- # Introduction

Les réseaux de senseurs sont des réseaux composés d'un très grand nombre de noeuds qui sont des capteurs. En plus d'être capable de récolter des données physiques tel que la température ou la pression, ils sont capable de stocker, d,exécuter des requête et de transmettre des données. Une certaine analogie peut se faire entre les réseaux de capteurs et les bases de données relationnelle car un capteur ou un ensemble de capteurs peut correspondre à un fragment d'une base de données.

Au cours des dernières années, des dispositifs de capteurs intelligents ont évolué au point qu'il est maintenant possible de déployer les grands réseaux distribués. Ils sont constitués de plusieurs dizaines ou centaines de nœuds autonomes qui fonctionnent sans intervention humaine (par exemple, la configuration du réseau, la recharge des batteries, ou le réglage des paramètres) pendant des semaines ou des mois à la fois.

les capteurs sont de petits composants avec une faible capacité de stockage, de calcul et sont alimentés avec une batterie ou avec des piles.
Pour qu'un réseau de capteurs reste autonome pendant une durée de quelques mois à quelques années sans intervention humaine, la consommation d'énergie devient le problème fondamental.
il est difficile (parfois impossible dans certaine applications) de changer la batterie de ces composants.
Ainsi que, le stockage et l'exploitation des données collectées par ces réseaux doit être transparent et optimal vu la capacité limitée de ces noeuds en terme de calcule et de stockage.

État de l'art

TinyDB: An Acquisitional Query Processing System for Sensor Networks

Au cours des dernières années, des dispositifs de capteurs intelligents ont évolué au point qu'il est maintenant possible de déployer les grands réseaux distribués. Ils sont constitués de plusieurs dizaines ou centaines de nœuds autonomes qui fonctionnent sans intervention humaine (par exemple, la configuration du réseau, la recharge des batteries, ou le réglage des paramètres) pendant des semaines ou des mois à la fois.

L’article se concentre sur le traitement des requêtes dans les réseaux de senseurs tout en prend en compte les contraintes liées à la consommation des ressources.

Une stratégie typique qui sert à minimiser la communication coûteuse en appliquant des opérations d’agrégation et de filtrage des messages circulant dans le réseau.

On cherche à optimiser les emplacements et les coûts d'acquisition de données, nous sommes en mesure de réduire considérablement la consommation d'énergie par rapport aux systèmes passifs traditionnels.

L’article explique la conception et la mise en œuvre d’un processeur qui intègre une technique d’acquisition appelée TinyDB

TinyDB est un processeur de requêtes distribuées il fonctionne sur chacun des nœuds dans un réseau de capteurs, il a beaucoup des caractéristiques d’un processeur de requêtes traditionnel, mais, intègre également un certain nombre d'autres fonctionnalités permettent de minimiser la consommation d'énergie par des techniques d’acquisition.

L’article répond à ces questions :

- Pour une requête particulière, quand est-ce que les échantillons pour cette requête devraient être prises et dans quel ordre?
- Quel est l’intervalle entre ces échantillons et les autres opérations ?
- Est-il intéressant de déployer la puissance de calcul ou de bande passante pour traiter et transmettre un échantillon particulier ?
L’article traite les points suivants :

Section 1 : une brève introduction au capteur réseaux
déjà faite !
Section 2 : cette section examine chacune de ces phases de l’ACQP:
On peut subdiviser la consommation de l’énergie dans les différents noeuds du réseau en plusieurs phases:
La phase de repos : représente l’état où le noeud attend son timer ou un évènement extérieur pour effectuer un traitement, le noeud est à son consommation minimale d’énergie.
La phase de traitement : correspond à la phase où requête résultats sont générés localement, elle présente une consommation 4 fois plus grande que le mode repos.
La phase transmission: est la phase où le noeud consomme le plus d’énergie et consiste à transmettre les résultats du traitement eux noeuds voisins.

