STDCS: A Spatio-Temporal Data-Centric Storage Scheme For Real-Time Sensornet Applications

Mohamed Aly Anandha Gopalan
Department of Computer Science
University of Pittsburgh
{maly,axgopala}@cs.pitt.edu

Jerry Zhao Adel M. Youssef
Google Inc.
{jerryzhao,adel}@google.com

Abstract

Sensor networks will shortly consist of globally deployed sensors providing real-time geo-centric information to users. Particularly, users with mobile devices will issue ad-hoc queries usually from within, or nearby, the queried area. In this paper, we propose an in-network Data-Centric Storage (DCS) scheme, namely the Spatio-Temporal Data-Centric Storage (STDCS) scheme, to efficiently answer these mobile user queries. STDCS is designed to maintain load-balancing among sensors to cope with query hotspots in the network. It is different from previous proposals in two aspects. First, it is based on the novel idea of using a temporally evolving spatial indexing scheme to balance querying load among sensors. Furthermore, STDCS uses dynamic mechanisms for query hotspot detection and decomposition. We conducted extensive simulations that showed our scheme's superiority to both local storage and spatial indexing (the only known geo-centric storage schemes) whenever experiencing query hotspots of different sizes.

1 Introduction

The next generation of sensor networks will be composed of sensors deployed everywhere [12]. Due to their huge number, sensors will mostly be clustered into small clusters/areas and relatively addressed within each cluster rather than being assigned absolute addresses. Sensors will mostly be plug-and-play battery-operated mote-like devices and will be accessible by mobile users/devices, such as robots, cell phones, and PDAs [16, 15]. Two examples of sensor clusters are the *Bronx Zoo* cluster and the *disaster management* cluster. In the first application, motes are deployed in Bronx Zoo for habitat monitoring. The park visitors are allowed to use their mobile devices to query sensors for real-time information about animals, their behaviors, the climate they live in, etc. In the second application, sensors are deployed in the area of a disaster/emergency in an ad-hoc manner. As the first responders roam in the disaster area, they query sensors for readings that would help them to better control the disaster.

Both applications are *geo-centric* and *real-time*, i.e., users issue *ad-hoc queries* asking for real-time data generated by sensors falling in a particular area. Ad-hoc queries can be issued anywhere in the cluster [14, 3] and target real-time readings of one or more sensors in the surrounding area. One way to answer this query type is to send the sensor readings to stationary Base Stations (BSs) and let the user directly query the BSs through a wireless or Internet connection. However, this approach may be questionable, either because BSs may not be available in some clusters, such as the disaster management cluster, or because a wireless or Internet connection may not be available in the cluster, as in the Bronx Zoo cluster. Additionally, answering the query through BSs does not take benefit of the *geographic locality* characteristic of our ad-hoc queries. To efficiently answer real-time geo-centric ad-hoc queries, the sensornet can store recent readings *temporarily* in the sensor caches, so that mobile users can immediately query the sensors when asking for real-time information. A mobile user can contact sensors for example using 802.5 or through an underlying mesh network. Periodic reading synopsis (or averages) may be sent for archival in BSs, or a reading may simply be dropped after a *lifetime*, e.g. 1 hour.

In-network Data-Centric Storage (DCS) schemes previously presented in literature adopted indexing based on the sensor reading values [14, 13, 10, 3]. In a DCS scheme, a *readings-to-sensors* mapping function assigns the responsibility for storing the reading of any sensor to a storage-sensor based on the value of that reading. As our queries are geo-centric, we realize that reading-

based indexing is not a good fit for our model as processing a geo-centric query will require flooding the whole cluster. In this paper, we present the Spatio-Temporal Data-Centric Storage scheme (STDCS), a novel DCS scheme for real-time geo-centric sensornet applications. In STDCS, data indexing is based on the sensor locations rather than the reading values. Hence, STDCS presents a *sensors-to-sensors* mapping instead of the previous readings-to-sensors mappings. Our scheme embeds the sensor geographic locations into the leaves of a k-d tree [4] and assigns virtual addresses to sensors based on their positions in the k-d tree. The virtual address of each sensor is used as an input to the mapping function as to determine its storage-sensor. Any point-to-point routing scheme can be then used to route readings to their storage-sensors, e.g. the Greedy Perimeter Stateless Routing protocol (GPSR) [7]. Query processing can be easily done locally and distributively by the sensors in a hop-by-hop manner.

