World Applied Sciences Journal 25 (4): 615-622, 2013
ISSN 1818-4952

DOI: 10.5829/idosi.wasj.2013.25.04.1327

Exploring TinyDB for a Low Traffic Solution and Query Optimization in Sensor Networks

[1]*Ateeq ur Rehman and* [2]*Muhammad Zakarya*

[1]Department of Computer Science, University of Southampton, UK
[2]Abdul Wali Khan University (AWKU), Mardan, Pakistan

Submitted: Aug 11, 2013; **Accepted:** Sep 19, 2013; **Published:** Oct 6, 2013

Abstract: Tiny DB is a database which is exclusively used in sensor network in Tiny OS (Tiny operating system) platform. It sends queries from a server to the network and gets back the result to the server in each epoch. One of the most challenging jobs in sensor networks is to reduce the traffic and to reduce the power consumption. We have derived a technique in case of network traffic which is significantly reducing the power consumption and network traffic cost than the current system. In our system, queries will be sent to the network in the first epoch during each data collection period and in all the other epochs it will just return the results. The motes will accept and store the first query that specifies the sample period and the data collection duration. It will then use the same query for retrieving the rest of the results at the end of each sample period for the entire data collection period. We used Tiny OS simulator (TOSSIM) which is embedded with Tiny DB and Tiny OS. We hope that it will be a great achievement in case of sensor network and sensor network database area.

Key words: Sensor Networks · TinyOS · TinyDB · TinySQL · TOSSIM · Power Efficient

INTRODUCTION

TinyDB is a query processing idea to dig out information from a group of TinyOS sensors, making a network. Distinct from on hand solutions for data processing in TinyOS [1], TinyDB does not impose you to write down embedded C code for sensors. In its place, TinyDB provides a clear-cut, SQL-like interface to recognize the statistics you desire to pull out, along with additional parameters, like the rapidity at which data should be re-energized – much as you would impersonate queries beside a conventional database. Given a query specifying your data interests, TinyDB [2] collects those facts from motes in the atmosphere and surroundings, filters it, aggregates it concurrently and direct it out to a machine. TinyDB does this through power-efficient in-network processing algorithms. We will talk about in this project, about methods designed to reduce and diminish power overheads through acquisitional techniques and methods. These techniques, taken in cumulative, can lead to orders of magnitude improvements in power utilization and expenditures *and* greater than before correctness and exactness of query results.

Fig 1 illustrates the fundamental architecture that we track throughout our research – queries are submitted at a powered PC (*base-station*), parsed, optimized and lastly sent into sensor network, where they are sprinkled and distributed and processed, with outcomes flowing back up the routing tree that was twisted and created as the queries were propagated [3, 4].

After a brief introduction to sensor networks in Section 1, Section 2 covers the related work and existing techniques along with some knowledge of query language, Section 3 highlights the existing problem in current system developed. Section 4 highlights our proposal on query optimization issues in power sensitive environments. Section 5 shows a mathematical and statistical proof to our proposed idea. In section 6 implementation and simulation results shows that our proposed idea is power efficient than other systems. In section 7 we will talk about some future research directions along with concluding remarks on our idea.

Related Work and Existing Techniques: In this section we discuss some existing mechanisms and techniques.

Corresponding Author: Ateeq ur Rehman, Department of Computer Science, University of Southampton, UK.

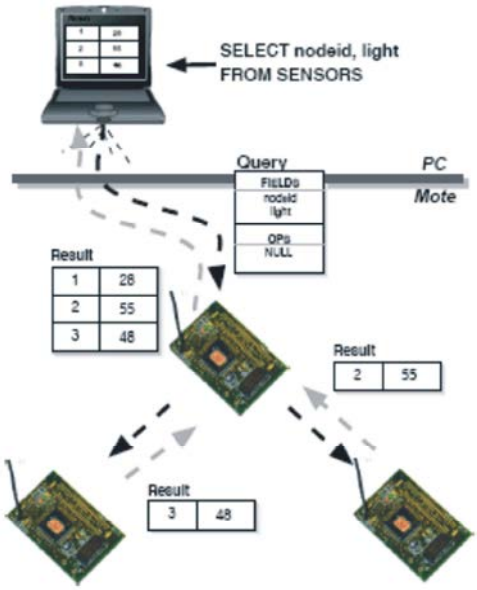


Fig. 1: A query and results propagating through the network..