TinyOS:

TinyOS se compose d'un ensemble de composants pour la gestion et l'accès au matériel du noeud. TinyOS a été porté sur une variété des plates-formes matérielles,
Il est intéressant de noter que TinyOS ne fournit pas le système d'exploitation traditionnel, TinyOS est tout simplement une bibliothèque qui fournit un certain nombre de logiciels pratique
et un niveau d'abstraction, y compris les composants de moduler les paquets à la radio, lire les valeurs de capteurs, synchroniser les horloges entre un émetteur et le récepteur, et de mettre
le matériel dans un état de faible puissance.

Section 3 : langage de requêtes,
Dans cette section, nous présentons notre langage de requête pour ACQP se concentrant sur des questions du genre quand et comment souvent des échantillons sont acquis.
Sur TinyDB, les tuples de capteurs appartenant à une table capteurs qui, logiquement, contient une ligne par noeud à un instant donné, avec une colonne par attribut *(par exemple,* la lumière, la température, *etc.)*
l'appareil peut produire, dans l'esprit du traitement en acquisition, les enregistrements dans cette table sont acquises seulement pour satisfaire la requête, et sont généralement stockées
seulement pour une courte période de temps ou remis directement sur le réseau.
Bien que nous imposons le même schéma sur les données produites par chaque appareil dans le réseau, nous permettons la possibilité que certains dispositifs manquent, certains capteurs physiques peuvent insérer les valeurs NULL pour les attributs correspondants à des capteurs manquants. Ainsi, dispositifs capteurs demandés dans une requête produira des données pour cette requête dans tous le cas, sauf si les valeurs NULL sont filtrés explicitement dans la clause WHERE.
Physiquement, la table des capteurs est distribuée entre tous les périphériques du réseau,
et chaque noeud produit et stock ses propres relevés .
Ainsi, en TinyDB, pour comparer les lectures de différents capteurs, ces lectures doivent être collectées à un noeud commun (le root)
le caractéristiques du langage:
Requêtes en TinyDB, comme dans SQL, se composent d'une -SELECT-FROM-GROUPBY premettant la sélection, jointure, projection, et l'agrégation.
La sémantique de SELECT, FROM, WHERE et clauses GROUP BY sont comme dans SQL.

Section 4 : les questions d'optimisation de l'énergie dans les endroits sensibles
Compte tenu du langage de requête pour les environnements ACQP, avec des caractéristiques spéciales basé sur les événements et la duré de vie des requêtes, nous passons maintenant à interroger des problèmes de traitement.
Les requêtes en TinyDB sont analysés localement et diffusés sous format binaire simple dans le réseau de capteurs, où ils seront exécutés.
Avant la diffusion des requêtes, le noeud base effectue une simple d'optimisation pour choisir l'ordre optimal des opérations (l'échantillonnage, les sélections, …). L'optimisation est basée sur le faite de choisir un plan de requête qui permettra de minimiser la consommation globale d'énergie.

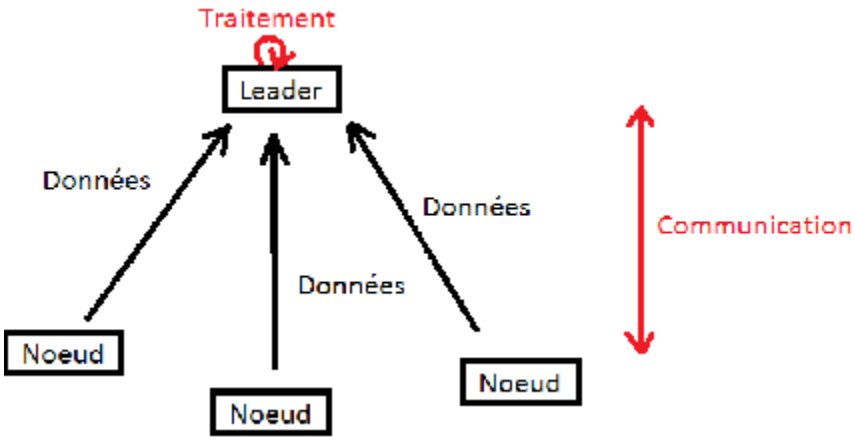
Chaque noeud dans TinyDB met un catalogue de méta-données qui décrit ses attributs locaux, les événements et les fonctions définies par l'utilisateur. Ces méta-données sont périodiquement copiés à la racine du réseau qui seront ensuite utilisées pour l'optimisation.
On peut optimiser la performance énergétique du système en ordonnant l'échantillonnage et les prédicats cette technique consiste à gérer est choisir le bon ordre des tâches surtout l'échantillonnage qui une opération coûteuse en terme de consommation de l'énergie. ainsi que la forme des prédicats.