Our major design goal for STDCS is *load-balancing*. Traffic skewness may easily occur in our applications due to the time-varying number of users and the hard task of expecting their behaviors at any point in time. The main skewness source in our model lies in *query hotspots* [1], where most of the mobile users issue queries targeting a fairly small number of sensors simultaneously. Query hotspots may arise because of the difference in *popularity* between the readings of different sensor nodes due to the reading type, location, time, etc. The task of predicting such queries, and thus, the traffic in the network, at any time is very difficult. Traffic skewness, and in particular query hotspots, is a major problem that may result in the early death of our battery-operated sensors, network partitioning, and a subsequent reduction in network lifetime.

To maintain load-balancing, we present the novel concept of spatio-temporal data indexing, where the mapping of readings to their storage-sensors depends not only on the location of the generating sensor, but also on the generation time of the reading. Spatio-temporal indexing balances the load, in terms of query accesses, among sensors with no dependence on the query distribution imposed on the cluster. Additionally, we present a separate load-balancing scheme to *adaptively* detect and decompose query hotspots. This scheme is based on the dynamic adjustment of the parameters of the spatio-temporal data indexing. Through extensive experimental evaluation, we show that the main advantages of STDCS are:

- Highly outperforming both local storage and spatial indexing when facing query hotspots.
- Minimizing load-balancing overhead by adaptively adjusting the needed load-balancing level based on the detected skewness level of the hotspot.

Paper Organization: The rest of the paper is organized as follows. Section 2 describes the components of STDCS. Experimental results are discussed in Section 3. Section 4 presents an overview of the related literature and Section 5 concludes the paper.

2 The Spatio-Temporal Data-Centric Storage Scheme (STDCS)

Our Spatio-Temporal Data-Centric Storage Scheme (STDCS) mainly proposes a new load-balancing technique for DCS schemes taking advantage of both the spatial and the temporal characteristics of sensor readings. We present STDCS for a cluster of sensors spanning a limited geographic area. At the start of the network operation, sensors are assigned addresses, i.e., (x, y) coordinates, relative to a common reference point within the cluster. We do not assume the existence of stationary BSs in our cluster. Sensor nodes are assumed to have the capacity for wireless communication, basic processing and storage, and they are associated with the standard energy limitations. There are two main components in DCS schemes: the *sensor-to-address* mapping that assigns a virtual (i.e., logical) address to each sensor, and the *reading-to-sensor* mapping that determines a *storage-sensor* for each reading, which is the sensor responsible of storing this reading [14, 13, 10, 3]. In light of these two general components, STDCS consists of the following components:

- The repetitive splitting of the geographic area to form the underlying k-d tree, and locally assign the virtual sensor addresses.
- The spatio-temporal data indexing which uses the virtual address of each sensor s and the reading generation time to map the sensed data of s to its storage-sensor.
- The point-to-point routing scheme to deliver the reading of any sensor to its storage-sensor.
- The query processing scheme that distributively process any query issued by any mobile user.
- The dynamic adjustment of the temporal change of the mapping function based on the hotspot skewness level.

We describe each of these components in details in the subsections below.

2.1 Local Virtual Address Assignment

Our virtual address assignment scheme is similar to that used in DIM [10]. At the start of the network operation, each sensor node populates its relative address