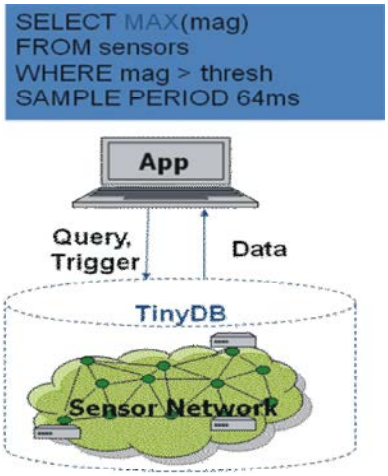


Fig. 2: TinyDB architecture

Tiny Database: TinyDB [5] makes available a top-level, declarative language for specifying queries. In a declarative language a programmer describe what he/she wishes for, but not how to attain it. Declarative languages are reasonably simple and easy to learn, with queries that are clear-cut to interpret and identify. Similarly they authorize the essential system to adjust how it runs a query, exclusive of requiring the query itself to be changed. This is noteworthy in sensor networks, where the best underlying implementation may need to change regularly and commonly.

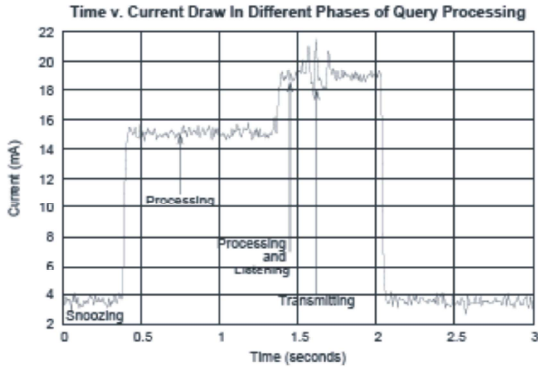


Fig. 3: Phase of Power Consumption in TinyDB

Sensor Networks: A sensor node [6, 7] is a battery power-driven, wireless computer. Characteristically, these nodes are bodily tiny (a few cubic centimeters in size) and tremendously low powered (a few tens of milliwatts versus tens of watts for a typical laptop computer) *[3]*. Power is of supreme and extreme significance. If used forthrightly, individual nodes will exhaust their energy supplies in only a few days *[4]*. In distinction, if sensor nodes are very frugal about power utilization and expenditure, months or years of life spann are possible. Mica motes, for example, when working at 2% duty cycle (between lively and snoozee modes) can accomplish and attain lifetimes in the 6 month range on a pair of AA batteries. This duty cycle confines the active time to 1.2 seconds per minute.

Power Consumption in Sensor Networks: In "Snoozing or sleeping" mode the node spends the majority of its time. The CPU and radio frequency are at rest, waiting for a timer to run out or any exterior happeningg to stir the gadget. In "Processing or active" mode the sensor node consumes more power than sleep mode and where query results are produced locally. In "Processing and Receiving" [8, 9] mode outcomes are collected and integrated from neighbors over the RF. To finish, in "Transmitting" mode, outcomes for the query are distributed by the local mote.

TinyOS: TinyOS [10-12] consists of a set of components for administration, supervision and accessing the mote hardware and a "C-like" programming language known as nesC. The major characteristics of TinyOS are:

- A group of software intended to make straightforward entrance to the lowest levels of hardware in an energy-efficient (Green computing) and conflict free way, and

- A programming replica planned to endorse extensibility and composition of software whereas maintaining a high level of concurrency and energy efficiency.

Query Language: Queries in TinyDB consist of a SELECT-FROM-WHERE clause supporting join, selection, aggregation, projection, sampling, windowing and sub-queries through materialization points. When a specific query is issued in TinyDB, it is assigned a unique identifier that is returned to the issuer. This identifier can be used to stop a specific query, limit a query execution to be executed for a specific time duration or include a stopping condition as an event.