Query Processing for Sensor Networks

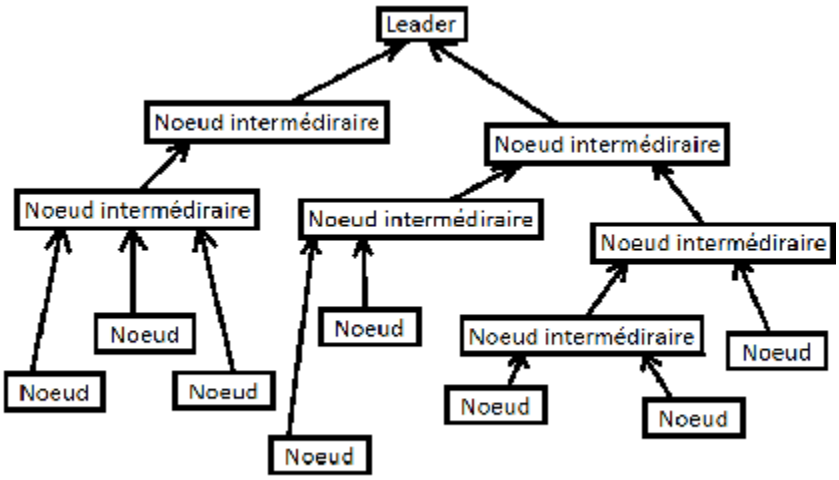
Les noeuds utilise le multi-hop routing protocol pour communiquer entre eux.
Les requêtes de réseaux de senseurs sont très semblables aux requêtes SQL. Les requêtes SELECT, FROM, WHERE, GROUP BY, HAVING sont les même que les requêtes SQL.
Cependant, deux autres requêtes ont été ajouté : DURATION et EVERY. DURATION
va servir a donner le temps de vie de la requête et EVERY la fréquence de la requête (ex : toutes les 10 secondes la requête se ré-exécute).

Un plan de requête peut être divisé en deux parties:

- communication : envoie des données des nœuds au nœud central (appelé leader)
- calcul/ traitement : traitement des données reçu au leader



Si on combine les deux parties d'un pan de requête, on peut créer des nœuds intermédiaire qui vont envoyer des données traitées à un nœud intermédiaire suivant jusqu'au nœud central. Ainsi on peut découper le traitement des données à plusieurs nœuds intermédiaires. Cela va permettre de diminuer le nombre de messages à envoyé et donc économiser de la batterie.



Exemple de schéma utilisant les nœuds intermédiaires

Il existes trois techniques pour combiner le traitement et la communication :

- Direct delivery :Chaque nœud envoie ses données grâce au multi-hop ad-hoc routing protocol.
- Packet merging : En réseau sans fil, il est plus coûteux d'envoyer plusieurs petits paquets plutôt qu'un gros (à cause de la taille des entêtes). Chaque paquet envoyé vers le leader sera « plein » (on envoie plusieurs données de nœuds différents dans un même paquet). Le traitement ne s'effectue que au nœud central.
- Partial aggregation : On utilise des nœuds intermédiaires qui vont faire des traitements sur les données des nœuds proche spatialement de lui. Ces données traitées vont être envoyées soit au nœud intermédiaire suivant soit au leader. Le leader va finalement faire le traitement des données reçues.

