ENGINEERING JOURNAL

Article

An Analysis of Deductive-Query Processing Approaches for Logic Macroprograms in Wireless Sensor Networks

Supasate Choochaisri[a] **and Chalermek Intanagonwiwat**[b,*]

Department of Computer Engineering, Chulalongkorn University, Bangkok 10330, Thailand
E-mail: supasate.c@student.chula.ac.th[a], chalermek.i@chula.ac.th[b,*]

Abstract. Logic macroprogramming paradigms for wireless sensor networks (WSNs) are rule-based abstractions for programming a network as a whole. Programmers only focus on the main objective of the network rather than the low-level implementation details on each node. Therefore, the low-level details are automatically handled by underlying middleware of the paradigms. To be viable, the middleware must efficiently handle the underlying issues as well as effectively minimize energy consumption and communication overhead. Not surprisingly, one major underlying issue in logic macroprogramming systems is deductive-query processing. In this paper, we analyze the characteristics of deductive-query processing and identify what have been overlooked in those previous approaches. Furthermore, we overview, analyze, and compare several recent approaches for deductive-query processing of logic macroprograms in WSNs. Our analysis reveals several important aspects that should be considered when designing such systems.

Keywords: Logic programming, macroprogramming, query processing, wireless sensor networks, analysis.

ENGINEERING JOURNAL Volume 16 Issue 4
Received 10 January 2012
Accepted 27 March 2012
Published 1 July 2012
Online at http://www.engj.org/
DOI:10.4186/ej.2012.16.4.47

DOI:10.4186/ej.2012.16.4.47

1. Introduction

Wireless-Sensor-Network (WSN) programming is notoriously tedious and difficult. To simplify WSN programming, several novel programming paradigms have been proposed. Many paradigms focus on programming the WSN as a whole (*i.e.*, Macroprogramming) [1, 2, 3, 4, 5, 6, 7, 8, 9] instead of multiple networked entities. In these macroprogramming paradigms, programmers only focus on the main objective of the network rather than the low-level implementation details of each node.

Some paradigms simplify WSN programming into logic programming [10, 11, 12]. With these paradigms, WSNs can be programmed declaratively and imperatively.

There are two classes of logic programming paradigms for WSNs: node-dependent and node-independent. In the node-dependent approaches, local node behaviors must be explicitly logic-programmed [10]. Conversely, WSNs are logic macroprogrammed in the node-independent approaches [11, 12]. Logic macroprogrammers can focus on how to use sensed data from the WSN rather than how to explicitly specify the low-level details (*e.g.*, local node behaviors, communications, query-processing techniques). The low-level details are automatically handled by the underlying middleware. To be viable, the middleware must minimize energy consumption and communication overhead due to limited resources of each sensor node.

A logic macroprogram contains facts and rules with at least one sub-goal. Users can query the system for answers related to the written rules. Query processing in WSNs is not trivial. To satisfy that query, the run-time engine (*i.e.*, middleware) attempts to match the query with a rule or a fact. This may result in cascading sub-queries because all sub-goals of the rule must also be matched. Furthermore, facts (*e.g.*, current sensor readings) in WSNs are dynamic values. It is unlikely to know priori which nodes currently sense values that can match the query. Therefore, communication overhead is unavoidably incurred and efficient query processing is undoubtedly required.

However, traditional query-processing approaches are designed for computers with plenty of power supply and processing power. In contrast, a WSN consists of several wireless sensing devices with extremely limited resources. As a result, novel query-processing mechanisms are required. Recently, we have proposed LogicQ [13], a semi-centralized approach for deductive sub-query processing. Later, Gupta *et al.* [7, 14] have presented various fully-distributed approaches. Nevertheless, no prior work indicates any design choice by considering the characteristics of logic macroprograms for WSNs. Consequently, in this paper, we point out these main characteristics as well as analyze and compare each query processing approach.

The rest of the paper is organized as follows. Section 2 covers the related works about logic programming for WSNs. In section 3, we introduce the fundamental concept of logic macroprogramming in WSNs. We briefly describe various logic-macroprogramming approaches for WSNs in section 4 and our approach in section 5. Then, in section 6, we analyze the cost of each approach and evaluate in section 7. We discuss further optimization in section 8. Finally, section 9 concludes the paper.

2. Related Work

During these past few years, researchers have simplified WSN programming into logic programming. Chu *et al.* [10] have proposed Snlog for logic-programming low-level details of each node. Unlike Snlog, our work focuses on a global abstraction that conceals low-level details from programmers.

The Semantic Streams framework [12] is a rule-based macroprogramming scheme with the semantic services. Services are described by a service description language whereas queries and rules are described by a logic-programming language. However, the Semantic Streams framework focuses on a service-oriented approach for wired sensor networks with plenty of power supply. In contrast, our work focuses on the deductive-query processing for wireless sensor nodes with extremely limited resources.

Recently, Gupta *et al.* [11, 14] have proposed a deductive framework for logic-macroprogramming that shares the same objective with our work. They present distributed approaches for joining related facts to produce complete answers for a query. The main focuses of their approaches are load-balancing and extending the network lifetime. However, their design choices do not consider some essential characteristics of a logic-macroprogram for WSNs. Due to these characteristics, their approaches are unavoidably load-unbalanced and practically power-inefficient. We directly compare our approach with various approaches presented in their paper.

3. Fundamental of Logic Macroprogramming

In this section, we describe the terminology of a logic macroprogram and a query-evaluation process. Our examples are quite simple but sufficiently general. The concept can be easily extended to more realistic complex applications as in [11, 12].

3.1. Terminology

A *logic-macroprogram* consists of *predicates*, *facts*, *rules*, and *queries*. A *predicate* is a relation of data (or a table in the relational-database terminology). For example, a predicate *temperature*(*NodeID, TemperatureValue*) is a relation named *temperature* of two variable arguments: *NodeID* and *TemperatureValue*.

A *fact* is a predicate whose arguments are all constants. For example, *temperature*(3,25) is a fact indicating that a node with ID 3 senses a temperature of 25 degree Celsius.

A *rule* is a clause that specifies a condition to deduce new facts from existing facts in the system. A rule contains head and body parts. For example, the following rule represents a condition to deduce *overThreshold* facts of nodes that sense temperature and humidity over some certain thresholds.

$$\begin{aligned} overThreshold(ID) :- \; & temperature(ID,T), \\ & humidity(ID,H), \\ & T > 50, H < 30. \end{aligned} \tag{1}$$

A query is a question that a user asks to retrieve data from the system. For example, ?-*overThreshold*(*X*) is a query for retrieving IDs of all nodes that satisfy the *overThreshold*(*ID*) rule. A query can be classified into two main classes; a query for retrieving all available answers and a query for existence checking. For instance, the above query ?-*overThreshold*(*X*) requests for all facts or deduced facts named *overThereshold*. Meanwhile, ?-*overThreshold*(5) only checks whether there exists such a fact or a deduced fact *overThreshold*(5) in the network or not.

We note that our logic-macroprogram can contain rules that specify *global* constraints or relations of facts across the entire network. Conversely, a Snlog logic-program [10] can contain only rules that specify local constraints or relations of facts in the same node.

3.2. Query Processing

Query processing is the crucial procedure of a logic macroprogram run-time engine. When a user sends a query into the system, the run-time engine has to process that query and return desirable answers to the user. Existing approaches (section 4) do not consider some characteristics of WSN logic-macroprogramming in their design. Hence, in this section, we explain the general concept of query processing and identify such main characteristics that affect the design of query processing mechanisms.

When the system receives a query, the run-time engine looks for a rule whose head part matches with the query. Then, each sub-goal (each predicate in the body part) of that matched rule is evaluated. A query is satisfied (has at least one answer) if all sub-goals are satisfied. If there is at least one unsatisfied sub-goal, the run-time engine will attempt to match with the next rule that contains the same head part (but different body part). If there is no satisfying rule, a false boolean flag is returned to the user. Furthermore, if a query is an existence checking query, the run-time engine does not necessarily send any satisfying answer to a user. Only a boolean flag (true or false) is sent.

In each rule, some variable arguments of one sub-goal may be bound in some previous sub-goals (two sub-goals share at least one identical variable argument). Consequently, some facts that seems to match a later sub-goal will not be eligible unless the shared argument is bound with the identical value as in the previous sub-goal. For example, from the above *overThreshold*(*ID*) rule, a fact *humidity*(5, 20) will not satisfy the second sub-goal if there is no fact *temperature*(5, _) whereby an *underscore* represents an arbitrary value. This process can be seen as joining of each sub-goal's predicate like joining each table in a relational database. This observation leads to a system characteristic that some facts of the later evaluated sub-goals may not be useful and not necessary to send toward the query processing engine.

Last but not least, after sending a query, a user generally waits for a reply from the system. Therefore, any answer must be sent back to a station where a user or a sub-process waits for a reply. This characteristic differentiates the query processing algorithm for logic-macroprograms from some algorithms in WSNs.

Those algorithms (*e.g.*, time synchronization, localization) are understandably better when being fully distributed. However, the query processing for logic-macroprograms is not necessary to be fully distributed, given that all answers are routed toward the same node.

In the next section, we describe each query processing approach and we raise some design issues related to the mentioned characteristics of logic-macroprograms for WSNs.

4. Existing Approaches

There are several approaches to process the query of logic macroprograms for WSNs. We classify them as follows.

4.1. Centralized Processing Scheme

The centralized approach has been mentioned in [13, 14] as a simple but inefficient approach. This approach is straightforward; all facts in the network must be periodically sent to the base station. Then, the run-time engine (at the base station) uses all collected facts to process each query. Alternatively, the run-time engine collects facts only when a query is sent into the system. Only facts that match with each sub-goal's predicate are collected. Figure 1 illustrates this scheme.

These fully centralized approaches are not appropriate for resource-limited WSNs because a vast amount of unrelated facts are sent to the base station. However, based on the observed characteristics of query processing, we can certainly improve this centralized approach into a much more energy-efficient one. We explain in details later (see section 5).

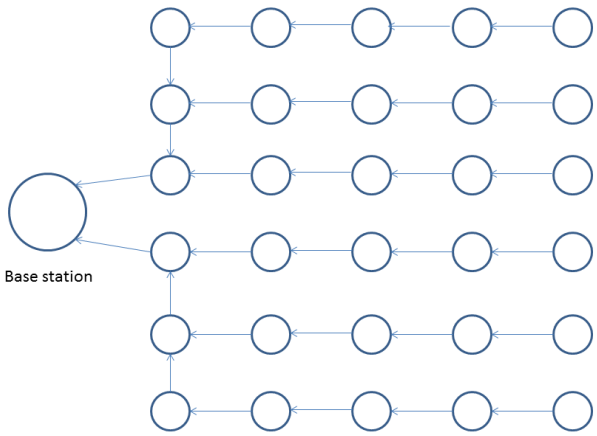

Fig. 1. Centralized Processing Scheme. Nodes route all facts to the base station.

4.2. Distributed Processing Scheme

Various distributed approaches have been concluded and presented in [14]. The main concept of prior presented approaches is to join all sub-goals of a rule in the network without helping from the base station.

A naive-broadcast approach (Fig. 2) is a simple distributed approach whereby each node floods every fact to the entire network. Later, the join process can be performed at any arbitrary node. This approach is undoubtedly infeasible for WSNs because a vast amount of memory is needed to store all facts. Furthermore, an excessive amount of energy is also wasted in flooding those facts.

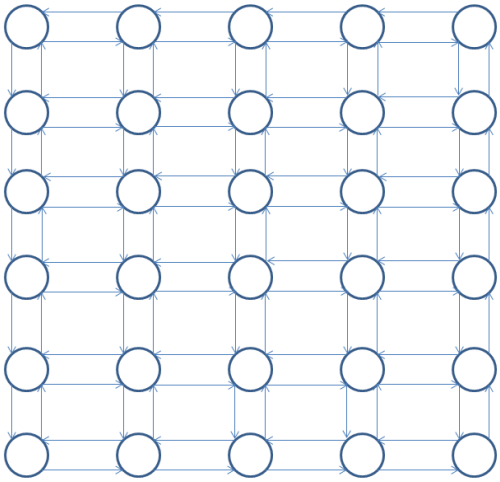

Fig. 2. Distributed Processing Scheme – Naïve Broadcast Approach. Nodes broadcast all facts throughout a network. Each node has a complete set of all facts.

In a Centroid Approach (CA), several centroid regions are constructed. Each centroid area only collects facts of only one sub-goal. Figure 3(a) demonstrates that each node routes its facts to corresponding centroid areas and a center node of each centroid area routes facts to collect at storage nodes in its centroid area. When performing a query evaluation, the process begins at the centroid area that collects facts of the first sub-goal. Then, partial results are sent to the second centroid area. These partial results are joined with collected facts of the second sub-goal to generate new partial results. Generated partial results are sent to the next area and so on. The complete results are generated at the last centroid area. Figure 3(b) demonstrates this process. In this approach, nodes around each centroid area unavoidably run out of energy early.

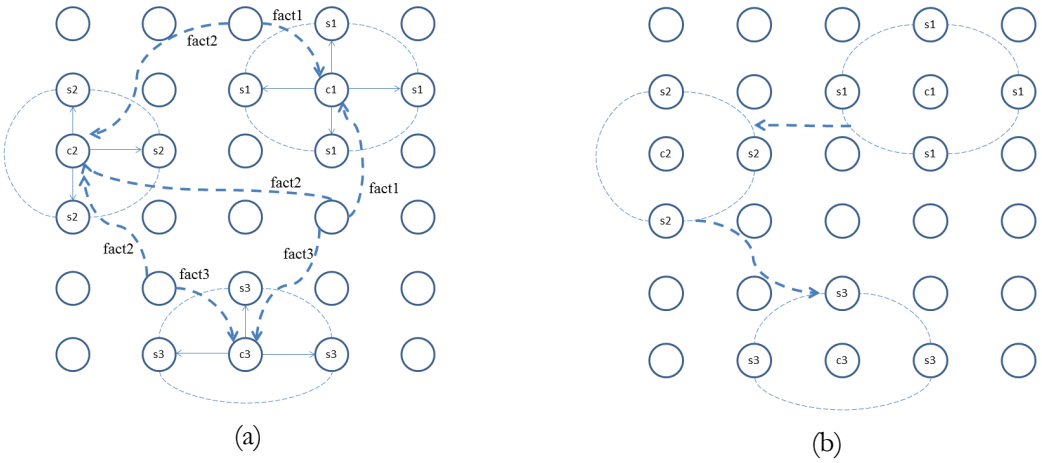

Fig. 3. Distributed Processing Scheme – Centroid Approach. Dash circles represent centroids corresponding to fact types. Thick dash lines represent logical routing from one node/centroid area to another node/centroid area. Thin solid lines represent that center nodes c1, c2, and c3 route facts to collect at storage nodes s1, s2, and s3 in their centroid areas. (a) Facts are collected at corresponding centroids. (b) Facts are routed to join with others from a centroid to another centroid.

DOI:10.4186/ej.2012.16.4.47

The Perpendicular Approach (PA) has been proposed as a load-balanced and communication-efficient approach for WSNs [14]. PA consists of two phases: Storage Phase and Join-computation Phase. In the Storage Phase, every node in the network broadcasts all local facts to be stored on each node along the virtual horizontal grid line. After that, in the Join-computation Phase, each fact of the first sub-goal is disseminated along the vertical grid line to one end. That fact is later disseminated to another end to join with every related fact stored on the vertical grid line.

PA has two join computation schemes: One-Pass Join Computation (OP) and Multi-Pass Join Computation (MP). In OP, nodes (along the vertical grid line) produce all possible partial results although some partial results do not satisfy the query-evaluation process. However, when that fact reaches another end, all complete results (generated by that fact) are stored along that vertical line in one turn. In the other hand, MP routes partial results from one end to another end in multiple turns. In the first turn, only one fact of the first sub-goal is routed to one end. Then, in the second turn, only partial results that are generated by joining that fact with facts of the second sub-goal are routed to other end. Finally, in the final turn, all complete results are generated. Figure 4 illustrates the PA approach.

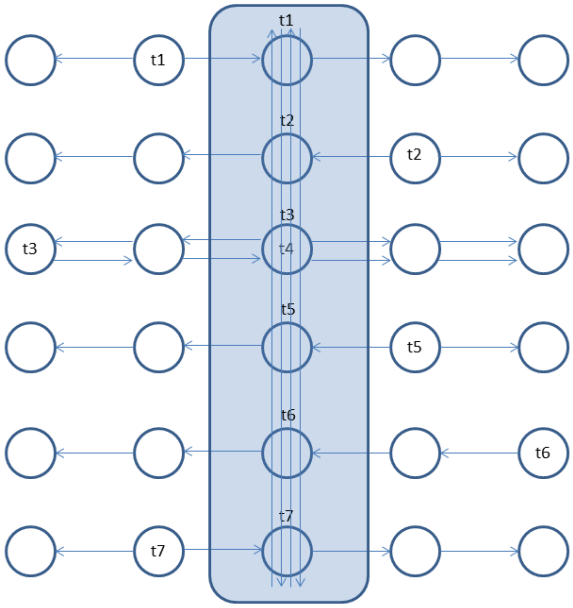

Fig. 4. Distributed Processing Scheme – Perpendicular Approach. Each node route its facts to store in every node along a horizontal line. Then, a node performs evaluation along a vertical line with one-pass or multi-pass.

The PA approach seems to be load-balanced and energy-efficient. However, all generated complete results must also be sent back to the base station. To send all completed results (stored around the network) consumes much energy. Therefore, nodes around the base station also run out of battery early as in the centralized approach. Furthermore, even without an involvement in the query evaluation process, a node may still run out of energy faster than in the centralized approach. The reason is that all nodes have to broadcast several facts from other nodes along the horizontal grid line.

5. Selective On-Demand Processing

The pitfalls of both centralized and distributed schemes presented above are due to no consideration of the mentioned query-processing characteristics in their design.

Given that all answers must be sent to the user at the base station, the design that only creates all complete results distributively stored in the network is not sufficient. Meanwhile, to naively collect all facts to be processed at the base station is not such a good design either.

Recently, our initial work LogicQ [13] is designed as a globally deductive database system for WSNs. The LogicQ system allows a user to write a logic macroprogram and to send a query for answers. The

simulation result of LogicQ in TOSSIM is encouraging but out of scope for this paper (see [13] for more details). The query processing approach of LogicQ is Selective On-Demand Processing Approach (SA).

SA processes a query at a central base station as in the centralized processing scheme. However, facts that cannot satisfy a current sub-goal are not selected or sent. The selection can be done by binding variable arguments of a current sub-goal with instantiated arguments. Then, an argument-bound sub-query is sent into the network.

For example, from the rule (1), if a user sends a query ?-*overThreshold*(2) to the system, the run-time engine will attempt to bind the constant value with all corresponding arguments in the body part. Consequently, the body part becomes *temperature*(2,*T*), *humidity*(2,*H*), $T > 50$, $H < 30$. Then, the run-time engine sends sub-queries ?-*temperature*(2,*T*) and ?-*humidity*(2,*H*) into the network. The temperature and humidity facts are sent back to the base station only when the first argument is 2. Meanwhile, all other facts (*e.g.*, *temperature*(5,30), *humidity*(4,20)) are not sent because those facts do not possibly satisfy the query (according to the mentioned characteristic of query processing). Thus, this processing approach minimizes unnecessary spent energy.

Additionally, we can use another distinguished characteristic about sub-queries to further improve the energy efficiency of the system. As mentioned before, sub-queries can be classified into two types: fact-existence checking and all-answers retrieving. Understandably, we need to disseminate all-answers retrieving queries to the entire network. Conversely, it will not be necessary for a node to further disseminate the fact-existence checking sub-query if that node has a fact to satisfy the sub-query. This sub-query can be suppressed because one fact is sufficient to confirm the existence. Therefore, only one answer is sent back to notify that the fact exists. Finally, only selected facts are processed by the base station. Figure 5 demonstrates this scheme with fact-existence checking.

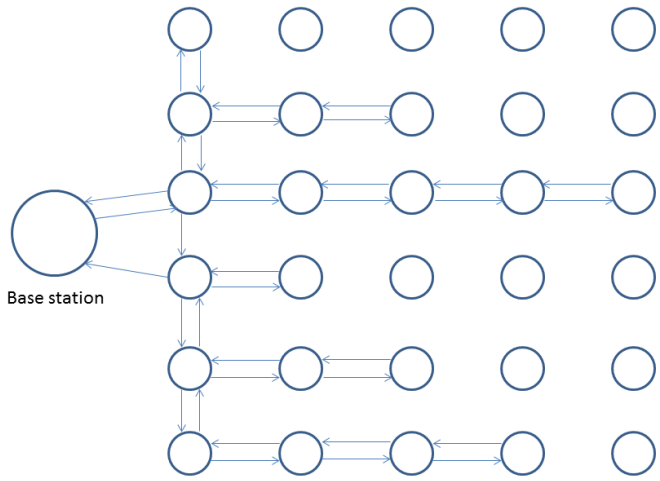

Fig. 5. Selective On-demand Processing Scheme. In fact existence checking, sub-queries do not necessary to reach all nodes and can be suppressed.

This query processing mechanism is simple but efficient. We compare this approach with others in the next section.

6. Cost Analysis

In this section, we analyze the communication cost of three processing schemes: centralized, distributed, and selective on-demand schemes. In all cases, we assume uniform distribution of facts in the network of size *M* nodes. $|G_i|$ represents the number of all available facts of the i^{th} sub-goal. $\sigma_{i_1 i_2}$ is a selectivity factor that is a fraction of the number of joined results to the number of all possible combinations between facts of the i_1^{th} sub-goal and facts of the i_2^{th} sub-goal.

6.1. Cost of Centralized Processing Scheme

Understandably, in centralized scheme, every fact in the network must be collected at the central base station. Therefore, the total communication cost is

$$C_{CP} = M + \overline{D}\sum_{i=1}^{n} |G_i|, \tag{2}$$

where M is the cost of query broadcasting which is equal to the number of nodes in the network and $\overline{D}$ is an average distance from arbitrary node to the base station.

6.2. Cost of Distributed Processing Scheme

In this scheme, we refer to [14] for all base equations but we use $|G_i|$ instead of $|R_i|$ to represent the number of available facts of the i^{th} sub-goal to comply with our terminology for comparison purpose. However, those equations in [14] are about the cost of producing answers from only one tuple of the first sub-goal. Their equations do not include the cost of sending all answers back to the base station. Therefore, we have completed their equations as follow.

6.2.1. Cost of Naïve Broadcast Approach

Although, the naïve broadcast approach is infeasible in practice for WSNs, it can be a base case for comparison.

$$C_{NB} = M\sum_{i=1}^{n} |G_i|, \tag{3}$$

In the naïve broadcast approach, all facts are flooded throughout the network. Hence, the cost of this approach is roughly a product of the network size and the total number of facts.

6.2.2. Cost of Centroid Approach

The cost of the centroid approach presented in [14] is only the cost of finding answers generated from only one fact from the first sub-goal. They do not include the cost of routing the facts to store on centroid storage areas of other sub-goals.

Let D_i be the average distance of each fact of the i^{th} sub-goal to the center node of that sub-goal's centroid area. r is memory capacity of each node. The cost of routing the facts to store on the centroid storage areas is $C_{CA\_Store} = \sum_{i=1}^{n} \left(|G_i| D_i + 1 + 2|G_i|/r\right)$, where $|G_i|/r$ is the minimum number of nodes required to store $|G_i|$ facts. A number of facts of a sub-goal $|G_i|$ is routed to the center node with the average distance D_i. $1 + 2|G_i|/r$ is the cost of searching for nodes with enough memory within the centroid area. The searching procedure is that the searching node broadcasts one storage-request message to neighbors, there are at least $|G_i|/r$ nodes response with sufficient-storage messages. Then the searching nodes routes $|G_i|/r$ messages to $|G_i|/r$ nodes to store facts which are aggregately included in each message. Thus, the total cost of searching is at least $1 + 2|G_i|/r$. Afterwards, each joined partial result is routed from the current sub-goal centroid area to the next sub-goal centroid area on an average of d hops away. Then, the final complete results are sent back to the base station.

Therefore, the overall cost of the centroid approach is

$$C_{CA} = |G_1|\left(\sum_{i=1}^{n-1}\left(d + |G_{i+1}|/r\right)(i)\left(|G_1|\ldots|G_i|\right)\prod_{1\le i_1,i_2\le i}\sigma_{i_1 i_2}\right) + \left(\prod_{1\le i_1,i_2\le n}\sigma_{i_1 i_2}\prod_{i=1}^{n}|G_i|\right)nD_f + C_{CA\_Store}, \quad (4)$$

D_f is the distance in hop(s) from the centroid area of the last sub-goal to the base station. $|G_1|$ is a multiplier to include every possible answer of the first sub-goal.

6.2.3. Cost of Perpendicular Approach

In the perpendicular approach, each fact must be routed to store on nodes along the horizontal line. L_h is an average hop length of the horizontal line. Thus, the cost of storage is $C_{PA\_Store} = L_h\sum_{i=1}^{n}|G_i|$. For both one-pass and multi-pass perpendicular approaches, all complete results are scattered around the network. Therefore, the cost of routing all complete results to the base station is $C_{PA\_Result} = \overline{D}\left(\prod_{1\le i_1,i_2\le n}\sigma_{i_1 i_2}\right)\prod_{i=1}^{n}|G_i|$, where $\overline{D}$ is an average distance in hop(s) from arbitrary node to the base station.

a) One-Pass Perpendicular Approach

For the one-pass perpendicular approach, each fact of the first sub-goal is routed to one end as a starting point. Then, it is routed L hops along the vertical line. All generated partial results except complete results are also routed along the vertical line from their origin nodes to another end. Therefore, the overall cost of the one-pass perpendicular approach is

$$C_{OP} = |G_1|\left(\sum_{l=1}^{L}\sum_{\overline{n}=2}^{n-1}\sum_{i=2}^{n}\overline{n}(L-l)N_l^i(\overline{n})|G_i|/l\right) + C_{PA\_Result} + C_{PA\_Store}. \quad (5)$$

$N_l^i(\overline{n})$ is the number of partial results generated at a node that is l hop(s) away from the starting node at one end.

$\overline{n}$ is the size of a partial result. For example, if a partial result is generated from only two first sub-goals, $\overline{n}$ is two.

b) Multi-pass Perpendicular Approach

For the multi-pass perpendicular approach, in the i^{th} round, partial results of size i are routed in L hops along the vertical line from one end to another end and new generated partial results of size $i + 1$ are routed in average L/2 hops from their origin nodes to another end to prepare for the next round.

Thus, the overall cost of multi-pass perpendicular approach is

$$C_{MP} = |G_1|\left(1.5L\left(\sum_{i=2}^{n-1} i\left(|G_2||G_3|\ldots|G_i|\right)\prod_{1\le i_1,i_2\le i}\sigma_{i_1 i_2}\right) + L\right) + C_{PA\_Result} + C_{PA\_Store}. \quad (6)$$

In both one-pass and multi-pass perpendicular approaches, $|G_1|$ is also a multiplier to include every possible answer of the first sub-goal like in the centroid approach.

6.3. Cost of Selective On-Demand Processing Scheme

In our Selective On-Demand scheme, we perform the join-computation process at the central base station and distributively collect only needed facts from the network.

In retrieving all satisfying answers, our scheme incurs broadcasting a sub-query for each sub-goal. However, only the selected facts (that satisfy the constraints caused by all previous sub-goals) for that sub-

query are sent back to the base station. In other words, previous satisfied sub-goals can suppress many unrelated facts in the network.

$$C_{SA} = nM + \overline{D}\sum_{i=1}^{n}\left(\prod_{1\le i_1,i_2\le i}\sigma_{i_1 i_2}\right)\left|G_i\right|. \tag{7}$$

Noticeably, our approach already includes the cost of producing all answers and the cost of sending all answers to the base station.

In [13], we have proposed two optimization techniques: superset caching and look-ahead data filtering. For superset caching, if a query requests for answers that are a subset of previous cached answers, the runtime engine is able to response with answers immediately without cost of searching in a network. For look ahead-data filtering, if a sub-goal can be bound with a constraint or a constant argument, they can be broadcasted together to filter out unnecessary answers. We refer readers to [13] for more details.

If we take these optimization techniques into account, the cost in Eq. (7) will be reduced to

$$C_{OSA} = p_t\left(nM + \overline{D}\sum_{i=1}^{n}\left(\prod_{1\le i_1,i_2\le i}\sigma_{i_1 i_2}\right)\sigma_{ij}\left|G_i\right|.\right), \tag{8}$$

where $i < j \le n$, from the look-ahead data filtering, σ_{ij} is the probability that a variable argument of the i^{th} sub-goal can be bound with a constant argument of the j^{th} sub-goal. From the superset caching, p_t is 0 if a query is in the superset cache before flushing. Otherwise, p_t is 1. The value of the probability σ_{ij} depends on queries and facts in an environment. Therefore, this probability value can be computed by statistical analysis. One feasible solution is analyzing the expected value in advance. Another feasible solution is collecting data for a while and deriving such a value with statistics from collected information.

7. Evaluation

Our Selective On-Demand Approach (SA) certainly performs better than the centralized and the naïve broadcast approaches do. Meanwhile, the Multi-Pass Perpendicular Approach outperforms the Centroid Approach and the One-Pass Perpendicular Approach when the selectivity factor is not too high [14]. Therefore, to find the best approach, we can simply compare our approach (SA) with MP.

We begin by analytically comparing SA and MP in section 7.1. Then, we conduct a sensitivity analysis to both approaches in section 7.2 – 7.4 to show how a parameter variation affects the performance. In each analysis, we set up an experiment with 25 nodes, 3 sub-goals, and 0.1 selectivity factor. We vary one parameter in each analysis to inspect the effect.