Event-Based Queries: Event based queries start running when a low-level "event" occurs. There are two steps concerned in authoring event-based queries:

- Defining the operating system event and
- Registering the query with a TinyDB.

To describe an event, you must write down a module that registers the event and signals that it has fired up at whatever time it occurs. Registering events is much similar to registering commands. Event based queries must be input in the text panel of the TinyDB GUI [13, 14].

Existing Problem: A close study on the current TinyDB could reveal a lot of slacks in performance related features notably energy consumption [15]. This is not because TinyDB is not properly designed but compatibility issues arising from the design of the conventional motes to the aims of TinyDB. In the current system of TinyDB, queries that constitute repetitive sampling are much more common than the other types of queries. In fact, repetitive queries are the major utility in TinyDB that exhibits most of the power consumption. TinyDB [16] was designed for conventional mote types the designers of which may not have thought of TinyDB architecture. Hence there was no query storage concept yet there is a flash storage concept that can allow for query results to be stored for TinyDB. Periodic queries traverse through the network repetitively consuming vital processing power and time. Exploring efficient in-network storage and processing can help remedy such limitations by triggering queries from within queried motes [17-19].

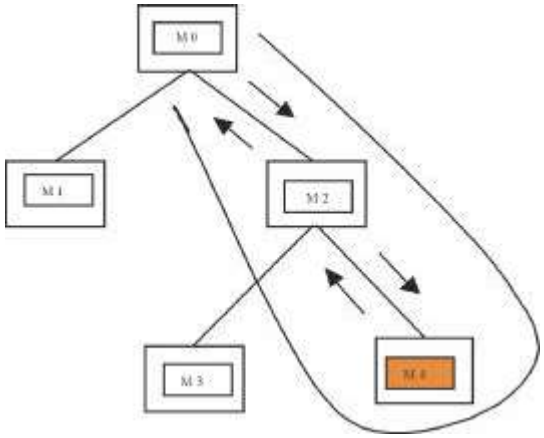


Fig. 4: Current System

Proposed Solution: We intend to put forward some approaches to solving the problems listed above in order to achieve the goal of TinyDB's ability to acquire and deliver desired data while conserving as much power as possible and satisfying desired lifetime goals.

Reducing Power and Traffic by Discarding Query Repetition: Suppose, we want the node id and temperature information and temperature should be above the threshold. The sample period is 1 second and query life time is 30 seconds.

- SELECT nodeid, temp
- From sensors
- WHERE temp > thresh
- SAMPLE PERIOD 1s FOR 30s

We know that TinySQL runs each query repeatedly, once per time-period or "epoch". That means for each sample period TinySQL runs each query repeatedly. As can be seen from Figure 4, at first time t-0, a query for the first time is injected into the network. Through the mote 2, the query comes to mote 4. Then fetch the result and send the result back to the root through mote 2 by aggregation and required filtration. This process is repeated for each epoch. In the case of the previous query the query is repeated for 30 times [20- 23].

From Figure 5, at time t-0, the query runs for the first time and it is injected into the network. The query comes to mote4 through mote 2. Assuming mote4 has the query storage capability then the requested 30 results can be returned by the mote without receiving any further requests.

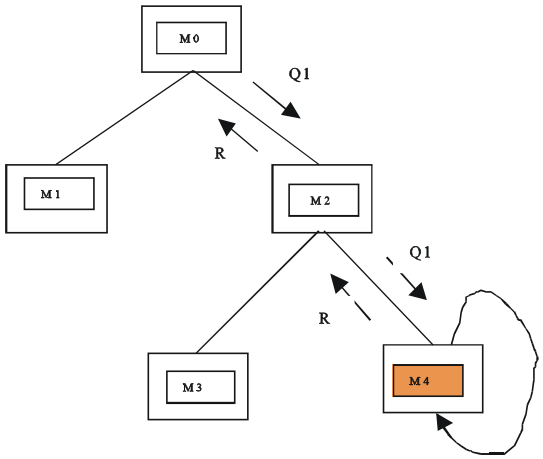


Fig. 5. Proposed System

That means for the entire data collection period the mote will receive a query only once (epoch 1) and for the rest of the epochs results will be returned consecutively.