Pour pouvoir utiliser le packet merging ou la partial aggregation, il faut synchroniser les nœuds. En effet puisque les nœuds intermédiaires contiennent des données de plusieurs nœuds « précédents » ; ce nœud doit avoir les données des nœuds précédents avant de pouvoir envoyer son paquet au nœud suivant.
Pour les closes SUM et AVG, il faut être sur d'avoir les données une et une seule fois.
Pour les closes MAX et MIN, pas besoin de vérifier la duplications des données.
La synchronisation doit pour chaque nœud à chaque round déterminer combien de paquets il doit recevoir et quand envoyer le paquet au nœud suivant.

Incremental time slot algorithm

A chaque début de round, chaque nœud démarre un compteur et attends un temps fixé que les données des nœuds enfants lui soient envoyées. Ce temps fixé dépend de la profondeur du nœud dans la structure.
Cet algorithme possède des défauts pour les réseaux de senseurs :

- Difficulté de prévoir le temps fixé pour collecter les données d'un nœud à un autre (due aux coupures de liens qui sont très dur à anticiper). Du coup, si le temps fixé est trop petit, les données envoyé peuvent être incomplète. Si ce temps est trop grand, le délai sera trop important au leader, le système sera trop lent pour pouvoir être utiliser
- Coût trop élevé en mise à jour pour le temps fixé si il y a beaucoup de coupures de liens.
- Les nœuds de senseurs ne sont jamais parfaitement synchronisés sans signal GPS ou des protocoles de synchronisation coûteux.

Sur une petite période de temps, la communication entre deux senseurs est constante. Il est donc possible de prédire la communication suivant entre ces deux senseurs.
Si *p* et *e* sont deux nœuds ; *p* est le parent de *e*. Si *p* reçoit un paquet de *e*, il peut recevoir au prochain round un autre paquet de *e*. *p* ajoute donc *e* à sa liste d'attente.
Cependant cette prédiction peu échouer :

- Si le lien entre *e* et *p* cassé et que suite au reroutage, *e* à un nouveau parent *p1*.
- Si une condition est posé sur la sélection des données de *e* à envoyer à *p* et que cette condition est fausse.

Les experts utilisent un timer pour savoir si un prédiction est fausse ou non.
Comme un enfant peut savoir si un parent attend un paquet venant de lui, le nœud enfant peut générer un paquet de notification si la prédiction du parent est fausse.

Les nœuds de senseurs envoient leurs enregistrements au nœud central qui va créer un paquet avec le résultat final. Ce nœud central va envoyer un paquet au nœud de passerelle pour pouvoir envoyer les informations à d'autres réseaux.
Pour envoyer des informations des nœuds au leader, un routage a été établi grâce au multi-hop protocol. Des nœuds intermédiaires ont été mis en place.
La gestion du routage ou des pertes de lien pour les réseaux sans fils sont plus compliqué à mettre en place que les réseaux filaires (car dans un réseau sans fils, chaque élément du réseau doit participer au routage).
Un niveau de routage dans le protocole de pile fournit un envoie et une réception au niveau supérieur en cachant le fonctionnement interne du protocole de routage sans fil.

Les objectifs d'un protocole de routage sont de découvrir les routes et de les mettre à jour (suppression/ insertion de routes) disponible dans le réseau. La découverte de route permet d'établir un lien entre deux nœuds. Les mises à jours permettent de réparer les routes en cas de rupture de lien.
Les protocoles de routages peuvent être classés en 3 catégories :

- préventif : crée et répare les routes avant qu'une demande lui soit faite possibles
- réactif : crée et répare les routes uniquement sur demande
- hybride : fusion des deux type de protocole ci-dessus

AODV (Ad hoc On Demand Distance Vector) est un protocole de routage utilisant 5 message dont 3 principaux :

- RREQ (route request) : fait un broadcast sur le réseau pour trouver le destinataire du message si il ne le connait pas.
- RREP (route reply) : renvoie le chemin le plus court au destinataire
- RERR(route error) : indique qu'un lien est cassé.

