Optimizing Concurrent-Query Execution in Wireless Sensor Networks

Dilini A. Muthumala#1, Udara S. Liyanage#2, Asanka P. Sayakkara#3, Jeevani S. Goonetillake#4

#University of Colombo School of Computing,

No. 35, Reid Avenue, Colombo 7, Sri Lanka

1diliniasanga@gmail.com

2udaraliyanage@gmail.com

3asanka@scorelab.org

4jsg@ucsc.cmb.ac.lk

Abstract— **Due to the underlying complexity of wireless sensor networks in acquiring data, database abstractions are used in many real-world WSN applications utilizing SQL-like queries. As it is the trend nowadays to share a WSN among multiple users, concurrent query execution has become an important concern in the domain. Since WSN nodes are extremely energy constrained entities, and query processing and query results communication consumes a significant amount of energy, it is necessary to perform optimizations to the concurrent query execution in WSNs. While many previous researches have attempted to address this problem, they have mostly assumed WSNs which consists of a single base station shared among multiple users. However, the existing concurrent query optimization solutions are not effective for a WSN with multiple base stations. In this paper, we present a novel query optimization strategy for the database abstraction of WSNs which consists with multiple users. The evaluation results show that the suggested scheme significantly decreases the energy usage not only in single base station scenarios but also in multiple-base station scenarios.**

Keywords— **WSN, Database Abstractions, Energy, Optimization**

I. Introduction

Wireless sensor networks (WSN) consist of groups of smart sensor devices which form a wireless network. WSNs are mainly used for monitoring certain environmental conditions, such as a monitoring wildlife, observing severe weather conditions and collecting data inside active volcanoes [4], [10]. Due to the nature of such WSN applications, WSN nodes may have to be used in large numbers. Due to the nature of these applications, once deployed, each WSN node has to operate as longer life time as possible before they run out of their battery power. Therefore, WSN nodes are manufactured with limited resource such as low processing capabilities, limited memory capacities and batteries to ensure a longer unattended operational life-time [1].

Development of software applications for resource constrained devices such as WSN nodes requires low-level programming languages and tools which can only be handled by experienced computer scientists. However, in real-world applications, the users of such WSN applications are normally biologists, environmental researchers, farmers and not computer scientists. The challenge of handling resource constrained devices enforce limitation on real-world WSN users demanding better middleware for WSN applications. Provision of different abstraction layers for WSN is such a solution where the complexities of the network is hidden under some user-friendly and familiar interface. One such widely used abstraction for WSN is database abstractions such as TinyDB [3] and TikiriDB [2] where the whole network is emulated as a database for the end users. The end-users are able to interact with WSN by issuing SQL-like declarative queries in order to obtain real-time data from the network. The distinct feature in this type of query is that the execution period and sample period can be specified within the query depending on the user requirement.

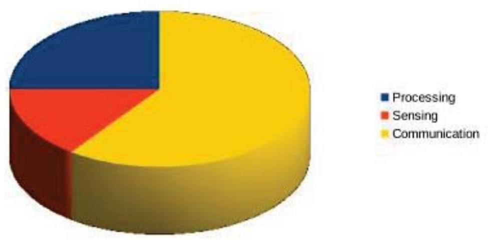

Fig. 1. The distribution of the energy consumption of a typical sensor node for the three tasks processing (15%-30%), sensing (6%-20%) and communication (about 60%) [7], [8].

As the capabilities of WSNs such as processing power, memory and onboard sensor types are getting improved, more and more people get attracted to use WSNs. However, due to the high cost involved in deploying a WSN, not everyone who wants to use a WSN is actually able to do so. As a solution to this problem, sharing of a WSN among multiple users has been suggested [2]. The underlying concept here is that once a WSN is deployed, different users are able to subscribe and get themselves connected to the network may be using fixed base stations or even through their laptops which could in turn be considered as base stations to obtain the data of their interest. Such WSNs can contain single or multiple base stations from any of which a user can inject a query to the network and due to the flexibility of getting connected to the network, connection among the base stations may not exist as shown in Figure 2. Due to such possibilities, in a particular time instance of the network, multiple queries can be running on a node creating concurrent query execution scenarios. Under such concurrent query situation, the database abstraction layer of a WSN can result in inefficiencies from energy point of view since the same data may be repeatedly sensed or communicated. In the following subsection, these issues are highlighted and thus we consider that the energy can be saved by optimizing concurrent query execution. Here the expected optimization is the minimization of sensing and communication cost.

