A DISTRIBUTED QUERY PROCESSING ENGINE

Supriyo Chatterjea[(1)], Paul Havinga[(2)]

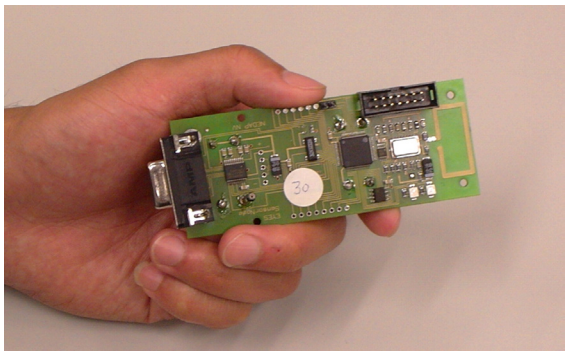
[(1)]Department of Computer Science, University of Twente, P.O. Box 217, 7500AE Enschede, The Netherlands, Email: supriyo@cs.utwente.nl
[(2)]Department of Computer Science, University of Twente, P.O. Box 217, 7500AE Enschede, The Netherlands, Email: havinga@cs.utwente.nl

ABSTRACT

Wireless sensor networks (WSNs) are formed of tiny, highly energy-constrained sensor nodes that are equipped with wireless transceivers. They may be mobile and are usually deployed in large numbers in unfamiliar environments. The nodes communicate with one another by autonomously creating ad-hoc networks which are subsequently used to gather sensor data. WSNs also process the data within the network itself and only forward the result to the requesting node. This is referred to as in-network data aggregation and results in the substantial reduction of the amount of data that needs to be transmitted by any single node in the network. In this paper we present a framework for a distributed query processing engine (DQPE) which would allow sensor nodes to examine incoming queries and autonomously perform query optimisation using information available locally. Such qualities make a WSN the perfect tool to carryout environmental monitoring in future planetary exploration missions in a reliable and cost effective manner.

1. INTRODUCTION

Wireless sensor networks (WSNs) are formed of tiny, extremely low-powered (typically around 16 milliwatts compared to the average notebook computer which runs at around 2 watts) sensor nodes that are equipped with built-in wireless transceivers. Fig. 1 shows a picture of a development board of an EYES [1] sensor node together with some of its specifications. It is currently used for testing purposes and the final version of an EYES node should be around the size of a one Euro coin. These nodes, which may be mobile, can be deployed in large numbers in unfamiliar environments. Once deployed, the nodes are capable of communicating with one another by autonomously creating ad-hoc networks which are subsequently used to gather sensor data. Considering the fact that these battery-powered nodes are supposed to operate for months (possibly even years) and that it is assumed that battery replacement is not a viable option due to the large numbers, one of the primary concerns of wireless sensor networks is how to extend the longevity of the network to the furthest possible extent.



Specifications: 16-bit, 5MHz processor, 60kB ROM, 2kB RAM, 2MB EEPROM, 115.2kbps data transfer rate,
Power consumption: 16mW (Transmit), 14.4mW (Receive), 0.015mW (Standby)

Fig. 1. A development board of an EYES sensor node

In this paper we first give a brief overview of one of the main techniques, data aggregation, that may be employed to minimise energy consumption of nodes in Section 2.1. Section 2.2 describes the architectures used in a few existing projects dealing with WSNs and highlights certain deficiencies in their designs with regards to energy consumption. After the concise primer on WSNs, section 3.2 explains why and how we feel wireless sensor networks can play a dominant role in data gathering operations in future planetary exploration missions by illustrating with a short example. Keeping the operational requirements of WSNs in mind, we then present a framework of a distributed query processing engine that will allow wireless sensor nodes deployed on the surface of a planet to autonomously gather and analyse data on-site in an energy efficient manner. We finally conclude the paper by stating the work that needs to be done in the future to build up on the framework presented.

Our work on WSNs is performed as part of the NWO funded CONSENSUS project [2] and the European EYES project (IST-2001-34734) [1] on self-organising and collaborative energy-efficient sensor networks. It addresses the convergence of distributed information processing, wireless communication and mobile computing.