La source envoie un RREQ pour trouver le chemin le plus court. Si le destinataire reçoit un RREQ, il va envoyer un RREP à la source. Si une RREP est reçu à la source, les données peuvent être transférées. Si la source ne reçoit aucun RREP, au bout d'un temps NET_TRANVERSAL_TIME, elle renvoie le message et répète ce processus RREQ_RRTRIES + 1 fois. Si aucun destinataire est trouvé au bout de RREQ_RRTRIES + 1 fois, on recommence 10 secondes plus tard.
Si la source est en train d'envoyer des données et reçoit un RREP qui a un chemin plus court que celui utilisé précédemment, le protocole met à jour le chemin et continue l'envoie de données.
Le lien entre la source et le destinataire se termine quand plus aucune données ne transitent vers le destinataire et que le temps ACTIVE_ROUTE_TIMEOUT a été dépassé.

Les chercheurs ont décidé d'utiliser le protocole AODC car :

- les réseaux de senseurs sont des grands réseaux (avec énormément de nœuds), il faut donc utiliser un protocole de routage réactif (qui permet de s’adapter rapidement à des coupures fréquentes)
- il ne génère pas de paquets en doubles (les doublons perturbent les calculs de SUM et AVG)
- il est très répandu et est implémenté sur beaucoup de simulateurs

Pour le paquet merging et le partial aggregation, il faut qu'un nœud intermédiaire puisse « intercepter » un message dont il n'est pas le destinataire. Pour ce faire, la couche des requêtes va devoir communiquer avec la couche réseaux pour intercepter les paquets destinés au leader.
En utilisant des filtres, au niveau du réseau, le paquets va passer par des fonction qui peuvent modifié ou même supprimé ce paquet. Au niveau des requêtes, un nœud intermédiaire peut intercepter des requêtes destiné au leader et cacher le résultat du traitement sur les requêtes. A un moment précis, le nœud intermédiaire va créer un nouveau paquet et l'envoyer au leader ou au nœud suivant. Tout ça se passe de façon transparente par rapport au niveau transport.

Modification du AODV

Les protocoles de réseau sans fil existant ont été crée pour établir et maintenir une connexion entre deux points. Dans un réseau de senseurs, plusieurs nœuds envoient des données au même nœud. Il faut donc modifier ces protocoles pour les réseaux de senseurs

Initialisation de la route : Dans un AODV classique, chaque nœud doit faire un broadcast pour déterminer le chemin le plus court jusqu'au leader. Les chercheurs ont décidés de faire un broadcast à partir du leader.

Ce message contient un compteur qui permet de définir la profondeur du nœud dans l'arbre. Ainsi chaque nœud peut sauvegarder le chemin inverse pour remonter au leader.
Maintenance de la route : Dans les réseaux de senseurs il est important que le protocole utilisé soit très fiable. En effet si l'on perd un paquet, le résultat traité au leader sera fortement erroné. Deux techniques peuvent être utilisé :

- Réparation locale : Dans un AODV lorsqu'un lien est cassé, le noeud concerné fait un broadcast pour trouver un chemin alternatif vers le leader.
- Réparation de groupe : La réparation locale ne va pas marché lorsque beaucoup de liens vont se casser simultanément ou si la topologie du réseau change suite à ces ruptures. Dans ce cas là, il est préférable de réparer toutes les routes directement depuis le leader. Cette réparation va être activé si le leader reçoit un nombre de tuples plus petit qu'une fraction définie par l'utilisateur (1 = reçoit tous les tuples/ 0 = reçoit aucun tuple)

Towards Sensor Database Systems

Les réseaux de senseurs sont énormément utilisés par des applications de mesures de détection et de surveillance. La plupart des applications existantes utilisent des systèmes centralisés pour collecter les données or ces systèmes ont de nombreux défauts. En premier lieu, ils manquent de flexibilité car les données sont extraites d'une manière prédéfinie. En second lieu, ils ne peuvent pas utiliser un très grand nombre de dispositifs à cause des volumes de données en jeu. Dans les systèmes distribuées, ce sont les senseurs qui décident quelles sont les données qui doivent être transférés. Dans ce papier les auteurs ont décrit un modelé de base de données de senseurs en représentant les données stockées en relation et les données des senseurs sont des séries temporelles.