A. Issues in concurrent query execution

As the scenario shown in Figure 2, multiple users are connected to the WSN via multiple base stations to acquire data from the network by issuing different queries. Four queries are issued by four different users at the same time which are named as Query-1, Query-2, Query-3 and Query-4.

Query-1

```
SELECT temp
FROM sensors
SAMPLE PERIOD 1s;
```

Query-2

```
SELECT temp
FROM sensors
SAMPLE PERIOD 5s;
```

Query-3

```
SELECT temp
FROM sensors
SAMPLE PERIOD 10s;
```

Query-4

```
SELECT temp
FROM sensors
SAMPLE PERIOD 5s;
```

As Figure 2 depicts, Query-1 and 2 are issued from the base station 1 by two different users while Query-3 and Query-4 are issued by two other users from base station 2. Even though the first two queries go through same base station, they are transmitted as two query packets two the network. Every intermediate packet forwarder should perform the transmission of two query packets individually under such a situation. Additionally, every sensor node of the network has to receive the two packets, process them individually and transmit their resulting data packets to the base station 1 separately. The base station 1 routes each resulting data packet to the relevant receiving end user. Even though both queries are originated from the same base station, separate treatment for the queries results in unnecessary packet transmissions between a particular sensor node and a base station. Since the cost for packet transmission is one of the highest energy consuming task on a WSN (Figure 1), this situation results in significant drainage of batteries on each node.

Meanwhile, Query-3 and Query-4 issued from base station 2 should also traverse through the network to reach all the sensor nodes to get executed for acquiring data. Consider a particular sensor node which has already received Query-1 and 2 separately which make the node to access temperature sensor on board periodically to cater the two queries. Note that a sensor reading acquired for Query-1 will not be used as a data for the Query-2 which gets acquired separately. The reception of Query-3 requires the node to access the same temperate sensor on board for a third time. Since each of these queries are treated separately, the node has to access temperature sensor multiple times for executing Query-1, 2, 3 and 4 when they all are running on a node concurrently. Such an uncoordinated access to the sensors on a node drastically consume the energy of the node decreasing the life-time of the network.

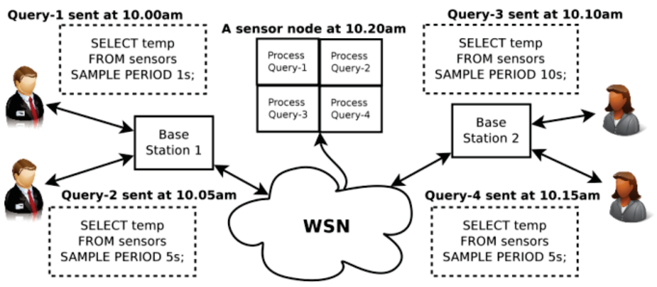

Fig. 2. A scenario where four users insert queries into the shared WSN at different times which gets executed concurrently at the nodes. At 10.20am in the network, a particular node will be running all the four queries concurrently as four separate threads.

The aforementioned scenario illustrates the negative effect of executing concurrent queries on WSN nodes without optimization under database abstractions having multiple base stations. Even though previous researches have attempted to address similar issues, such solutions are unable to handle the problem effectively under multiple base station scenarios. To address this problem, this paper presents a novel concurrent- query optimization mechanism that not only works on single-base station-WSNs, but also on multiple base station-WSNs. The aim of this optimization scheme is to reduce the energy spent on communication, sensing and processing operations, performed by sensor motes when processing queries. The contributions of this research are as follows:

- We designed, implemented and evaluated an optimal way to process concurrent queries.
- This new optimization scheme can be applied not only to single-base station-WSNs, but also to multiple-base station-WSNs. To the best of our knowledge, this is the first optimization scheme which handles both of these cases.
- We have proved using our evaluation that the proposed approach can bring benefits by saving costs which are caused by communication and sensing operations.

The rest of the paper is organized as follows. In Section II, we present our proposed optimization mechanism for con- current query execution on WSNs. Section III evaluates the suggested optimization mechanism. We present the related research work of this problem domain in Section IV and then finally concludes the paper in Section V.

II. Optimization of Concurrent Query Execution

When designing our optimized concurrent query execution strategy for database abstraction layer of WSNs to reduce the energy wastage on communication, processing and sensing stages of the network, we considered two potential end points where we can perform the optimization. Those two end-points are namely, base station and sensor nodes. The base station is responsible for accepting multiple queries from multiple users which are injected to the network. Therefore, a concurrent query optimization needs to be performed at a particular base station. However, due to the existence of multiple base stations, each sensor node in the network can receive multiple similar kind of queries to be executed concurrently which are originated from different base stations. Therefore, an optimization at the base station is also necessary to achieve a higher level of efficiency.