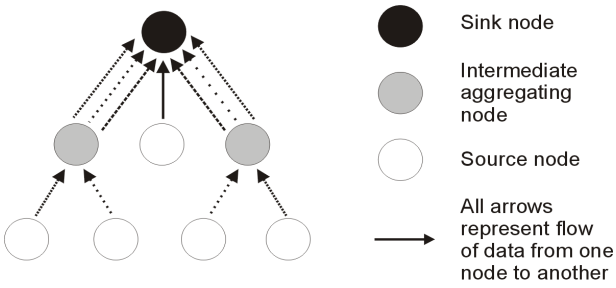


Fig. 2. Data flow without aggregation

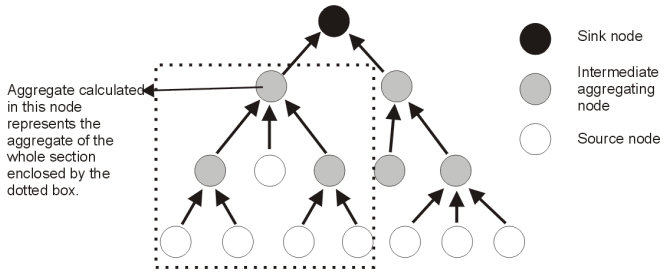


Fig. 3. Data flow with aggregation

2. BACKGROUND

2.1. Overview of data aggregation

The most energy consuming operation a node can perform is the transmission of data. In fact, transmitting just 1Kb of data a distance of 100 metres is approximately equal to the cost of executing three million CPU instructions [6]. Keeping this fact in mind, apart from simply collecting data, WSNs are designed to process data within the network itself and subsequently forward the result to the requesting node. This is referred to as in-network data aggregation and results in the substantial reduction in the amount of data that needs to be transmitted within the network as a whole which in turn translates into substantial energy savings as shown in Fig. 2 and Fig. 3. Data aggregation can be performed by intermediate nodes that lie between the sink (a node that injects a query into a network) and source (a node that responds to a query by sensing some physical parameter) node. These intermediate nodes carry out partial computation of the data obtained from sensor nodes thus ensuring that each node only has to transmit one data message. This also implies that bandwidth requirements between neighbouring nodes remain constant regardless of their position in the tree.

2.2. Architectures of current query processing systems

While performing data aggregation within the network may result in extending the operational lifetime of sensor nodes, it is important to note that there may be numerous ways to evaluate any particular query. Out of all these possibilities, only a handful might actually lead to energy savings. Thus it is important to develop a system that can analyse every incoming query and work out the optimal solution using the *current* network dynamics to ensure accurate decision making. In certain existing query processing systems such as COUGAR [3] and TinyDB [5], network statistics (or *network metadata*) such as patterns of data produced by individual nodes, location information and energy reserves of nodes, etc. are sent back periodically to the central node (server) which originally injected a query into the node.

Using the centrally collected data, the server, which now has a detailed overview of the status of the entire network of nodes, calculates the optimal method in which the query may be evaluated. So the server generates a set of instructions that are then sent out to the individual nodes explaining the role individual nodes will play in evaluating the query, e.g. the server may stipulate which specific nodes would be required to perform aggregation of data.

This central architecture has a number of inherent drawbacks. Firstly, as stated earlier, transmission of data is the most energy consuming operation that can be performed by a node. Thus having every node relay its network metadata back to the server on a periodic basis is a very expensive operation due to the high amount of overhead involved. Also, since network metadata is only relayed back periodically, it is not possible for the server to always maintain an updated view of the whole network. Thus a node which is pre-assigned by a server to carry out aggregation may not actually be available once the server sends out the specific instructions to evaluate a certain query (e.g. a node might die due to loss of power). Naturally having a continuously updated view would also mean that traffic would increase exponentially closer to the server and this would result in network delays.

3. A DISTRIBUTED QUERY PROCESSING ENGINE

3.1. Overview

In order to address the problems mentioned above, we have suggested a new framework for a *completely* distributed query processing engine (DQPE) for wireless sensor networks. The primary difference from the existing models is that in our framework, we transfer the task of generating query plans from a central server to the sensor nodes that lie within the network.

Therefore, instead of query plans being generated using a single global view, nodes generate query plans using information that is available locally. The plan that is eventually generated may not be optimal compared to the one that is generated centrally but substantial savings would be made in terms of transmissions. This is because network metadata from any particular node would not have to be relayed all the way back to the server but would only be dissipated in the vicinity of the node itself. As nodes rely fully on locally available data, every node would be able to detect changes in its vicinity and make necessary changes to its query execution methods almost immediately. Also, the user who injects the query into the network need not bother about how to evaluate a certain query in an efficient manner under the current network conditions, i.e. the query evaluation procedure is carried out autonomously by the nodes and is completely transparent to the user. The user need not be concerned about the current network dynamics.