7.1. Analytical Comparison

In this section, we prove that SA performs better than MP in most cases. Formally, $C_{SA} \le C_{MP}$ when $L \le \left|G_i\right|$ and $\overline{D} \le L$ where L is the number of hops along the vertical line, $\left|G_i\right|$ is the number of available facts of the i^{th} sub- goal, and $\overline{D}$ is an average distance from arbitrary node to the base station.

Proof. We assume $\prod_{i=1}^{n}\left|G_i\right| \ge 1$ (there is at least one combination of facts in the network). We note that if the i^{th} sub-goal has no related fact, we assign $\left|G_i\right| = 1$ to prevent a zero production. From the C_{MP} Eq. (6), we expand C_{PA_Result} and C_{PA_Store} to Eq. (9). Then, the equation is rearranged into the three-term Eq. (10) for comparison.

$$C_{MP} = |G_1|\left(1.5L\sum_{i=2}^{n-1}\left(i(|G_2|\ldots|G_i|)\prod_{1\le i_1,i_2\le i}\sigma_{i_1i_2}\right)+L\right)+\overline{D}\left(\prod_{1\le i_1,i_2\le n}\sigma_{i_1i_2}\right)\prod_{i=1}^{n}|G_i|+L_h\sum_{i=1}^{n}|G_i| \quad (9)$$

$$= \sum_{i=2}^{n-1}1.5iL(|G_1|\ldots|G_i|)\prod_{1\le i_1,i_2\le i}\sigma_{i_1i_2}+L|G_1|+\overline{D}\left(\prod_{1\le i_1,i_2\le n}\sigma_{i_1i_2}\right)\prod_{i=1}^{n}|G_i|+L_h\sum_{i=1}^{n}|G_i|$$

$$= \sum_{i=2}^{n-1}\left(L_h+1.5iL(|G_1|\ldots|G_{i-1}|)\prod_{1\le i_1,i_2\le i}\sigma_{i_1i_2}\right)|G_i|+(L_h+L)|G_1|+\left(L_h+\overline{D}\prod_{1\le i_1,i_2\le n}\sigma_{i_1i_2}\prod_{i=1}^{n-1}|G_i|\right)|G_n| \quad (10)$$

From Eq. (7), *M* in SA equals to LL_h in MP. Thus, we obtain the following Eq. (11).

$$C_{SA} = nM+\overline{D}\sum_{i=1}^{n}\left(\prod_{1\le i_1,i_2\le i}\sigma_{i_1i_2}\right)|G_i|$$

$$= \sum_{i=1}^{n}M+\overline{D}\sum_{i=1}^{n}\left(\prod_{1\le i_1,i_2\le i}\sigma_{i_1i_2}\right)|G_i|$$

$$= \sum_{i=1}^{n}\left(LL_h+\overline{D}\left(\prod_{1\le i_1,i_2\le i}\sigma_{i_1i_2}\right)|G_i|\right) \quad (11)$$

We assume $\forall i\in \mathrm{N}: L\le|G_i|$. This assumption is always valid in a homogeneous sensor network and valid in most cases of a heterogeneous sensor network. Therefore, we obtain Eq. (12) and derive it into the three-terms Eq. (13) for comparison.

$$C_{SA}\le\sum_{i=1}^{n}\left(L_h+\overline{D}\prod_{1\le i_1,i_2\le i}\sigma_{i_1i_2}\right)|G_i| \quad (12)$$

$$\le\sum_{i=2}^{n-1}\left(\left(L_h+\overline{D}\prod_{1\le i_1,i_2\le i}\sigma_{i_1i_2}\right)|G_i|\right)+(L_h+\overline{D})|G_1|+\left(L_h+\overline{D}\prod_{1\le i_1,i_2\le n}\sigma_{i_1i_2}\right)|G_n| \quad (13)$$

Finally, we compare Eq. (10) with (13). If $\overline{D}\le L$ (a general case in WSNs), each term in (13) is less than its corresponding term in (10), respectively. Therefore, we conclude that

$$C_{SA}\le C_{MP} \qquad \square$$

We note that even when $L>|G_i|$ or $\overline{D}>L$ (the assumptions are broken), C_{SA} is still mostly lower than C_{MP}. C_{SA} is greater than C_{MP} only when $|G_i|$ is much lower than the number of nodes in a network. In other words, C_{SA} is greater than C_{MP} only when there are a few (satisfiable and unsatisfiable) facts in a network. In Fig. 6, these assumptions are invalid with different network sizes (5x10, 13x6, and 9x11 nodes) where *n* = 3 and selectivity factors = 0.1. We also notice that, when C_{SA} is greater than C_{MP}, the cost difference is not significant.

DOI:10.4186/ej.2012.16.4.47

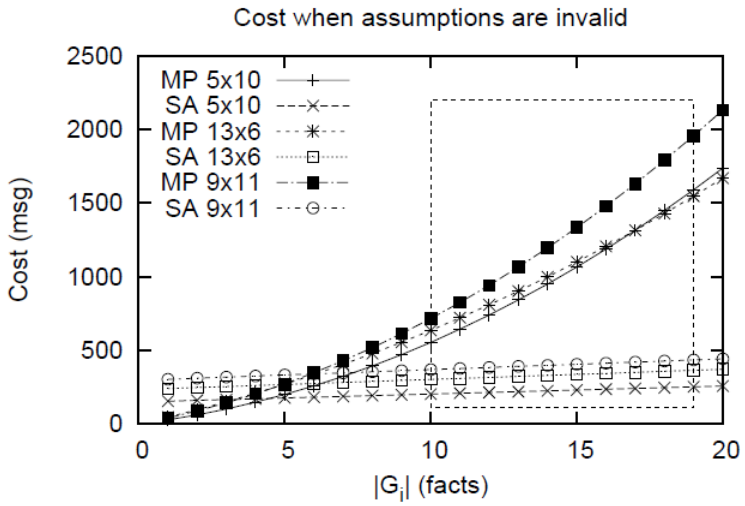

Fig. 6. Cost when assumptions are invalid.

7.2. Effect of Network Size

The effect of the network size indicates how scalable an approach is. We vary the network size and calculate the number of all transmitted messages. In Fig. 7, SA's cost scales linearly because there are only new transmissions from added nodes. Conversely, MP's cost exponentially grows. When the number of nodes increases, there are more possible answers. Therefore, partial results in MP also increase. MP has to send these partial results multiple turns until the complete results are constructed. These increased partial results will also be routed in more hops as the network size increases. Consequently, transmitted messages exponentially grow with the network size.