There are two different types of benefits we achieve by designing optimizations at the two end points in the network.

1) Base station level optimization:

The objective of the base station-level optimizer is to minimize, if not possible to prevent, redundant concurrent queries from getting executed by sensor nodes in the network. From eliminating or minimizing redundant query insertion into the WSN, the base station-optimizer expects to reduce mainly the communication cost, sensing cost and processing cost which are spent by sensor nodes on query execution.

2) Node level optimization:

A node-level optimization is introduced to handle possible redundant queries posed from different base stations. The objective is to reduce the consumption of memory and processing power of each node whilst concurrent query execution.

In the following subsections, we describe our concurrent query execution strategies in both at the base station level and the sensor node level.

A. Optimizations at base station

The base station-level optimizer attempts to satisfy new query submissions from the result stream of the queries which have been inserted to the WSN by the base station previously and are still being executed. Upon arrival of a new query, the base station will check which of the attributes are *satisfiable* from the result-stream that is already coming in to the base station. A certain attribute is satisfiable, if the base station is already acquiring the same attribute at a rate either equal to or faster than the *SAMPLE PERIOD* of the new query.

Figure 3 depicts the algorithm utilized at each base station of the network to generate a new query after processing multiple queries received to the base station from multiple users. There are an infinite number of possible scenarios which a base can encounter, upon arrival of a new query. However, three categories can be identified to which any possible scenario would exclusively belong as given below.

The new query q is,

- Fully Satisfiable
- Partially Satisfiable
- Not Satisfiable at all

To demonstrate the algorithm shown in Figure 3, we consider one example from each category and explain how the algorithm handles it.

1. *The new query is fully satisfiable:* Assume the following scenario. A user inserts Query-4 from a certain base station; at the same time, another user inserts Query-5 from the same base station. Notice that Query-4 already has demanded Temperature at a rate of one second which can be used to satisfy Query-5 as it demands the same attribute (temperature) at the same rate (one second). As such, the Base Station- level Optimizer will prevent the insertion of Query-5, with the following expectations.

```
1: procedure BASE-STATION-LEVEL-OPTIMIZER()
2:     Upon arrival of a new query $q$
3:     $s \leftarrow$ sample period of $q$
4:     for all $attribute_a \in q$ do
5:         $satisfiable \leftarrow FALSE$
6:         for all sample period $sp \in samplePeriodList[a]$ do
7:             if $s \% sp = 0$ then
8:                 $satisfiable \leftarrow TRUE$
9:                 break out of the inner most for-all loop
10:            end if
11:        end for
12:        if $satisfiable = FALSE$ then
13:            add $a$ to unsatisfiable-list
14:        end if
15:    end for
16:    if $unsatisfiable - list \neq empty$ then
17:        $q \leftarrow$ CreateQuery(unsatisfiable-list, $s$)
18:        insert $q$ into the WSN
19:        update $samplePeriodList$
20:    end if
21: end procedure
```

Fig. 3. Base station-level optimizer algorithm.

The expected outcomes are three-folds: (1) Reduction in communication cost: Since Query-5 does not get inserted in to the WSN, query propagation cost would be saved. Furthermore, since Query-5 would not send any results, result- propagation cost too will be saved. (2) Reduction in sensing cost: Since Query-5 does not get executed in sensor nodes, it would not cause any sensory data acquisitions. (3) Reduction in processing cost: Query-5 does not get processed in the network, thus would save processing energy of sensor nodes.

Query-4

```
SELECT temp,
humid FROM
sensors SAMPLE
PERIOD 1s;
```

Query-5

```
SELECT temp
FROM sensors
SAMPLE PERIOD 1s;
```

2. *The new query is partially satisfiable:* The scenario in consideration is as follows. A user inserts Query-6 from a certain base station; at the same time, another user inserts Query-7 from the same base station. Since Query-7 requests more attributes (Humidity) than the Query-6, the base station has to send a request to the WSN, demanding for the additional data. However, since Temperature is already

acquired at one- second rate, the base station can satisfy a part of the query. In a situation like this, the base station will insert the part of the Query-7 which is unsatisfiable to the WSN.