3.2. A possible application scenario

We believe that due to the above mentioned properties of our framework for WSNs, they would fit perfectly into the niche of gathering data in future planetary exploration missions where long node lifespan, reliability, fully autonomous operation and the possibility of covering large geographic areas without incurring high costs are of paramount importance. To illustrate this idea, we build on the BepiColombo mission as an example [4]. In this mission, it may be possible for the lander of the Mercury Surface Element (MSE) to scatter a large number of sensors nodes (by the hundreds or thousands) seconds before the lander touches down onto the surface of Mercury. This would allow a large geographical area to be examined in an energy-efficient manner without actually requiring a rover to move around and gather data. Upon landing, the MSE could send out queries to the thousands of sensor nodes scattered around it. Once a query has been received, the nodes within the network will set up routes and query plans automatically, collect data, aggregate them and start sending results back to the MSE. Due to the hostile environmental conditions on the surface of the planet, and also due to the limited power supply of each node, the sensor nodes will be prone to failure after a certain amount of time. However, having a high density of sensor nodes will allow the nodes to autonomously modify existing routing mechanisms and query execution plans to adapt to the volatile network conditions thus increasing the robustness of the network as a whole.

3.3. Architecture

The DQPE, which lies on top of the operating system, is structured as shown in Fig. 4. The seven components shown in Fig. 4 can be separated into two sections based on their functionality. Blocks 1, 2, 3 and 7 are the main components involved in breaking down an incoming query, analysing it, optimising it and subsequently sending out a restructured query to a neighbouring node. Blocks 4, 5 and 6 form a feedback loop that is used by the DQPE to check if the incoming results match up to expectations. The feedback loop is also used to update the network metadata parameters to ensure that optimisation performed on future queries can be carried accurately using the most recent network statistics. The following subsections describe the role of each block in Fig. 4.

3.3.1. Query Decomposition

This is the first block that encounters an incoming query. At this stage, it is assumed that incoming queries are syntactically correct. Queries injected into a network will be in binary format. We currently assume that the syntax of the query language will be similar to SQL and may be modified to support more complex data aggregation functions, depending on the requirements of the application. An incoming query is broken down and analysed semantically to ensure that an incorrect query is detected and rejected as early as possible. For instance a query that requests two incompatible tables to be joined would result in an error. Redundancies in queries are also eliminated using certain idempotency rules. In the last stage of query decomposition, a list of *operator trees* are generated using *transformation rules*. Each operator tree describes a single way in which the incoming query can be interpreted and subsequently executed. Although all the operator trees may be equivalent in terms of the final result obtained, some might require a higher execution cost than others. In order to save the Query Optimisation block 7) from having to compare all the possible trees based on their predicted cost, transformation rules are used to restructure operator trees in a systematic way so that "bad" operator trees are eliminated at the very first stage itself.

3.3.2. Data localisation

The query decomposition block did not take the distribution of data into account. The main role of the data localisation block is to localise the query's data using data distribution information which is obtained from the Fragmentation Schema block.

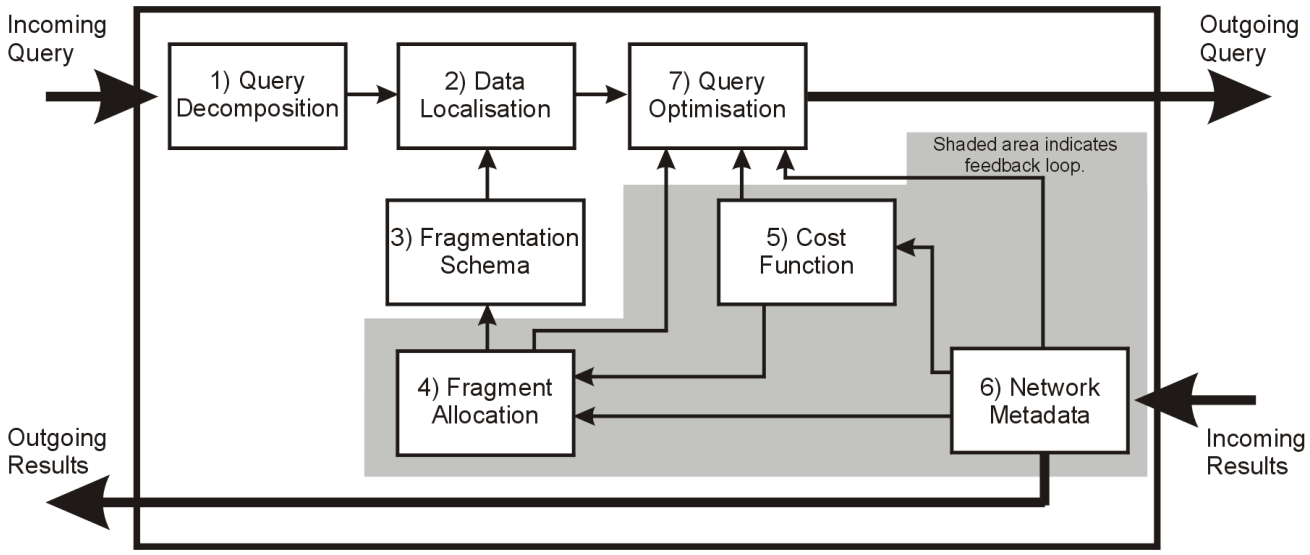


Fig. 4. Framework of the distributed query processing

As data in sensor networks may be flowing as a stream of data, we plan to cache some of the accumulated data at certain nodes throughout the network. Naturally, due to the memory constraints of every node, it will not be possible to keep all the data cached in a single node. Thus the collected data needs to be *fragmented* among several nodes. More details about fragmentation are mentioned under the Fragment Schema and Fragment Allocation blocks in Sections 3.3.3 and 3.3.4 respectively.

Thus data localisation examines the incoming query from the query decomposition block and determines which fragments of data are involved in the query. It also retransforms incoming queries into simpler and more optimised forms by using different reduction techniques depending on how the data has been fragmented. For example, when selections on fragments are made that have a qualification contradicting the qualification of a fragmentation rule, empty relations (or redundancies) are generated. Reduction rules ensure that such empty relations are eliminated.

3.3.3. Fragmentation schema

Data from sensors may be cached in certain sensors in tables (also known as *relations*). As mentioned earlier, it may not be possible to store all the data in a single node. Thus relations may be horizontally or vertically fragmented among several nodes. The purpose of the Fragmentation Schema is to describe how the various fragments may be related to one another to form a complete relation. The Fragmentation Schema extracts this information from the Fragment Allocation block. Fig. 5 shows how a relation, R, may be divided into five separate fragments. Using the fragmentation information, it would be possible to reconstruct or *materialise* the relationship R from the various fragments.

3.3.4. Fragment allocation

The task of the Fragment Allocation block is to decide at which node a certain fragment of a relation should be stored. Suppose there are a set of fragments $F = \{F_1, F_2,\ldots, F_N\}$ and a network of sensor nodes, $S = \{S_1, S_2,\ldots, S_N\}$, the Fragment Allocator needs to find the optimal distribution of F to S. There are numerous parameters that need to be considered during the optimisation process, e.g. cost of communication between any two pairs of sites, S_i and S_j, varying access patterns of various nodes, mobility patterns, cost of storing each F_i at a site S_j, cost of querying F_i at a site S_j, remaining energy reserves of a node and performance parameters such as throughput and response time. The Fragment Allocator will also have a part to play in taking care of the reliability issues by deciding on whether certain fragments need to be *replicated*, and if so the strategy of replication required. While the allocator attempts to minimise the combined cost, it is important to note that due to the large number of parameters involved obtaining an optimal solution is not computationally feasible. This is all the more true when considering the limited processing power of each individual node. Thus the main strategy will be to attempt to find good heuristics which in turn can be used to provide suboptimal solutions.

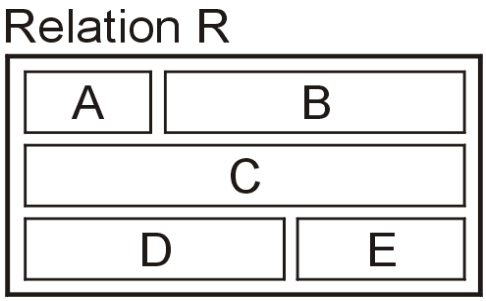


Fig. 5. Fragmentation of Relation R

3.3.5. Cost function

The Fragment Allocator makes decisions based on the inputs it receives from the Cost Function and Network Metadata Block. The Cost Function block can be viewed as a database that stores the cost of some of the parameters considered by the Fragment Allocator such as inter-node communication cost, cost of storing at and querying a particular node. There are typically two categories of cost parameters stored by the cost function. The first consists of a set of parameters whose costs are static and are generally defined prior to deployment of the sensor network. For example, the nodes may be pre-programmed with a set of MAC and routing protocols each of which would have a fixed cost depending on the task at hand. The second category is made up of parameters whose costs are dynamic and may change depending on the network conditions, e.g. the cost of communication between nodes S_i and S_j may vary depending on the current traffic conditions, distance between the two nodes, etc. The costs of these dynamic parameters are obtained from the Network Metadata Block which is described below.