Advantages :

- It will reduce the traffic and routing overhead.
- It will also reduce power consumption for root node and distribute power consumption.
- We hope that it will give a better result for a long tree.

Mathematical Proof: The mathematical verifications involved here are rather trivial. We considered a simple mote network as shown in figure 4.3. We assumed that the current draw by various components of each mote in the network for the same query message is the same. This is best realized with a network with the same type of motes deployed. To formulate an accurate mathematical representation, the following parameters should be assumed:

- C0total power overhead on a mote for receiving a query from its parent and transmitting the same query to a neighbor node. This power is the total of that consumed by the radio, leds, sensor board, cpu, adc and the EEPROM.
- C1total power overhead on a mote for receiving a result from its child node and transmitting the same result to a parent node. This power is the total of that consumed by the radio, leds, sensor board, cpu, adc and the EEPROM.

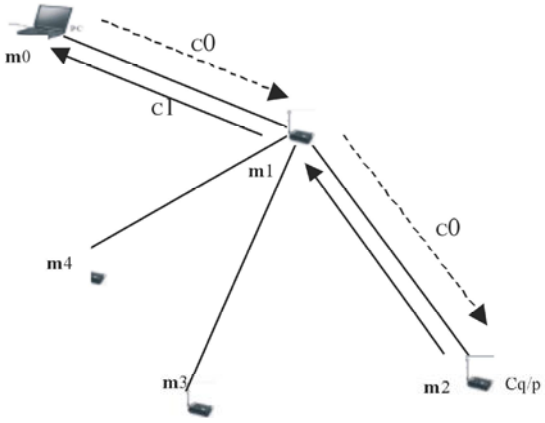


Fig. 6: Schematic of a sample query sent to mote m^2

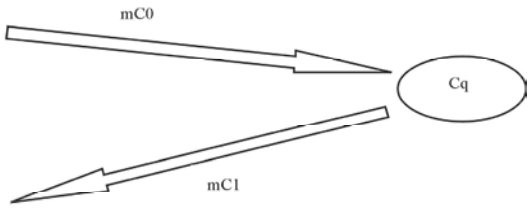


Fig. 7:

- Cqthe total power overhead on a queried node when the query is not stored.

- Power consumed for fetching a result for the query.
- The power consumed for receiving the query from a parent node and sending the fetched result back to a parent.

Cpthe total power overhead on a queried node when the query is stored.

- Power consumed by the process that triggers the query processing when the clock for an epoch expires.
- Power consumed for sending the fetched result back in the network.
- Extended power consumption due to mote sleep time limitations imposed by stored queries.
- Mthe number of motes along the route from the query originator to the queried node. This excludes the queried node itself.
- Nthe number of epochs specified by the query originator.

In current systems, the power consumption for repetitive queries can then be calculated as follows.

For a Single Query Sent;
Power = mC0 + Cq + mC1⇒m(C0 + C1) +Cq
For a query of n epochs;
Power = n[m(C0 + C1) +Cq]

In our proposed system, the power consumption for repetitive queries can be calculated as follows.

For a single query sent;
Power = mC0 s+ Cp + mC1m(C0 + C1) +Cp

For a query of n epochs;
Power = n[m(C0 + C1) +Cp] – n-1(mC0)

Power conserved

This can be further simplified as;
Power = n[mC1 +Cp] + mC0

The equations can be further improved to show important achievement;

- It is known that the power difference between forwarding a query and forwarding a result is quite negligible, since both processes take the form of active messages. Hence C0 ˜˜ C1
- The difference in value between Cp and Cq can be negligible if the duration of query epoch is carefully selected not to hamper the mote sleep time. Cq ˜˜ Cp

Hence equation 4.1 can be rewritten as;
Power = n[2mC0 +Cq]
Hence equation 4.3 can be rewritten as;
Power = n[mC0 +Cq] + mC0

From the above analysis, we can conclude that;

- Apart from the energy consumed by a queried node, the power consumption of our design is approximately one half of the original system if not for the value of mC0 which is the initial forward traffic.
- For a very large network of mote, the value of mC0 will be negligible and the power saving advantage will be easily realized.
- The Cp value is another important factor that can be controlled to some extent. Mica motes for example have a standard period of 4 seconds between sleep times when there is no activity. The epoch period must be set not expire during the period in which a mote is asleep.