The expected outcomes are three folds: (1) Reduction in communication cost: Since both the queries are originating from the same base station, and they demand data at the same rate, sending only one packet every second over to the base station would suffice to satisfy both the queries. From doing this communication cost is expected to be reduced. (2) Reduction in sensing cost: Since the sensor node would share temperature readings among the two queries, sensing cost would be reduced. (3) Reduction in processing cost: Since only a part of the Query-7 is get inserted into the WSN, processing energy spent by sensor nodes are expected to be reduced.

Query-6

```
SELECT temp
FROM sensors
SAMPLE PERIOD 1s;
```

Query-7

```
SELECT temp,
humid FROM
sensors SAMPLE
PERIOD 1s;
```

3.*The new query is not satisfiable at all:* Consider the following scenario. A user inserts Query-8 from a certain base station; at the same time, another user inserts Query-9 from the same base station. In this scenario, the queries request different attributes: one query requests temperature whilst the other requests humidity. In a situation similar to this one, the Base Station-level Optimizer cannot satisfy one query from the other; neither fully nor partially. As such, it will insert Query-9 in to the WSN..

However, the node takes advantage of the fact that both the queries are originating from the same base station. Rather than sending two packets, one for Query-8 and another for Query-9 over to the same base station, the node will put the data demanded by Query-8 and Query-9 in one packet, and send to the Base Station. Using the received aggregated packet, the optimizer at the base station takes care of generating two separate packets to serve the two users with relevant data requested.

Query-8

```
SELECT temp
FROM sensors
SAMPLE PERIOD 1s;
```

Query-9

```
SELECT humid
FROM sensors
SAMPLE PERIOD 1s;
```

B. Optimizations at each node

A Node-level Optimization component is introduced because there is no other entity in a WSN that can handle possible redundancies among queries which are coming from different base stations. However, since a sensor mote is an extremely resource-constrained entity, pushing an optimization component into such an entity might not sound as a wise decision. However, this research introduces a node-level optimization component with the hope of saving communication and sensing energy (which are the most energy- hungry operators), at the cost of added-processing-energy (which is less energy-consuming, compared to communication and sensing operations) as shown in Figure 1. The proposed node-level optimizer utilizes a technique for improving the energy efficiency on the network which we describe using the following example.

Query-10

```
SELECT temp,
humid FROM
sensors SAMPLE
PERIOD 1s;
```

Query-11

```
SELECT temp
FROM sensors
SAMPLE PERIOD 1s;
```

A certain node receives two queries at the same time: Query-10 which is sent from Base Station-1 and Query-11 which is sent from Base Station-2. Existing systems such as TikiriDB assigns a separate thread for each of the above two queries. The thread which process Query-10 will sense temperature and humidity every second. The other thread will sense Temperature every second. In contrast, the node- level optimizer will not do separate sensing operations for each query. Rather, it will sense temperature only once, every second, and share the sensed value among the two queries. This will eliminate a redundant temperature-sensing operation which used to occur every second. Thus the example above highlights the technique which is used by the node-level optimizer; *read a certain sensor once on behalf of all queries and share*. Figure 4 depicts the algorithm which is performed at each sensor node of the network to achieve the functionality of the node-level optimizer.

Upon arrival of a new query, the QUERY-HANDLER() procedure (Figure 4) will be invoked. This procedure will invoke a sub-procedure: UPDATE-DATA-STRUCTURES(q); which will update a certain data structure which the sensor mote itself maintains, to keep track of requests made by concurrent queries. Simply, this data structure contains data such as which base station demanded for which attributes at which rates. Afterwards, csr and epoch will be updated.

’csr’ refers to the clock-strike rate for the sensor mote; which initially equals to zero. Each mote maintains a clock which strikes at a rate equal to the clock-strike rate. To save energy, the sensor mote will only wake up when this clock strikes. As such, it is important that the mote wakes up every time a data acquisition is due. Hence, the clock-strike rate is updated upon arrival of each query, to make sure the wake up will happen correctly to compensate the newly arrived query.

```
procedure QUERY-HANDLER()
    $s \leftarrow$ sample period of $q$
    $UPDATE - DATA - STRUCTURES(q)$
    if the mote does not execute any other queries then
        $csr \leftarrow s$
        $epoch \leftarrow csr$
        CLOCK-STROKE-EVENT-HANDLER()
    else
        $csr \leftarrow Greatest - Common - Divisor(csr, s)$
        $epoch \leftarrow csr$
    end if
end procedure

procedure CLOCK-STROKED-EVENT-HANDLER()
    for all constant $A \in supportedAttributeSet$ do
        for all sample period $sp \in samplePeriodList[A]$ do
            if $epoch \% sp = 0$ then
                $shouldBeAcquired \leftarrow TRUE$
                mark which base stations need $A$
            end if
        end for
        if $shouldBeAcquired = TRUE$ then
            ACQUIRE($A$)
        end if
    end for
    for all base station $b$ do
        send all demanded data in one packet
    end for
    $epoch \leftarrow epoch + csr$
    set CLOCK-STROKE-EVENT-HANDLER() in another $csr$ seconds
end procedure
```