3.3.6. Network metadata

The network metadata block plays a crucial role in the overall performance of the DQPE as it provides the input to the Query Optimisation, Fragment Allocation and Cost Function blocks which together hold the key to how well a particular node responds to the dynamics of the network. This block monitors every single query or result that the node hears, i.e. the query or result may not be addressed to it specifically but it might overhear a certain message from a neighbouring node that is within its transmission range. This allows the node to gather statistics about other node operations in its local environment. Sizes of relations, patterns of query flow, energy reserves of a particular node, mobility issues are just some of the parameters that may be monitored. While certain parameters may be specifically broadcast by a neighbouring node, others may be inferred by the receiving node by analysing message packets that are overheard. Since transmission of a message by a node is such an energy consuming process, eavesdropping on messages helps to ensure that every message transmitted is utilised to the maximum.

3.3.7. Query optimisation

The Query Optimisation block receives several execution strategies (or operator trees) for a single query from the Data Localisation Block. It is within this block that the DQPE actually takes into account the distribution of data fragments, various costs involved and the current network dynamics in order to generate a query execution plan.

At this stage, the query is partially optimised and information about how certain fragments can be reconstructed using certain inter-fragment relationships is presented to the Query Optimisation block. The optimiser tries to perform the most selective operations that reduce the amount of data involved as early as possible.

Theoretically, this block should try to choose the best possible solution in the solution space of all possible execution strategies by comprehensively predicting the cost of each and every strategy and subsequently selecting the strategy with the minimum cost. Since the solution space may be extremely large (due to the large number of parameters involved) measures need to be taken to try and obtain solutions which are “very good” rather than perfect. The amount of resources spent on optimising a certain query might well be dependent on the query itself. For instance, if a certain query is subsequently followed by multiple executions, it might be wise to initially spend a little more effort on the optimisation phase.

4. CONCLUSION AND FUTURE WORK

We have presented a framework that would allow sensor nodes to execute queries autonomously in an energy efficient manner so as to extend the longevity of the network as a whole. This is done by transferring the process of query optimisation and planning, to within the network itself. However, there are a number of issues which require further study. Firstly, we are assuming that there will be two types of nodes within the network – nodes which have sensors attached to them, and nodes which act as gateways. While both sensor nodes and gateway nodes may be able to perform query optimisations, preference will be given to the gateway nodes. The degree of optimisation that can be performed by each node needs to be examined. Also, measures need to be introduced to control the number of nodes that are involved in the query optimisation process for any single query, as latency issues need to

be taken into account. While we have mentioned that the nodes will be making use of only local information, we have not specifically explained what is meant by "local information". This is because simulations need to be carried out to study the tradeoffs between efficiency and using local information that may be one, two or even three hop counts away. Also, considering the complexity of the framework presented here, the current memory specifications of the EYES nodes are unlikely to be adequate. A prototype implementation would indicate the actual memory requirements and it would then be possible to measure the tradeoffs between performance and memory and strike an acceptable balance.

One of primary considerations of every planetary exploration mission is to execute the mission using a small budget and yet it should be a highly reliable system – one that is unlikely to fail even under the hostile environmental conditions that may be experienced in space. We believe that using the framework presented, WSNs would then provide the perfect solution as scientists can migrate away from a system where environmental measurements are obtained by high cost vehicles such as rovers, which are prone to mechanical failure and are also unable to cover large geographical areas.

5. REFERENCES

1. EYES homepage: http://eyes.eu.org.
2. CONSENSUS homepage: http://www.consensus.tudelft.nl.
3. Bonnet P., Gehrke J. and Seshadri P. Towards sensor database systems. In *2nd International Conference on Mobile Data Management*, Hong Kong, January 2001.
4. Grard R., Novara M., and Scoon G.. BepiColombo – A Multidisciplinary Mission to the Hot Planet. In *ESA Bulletin*, 1, pp. 11-19, Aug. 2000.
5. Madden S., Szewczyk R., Franklin M. J. and Culler D. Supporting Aggregate Queries Over Ad-Hoc Wireless Sensor Networks. In *4th IEEE Workshop on Mobile Computing Systems and Applications*, Callicon, NY, June 2002.
6. Pottie G. and Kaiser W., Wireless integrated network sensors. In *Communications of the ACM*, 2000.