Les capteurs actuels ne sont plus un outil de transformation d'une données physique en une données numériques seulement. ils sont capables de stocker, de calculer et de transférer ces données. On les appelle des bases de données de capteurs et les requêtes exprimées dans ce réseau sont appelle des requêtes capteurs. Afin de tester leurs systèmes dans un cas réel, les auteurs ont choisi de l'appliquer dans une factor Warehouse dans lequel on aurait placer des capteurs de température. Ces capteurs sont munis de deux fonctions *getTemperatures()* permettant de mesurer la température à des intervalles réguliers et *detectAlarmeTemperatur(threshold)* qui retrouve la température à chaque fois qu'elle dépasse un certain seuil. Chaque capteurs a la capacité de stocker localement ces données ainsi que de les envoyer. La base de données de capteurs qui est connecté au réseau de capteurs stocke les identifiant de tous les capteurs et leurs positions.
Le type de requêtes qui sont en exécution peuvent être de cette forme:

- Requête 1: retourner les températures anormales mesurés par tous les capteurs
- Requête 2: Retourner les température mesurées par tous les capteurs au troisième étage toute les minutes.
- Requête 3: Générer une notification à chaque fois que deux senseurs espace d'une certaine distance x mesurent une température anormale simultanément.
- Requête 4: Retourner la température maximale mesurés pendant les dernières 5 minutes.
- Requête 5:Retourner la température moyenne mesurée à chaque étage pendant les 10 dernières minutes

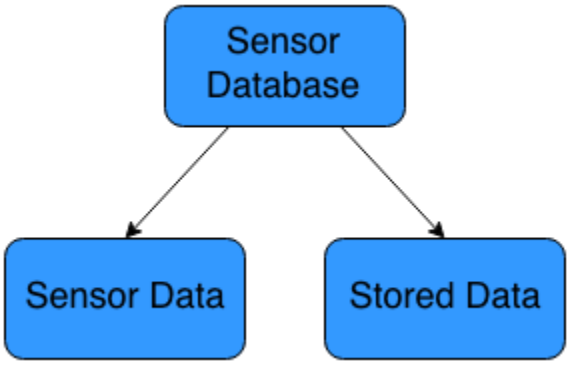
Les requêtes sont faciles à exprimer en faisant une analogie avec les bases de données relationnelles cependant à cause de la nature des données des capteurs qui ne sont pas stocker dans une base centrales et qui changent au cours du temps, il y a une certaine modification à apporter à l'analogie faite avec les bases de données relationnelles. Pour ce faire, il existe deux approches:

- L'approche centralisée dans lequel on sépare le calcul des requêtes qui se fait dans la base centralisée et l'accès au réseau de capteurs qui se fait d'une façon prédéfinie. Cette approche est adapté lorsque les requêtes sont prédéfinies et que les données ne varient pas beaucoup.
- L'approche distribuée: Cette approche est plus flexible car différents requête peuvent extraire différents données du réseau et elle est plus efficace car seules les données pertinentes sont extraites du réseau en plus du fait que cette approche permet au base de données des capteurs d'alléger sa charge de travail en distribuant les calculs dans les capteurs ainsi une requête peut être exécute dans les capteurs

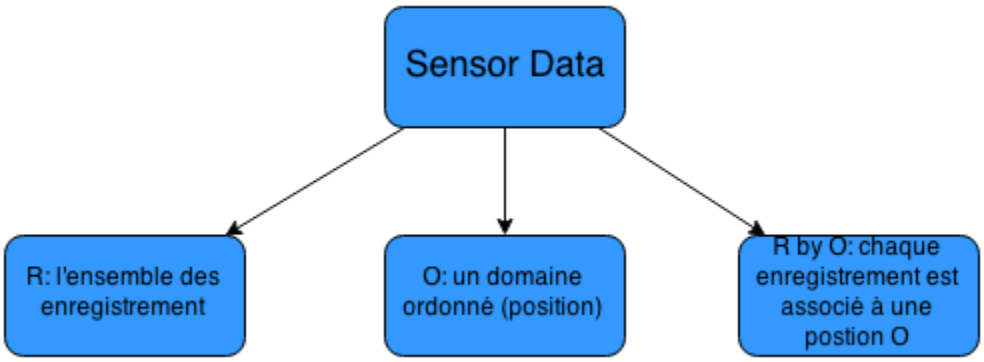
1 Un modelé de base de données de senseurs