Fig. 4. Sensor node-level optimizer algorithm

When the mote executes only one query, waking up at a rate equal to the sample period of the query would suffice. However, when there are concurrent queries executing, the mote should wake up at a rate equal to the greatest common divisor of the sample periods of all concurrent queries, so that it is ensured that the mote is awake whenever there is a sampling due for any query.

To keep track of the amount of time elapsed, the mote maintains a counter - named 'epoch'. Upon arrival of a new query, the epoch will be set to csr. This is because the epoch will be divided by sampling rates to decide whether a sampling is due; and to make epoch properly divisible by a sampling rate, epoch should be a multiple of all sampling rates; hence it is reset to a value equal to the greatest common divisor of all sampling rates. In case, the newly arrived query is the only query the mote executes, this procedure will invoke another procedure: CLOCK-STROKE-EVENT-HANDLER(). If there are any other queries that the mote already executes, then invoking of this procedure does not happen since those queries will make sure to invoke the CLOCK-STROKE-EVENT-HANDLER() within the same procedure itself.

The CLOCK-STROKE-EVENT-HANDLER() is first invoked by the first ever query the mote receives. Afterwards, it is invoked by itself every csr seconds; hence no subsequent query need to invoke it. Upon invocation, this handler iterates over a list of constants, where each constant represents an attribute that the mote support sensing. For example, a certain mote could be capable of sensing Temperature, Humidity and Light; hence the list of constants would be TEMPER- ATURE=0, HUMIDITY=1, LIGHT=2. Let the constant in a certain iteration be A, representing the attribute Temperature. Temperature could have being demanded by zero or more queries. The mote keeps a list of such demanded rates: samplePeriodList[A]. The mote checks if at least one of the elements in samplePeriodList[A] properly divides epoch. If so, it indicates that at least one query has demanded Temperature at the current epoch. For example, when the current epoch is 10 seconds, a query demanding temperature with a sample period of 2 wants Temperature to be acquired at the current epoch. As such, the mote will mark the fact that at least one query has demanded Temperature at this epoch and continue iterating the samplePeriodList[A]. The reason for continuing iterating over the samplePeriodList[A] is that, even though the mote knows that Temperature need to be acquired at this epoch, the mote also wants to know all the base stations which demanded Temperature at the current epoch. After noting which base stations have demanded for Temperature, it will stop iterating over the samplePeriodList[A] and acquire Temperature. In the existing systems, how this would happen is, Temperature would be acquired for each query that demanded for it. In contrast, in the optimized way, Temperature will be acquired only once, on behalf of all the queries that demanded for Temperature.

Likewise, the same procedure will be executed for other sup- porting attributes. Finally, since the mote has noted which base stations has demanded which attributes at the current epoch, the mote will send each base station only one result packet, containing all the demanded sensory data. This contrasts with the existing systems as follows. In the existing systems, a separate packet will be sent to a certain base station for each query that the base station has produced. In the optimized way, only one packet will be sent to the base station, containing all the data requested by all the queries, produced by the base station. Base station, typically being a powerful entity, can then make sure to extract sensory data from the result packet and deliver those to the relevant queries. After this process, the CLOCK-STROKE-EVENT-HANDLER() will set itself to call itself back in csr seconds.

Likewise, the same procedure will be executed for other sup- porting attributes. Finally, since the mote has noted which base stations has demanded which attributes at the current epoch, the mote will send each base station only one result packet, containing all the demanded sensory data. This contrasts with the existing systems as follows. In the existing systems, a separate packet will be sent to a certain base station for each query that the base station has produced. In the optimized way, only one packet will be sent to the base station, containing all the data requested by all the queries, produced by the base station. Base station, typically being a powerful entity, can then make sure to extract sensory data from the result packet and deliver those to the relevant queries. After this process, the CLOCK-STROKE-EVENT-HANDLER() will set itself to call itself back in csr seconds.

III. Experiments and Results

Since this research explores the possibility of optimizing concurrent queries in the presence of single or multiple-base stations, the evaluation has two parts:

1) Evaluation for single-base station WSNs
2) Evaluation for multiple-base station WSNs.

In a realistic WSN, there could be an infinite number of possible scenarios that could occur. Since it is practically impossible to evaluate the suggested optimization scheme for each such scenario, several representative scenarios were chosen and were evaluated. Some of such scenarios are presented in this section..

A. Single-Base Station WSNs

In Section II-A: Examples, three states were identified at which a base station can exclusively be, upon arrival of a new query. Since any possible query falls under one of these states, for the sake of completeness of the evaluation, one scenario from each state was evaluated.

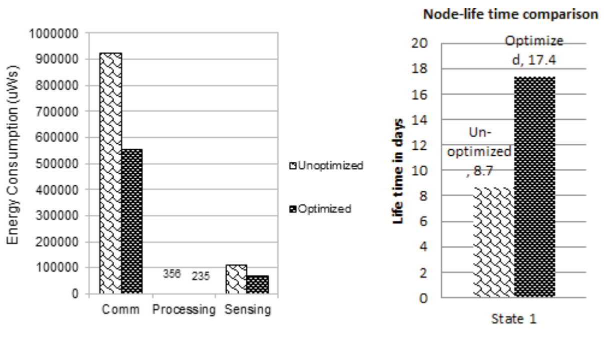

Fig. 5. Energy Consumption in Un-optimized scenario and Optimized scenario, with respect to the three operators: Communication, Sensing and Processing.

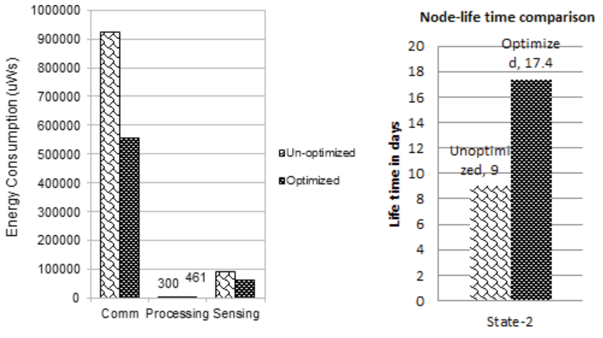

Fig. 6. Energy consumption comparison, with respect to state-2: The new query is partially satisfiable.

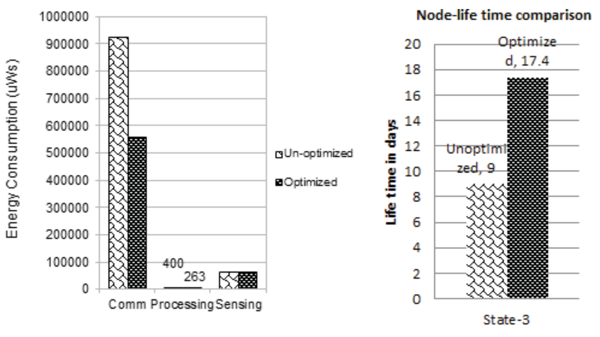

Fig. 7. Energy consumption comparison, with respect to state-3: The new query is not satisfiable at all.

1.*The New Query is Fully Satisfiable:* Scenario: A particular user issues Query-12 from a certain base station. After this query is being executed for 10 seconds, another user inserts Query-13 from the same base station. Afterwards, both the queries get executed concurrently and node energy consumption was measured for 20 seconds.

Query-12
```
SELECT temp,
humid FROM
sensors SAMPLE
PERIOD 1s;
```

Query-13
```
SELECT temp,
humid FROM
sensors SAMPLE
PERIOD 1s;
```

The graph in Figure 5 shows that the optimization scheme has saved the energy consumption in all three possible ways: communication, processing and sensing. This marks a realization of the expected outcomes from base-station level optimizer.

Furthermore, the graph on the right in Figure 5 shows the life time comparison between unoptimized and optimized cases. The lifetime increment is from 8.7 days to 17.4 days, which is a significant improvement.

2.*The New Query is Partially Satisfiable:* Similar to the previous scenario, the partially satisfiable scenario was evaluated for the two queries 6 and 7 given under Section II-A. The results obtained are as follows.

The graph in Figure 6 shows that the optimization scheme has dropped the energy consumption in communication, processing and sensing aspects. It can also be noted that processing cost is increased in this scenario. This could be the overhead introduced by the Node-level Optimizer. However, most importantly, the graph on the right hand side shows that, in overall, there is an increment in sensor nodes lifetime. This is due to reducing energy-hungry operations: communication and sensing, at the cost of increased processing.

3.*The New Query is not Satisfiable at all:* This scenario was evaluated for the two queries given under the same state in Section II-A. The results obtained are shown in Figure 7.

In this scenario, attribute values cannot be shared among the queries. Thus there is no reduction in sensing cost. However, as the queries are coming from the same base station, a significant amount of communication cost could be saved from sending one packet on behalf of both the queries, as seen in the graph.

B. Multiple-Base Station WSNs

1.*Attributes Can be Shared: Scenario:* A particular user issues Query-14 from a certain base station. At the same time, another user inserts Query-15 from a different base station. Afterwards, both the queries get executed concurrently and node energy consumption was measured for 50 seconds.

Query-14

```
SELECT temp,
humid FROM
sensors SAMPLE
PERIOD 1s;
```

Query-15

```
SELECT temp,
humid FROM
sensors SAMPLE
PERIOD 1s;
```

As shown in the graph in the left side in Figure 8, the communication cost has stayed same in both unoptimized and optimized cases. This is because the node cannot send one packet for both queries as they are originating from different base stations. Further it can be noted that processing cost has increased in the optimized case, while the sensing cost has being reduced.

The graph on the right hand side shows that, on overall, the optimization scheme has increased the lifetime of the sensor node. This shows that the increment in processing cost is compensated from the reduced sensing cost.

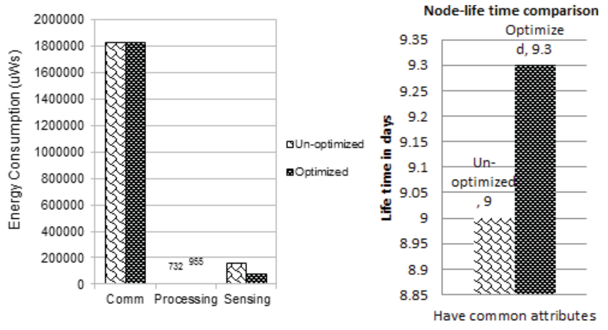

Fig. 8. Energy consumption comparison in the multiple-base station scenario, when the queries have common attributes.

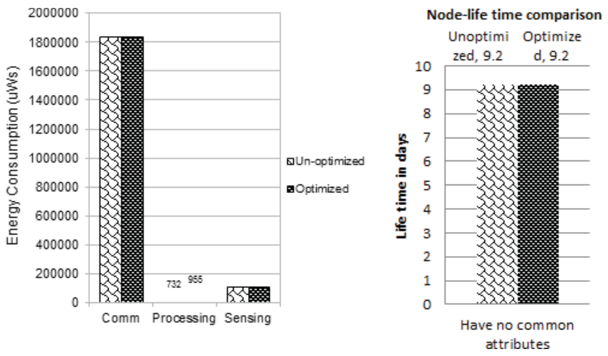

Fig. 9. Energy consumption comparison in the multiple-base station scenario, when the queries have common attributes.

2.*Attributes Cannot be Shared:* Scenario: A particular user issues Query-16 from a certain base station. At the same time, another user inserts Query-17 from a different base station. Afterwards, both the queries get executed concurrently and node energy consumption was measured for 50 seconds.

Query-16

```
SELECT humid
FROM sensors
SAMPLE PERIOD 1s;
```

Query-17

```
SELECT temp
FROM sensors
SAMPLE PERIOD
1s;
```

Since the queries have no attributes in common, the node- level optimizer cannot share sensory data as done previously. As such, sensing cost in the optimized case stays same as in the unoptimized case. This can be observed in Figure 9.

Communication cost too had stayed same while the processing cost has increased. However, the lifetime-graph, on the right, shows that this increment is insignificant as the lifetime of the sensor node has not changed to a visible extent.

IV. RELATED WORKS

As an early work for shared wireless sensor networks with database abstractions, Tikiridb [2] facilitates concurrent query executions which arrives from multiple base stations and multiple users. However, in its implementation, each query received by a node through the network is treated individually in separate threads without considering any possible optimizations on sensor nodes. Similarly, the base stations of TikiriDB just act as routers which pass queries and data between the network and user without considering optimization possibilities.

It is crucial to ensure, yet open to doubt, whether concurrent queries could be processed in the optimal way by query layers such as TinyDB and TikiriDB. However, many researchers in the past have tried to optimize concurrent queries [5], [12], [6], [11], [13], [9]. Different approaches have been followed such as the universal-query approach [6], two-tier multi- query optimization approach [12] and merge-split-parallelize approach [5].

Several researchers have succeeded in reducing the energy consumption of sensor motes by optimizing concurrent queries. For instance, [5] suggests a method of rewriting a set of concurrent queries, accumulated at the base station, in to a new query-set in such a way that the new query-set will consume a less amount of energy from the sensor motes than the original-set would do. The problem with the existing optimization mechanisms is that they seem to have been designed assuming that a WSN has only one base station.

When a WSN has multiple-base stations, such mechanisms would either be inapplicable; or fail to optimize concurrent queries, originating from different base stations.

V. CONCLUSIONS

This research explores the possibility of optimizing concurrent queries in the presence of single or multiple-base stations. The aim is to reduce the energy spent by extremely energy-constrained sensor nodes with respect to three specific operations in query handling: communication, sensing and processing.

In the presence of concurrent queries, the base station-level optimizer reduces energy consumption in many ways. When a newly inserted query is fully or partially satisfiable within a base station, the base station-level optimizer reduces energy consumption with respect to all three operations: communication, sensing and processing. Among these savings, communication-cost saving was the most significant, then sensing cost, finally processing cost, which was relatively less significant. However, when a newly inserted query is not satisfiable, the base station does not do any energy saving.

The node-level optimizer successfully saves energy to a significant extent, using the two techniques: (1) sharing sensor reading among queries, (2) sending one packet for multiple queries coming from the same base station. From sharing sensor reading once in every second, the node level optimizer has been able to extend the lifetime of the sensor nodes from 9 days to 9.3 days. From sending one packet for multiple queries coming from the same base station, the node level optimizer was able to extend the lifetime of the sensor network from 9 days to 17.4 days.

The expected gain would become significant with respect to the queries from different base stations if there are many similar queries. Although there is a processing cost, that is compensated by saving energy from sensing. Most importantly, when the optimization scheme does not have any chance to apply its techniques, still the optimization has not brought any significant disadvantage. Thus it can be used even when it is uncertain whether there will ever be any chance for applying optimization techniques.

REFERENCES

2. Advanticsys. Tmote sky sensor mote. http://www.advanticsys.com/.
3. N. M. Laxaman, M. D. J. S. Goonatillake, and K. D. Zoysa. Tikiridb: Shared wireless sensor network database for multi-user data access. *CSSL*, 2010.
4. S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: An acqusitional query processing system for sensor networks. *ACM TODS*, 2005.
5. A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, pages 88–97, New York, NY, USA, 2002. ACM.
6. R. Mu¨ller and G. Alonso. Optimization of concurrent queries in wireless sensor networks (technical report tr-589).
7. R. Mu¨ller, G. Alonso, G. Alonso, and G. Alonso. *Shared queries in sensor networks for multi-user support*. ETH, Department of Computer Science, 2006.
8. A. Nechibvute, A. Chawanda, and P. Luhanga. Piezoelectric energy harvesting devices: an alternative energy source for wireless sensors. *Smart Materials Research*, 2012, 2012.
9. V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava. Energy-aware wireless microsensor networks. *Signal Processing Magazine, IEEE*, 19(2):40–50, 2002.
10. N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman. Multi-query optimization for sensor networks. In *Distributed Computing in Sensor Systems*, pages 307–321. Springer, 2005.
11. G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh. Deploying a wireless sensor network on an active volcano. *Internet Computing, IEEE*, 10(2):18–25, 2006.
12. S. Xiang, H. B. Lim, and K.-L. Tan. Impact of multi-query optimization in sensor networks. In *Proceedings of the 3rd workshop on Data management for sensor networks: in conjunction with VLDB 2006*, pages 7–12. ACM, 2006.
13. S. Xiang, H. B. Lim, K.-L. Tan, and Y. Zhou. Two-tier multiple query optimization for sensor networks. In *Distributed Computing Systems, 2007. ICDCS'07. 27th International Conference on*, pages 39–39. IEEE, 2007.
14. S. Xiang, Y. Zhou, H.-B. Lim, and K.-L. Tan. Query allocation in wireless sensor networks with multiple base stations. In *Database Systems for Advanced Applications*, pages 107–122. Springer, 2009.