, Z., Florescu, D., Friedman, M., Levy, A., Weld, D. *An adaptive query execution system for data integration*. ACM SIGMOD International Conference on Management of Data - COMAD, pg. 299–310, Junho de 1999.

[36] Kalpakis, K., Dasgupta, K., Namjoshi, P. *Maximum lifetime data gathering and aggregation in wireless sensor networks*. IEEE International Conference on Networking - ICN, pg. 685-696, Agosto de 2002.

[37] Koudas, N., Srivastava, D. *Data stream query processing: a tutorial*. XXI Simpósio Brasileiro de Banco de Dados – SBBD, Outubro de 2006.

[38] Lindsey, S., Raghavendra, C. *PEGASIS: power-efficient gathering in sensor information systems*. IEEE Aerospace Conference, pg. 1–6, Março de 2002.

[39] Madden, S., Franklin, M., Hellerstein, J. *TinyDB: an acquisitional query processing system for sensor networks*. ACM Transactions on Database Systems - TODS, Vol. 30, N$^{o.}$ 1, pg. 122-173, Março de 2005.

[40] Madden, S., Franklin, M., Hellerstein, J., Hong, W. *The design of an acquisitional query processor for sensor networks*. ACM SIGMOD International Conference on Management of Data - COMAD, pg. 491-502, Junho de 2003.

[41] Madden, S., Franklin, M. J., Hellerstein, J., Hong, W. *TAG: a Tiny AGgregation service for ad-hoc sensor networks*. Symposium on Operating System Design and Implementation – OSDI, pg. 171–182, Dezembro de 2002.

[42] Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G. S., Olston, C., Rosenstein, J., Varma, R. *Query processing, approximation, and resource management in a data stream management system*. Conference on Innovative Data Systems Research – CIDR, pg. 245-256, Janeiro de 2003.

[43] Pottie, G.J., Kaiser, W.J. *Wireless integrated network sensors*. Communications of ACM, 43(5), pg. 51–58, Maio de 2000.

[44] Rabaey, J.M., Ammer, M. J., Silva, J.L., Roundy, D. P. *PicoRadio supports ad hoc ultra-low power wireless networking*. IEEE Computer Magazine, 33(7), pg. 42–48, Julho de 2000.

[45] Raghunathan, V., Schurgers, C., Park, S., and Srivasta, M. *Energy aware wireless microsensor networks*. IEEE Signal Processing Magazine, vol.19, n^{o}2 , pg. 40-50, Março de 2002.

[46] Ratnasamy, S., Estrin, D., Govindan, R., Karp, B., Shenker, S., Yin, L., Yu, F. *Data-centric storage in sensornets*. Workshop on Wireless Sensor Networks and Applications - WSNA, pg. 78-87, Setembro de 2002.

[47] Ruiz, L. B., Nogueira, J. M., Loureiro, A. *Sensor network management*. Handbook of sensor networks: compact wireless and wired sensing systems. Editado por Mohammad Ilyas e Imad Mahgoub, CRC Press, pg. 57 – 84, Julho de 2004.

[48] Schneider, Donavan A. *A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment.* ACM SIGMOD-International Conference on Management of Data - COMAD, pg. 110-121, Junho de 1989.

[49] Shih, E., Cho, S., Ickes, N., Min, R. et al., *Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks*. ACM/IEEE International Conference in Mobile Computing and Network – MOBICOM, pg. 272–287, Julho de 2001.

[50] Silberschatz, A., Korth,H. F. and Sudarshan, S. Database Systems Concepts. McGraw-Hill Higher Education , 4th edition, 2001.

[51] Solis, I., Obraczka, K. *In-Network aggregation trade-offs for data collection in wireless in sensor networks*. Technical Report, University of California, Agosto de 2003.

[52] Srisathapornphat, C., Jaikaeo, C., Shen, C. *Sensor information networking architecture and applications*. IEEE Personal Communication, 8(4), pg. 52–59, Agosto de 2001.

[53] Su, W., Cayirci, E., Akan, O. B. *Overview of communication protocols for sensor networks*. Handbook of sensor networks: compact wireless and wired sensing systems. Editado por Mohammad Ilyas e Imad Mahgoub, CRC Press, pg. 314 – 329, Julho de 2004.

[54] Tucker, P., Maier, D., Sheard, T., Fegaras, L. *Exploiting punctuation semantics in continuous data streams*. IEEE Transactions on. Knowledge and Data Engineering, 15(3), pg. 555-568, Maio de 2003.

[55] Urhan, T., Franklin, M. *Xjoin: a reactively scheduled pipelined join operator*. IEEE Data Engineering Bulletin, 23(2), pg. 27-33, Junho de 2000.

[56] Yao, Y., Gehrke, J. *Query processing for sensor networks*. Conference on Innovative Data Systems Research – CIDR, pg. 233-244, Janeiro de 2003.

[57] Yao, Y., Gehrke, J. *The Cougar approach to in-network query processing in sensor networks*. SIGMOD Record, Vol. 31, N^{o} 3, pg. 9-18, Setembro de 2002.

[58] Younis, M., Akkaya, K., Eltoweissy, M., Wadaa, A. *On handling QoS traffic in wireless sensor networks*. IEEE Hawaii International Conference on Systems Sciences **-** HICSS, Vol. 9, pg. 90292.a, Janeiro de 2004.

[59] Waneke, B., Lebowitz, B., Pister, K. S. J. *Smart dust: communicating with a cubic-millimeter computer*. IEEE Computer, pg. 2-9, Janeiro de 2001.

[60] Wang, Q., Hassaneim, H. *A comparative study of energy-efficient (E2) protocols for wireless sensor networks.* Handbook of sensor networks: compact wireless and wired sensing systems. Editado por Mohammad Ilyas e Imad Mahgoub, CRC Press, pg. 239 – 360, Julho de 2004.

[61] Wilschut, Annita N., Apers, Peer M.G. *Dataflow query execution in a parallel main-memory environment.* International Conference on Parallel and Distributed Information Systems - PDIS, pg. 68-77, Dezembro de 1991.

[62] Zhao, F., Guibas, L. *Wireless sensor networks – An information processing approach*. Morgan Kaufmann press, Julho de 2004.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo