KSpot: Effectively Monitoring the K Most Important Events in a Wireless Sensor Network

Panayiotis Andreou[#], Demetrios Zeinalipour-Yazti[*], Martha Vassiliadou[#], Panos K. Chrysanthis[‡], George Samaras[#]

[#] Department of Computer Science , University of Cyprus, P.O. Box 20537, 1678 Nicosia, Cyprus
[*] Pure and Applied Sciences, Open University of Cyprus, P.O. Box 24801, 1304 Nicosia, Cyprus
[‡] Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260, USA
p.andreou@cs.ucy.ac.cy, zeinalipour@ouc.ac.cy, cs04mv1@ucy.ac.cy, panos@cs.pitt.edu, cssamara@cs.ucy.ac.cy

***Abstract*— This demo presents a graphical user interface and ranking system, coined KSpot, for effectively monitoring the K highest-ranked answers to a query Q in a Wireless Sensor Network. KSpot deploys state-of-the-art distributed Top-k query processing algorithms in order to realize both *snapshot queries* and *historic queries* minimizing the consumption of system resources and prolonging the lifetime of the deployed sensor network. Additionally, KSpot is *user-friendly and customizable* featuring an intuitive user interface that enables a user to express declarative SQL-like queries over any ad-hoc scenario and to display the results graphically as opposed to the traditional tabular representation.**

To demonstrate the applicability of our system during the conference, we will continuously identify the K conference rooms with the highest sound level and display them such that conference attendees will be able to quickly determine the rooms with the most active discussions. The demo will also allow attendees to customize the system by changing the target scenario (e.g., by adapting the K value, the sensed parameter, etc.). Finally, we will present KSpot's system panel which continuously displays the savings in energy and messages that our system yields.

I. Introduction

Wireless Sensor Networks (WSNs) is an innovative technology that enables users to monitor and study the physical world at an extremely high resolution. Numerous applications have already emerged in environmental and habitant monitoring [1], [2], structural monitoring [3] and urban monitoring [4]. Query processing in such ad-hoc environments is a challenging task due to the complexities imposed by the inherent energy and communication constraints [5], [6], [7], [8]. To this end, the research community has proposed to take into account user-defined parameters in order to derive the K most relevant (or Top-K) answers quickly and efficiently. A Top-K query returns the subset of most relevant answers, in place of all answers, for two reasons: i) to minimize the cost metric that is associated with the retrieval of all answers; and ii) to improve the *recall* and the *precision* of the answer set, such that the user is not overwhelmed with irrelevant results. Top-K Queries can support a wide range of applications in the below categories:

i. **Snapshot Top-K Queries:** These refer to the current readings of the sensor devices. An instance in this category is a query of the type: *"Find the K nodes with the highest temperature"*, or a K-Nearest Neighbor snapshot query of the type: *"Find the K closest rangers to the fire"*, or finally a continuous snapshot query of the type: *"For the next one hour continuously report the K rooms with the highest average temperature"*.
ii. **Historic Top-K Queries**: These refer to historic data that is buffered locally on each sensor. Since a sensor has limited storage capacity it can conduct the buffering in a sliding window fashion either in main memory, for devices with adequate SRAM such as the IMote2 [9], or on secondary memory [10]. An instance in this category is a query of the type: *"Find the K time instances with the highest average temperature during the last 3 months"*, or a spatio-temporal query of the type: *"Find the K zebras with the most similar trajectories to zebra X"*.

For ease of exposition, let us consider the scenario in Figure 1 where we illustrate a deployment of 9 sensors in a 4-room building. Also, let us assume a continuous snapshot query that aims to discover the average sound level (as a percentage) of the `k=1` rooms every one minute. In particular, such a query can be expressed in an SQL-like syntax as follows:

```
SELECT TOP 1 roomid, AVERAGE(sound)
FROM sensors
GROUP BY roomid
EPOCH DURATION 1 min
```

Using the logic of the TAG in-network aggregation approach [11] utilized in the TinyDB system [5], each node could forward tuples of the form (`roomid`, `sum`, `count`) to its parent every one minute. One could then easily implement a new top-k operator at the sink (querying node) that would derive the correct answers in a centralized manner by pruning the answer-space (although none of the existing systems currently supports this feature). Even if such a technique was readily available, it is not cost effective because all tuples need to be transferred to the querying node.

To improve the performance of the above technique the

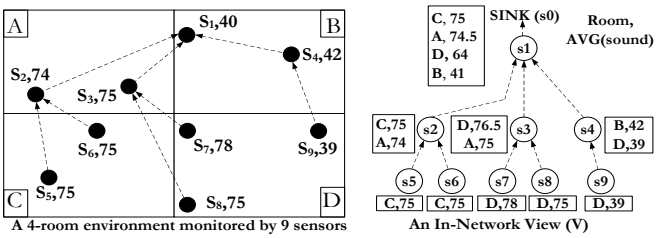

Fig. 1. The left figure illustrates a sensor network scenario that consists of 9 sensors $\{s_1, ..., s_9\}$ deployed in four rooms $\{A, B, C, D\}$. The label next to each sensor denotes the identifier of the node and the local noise level (as a percentage). The figure on the right presents the tree of readings for the respective query. The label next to each node indicates the local averages.

KSpot system conducts the pruning locally at each sensor in an *in-network* manner. This pruning goes beyond the straightforward and wrongful elimination of the local top-k results of each node (an example will be presented in Section III-A).

The KSpot system presented in this demonstration implements several top-k processing algorithms in order to overcome ranking challenges for both snapshot and historic queries. Our system consists of two software subsystems: *KSpot server* and *KSpot client*. The *KSpot server* software written in JAVA integrates distributed top-k query processing operators to the TinyDB [5] server and also features a graphical user interface that displays the requests made by users. This is essentially the base station through which user requests are disseminated to the deployed sensor network. The *KSpot client* software on the other hand is written in nesC and that unit integrates local access methods and top-k query processing functionality to a TinyOS [12] capable mote device. This is the software running on each KSpot sensor node.

Contributions:

- KSpot enables *user-friendly* and *customizable* top-k query execution in WSNs. We expect that this will enable a wide range of new applications;
- KSpot seamlessly integrates state-of-the-art distributed top-k query processing algorithms to existing data acquisition software; and
- KSpot is to our knowledge the first real prototype that addresses the query ranking problem in an in-network manner yielding enormous savings in messaging and energy.

To demonstrate the applicability of our system during the conference, we will continuously identify the K conference rooms with the highest sound level and project them on a wall/screen (or display them on a monitor) such that conference attendees will be able to quickly determine the rooms with the most active discussions. The demo will also allow conference attendees to customize the system by changing the target scenario (e.g., by adapting the K value, the sensed parameter, etc.) Finally, we will also present KSpot's system panel which continuously projects the savings in energy and messages that our system yields.

In Section II we will briefly overview the system architecture of the KSpot system. In Section III we will sketch the two algorithms we have implemented for realizing snapshot and historic queries and in Section IV we will present our equipment and demonstration settings.

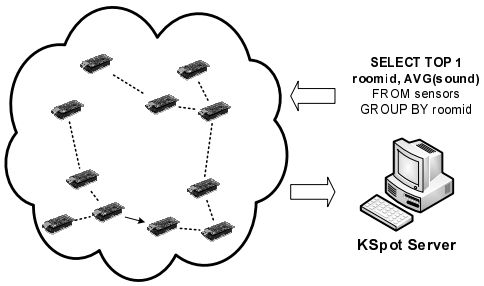

Fig. 2. The System Architecture of the KSpot System.

II. System Architecture

KSpot features a two-tier architecture that is illustrated in Figure 2. The first tier consists of a number of wireless sensor nodes which are positioned in predefined areas of interest (in our case the conference rooms). The devices are loaded with the KSpot client software running on the TinyOS [12] operating system which enables the sensors to continuously transmit their sensed data to the sink node.

The network interface of a KSpot client directs instructions (i.e., queries and other commands) from the KSpot server to the KSpot client. The local query parser of the KSpot client internally implements a query router that disseminates basic `SELECT` and `GROUP-BY` queries to the existing local query processing engine while `TOP-K` queries are routed to a specialized top-k query operator which is interfaced directly on the query parser of the KSpot client.

The second tier is a graphical user interface (KSpot GUI), which is used by users to post new queries and for displaying the query results in a manner that highlights the ranking properties of the executed query. In particular, the KSpot GUI consists of three panels (see Figure 3):

i. The *Configuration Panel* (Figure 3, top-left), which enables the user to load a new scenario from a configuration file or to create a new scenario that can be stored in a configuration file. Through this panel the user can specify which nodes belong to (are *clustered*) in the same physical region (e.g., Auditorium, Conference Rooms, Coffee Stations, etc.)

ii. The *Query Panel* (Figure 3, bottom-left), which enables the user to specify aggregate (AVG, MIN and MAX) and non-aggregate SQL-like queries either graphically or manually. The constructed query is subsequently forwarded to a modified TinyDB instance, which has been extended with Top-k operators for snapshot and historic queries (described in the next section).

iii. The *Display Panel* (Figure 3, right), which allows a user to load a JPG image representation of the scenario map. Subsequently, the user can drag-and-drop the sensing devices to the respective positions on the map. Our system allows the user to choose among a wide range of sensor devices, coming in various shapes and sizes,

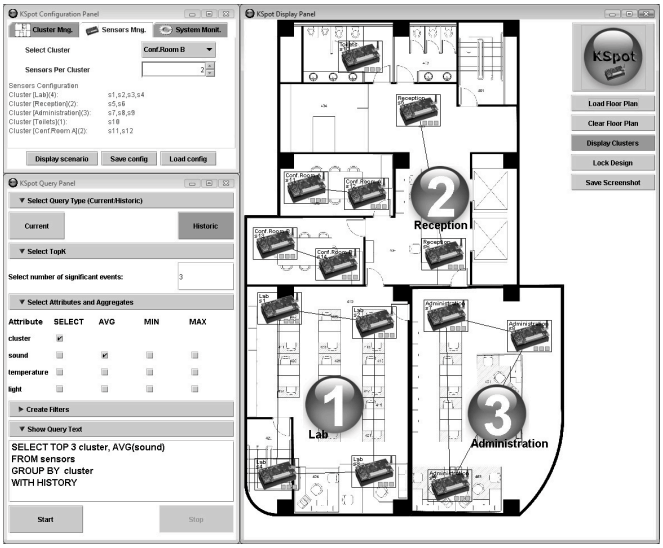

Fig. 3. KSpot's Graphical User Interface (GUI) allows users to administer the execution of Top-K Queries through an intuitive and declarative user interface. The above scenario conducts a Top-3 query over a 14-node sensor network organized in 6 clusters. The Display Panel (on the right) illustrates the three KSpot-Bullets for the three highest-ranked sensor clusters.

in order to accommodate crowded map configurations. Note that the Display Panel links together nodes of the same cluster using a black line. Additionally, the panel highlights the K-highest ranked clusters by utilizing a red bullet, coined the *KSpot Bullet*, which projects the rank of the given cluster at any given time instance. Subsequently, the KSpot bullets are continuously re-ranked such that the user is informed about the K highest ranked answers instantaneously.

III. Top-K Query Processing in KSpot

Top-k Query Processing has been studied in a variety of contexts including middleware systems, web accessible databases, stream processors, peer-to-peer systems and other distributed systems. The wave of centralized top-k query processing algorithms was succeeded in recent literature by distributed and in-network algorithms for *historic queries* (namely *TPUT* [13], *TJA* [14], *TPAT* [15]) and *snapshot queries* (namely *MINT* [16], *FILA* [17] and *BABOLS* [18]).

The KSpot system proposed in this work integrates algorithms from both classes of queries. The rationale behind this separation is the fact that there exists no universal algorithm that is optimized for both classes of queries, rather there is a pool of data processing algorithms for each class. KSpot intelligently exploits this by executing a different query processing algorithm based on the query semantics. In particular, it implements our TJA algorithm for realizing historic queries and our MINT algorithm for realizing snapshot queries. Both classes and their execution strategies are further described in the next subsections.

A. Snapshot Top-K Query Processing

A snapshot query refers to the current readings of sensor devices. The below query provides an example of such a query:

```
SELECT TOP K roomid, AVERAGE(sound)
FROM sensors
GROUP BY roomid
```

The KSpot system utilizes the MINT algorithm [16] to efficiently address this class of queries by constructing an in-network hierarchy of views, where ancestor nodes in the routing hierarchy maintain a superset view of their descendants. In particular, the MINT algorithm is divided in three phases:

1) *The Creation Phase*, executed during the first acquisition of readings from the distributed sensors. This phase results in n distributed views V_i ($i \leq n$);
2) *The Pruning Phase*, during which each sensor s_i locally prunes V_i and generates V_i' ($\subseteq V_i$). V_i' contains only the tuples that are among the final top-k results; and
3) *The Update Phase*, executed once per epoch, during which s_i updates its parent node with V_i'.

Note that in-network pruning of tuples problem is challenging as all tuples need to recursively percolate up to the sink. In particular, a naive local greedy pruning strategy may easily discard tuples that will finally be among the k highest-ranked answers. To understand this problem consider again the example of Figure 1 and assume that each node naively eliminates any tuple below its local top-1 result. Obviously, such a strategy will lead to the erroneous answer $(D, 76.5F)$, while the correct answer is $(C, 75F)$. This problem occurred because $(D, 39F)$ was eliminated by s_4. In order to avoid this, MINT utilizes a set of descriptors γ which are utilized to bound above the attributes in V_0 and subsequently enable a powerful pruning framework.

B. Historic Top-K Query Processing

A historic query refers to the case where each sensor can buffer sensor readings locally in a sliding window fashion (either in main memory or on flash). When the query refers to *horizontally fragmented* data, in which case the pruning can be conducted locally, the KSpot system extends the snapshot query described previously by conducting a local search and filtering in the respective history window before transmitting the results upwards in the query tree hierarchy. The below query provides an example of such as query:

```
SELECT TOP K roomid, AVERAGE(sound)
FROM sensors
GROUP BY roomid
WITH HISTORY {interval}
```

On the contrary, when the query refers to data that is *vertically fragmented* over the n sensors, in which case the pruning can only be conducted when the readings from all n sensors are combined, the KSpot system implements our TJA algorithm. An example of such a query is the following: *"Find the K time instances with the highest average temperature*

during the last 3 months." The TJA algorithm implemented in the KSpot system works in the following three phases:

1) The *Lower Bound* phase, in which the sink collects the union of the top-k results from all nodes in the network (denoted as L_{sink}=$\{l_1, l_2, \ldots, l_o\}$, $o \geq K$),
2) The *Hierarchical Joining* phase, in which each node uses L_{sink} for eliminating any tuple that has a value below the least ranked item in L_{sink},
3) The *Clean-Up* phase, in which the final top-k results are identified.

IV. Demonstration Settings

A. Equipment

For the actual demo at the conference we will use a set of 15 MICA2 sensors equipped with the MTS310 sensor boards. MICA2 features a low power processor and a radio module operating at 868/916 MHz enabling data transmission at 38.4Kbits/s with a outdoor range of 150 meters. MICA2 is supported by the TinyOS operating system that enables users to develop applications using the nesC language which was used to develop the KSpot client. Furthermore, MICA2 supports an expansion connector for attaching various sensor boards. We utilize the MTS310 sensor board which supports a sensor board with a variety of sensing modalities. These modalities include 2-Axis Accelerometer, 2-Axis Magnetometer, Light, Temperature, Acoustic, and Sounder.

Our base station is a MIB520 gateway which provides USB connectivity from the KSpot server to the MICA/MICA2 family of sensors. When attaching a single sensor on the gateway, it functions as the base station (sink) by collecting all results from the network. In addition to data transfer, MIB520 also provides the programming interface.

B. Demo Plan

At the conference site, and prior our demonstration, we will design a floor plan in an external CAD tool. This will yield a JPG image which will be loaded directly into the KSpot GUI. We will then position one or more sensors at various key positions throughout the conference (e.g., Auditorium, Conference Rooms, Coffee Stations, etc.) and configure the KSpot interface to display them accordingly. In particular, we will drag-n-drop the displayed sensors to the correct positions on the pre-loaded JPG plan (note that our sensing devices do not support GPS, thus topological information cannot be inferred automatically).

For the server, we will use a laptop and a projector which will present the ranking of pre-specified rooms on a screen or a white wall, such that conference attendees will have the opportunity to determine the rooms with the highest activity in discussions (i.e., sound level) at a glance. Additionally, conference attendees will have the ability to issue new Top-K queries to the network using KSpot's Query Panel. As soon as a query is issued, the audience will have the ability to view the rooms that present the highest Top-K ranking based on their provided criterion. Furthermore, conference attendees will have the ability to experiment with the System Panel which continuously displays network statistics that illustrate the efficiency of the KSpot system with regards to the number and size of the communication messages transmitted through the sensor network.

Acknowledgments

This work was supported in part by the Open University of Cyprus under the project SenseView, the US National Science Foundation under the project AQSIOS (#IIS-0534531), the European Union under the projects mPower (#034707) and IPAC (#224395).

References

[1] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler, "An analysis of a large scale habitat monitoring application," in *ACM SenSys*, 2004.
[2] T. Liu, C. M. Sadler, P. Zhang, and M. Martonosi, "Implementing software on resource-constrained mobile sensors: experiences with impala and zebranet," in *ACM MobiSys*. NY, USA: ACM, 2004, pp. 256–269.
[3] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Health monitoring of civil infrastructures using wireless sensor networks," *ACM IPSN*, pp. 254–263, 2007.
[4] R. Murty, A. Gosain, M. Tierney, A. Brody, A. Fahad, J. Bers, and M. Welsh, "CitySense: A Vision for an Urban-Scale Wireless Networking Testbed," TR-13-07, Harvard University, 2007, Tech. Rep.
[5] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: an acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, 2005.
[6] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," *SIGMOD Rec.*, vol. 31, pp. 9–18, 2002.
[7] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *ACM MobiCom '00*. New York, NY, USA: ACM, 2000, pp. 56–67.
[8] A. Sharaf, J. Beaver, A. Labrinidis, and K. Chrysanthis, "Balancing energy efficiency and quality of aggregate data in sensor networks," *VLDB*, 2004.
[9] "Crossbow technology inc., http://www.xbow.com/."
[10] D. Zeinalipour-Yazti, S. Lin, V. Kalogeraki, D. Gunopulos, and W. A. Najjar, "Microhash: An efficient index structure for flash-based sensor devices," *USENIX FAST*, 2005.
[11] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," *OSDI*, 2002.
[12] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," *ASPLOS*, 2000.
[13] P. Cao and Z. Wang, "Efficient top-K query calculation in distributed networks," *ACM PODC*, pp. 206–215, 2004.
[14] D. Zeinalipour-Yazti, Z. Vagena, D. Gunopulos, V. Kalogeraki, V. Tsotras, M. Vlachos, N. Koudas, and D. Srivastava, "The threshold join algorithm for top-k queries in distributed sensor networks," *VLDB's DMSN*, pp. 61–66, 2005.
[15] H. Yu, H. Li, P. Wu, D. Agrawal, and A. El Abbadi, "Efficient Processing of Distributed Top-k Queries," *Dexa)*, 2005.
[16] D. Zeinalipour-Yazti, P. Andreou, P. Chrysanthis, and G. Samaras, "MINT Views: Materialized In-Network Top-k Views in Sensor Networks," *IEEE MDM*, pp. 182–189, 2007.
[17] M. Wu, J. Xu, X. Tang, and W. Lee, "Monitoring top-k query in wireless sensor networks," *IEEE ICDE*, 2006.
[18] B. Babcock and C. Olston, "Distributed top-k monitoring," *ACM SIGMOD*, pp. 28–39, 2003.