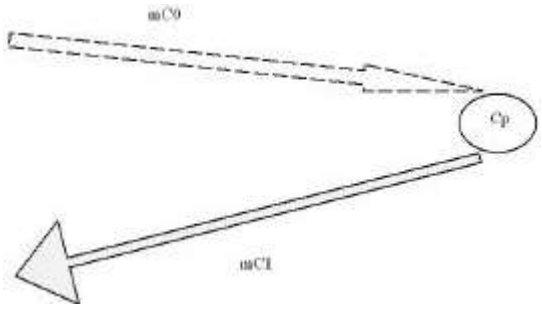


Fig. 8:

Fig. 9:

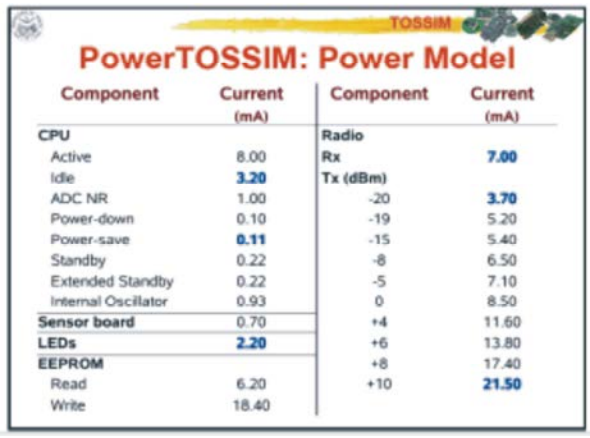
Implementation, Simulation and Results: We used the Tossim simulator here which is a discrete event simulator for TinyOS applications such as TinyDB. We focus on a specific sensor type –the Mica2 sensor node- developed by UC Berkeley. For the sake of understanding and simplicity, we choose a simple star topology of 4 nodes with node 0 serving as the base station [29, 30].

We then send a typical periodic query from the base station to the other nodes using the Tossim simulator. PowerTossim, a component of the Tossim simulator captures all associated power events (states) and then applies a specified power model to those events in order to compute their individual power consumptions as well as the sum total. The power model for a Mica2 mote can be seen in the next section.

The Power Model and Measurement: The power model is specifically for a Mica2 mote. This device consists of a 7.3 MHz ATmega128L processor, 128KB of code memory, 512KB EEPROM, 4KB of data memory and a ChipC on CC1000 radio capable of transmitting at 38.4 Kbps with an outdoor transmission range of approximately 300m. The device measures 5.7cm × 3.1cm × 1.8cm and is typically powered by 2 AA batteries.

Table 1 presents the resulting power model for the Mica2 hardware platform.

Table 1: Power Model for the Mica2. The Mote Was Measured with the Micasb Sensor Board and a 3v Power Supply.



PowerTOSSIM: Power Model

Component	Current (mA)	Component	Current (mA)
CPU		**Radio**	
Active	8.00	**Rx**	**7.00**
Idle	**3.20**	**Tx (dBm)**	
ADC NR	1.00	-20	**3.70**
Power-down	0.10	-19	5.20
Power-save	**0.11**	-15	5.40
Standby	0.22	-8	6.50
Extended Standby	0.22	-5	7.10
Internal Oscillator	0.93	0	8.50
Sensor board	0.70	+4	11.60
LEDs	**2.20**	+6	13.80
EEPROM		+8	17.40
Read	6.20	+10	**21.50**
Write	18.40		

Power Monitoring of a Sample Query: In order to monitor power for a simple tinydb query, power state transition messages must be enabled. Also the tinydb application must be compiled and run with power specific options. In our experiment, we run the tinydb application for 4 mins (240secs) and collect vital data from the debug messages into a trace file which will later be processed for energy consumption analysis in the next section. For the complete procedure of what is described above, see section 1 of the appendix [24].