1.1 les données des capteurs

- les "Stores Data" sont stockés dans la base de données des capteurs. Ce sont l'ensemble des caractéristiques des capteurs (Ex. la position du capteur) ou des caractéristique de l'environnement physique. Ces données sont représenté en relation



- Les "sensor Data" sont générés à partir des fonctions de traitement de signal. Leurs représentation doit faciliter la formulation de requête de senseurs. Comme le temps joue un rôle central dans ces requêtes, les "sensor Data" sont représenté en séries temporelles. C'est une séquence de 3 tuples comme le montre la figure qui suit



Les opérateurs séquences agissant sur ces séquences sont n-aires et produisent un unique résultat séquence. Ces opérateurs incluent: la sélection, la projection, la jointure

1.2 les requêtes de senseurs

A chaque requête est associée un inter val temps de la forme [O, O+T] ou O est l'instant durant lequel la requête a été soumise et T le nombre de quantum durant son exécution.
Une requête représente est une vue persistante reflétant les mises à jour effectuées pendant une période T donnée

2 Le système de base de données COUGAR

Les capteurs physiques sont représentés dans ce système de base de données relationnelle orienté objet en fonctions d'objet de type Abstract Data Type (ADT). Ce type de représentation permet d'encapsuler les données à travers des fonctions. Les sorties répétitives des fonctions d'un ADT sont le résultats d'exécution successives des ces fonctions scalaires. Ce choix entraîne des contraintes concernât l'expression d'une contrainte temporelle.
Les requêtes sont formulées en SQL avec une légère modification. Le "From" doit contenir un attribut d'un ou plusieurs senseurs. Dans ce cas, la base de données des capteurs est une relation *R(loc point, floor int, s sensorNode)* où *loc* est un point ADT dans lequel est enregistré la position d'un capteurs, *floor* est le numéro de l'étage et *sensorNode* est un ADT qui supporte les méthode *getTemp()* et *detectAlarmeTemp(thershold)*
Les requêtes SQL
Les requêtes 4 et 5 ne peuvent pas être exprimés dans cette version car les intervalles temporels ne sont pas supportés.
Les requêtes sont exécutes dans la base de données front-end alors que les fonctions de traitement de signal sont exécutée dans les capteurs appartenant à la requête.

3 Les problèmes avec les fonctions traditionnelles des ADT

Lors de l'exécution d'une fonction ADT, une fonction locale est appelée pour obtenir la valeur de retour. Cette valeur est lisible à la demande si on se refaire à la première version de COUGAR or cette hypothèse n'est pas réalisable car il peut y avoir des problèmes de latence et lors de l'évaluation d'une requête long-execution (durant un intervalle Ex. Requête 5), la fonction ADT retourne plusieurs valeurs. Afin de régler ce problème, un nouvel opérateur a été introduit, c'est le variante de la jointure entre la relation qui contient l'attribut du capteur et la fonction représenté sous une forme tabulaire. cette dernière est une relation virtuelle qui contient les argument d'entrée et de sortie d'une méthode. Comme exemple, si une méthode M prend m arguments alors le schéma de la relation virtuelle est de m+3, où le premier attribut est l'id du capteur, le deuxième jusqu'à m+1 correspondent aux arguments de M, m+2 est la valeur de sortie de M, et m+3 l'intervalle de temps qui a été nécessaire pour obtenir le résultat de sortie.

id	arguments de M (m colonnes)	résultat de M	durée d'exécution

On ajoute une donnée à une relation virtuelle que lorsque une fonction retourne un résultat. Ces relations sont réparties sur un ensemble de noeuds.

