The Threshold Join Algorithm for Top-k Queries in Distributed Sensor Networks

D. Zeinalipour-Yazti, Z. Vagena, D. Gunopulos, V. Kalogeraki, V. Tsotras
University of California - Riverside
Riverside, CA, USA
{csyiazti, foula, dg, vana, tsotras}@cs.ucr.edu

M. Vlachos
IBM T.J Watson Research
Hawthorne, NY, USA
vlachos@us.ibm.com

N. Koudas
University of Toronto
Toronto, ON, Canada
koudas@cs.toronto.edu

D. Srivastava
AT&T Research Labs
Florham Park, NJ, USA
divesh@research.att.com

ABSTRACT

In this paper we present the *Threshold Join Algorithm (TJA)*, which is an efficient TOP-k query processing algorithm for distributed sensor networks. The objective of a top-k query is to find the k highest ranked answers to a user defined similarity function. The evaluation of such a query in a sensor network environment is associated with the transfer of data over an extremely expensive communication medium. *TJA* uses a non-uniform threshold on the queried attribute in order to minimize the number of tuples that have to be transferred towards the querying node. Additionally, *TJA* resolves queries in the network rather than in a centralized fashion, which minimizes even more the consumption of bandwidth and delay. Our preliminary experimental results, using our trace driven simulator, show that TJA is both practical and efficient.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Experimentation, Performance

Keywords

Top-K Queries, Sensor Networks, Distributed Systems

1. INTRODUCTION

The advances in wireless communications along with the exponential growth of transistors per integrated circuit lead to a rapid evolution of *Wireless Sensor Devices (WSDs)*, that can be used for monitoring environmental conditions at a high fidelity. WSDs are extremely resource constrained devices with a limited energy budget [7, 8, 10]. Additionally, the communication over the radio is orders of magnitude more energy demanding than local processing.

In this paper we present the *Threshold Join Algorithm (TJA)*, which is an efficient TOP-k query processing algorithm for distributed sensor networks. The objective of a top-k query is to find the k highest ranked answers to a user defined similarity function. An example of a top-k query might be *"Find the three moments on which we had the highest average temperature in the last month?"*. Our algorithm is designed for queries where the user is not continuously interested in having the k most relevant answers. For example, biologists analyzing a forest might be interested in long-term monitoring that will allow them to understand changes and conditions over long periods of time. Therefore sensors can keep sensor readings locally and transmit their readings to a sink selectively, when certain preconditions are met, or when the sensors receive a query over the radio.

Since the execution of a query is typically associated with the transfer of data over the extremely expensive network medium, TJA's objective is to minimize this burden by transferring fewer readings to the sink and to execute in a fixed number of round trips. This keeps the radio less busy which results in tremendous energy savings. Additionally, *TJA* resolves queries in the network rather than in a centralized fashion, which further minimizes the consumption of bandwidth and delay.

The requirement of an efficient top-k query processing algorithm for WSDs, becomes evident in environments where most of the sensor readings are kept *in-situ* (at the generating sensor). This creates a network of tiny databases [12, 10] as opposed to the prevalent model of a centralized database that collects readings from many sensors [8]. The presented solutions are developed in the context of the *RISE (Riverside Sensor)* hardware platform [10], which is a wireless sensor platform with an external SD Media card which accommodates each sensor with several MBs of storage.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DMSN'05, August 29, 2005, Trondheim, Norway.
Copyright 2005 ACM 1-59593-206-2/05/0008 ...$5.00.

2. PROBLEM DEFINITION

In this section we will formalize our basic terminology.

v_1 oid,val	v_2 oid,val	v_3 oid,val	v_4 oid,val	v_5 oid,val	TOP-5 oid,val
o_3, .99	o_1, .91	o_1, .92	o_3, .74	o_3, .67	o_3, 4.05/5 = .81
o_1, .66	o_3, .90	o_3, .75	o_1, .56	o_4, .67	o_1, 3.63/5 = .73
o_0, .63	o_0, .61	o_4, .70	o_2, .56	o_1, .58	o_4, 2.07/5 = .41
o_2, .48	o_4, .07	o_2, .16	o_0, .28	o_2, .54	o_0, 1.88/5 = .32
o_4, .44	o_2, .01	o_0, .01	o_4, .19	o_0, .35	o_2, 1.75/5 = .29

Table 1: The local scores of five objects $o_1..o_5$ which are located at nodes $v_1..v_5$. The last column displays the sum of scores (overall rank).

Let R be a relation with n attributes $s_1, s_2, \ldots, s_n$, each featuring m objects $o_1, o_2, \ldots, o_m$. The j^{th} attribute of the i^{th} object is denoted as o_{ij}. Also let $G(V, E)$ denote an undirected network graph that interconnects the n vertices in V using the edge set E. The edges in E, represent the connections between the vertices in V (the set of sensor nodes). We assume that each vertex is connected to only d ($d << n$) other vertices (i.e. the average degree of the graph is d). Now assume that each sensor s_i is mapped to the elements of the vertex set $V = \{v_1, v_2, \ldots, v_n\}$ using a 1:1 mapping function $f : s_i \rightarrow v_i, \forall i$. This happens because each sensor maintains only local information (i.e. a single dimension in the n-dimensional space).

Consider $Q = (q_1, q_2, \ldots, q_n)$, a top-k query with n attributes. Each attribute of Q refers to the corresponding attribute of an object and the query attempts to find the k objects which have the maximum value in the following scoring function:

$$Score(o_i) = \sum_{j=1}^{n} w_j * sim(q_j, o_{ij}) \qquad (1)$$

where $sim(q_j, o_{ij})$, is a similarity function which evaluates the j^{th} attribute of the query Q against the j^{th} attribute of an object o_i and returns a value in the domain [0,1] (1 denotes the highest similarity). Since each attribute might have a different factor of importance, we also use a weight factor w_j ($w_j > 0$), which adjusts the significance of each attribute according to the user preferences. For instance, if the readings acquired by node v_l in the network are more important than the readings acquired by the other nodes then w_l might be set to a large value. Note that, similarly to [5, 1], we require the score function to be *monotone*. A function is monotone if the following property holds: if $sim(q_j, o_{1j}) > sim(q_j, o_{2j})$ ($\forall j \in n$) then $Score(o_1) > Score(o_2)$. This is true when $w_j > 0$ ($\forall j \in n$).

For example, suppose that we have a collection of temporal climate objects from three sensors s_1, s_2 and s_3 at two time moments o_1:(s_1=100F, s_2=90F, s_3=80F) and o_2:(s_1= 100F, s_2=70F, s_3=80F) and each attribute is of the same importance (i.e. w_j=1 $\forall j$). Furthermore assume that the distance function $sim(q_j, o_{ij})$ represents the percentage of similarity to the most similar object in dimension j. Given that some function *max* calculates the highest temperature in a given dimension, the top-1 object to the query Q= (max(temp), max(temp), max(temp)), would be o_1 because Score(o_1)=3.0 (i.e. 1*1.0 + 1*1.0 + 1*1.0) and Score(o_2)= 2.77 (i.e. 1*1.0 + 1*0.77 + 1*1.0).

3. RELATED WORK

There has been a lot of work in the area of top-k query processing in the database community. In this section we provide a brief overview of the proposed algorithms. For ease of exposition, we use a generic example which captures the problem setting of the described algorithms. Our setting includes five nodes ($v_1..v_5$) each of which maintains locally the scores of five objects ($o_0..o_4$). On table 1, we print the local scores of the five objects (ordered by highest score). The task is to find the one object which maximizes the sum of local scores across all sensors (i.e. o_3).

A querying node QN, can easily compute an answer to the top-1 query by first transferring all $m * n$ o_{ij} pairs to itself, then perform a local join, and finally extract the desired results. Assuming that each node transmits its local scores to QN either directly or over a multi-hop network, this approach would require $\sum_{i=0}^{n} \delta(v_i) * m$ messages, where $\delta(v_i)$ is the depth of node v_i from QN. Obviously this algorithm, denoted as the *Centralized Join Algorithm (CJA)*, is extremely expensive in practice. However when intermediate nodes perform some local aggregation similarly to [7], then this cost can be reduced to $(n-1) \cdot m$ which is essentially one message per edge (let this be denoted as the *Staged Join Algorithm (SJA)*). An important problem is that in our environment the value m might be arbitrary large. Therefore this approach might still be quite expensive for large-scale environments or when the query window m is large.

The *Fagin Algorithm (FA) [5]*, is one of the first top-k query processing algorithms over middleware systems which interact with a number of autonomous data sources. In FA, QN performs a two phase retrieval which consists of a sorted access and random access phase. Initially, QN accesses the n lists in parallel until it locates k objects which belong to all lists. In our working example, this would be equivalent to retrieving the first two rows of the table. QN then requests from each node to send the score for any object whose score could not be computed exactly (i.e. o_1 and o_4). Assuming a star topology, FA has with very high probability a cost of $O(m^{(n-1)/n} \cdot k^{1/n})$. However in a distributed environment, FA's execution is quite inefficient as the sorted phase might take an arbitrary large number of round trips. Additionally, the fact that each list is accessed at the same depth during the sorted phase, results in the retrieval of a large number of unnecessary object scores.

The most widely recognized algorithm for top-k queries in a centralized environment is the *Threshold Algorithm (TA)* [5]. TA starts out by performing a parallel sorted access to the n lists. While an object o_i is seen by QN, TA performs a random access to the other lists to find the exact score for o_i (i.e. $\sum_{j=1}^{n} o_{ij}$). After finding the exact score for each object in the current row[1], it computes a *threshold* value τ as the sum of all exact scores in the current row. The algorithm stops after k objects have been found with a score above τ. While the TA algorithm accesses less objects than FA, it also uses more round trips as it invokes several small random accesses. This would again translate into an arbitrary large number of phases, which is highly undesirable for a hierarchical environment.

Top-k algorithms have also been studied in other settings where the pruning of the retrieval space is highly desirable. Bruno et al. [1] discuss the problem of answering top-k queries over web accessible databases. The problem of continually providing top k answers in a distributed environment is discussed in [2]. The problem is tackled by installing

[1]Sorted access is executed on a row-at-a-time basis

arithmetic constraints at each node which define the current top-k scores at any point. The problem was later extended to hierarchical environments [4]. The TPUT[3] algorithm proposed by Cao and Wang, uses three phases in order to resolve top-k queries in star topologies. The algorithm constructs a bound which is uniform for all lists, similarly to FA, which is too coarse in practice.

Finally the very recent work in [9], examines the problem of approximate top-k queries in distributed environments. The paper assumes that each node maintains an approximation of the local scores instead of the actual scores. The approximation essentially consists of an equi-width histogram on the local scores along with a bloom filter per histogram bucket which captures object identifiers inserted into the specific bucket.

Most of the above approaches assume a star communication topology, in which all nodes are directly accessible by the querying entity. On the other hand, in our work we focus on the challenges of a hierarchical topology which is ubiquitous in sensor network environments. Specifically, our contributions can be summarized as following: i) We study the feasibility and applicability of top-k queries in hierarchical networks, ii) We propose the TJA algorithm which is a new algorithm that resolves top-k queries in a hierarchical environment using a fixed number of phases. iii) We propose methods for in-network query processing of top-k queries which reduces network traffic.

4. THE TJA ALGORITHM

We now introduce TJA, which is an efficient top k query processing algorithm for Sensor Networks. For clarity, we will refer to the collection of local scores at each node v_i, as $list(v_i)$. TJA decreases the number of objects that are required to be transmitted from each $list(v_i)$, by using an additional probing and filtering phase. In addition, the algorithm seeks to optimize the use of the network resources by pushing computation into the network. More specifically, the algorithm consists of three phases: i) the *Lower Bound* phase, in which the querying node finds a lower bound on the lists by probing the nodes in a network ii) the *Hierarchical Joining* phase, in which each node uses the lower bound for eliminating the objects that are below this bound and join the qualifying objects with results coming from children nodes and iii) the *Clean-Up* phase, in which the actual top-k results are identified. The three phases of the algorithm are presented below:

4.1 Lower Bound (LB) Phase

This phase identifies a set of objects that are used to construct a threshold. The top-k results are guaranteed to have a score above this threshold. The phase works by having each node v_i sort in descending similarity order the elements in $list(v_i)$. v_i then extracts from the sorted $list(v_i)$, the objectIDs of the k local highest ranked objects (we denote this new set as $list_k(v_i)$). As soon as some vertex v_i receives $list_k(v_j)$ from all its children v_j, it performs a union of all collected lists and creates a partial lower bound $L(v_i)$, with the following function:

$$L(v_i) = list_k(v_i) \bigcup (\bigcup_{\forall j \in children(v_i)} list_k(v_j))$$

$L(v_i)$ is then propagated to the parent of v_i and when this recursive operation terminates, the querying node will receive a list of object identifiers that define the complete lower bound $L_{queryNode}$=L_{total}=$\{l_1, l_2, \ldots, l_o\}$, $o \geq k$.

Consider our working example in table 1 (which is also summarized in figure 1c). Assume that the querying node is v_1, and that it issues the top-1 query: "Find the time moment with the highest average temperature." We note that it is easy to express such a query in our framework, by setting $w_j = 1$ and setting $sim(q_j, o_{ij})$ equal to the normalized temperature of node i at time j (this is equivalent to asking for the time moment with a vector closest to (1,..,1)).

In figure 1a, we illustrate the query spanning tree of the LB phase. Each node sends its top value: $list_1(v_i)$ for the vertices v_1, v_2, v_3, v_4, v_5 is o_3, o_1, o_1, o_3 and o_3 respectively. In our working example the lower bound is L_{total}=$\{o_1, o_3\}$. The figure shows how each intermediate node v_i calculates the partial lower bound $L(v_i)$. For example at node v_4, we perform the following union $L(v_4) = \{o_3 \cup o_3\}$, therefore v_4 is only required to propagate o_3 to v_2 rather than all the pairs in its local list.

4.2 Hierarchical Join (HJ) Phase

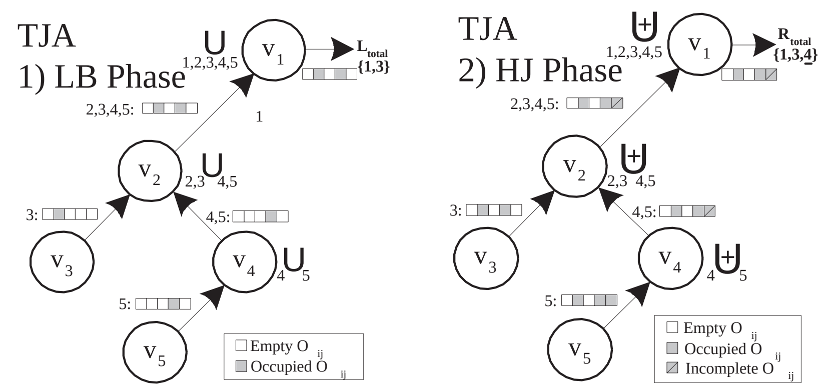
In the second phase the querying node propagates L_{total} to all nodes in the network, possibly by using the same hierarchy created in the LB phase. This requires only $n-1$ messages, where n is the number of nodes in the network. Each node receiving L_{total}, searches its local sorted $list(v_i)$ in order to identify the index of the lowest ranked object that belongs to L_{total}. More precisely, a procedure *FindMinRank* locates the lowest ranked object that belongs to L_{total}. All objects above idx are candidates for the result.

```
1: procedure FINDMINRANK($L_{total}$, $list(v_i)$)
2:     $idx = -1$
3:     for $j = 1$ to $|L_{total}|$ do
4:         if ($idx < rank(L_{total}[j], list(v_i))$) then
5:             $idx = rank(L_{total}[j], list(v_i))$
6:         end if
7:     end for
8:     return $idx$
9: end procedure
```

In the next step, each node uses the locally generated idx in order to extract the top-idx from its sorted $list(v_i)$. Let $list_{idx}(v_i)$ denote the set of o_{ij} pairs generated by this procedure. If a node is a leaf node, it simply forwards $list_{idx}(v_i)$ towards its parent. Otherwise a node waits until it receives all $list_{idx}(v_j)$ from one of its children v_j, at which point it performs a full outer join using the *FullOuterJoin* procedure illustrated next. We note that in a full outer join of two relations A and B, in addition to the rows that join on the objectID, the rows of both A and B without a match also appear in the result. However, the rows that don't match in both A and B, are marked with a *incomplete* flag. Below we present how *FullOuterJoin* ($\biguplus$) works:

$$R(v_i) = list_{idx}(v_i) \biguplus (\biguplus_{\forall j \in children(v_i)} list_{idx}(v_j))$$

The above procedure creates a local *partial result* $R(v_i)$. During this computation the algorithm computes a partial score for each object o_j in $R(v_i)$. If object o_j appears in the result list of v_i and in the result list of all its children this partial score can be computed exactly using formula 1. If on



v_1	v_2	v_3	v_4	v_5
o3,.99	o1,.91	o1,.92	o3,.74	o3,.67
o1,.66	o3,.90	o3,.75	o1,.56	o4,.67
o0,.63	o0,.61	o4,.70	o2,.56	o1,.58
o2,.48	o4,.07	o2,.16	o0,.28	o2,.54
o4,.44	o2,.01	o0,.01	o4,.19	o0,.35

— — LB ----- HJ ⬭ CL

Figure 1: The Query Spanning Tree for the phases of the TJA Algorithm: 1) Lower Bound (LB), 2) Hierarchical Join (HJ) and 3) The table shows the objects qualifying in each phase. The Clean-Up (CL) Phase is omitted as it does not contribute to the final result.

the other hand, o_j does not appear in all of the lists (and so it is marked as *incomplete* by the *FullOuterJoin* procedure), the algorithm computes an upper bound estimation on the score.

For example if object o_l does not appear in the result list of some node v_i, we compute an upper bound on the partial score of o_l by substituting $sim(q_i, o_{il})$ with $min_{a \in list_{idx}(v_i)}[sim(q_i, a)]$. Before forwarding its result list, v_i marks (using an extra bit) those score computations that are upper bounds (to differentiate them from the exact partial scores). v_i then forwards $R(v_i)$ to its parent and when this recursive operation terminates, the querying node will receive a superset of the final top-k result $R_{queryNode}$=R_{total}=$\{r_1, r_2, \ldots, r_o\}$, $o \geq k$.

In our working example the superset result is $R_{total} = \{(o_1, 3.63), (o_3, 4.05), (o'_4, 3.54)\}$ and the querying node just extracts the highest rank answer o_3 which is the result with the highest score. In figure 1b, we illustrate the QST for the HJ phase of our working example. The figure shows how each intermediate node v_i calculates the partial result $R(v_i)$. At node v_4, the full outer join of $list_{idx}(v_4) = \{(o_3, .74), (o_1, .56)\}$ and $list_{idx}(v_5) = \{(o_3, .67), (o_4, .67), (o_1, .58)\}$ generates the following partial result $R(v_4) = \{(o_3, 1.41), (o_1, 1.14), (o'_4, 1.23)\}$, were the result for o_4 is an upper bound, and the others are marked as exact.

4.3 Clean-Up (CL) Phase

In the last phase of the algorithm the querying node has collected a list of objects for which either the complete score or an upper bound of this score has been computed.

The querying node finds those objects that have upper bounds higher than the k-th complete result and computes the exact scores for these by requesting the exact scores from its children. We note that this request has to be forwarded only to those nodes that send an upper bound to the scores of these objects. Each node receiving a clean-up request uses the function $objects_{R'}(v_i)$, which fetches from the local storage of node v_i all objects in R'. As soon as v_i retrieves the list $objects_{R'}(v_j)$ from each child v_j, it joins all collected lists and creates a full score for each object in R'. This is illustrated using the below function:

$$C(v_i) = objects_{R'}(v_i) \bowtie (\bowtie_{\forall j \in children(v_i)} objects_{R'}(v_j))$$

When the querying node receives C_{total}, it computes the final top-k answers. This is achieved by joining C_{total} with R_{total}, where R_{total} is the partial result generated in the HJ phase. In our example the querying node has calculated an upper bound of 3.54 for o_4, which is less than the score of o_3 (i.e. 4.05), and so the querying node does not have to execute the CL phase.

4.4 Discussion

The advantage of the TJA algorithm over other query processing algorithms such as TA and FA is twofold:

1. Instead of performing random accesses for individual objects, TJA performs them all together in the clean-up phase. This minimizes the number of messages, and therefore also the number of transmitted bytes[2]. Additionally, it also minimizes the delay to find the expected answer.

2. By structuring the communication into three phases, we are able to increase aggregation in the query tree hierarchy. This happens because individual random accesses yield less aggregation than by combining several random accesses.

5. EXPERIMENTAL EVALUATION

In this section we describe our experimental framework which consists of a distributed trace-driven simulator written in Java that implements CJA, SJA and TJA. Our evaluation focuses on the number of transmitted bytes, the number of messages and the required time to obtain the final result in our simulation environment. While our simulation environment does not allow us to obtain fine-grained execution time, it allows us to relatively compare the algorithms.

In all cases, the presented algorithms return exactly the same result with what is returned by evaluating the query over a centralized collection of lists. In the case where the

[2]Note that each message has a fixed overhead (a header)

network suffers from failures, we also study the *average error* of the algorithms, that is, how far are the results from the correct ones.

Our experiments are based on a real dataset of atmospheric data collected at 32 sites in Washington and Oregon, by the Department of Atmospheric Sciences at the University of Washington.[3] More specifically, each of the 32 sites maintains the average temperature on an hourly basis for 208 days between June 2003 and June 2004 (i.e. 4990 time moments).

In our configuration each message has a 23 bytes header, which includes the unique identifier of the message, the payload size (i.e. how many objects are packed in the message), as well as other implementation specific parameters. Each element in $list(v_i)$ (i.e. an o_{ij} pair), requires 8 byte (oid=4 bytes, val=4 bytes) and these pairs are consecutively packed after the message header. Our query is to find the three hours (moments), at which the average *temperature* among all monitors was maximum. Our network topology consists of a connected random graph with average degree $d = 4$ and diameter 6.

5.1 Performance Evaluation

In figure 2a (f=0)[4], we plot the number of required bytes for the three algorithms when there are no failures in the network. The figure indicates that TJA requires an order of magnitude less bytes than SJA for calculating the results. More specifically TJA requires 355KB while SJA and CJA require 1.28MB and 4.4MB respectively. Figure 2b (f=0), displays the respective time for the three algorithms. As we can see TJA calculates the top k result in only 3,797 ms (LB=1,059ms, HJ=2,730ms and CL=8ms) while SJA requires 8,224 ms and CJA 18,666 ms. Finally in figure 2c (f=0), we plot the number of messages. The figure indicates that TJA requires more messages than SJA. More specifically TJA requires 246 messages while CJA requires 259 messages and SJA 183 messages. However it is important to mention that most of the messages used by TJA are small in size.

5.2 Experimentation under Failures

Node failures, battery lifetime and collisions at the MAC layer generate a dynamic environment in which sensors appear to be leaving or joining the network in an ad-hoc manner. This might not allow the querying node to obtain the correct top-k results during its execution. Therefore we define the *Average Error* function Φ, that measures the error in the rank of the each object o_i compared to its real rank (the one obtained in a stable environment). Formally Φ is defined as:

$$\Phi = \frac{1}{k} * \sum_{i=0}^{k} penalty(o_i) \qquad (2)$$

where $penalty(o_i)$ is equal to $realrank(o_i) - rank(o_i)$ if $realrank(o_i) > rank(o_i)$ or zero otherwise. Note that $realrank(o_i)$ is obtained by running a top-k algorithm locally while $rank(o_i)$ is obtained from the results coming from the network. We simulate failures by disconnecting nodes randomly and uniformly across the network and then join them back after some fixed interval. At any given moment the percentage of disconnected nodes is statistically no more than a given threshold f.

[3] http://www-k12.atmos.washington.edu/k12/grayskies/ .

[4] The rest f values will be discussed in the next subsection.

The plots in figure 2, show the number of bytes, the required time and the number of messages utilized by each of the described techniques when the failure factor is set to 10%, 20% and 30% respectively. The graphs indicate that in all techniques the number of bytes and messages decreases linearly with increasing failures because more nodes tend to loose their parents. On the other hand the required time to execute a query under failures increases for SJA and TJA. This happens because each node v_i has a timer τ_{v_i}=$\tau_{max}$$*$$ttl(v_i)$, where τ_{max} is a given threshold and $ttl(v_i)$ the time-to-live parameter taken from the query when it arrives at v_i. This decreases the waiting period of nodes deeper in the QST hierarchy. Nodes in CJA on the other hand do not use any timer as messages might arrive from an arbitrary large number of nodes (not just the children) and therefore the waiting does not remain fixed.

Finally in figure 2c we plot the Φ parameter for the different failure thresholds. The figure shows that TJA always achieves excellent resilience $\Phi < 1$. This means that the rank of each results is on average less than one position wrong. On the other hand CJA and SJA have a larger Φ value because they take longer to complete and therefore miss more results.

6. CONCLUSION & FUTURE WORK

In this paper we study the problem of finding the k highest rank answers to user query in a sensor network environment. We propose the TJA algorithm which is a new algorithm for resolving top-k queries in a hierarchical environment using a fixed number of phases. We additionally deploy in-network aggregation to minimize the utilization of the network. Our preliminary results showed that our approach is both efficient and practical. We believe that our algorithm will be a useful component for query processing engines of sensor data management systems. We are currently developing a prototype of our algorithm in nesC and plan to evaluate its efficiency using the RISE sensor [10].

7. REFERENCES

[1] N. Bruno, L. Gravano and A. Marian, “Evaluating Top-k Queries Over Web Accessible Databases”, In *Proceedings of the 18th International Conference on Data Engineering*, San Jose, CA, USA, Pages 369, 2002.

[2] B. Babcock and C. Olston, “Distributed Top-K Monitoring”, In *Proceedings of the ACM SIGMOD international conference on Management of data*, San Diego, CA, USA, Pages 28-39, 2003.

[3] P. Cao and Z. Wang, “Efficient Top-K Query Calculation in Distributed Networks”, In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, St. John’s, Newfoundland, Canada, Pages 206-215, 2004.

[4] A. Deligiannakis, Y. Kotidis, N. Roussopoulos “Hierarchical in-Network Data Aggregation with Quality Guarantees”, In *9th International Conference on Extending Database Technology*, Heraklion, Greece, March 14-18, Pages 658-675, 2004.

[5] R. Fagin, “Combining Fuzzy Information from Multiple Systems”, In *Proceedings of the fifteenth*