Power State consists of a single interface with one command for each possible state transition. Each function tests if power profiling is enabled and if so, emits a log message detailing the mote number, the specific power state transition and the current simulation time. An excerpt from the trace file is shown below:

0 : POWER: Mote 0 LED_STATE RED_OFF at 18677335
0 : POWER: Mote 0 LED_STATE YELLOW_OFF at 18677335
0 : POWER: Mote 0 ADC SAMPLE RSSI_PORT at 18990479
0 : POWER: Mote 0 ADC DATA_READY at 18990679
0 : POWER: Mote 0 RADIO_STATE TX at 18993551
0 : POWER: Mote 0 RADIO_STATE RX at 19199375

Power Consumption Analysis (Per Mote Basis): Once the power and CPU state data has been obtained by the simulation run, several tools are available to permit analysis and visualization. In this experiment, we use the PowerTossim tool to compute the power consumption from the traces of transition messages obtained in the previous section. The procedure for using this feature of powerTossim can be found in section 2 of the appendix.

C.1 **Total energy vs. Time**
Current model

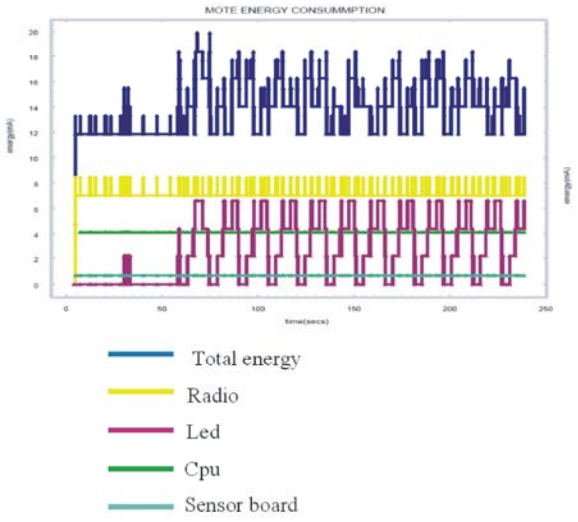


Fig. 10: Component power consumption in current systems for 10 epochs

Our model

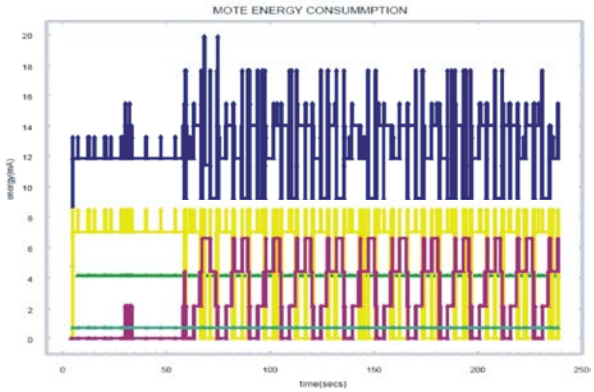


Fig. 11: Component power consumption in our model for 10 epochs

From the comparison of Figure 4.7 and Figure 4.8 it is clearly visible that total energy consumption of our model is rather less than that of the current systems. It is clear that the minimum consumption for the current systems at any moment is not less than 12mA. In our model, the power consumed can be as low as 9.5mA. This is the time when a mote had a chance to sleep putting off the radio since there is no query reception time involved.

For queries that last for very long periods, the cumulative energy consumption given by our model – the green plot – proved very impressive energy conservation over the conventional systems [23, 24]. After 4 minutes (240 seconds) about 2500mA was conserved for a single mote. This is a great achievement in a sensor network.

C.2 Cumulative energy Vs. Time

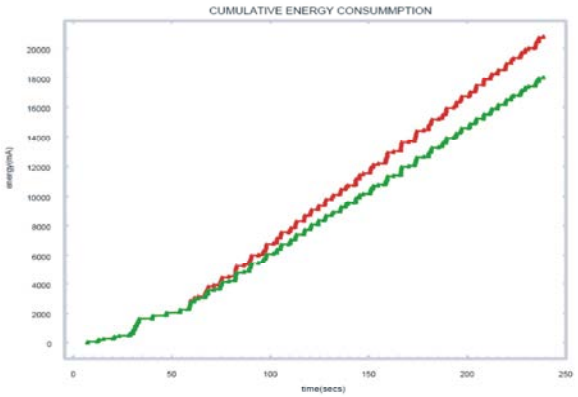


Fig. 12: Cumulative power consumption in the two systems for 10 epochs within 4minutes

C.3 Total energy consumption per mote

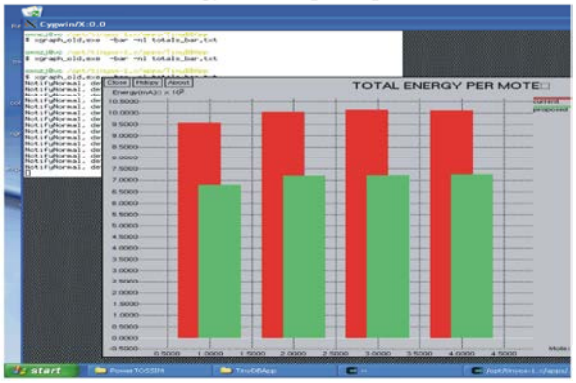


Fig. 13: Energy consumption per mote for both systems

Below is a figure that shows a comparative energy consumption chart between two systems across all four nodes that where simulated. It should be noted that mote0 is the root node sending the queries and in no means processes its own query for a result which makes is consume a slightly less energy. It should also be noted that all motes simulated are of a mica2 type.

CONCLUSION

It can be seen that the approached above is a big steps towards enhancing query performance in TinyDB. For a considerably large sensor network, the benefits of query storage are clearly notable. It will not only serve as a means of saving vital power but also provides network traffic optimization advantage. With frequent flow of queries through a large network, congestion and slow response times may result [25]. Real time processing is an important phenomenon in sensor networks which cannot be realized with large amounts of network congestion [27, 28]. The concepts discussed here can also be very useful in networks where queries may be carried out for long periods of time with relatively long epoch periods for each query. The advantages of the concept will be better realized when mote sleep intervals are less hampered with, in other words, when the epoch periods are not less than this interval. Although current mote types support flash storage, specially built types that will provide for all the associated constraints will be far more superior. However, this is not a constraint that overshadows the performance advantages brought about by the concept [26]. Finally, it is not difficult to observe that the requirements and limitations of these findings are quite negligible when balanced with the performance and energy saving advantages that comes in. this is clearly demonstrated in both the mathematical analysis and the experimental results.

REFERENCES

1. Madden, S.R., M.J. Franklin, J.M. Hellerstein and W. Hong, 2005. TinyDB: An acquisitional query processing system for sensor networks. ACM Transactions on Database Systems (TODS), 30(1): 122-173.
2. Zakarya, M. and A.A. Khan, 2012. Cloud QoS, High Availability and Service Security Issues with Solutions. IJCSNS, 12(7): 71.
3. Madden, S., J. Hellerstein, and W. Hong, 2003. TinyDB: In-network query processing in tinyos. Version 0.4, September.
4. Madden, S., M.J. Franklin, J.M. Hellerstein, and W. Hong, 2002. TAG: A tiny aggregation service for ad-hoc sensor networks. ACM SIGOPS Operating Systems Review, 36(SI): 131-146.
5. Munthe-Kaas, E. and J. Søberg, 2009. Data management in sensor networks. University of Oslo, INF5100, Autumn.
6. Zakarya, M., A.A. Khan and H. Hussain, 2010. Grid High Availability and Service Security Issues with Solutions.
7. Akyildiz, I.F., W. Su, Y. Sankarasubramaniam and E. Cayirci, 2002. Wireless sensor networks: a survey. Computer Networks, 38(4): 393-422.
8. Sam, M., H. Joe and H. Wei, 2003. TinyDB: in-network query processing in TinyOS. USA: University of California Berkeley: Computer Department.
9. Madden, S., 2006. Data management in sensor networks. In Wireless Sensor Networks Springer Berlin Heidelberg. pp: 1-1.