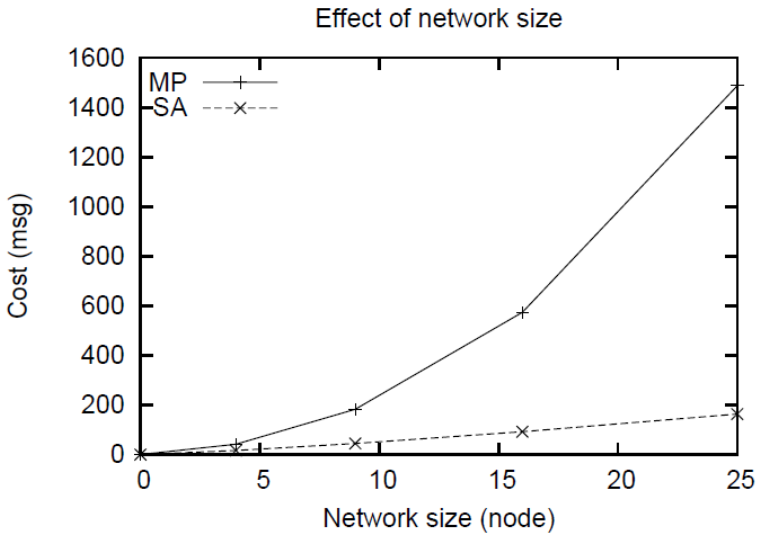

Fig. 7. Effect of network size.

7.3. Effect of Selectivity Factors

In Fig. 8, the higher selectivity factors result in more partial results generated. Similar to the effect of the network size, MP grows exponentially whereas SA is linearly affected. MP is exponentially affected by an increasing number of partial results. However, in SA, the increasing number of partial results can be viewed as there are additional nodes in the network.

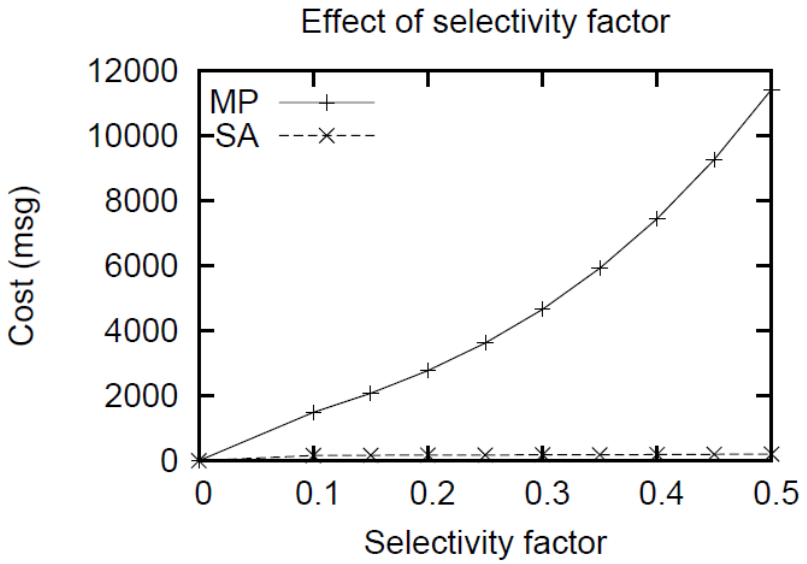

Fig. 8. Effect of selectivity factor.

7.4. Effect of the Number of Sub-goals

Finally, in Fig. 9, we vary the number of sub-goals to evaluate how the complexity of a rule affects the performance. SA grows linearly after the first sub-goal. In SA, all nodes send their facts for the first sub-goal whereas only some nodes send their facts for the later sub-goals. In the other hand, MP grows exponentially for the first three sub-goals due to the increasing number of partial results. After the third sub-goal, the exponential cost component starts to disappear and the cost becomes linear. The reason is that, with 0.1 selectivity factors, later sub-goals have very low probabilities to produce next partial results that are also constrained by previous sub-goals' values.

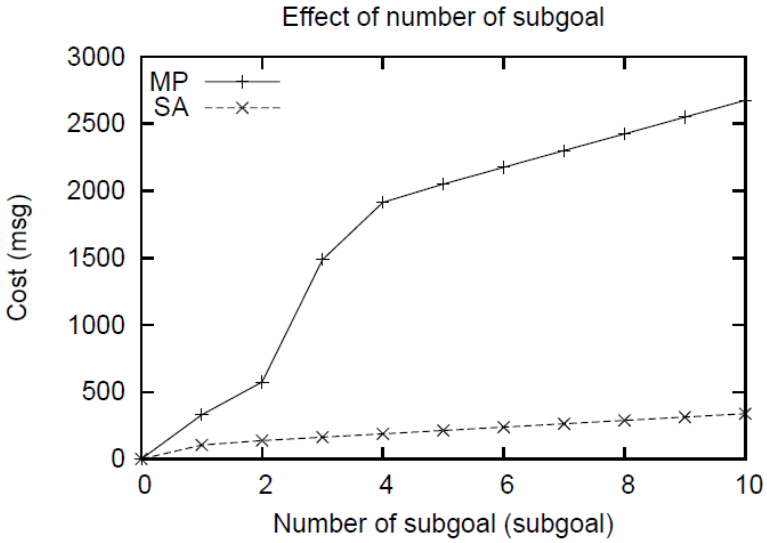

Fig. 9. Effect of the number of sub-goals.

8. Discussion on Optimization

In our Selective On-Demand Approach, we propose how to process a query by considering the characteristics of logic-macroprograms for WSNs. However, it is of a particular interest to explore how this approach can be further optimized. For instance, several rules usually contain a sub-goal to constrain a variable argument with a certain range of values. An example body of such a rule is *subgoal1*(*X*, *Y*),

subgoal2(*Y*,*Z*), $Y < 50$. In this situation, we can simply send the constraint $Y < 50$ along with the sub-query *subgoal1*(*X*, *Y*). Only facts *subgoal1* with the second argument that is smaller than 50 are selected to be sent as answers. Others are suppressed. This optimization technique is certainly applicable to most approaches.