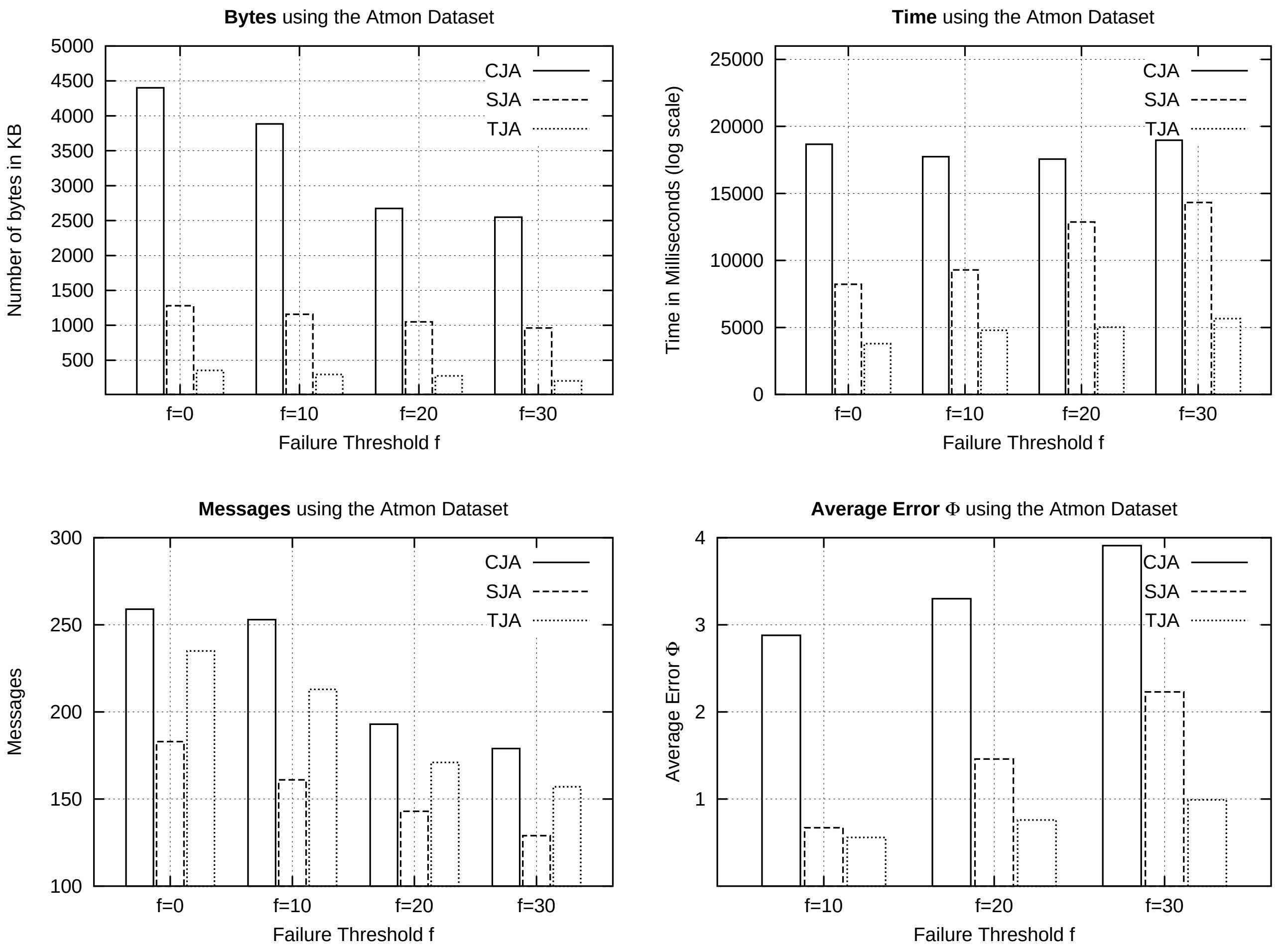


Figure 2: a) Bytes, b) Time, c) Messages and d) Average Error Φ for failure thresholds (0.0, 0.1, 0.2 & 0.3). The figures indicate that TJA requires an order of magnitude less bytes than SJA for calculating the results. TJA also minimizes the query execution time and works well under failures. The fact that TJA executes in three, rather than one phase, slightly increases the number of messages. However these messages are smaller than the huge and monolithic messages utilized by CJA and SJA.

ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, Montreal, Quebec, Canada, Pages 216-226, 1996.

[6] R. Fagin, A. Lotem and M. Naor, "Optimal Aggregation Algorithms For Middleware", In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, Santa Barbara, CA, USA, Pages 102-113, 2001.

[7] S.R. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks", In *Proceedings of the 5th symposium on Operating systems design and implementation*, Boston, MA, USA, Pages 131-146, 2002.

[8] S.Madden, M.Franklin, J.Hellerstein, W.Hong, "The Design of an Acquisitional Query Processor for Sensor Networks", In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, San Diego, CA, USA, Pages 491-502, 2003.

[9] S. Michel, P. Triantafillou, G. Weikum "KLEE: A Framework for Distributed Top-k Query Algorithms", In *31st conference in the series of the Very Large Data Bases*, Trondheim, Norway, 2005.

[10] S. Neema, A. Mitra, A. Banerjee, W. Najjar, D. Zeinalipour-Yazti, D. Gunopulos, V. Kalogeraki, "NODES: A Novel System Design for Embedded Sensor Networks", Demo at *4th International Symposium on Information Processing in Sensor Networks*, Los Angeles, CA, USA, April 25-27, 2005.

[11] Y. Yao, J.E. Gehrke, "Query Processing in Sensor Networks", In *First Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, USA, January 5-8, 2003.

[12] D. Zeinalipour-Yazti, S. Neema, D. Gunopulos, V. Kalogeraki and W. Najjar, "Data Acquision in Sensor Networks with Large Memories", In *1st IEEE International Workshop on Networking Meets Databases*, Tokyo, Japan, April 8-9, 2005.