10. Madden, S., M.J. Franklin, J.M. Hellerstein and W. Hong, (2003, June). The design of an acquisitional query processor for sensor networks. In Proceedings of the 2003 ACM SIGMOD international conference on Management of data (pp: 491-502). ACM. '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pages 491--502, 2003
11. Zakarya, M., N. Dilawar, M.A. Khattak and M. Hayat, 2013. Energy Efficient Workload Balancing Algorithm for Real-Time Tasks over Multi-Core. World Applied Sciences Journal, 22(10): 1431-1439.
12. Madden, S.R., M.J. Franklin, J.M. Hellerstein and W. Hong, 2005. TinyDB: An acquisitional query processing system for sensor networks. ACM Transactions on Database Systems (TODS), 30(1): 122-173.
13. Madden, S., J. Hellerstein and W. Hong, 2003. TinyDB: In-network query processing in tinyos. Version 0.4, September.Sam Madden, et al. The design of an acquisitional query processor for sensor networks, SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pp: 491-502.
14. Zakarya, M., I. Rahman and I. Ullah, 2012. An Overview of File Server Group in Distributed Systems.
15. Samuel, R., Madden, Michael J. Franklin, Joseph M. Hellerstein and Wei Hong, TinyDB: In-network query processing in TinyOS, the TinyDB documentation
16. Hong, W. and S. Madden, 2004. Implementation and research issues in query processing for wireless sensor networks. In Data Engineering, 2004. Proceedings. 20th International Conference on (pp: 876-876). IEEE.
17. Gehrke, J. and S. Madden, 2004. Query processing in sensor networks. Pervasive Computing, IEEE, 3(1): 46-55.
18. Levis, P., S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, and D.E. Culler, 2004. The Emergence of Networking Abstractions and Techniques in TinyOS. In NSDI, 4: 1-1.
19. Levis, P., S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo and D. Culler, 2005. TinyOS: An operating system for sensor networks. In*Ambient intelligence* (pp: 115-148). Springer Berlin Heidelberg.
20. Li, J.Z., J.B. Li and S.F. Shi, 2003. Concepts, issues and advance of sensor networks and data management of sensor networks. Journal of software, 14(10): 1717-1727.
21. Wei Hong (Intel Research, Berkeley), Sam Madden (MIT), Implementation and Research Issues in Query Processing for Wireless Sensor Networks.
22. Wei Hong, Intel Research Berkeley, Sam Madden, M.I.T, Query in physical world by Joe Hellerstein UC Berkeley and Intel Research Berkeley
23. Zakarya, M., 2013. DDoS Verification and Attack Packet Dropping Algorithm in Cloud Computing. World Applied Sciences Journal, 23(11): 1418-1424.
24. Zakarya, M. and I.U. Rahman, A Secure Packet Drop Defense Mechanism in Wireless Mobile Ad-hoc Networks.
25. Aberer, K. M. Hauswirth and A. Salehi, (2007, May). Infrastructure for data processing in large-scale interconnected sensor networks. In Mobile Data Management, 2007 International Conference on (pp: 198-205). IEEE.
26. Demers, A., J. Gehrke, R. Rajaraman, N. Trigoni and Y. Yao, (2003, October). Energy-efficient data management for sensor networks: A work-in-progress report. In 2nd IEEE Upstate New York Workshop on Sensor Networks, pp: 1-4.
27. Perrone, L.F. and D.M. Nicol, 2002. A scalable simulator for TinyOS applications. In Simulation Conference, 2002. Proceedings of the Winter (1: 679-687. IEEE.
28. Khan, A.A. and M. Zakarya, 2010. Performance Sensitive Power Aware Multiprocessor Scheduling in Real-time Systems. Technical Journal UET Taxila (Pakistan).
29. Zakarya, M., I.U. Rahman, N. Dilawar and R. Sadaf, An integrative study on bioinformatics computing concepts, issues and problems. International Journal of Computer Science (IJCSI), 8(6).
30. Woo, A., S. Madden and R. Govindan, 2004. Networking support for query processing in sensor networks. Communications of the ACM, 47(6): 47-52.