Le tableau ci-dessous représente l'emplacement de chaque entité:

Base de données d'un capteur	distribué sur chaque capteur
Relation virtuelle	distribué sur un ensemble de capteurs
Relation de base	base de donnée front-end

4 Plan d'exécution d'une requête

Les relations virtuelles se retrouvent dans le plan d'exécution d'une requête au même titre qu'une relation de base. Elles sont accessibles à travers un scan virtuel indexé. Les jointures virtuelles sont une façon efficace de régler le problème des requêtes qui s'exécutent sur un intervalle de temps et les requête bloquantes.

- Comparaison

Suite à l'analyse des articles précédant, chaque partie traite un aspect différent des réseaux de senseurs. En effet, la première partie traite de l'optimisation des requêtes ainsi que la consommation d'énergie. La seconde borde la transmission des données entre chaque noeud ainsi que les protocoles à utiliser. La dernière partie débat de la mise en place d'un système qui permet d'interroger des capteurs avec différents paramètres.

- Synthèse

Les différentes technologies présentées permettent d'avoir une vue assez générale sur les applications qui existent utilisant les réseaux de capteurs. Ces réseaux sont des entités composes de plusieurs noeuds ou chaque noeud est un composants fonctionnant indépendamment des autres noeuds. Ayant une capacité de stockage, de traitement, et de transfert des données, ces capteurs forment un réseau qui peut être assimile a une base de données distribuées où un ensemble de noeuds représente un fragment de la base. Des opérations sur ces noeuds ont été définit précédemment permettent de distribuer le traitement des requêtes et ainsi bénéficier des plusieurs avantages:
Un gain conséquent en terme de consommation d énergie. Afin d exécuter une requête le système n a plus besoin de collecter les donnes de tout le réseau, il a juste à rassembler les données dont il a besoin d'où les coûts de transfert de données sont revue a la baisse
le temps de réponse diminue car au sein même des noeuds il y a la possibilité d exécuter les requêtes et de ne renvoyer que le résultat
Il est possible d'utiliser un plus grand nombre de noeud. Cette possibilité découle des deux raisons précédentes et du fait que chaque noeud est un composant indépendant.
L'optimisation de la consommation d'énergie est un facteur important, Une des techniques permettant de faire cela est celle citée précédemment. En effet, l'utilisation des ACQP permet à chaque noeud du réseau de fonctionner selon de cycles (traitement et repos) ce qui conduit à une optimisation considérable de l'énergie.
Des techniques d'exécution de requête dans ce réseau ont été cite. Ces techniques permettent d'extraire les informations désirées avec une optimisation assez poussée. Ce qui est intéressant, c'est de remarquer que

les techniques citées ressemblent beaucoup aux techniques utilisées dans les bases de données distribuées ou chaque fragment représente un noeuds qui est capable de stocker et traiter des requête et de transférer des données.
Concernant la transmission des données, il est préférable d'utiliser le protocole AODV comme protocole de transmission. De plus, utiliser la partial agrégation permet de paralléliser les requêtes et ainsi avoir un coût pour une requête plus petite.

bibliographie:

- J. Gherke, S. Madden. (2004). Query Processing in Sensor Networks. *IEEE Pervasive Computing*, 3(1), 46-55.
- S.R. Madden. (2003). The Design and Evaluation of a Query Processing Architecture for Sensor Networks. Proc. MDM 2001.
- P. Bonnet, J. Gehrke, P. Seshadri. (2001). Towards sensor database systems. Proc. MDM 2001.
- S. Madden, M.J. Franklin, J.M. Hellerstein. (2005). TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. on Database Sys.*, 30(1), 122-173.
- H.C.Le ,Optimisation d'accès au médium et stockage de données distribuées dans les réseaux de capteurs
- Jaydip SenWireless Sensor Network Databases Innovation Labs, Tata Consultancy Services