As mentioned before, in SA, an existence-checking query can be suppressed by a node that already contains a satisfying fact. This suppressing node does not need to further disseminate the query to nodes under its sub-tree. The cost saving of this optimization technique is essentially the number of suppressed messages (or the number of nodes under the sub-tree rooted at the suppressing node). Let C_i^d be the number of nodes in the sub-tree whose root node *i* is on the d^{th} level depth. Thus, the cost saving can be formally expressed as

$$C_i^d = \begin{cases} 1 & \text{if node } i^{th} \text{ is a leaf node,} \\ 1 + \sum_{j \in S_i} C_j^{d+1} & \text{otherwise.} \end{cases}$$

We believe that there are open research areas in optimizing either centralized or distributed approaches. However, in this paper, we learn that neither a distributed nor a centralized approach overcomes the other without considering the characteristics of logic macroprograms for WSNs (as seen in the previous section). Regardless of the approaches, the complete results must be sent back to a waiting user at the base station. This results in heavy loads on nodes around the base station. To avoid the heavy loads and to improve the performance, the load-balanced tree protocol [15] can be applied.

9. Conclusion

In this paper, we examine various query processing schemes in logic macroprograms for WSNs. Prior works are fully distributed to avoid drawbacks of the centralized approach. However, we notice that all complete results produced by those distributed approaches are still scattered in the network. Eventually, those results must be sent back to a sink node or a base station. Hence, being fully distributed may not be advantageous as normally perceived.

We present the Selective On-Demand Processing scheme that combines benefits of centralized and distributed approaches. We also analyze and compare the communication cost of our approach with that of previous approaches. Our analysis reveals several important aspects that should be considered when designing a logic-macroprogramming system for WSNs. There is room to optimize the query processing for a logic-macroprogramming system to further reduce the communication cost and to balance the network load. We believe that a further deeper analysis with recursive queries can be extended from this work. A recursive rule can be expanded to a long-tail non-recursive query that consists of several repeated predicates. However, the exact numbers of predicates is now known a priori and depends on environmental data to be collected. The stochastic analysis can be one of feasible solutions for further deeper analysis.

Acknowledgement

We would like to thank Rawin Youngnoi and anonymous reviewers for their valuable suggestions to improve this work. This work was supported by the Thailand Research Fund (TRF) under grant MRG5080449 and the CP CU Academic Excellence Scholarship from Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University.

References

[1] C. Borcea, C. Intanagonwiwat, P. Kang, U. Kremer, and L. Iftode, "Spatial programming using smart messages: Design and implementation," in *the 32th International Conference on Distributed Computing Systems*, Tokyo, Japan, 2004, pp. 690-699.

[2] R. Gummadi, N. Kothari, R. Govindan, and T. Millstein, "Kairos: a macro-programming system for wireless sensor networks," in *the 20th ACM Symposium on Operating Systems Principles*, Brighton, United Kingdom, 2005, pp. 1-2.

[3] C. Intanagonwiwat, R. K. Gupta, and A. Vahdat, "Declarative resource naming for macroprogramming wireless networks of embedded systems," *ALGOSENSOR*, vol. 4240, pp. 192-199, 2006.
[4] M. Karpinski and V. Cahill, "Stream-based macro-programming of wireless sensor, actuator network applications with SOSNA," in *the 5th workshop on Data management for sensor networks*, Auckland, New Zealand, 2008, pp. 49-55.
[5] L. Mottola and G. P. Picco, "Programming wireless sensor networks with logical neighborhoods," in *the First International Conference on Integrated Internet Ad Hoc and Sensor Networks*, Nice, France, 2006, Article 8.
[6] R. Newton, G. Morrisett, and M. Welsh, "The regiment macroprogramming system," in *the 6th International Conference on Information Processing in Sensor Networks*, Cambridge, MA, 2007, pp. 489-498.
[7] R. Newton and M. Welsh, "Region streams: functional macroprogramming for sensor networks," in *the 1st International Workshop on Data Management for Sensor Networks: in conjunction with VLDB 2004*, Toronto, Canada, 2004, pp. 78-87.
[8] A. Pathak, L. Mottola, A. Bakshi, V. K. Pasanna, and G. P. Picco, "Expressing sensor network interaction patterns using data-driven macroprogramming," in *the 5th IEEE International Conference on Pervasive Computing and Communications Workshops*, White Plains, NY, 2007, pp. 255-260.
[9] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler, "Hood: a neighborhood abstraction for sensor networks," in *the 2nd International Conference on Mobile Systems, Applications, and Services*, Boston, MA, 2004, pp. 99-110.
[10] D. Chu, L. Popa, A. Tvakoli, J. M. Hellerstein, P. Levis, S. Shenker, and I. Stoica, "The design and implementation of a declarative sensor network system," in *the 5th International Conference on Embedded Networked Sensor Systems*, Sydney, Australia, 2007, pp. 175-188.
[11] H. Gupta, X. Zhu, and X. Xu, "Deductive framework for programming sensor networks," in *the 2009 IEEE International Conference on Data Engineering*, Shanghai, China, 2009, pp. 281-292.
[12] K. Whitehouse, F. Zhao, and J. Liu, "Semantic streams: A framework for composable semantic interpretation of sensor data," in *the 3rd European Workshop on Wireless Sensor Networks*, ETH Zurich, Switzerland, 2006, pp. 5-20.
[13] S. Choochaisri and C. Intanagonwiwat, "A system for using wireless sensor networks as globally deductive databases," in *the 2008 IEEE International Conference on Wireless & Mobile Computing, Networking & Communication*, Avignon, France, 2008, pp. 649-654.
[14] X. Zhu, H. Gupta, and B. Tang, "Join of multiple data streams in sensor networks," *IEEE Trans. on Knowl. and Data Eng.*, vol. 21, no. 12, pp. 1722-1736, Dec. 2009.
[15] T. S. Chen, H. W. Tsai, and C. P. Chu, "Gathering-load-balanced tree protocol for wireless sensor networks," in *the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, Taichung, Taiwan, 2006, pp. 8-13.