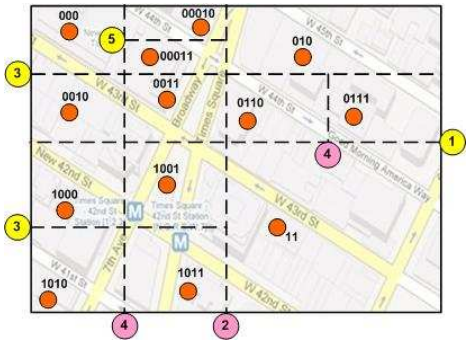

Figure 1. Virtual addresses in a cluster

(in the form of (x, y) coordinates) to its neighbors (sensors falling in its communication range). At the end of this process, each sensor makes a list of all its neighbors, $neighbors\_list$. Each sensor uses its $neighbors\_list$, together with the knowledge of the approximate boundaries of the geographic area spanned by the cluster to *locally* form its virtual address as follows. The sensor uniformly splits (i.e., bisects) the overall service area in a round robin fashion, horizontally then vertically, left shifting its bit-code with every split by 0 (or 1) bit when determining that it falls above (or below) the horizontal split line (similarly, by a 0 bit if falling on the left of the vertical split line, or a 1 bit otherwise). This *static* process partitions the area into zones, where a zone is defined to be a rectangular area well defined by an $[x_1, x_2]$ range and a $[y_1, y_2]$ range. The sensor stops applying this process the first iteration it determines that it falls by itself in a zone. At this point, the sensor relative address becomes the zone bit-code. When all sensors are done with this process, the global address assignment process ends with a complete partitioning of the service area into zones, with each zone having exactly one sensor in it. Thus, the length of the binary address of each sensor (in bits) represents its depth in the underlying k-d tree. Figure 1 shows the virtual addresses assigned to a sensor cluster. The circled numbers in the Figure represent the increasing order of the bisector.

2.2 Spatio-Temporal Data Indexing

Based on the sensor virtual addresses, STDCS uses a spatio-temporal data indexing scheme. We start by the spatial aspect of the indexing scheme, then, we describe its temporal aspect.

Our spatial data indexing scheme determines the storage-sensor of every sensor s_i in the cluster using the virtual address of s_i. The scheme uses two main system parameters: $prefix$ and $offset$. The main idea of the scheme is to partition the cluster into a set of subclusters and map sensors in each subcluster to the corresponding, or the most similar, ones in another subcluster. This subcluster mapping is *unidirectionally unique* in the sense that a cluster c_1 mapped to cluster c_2 means that every sensor in c_1 is mapped to a storage-sensor in c_2 and that sensors in c_2 are not necessarily mapped to c_1's sensors. To do so, the scheme uses the most significant bit-code of s_i's virtual address, where the length of this bit-code is equal to the value of $prefix$. Furthermore, let $base$ be a number equal to 2^{prefix} and $offset$ be defined as a random number, initially chosen between 0 and $base - 1$. Both $prefix$ and $offset$ are either set prior to the network operation for all sensors or picked by a central authority in the cluster, such as a BS (if available).

The actual spatial mapping is done as follows. Let sensors s and t be two sensors such that our spatial scheme maps s to t. The address of s is denoted by $address(s)$ with bit code length of l . Let m be the number formed by the most significant $prefix$ bits of $address(s)$, that is: $m = (address(s) >>_{l-prefix})$, where $<<_i$ and $>>_i$ are defined to be bitwise shift-left and shift-right for i bits, respectively. Let the value of h be equal to $(m + offset)\%base$, where h is the defined to be the mapped value of m. Given the value of h, the address of sensor t is formed as follows:

$$address(t) = h <<_{(l-prefix)} |(address(s)\&(2^{l-prefix}-1))$$

where $|$ and $\&$ are bitwise *or* and bitwise *and* operators, respectively. In other words, we hash the number resulting from the most significant prefix bits of the virtual address to another number of equal bit-length. Then, we concatenate the bit-code of h with the least-significant bit-code of the original address of s to get the address of sensor t. The effect of this hashing scheme is to map all the sensors in the subcluster where all sensors have the prefix bit-code equal to m to the subcluster where sensors have the prefix bit-code equal to h.

Figure 2 shows an example for the spatial mapping in a sensor cluster with $prefix = 2$ and $offset = 1$. As an example from the Figure, sensor 0110 is mapped to sensor 1010. This is done by taking the most-significant 2 bits of sensor 0110, which is the bit-code 01. As the value of this bit-code is 2, thus, the mapped bit-code will be $2+offset = 2+1 = 3$. As $3\%2^{prefix} = 3\%4 = 3$, the resulting most-significant bit-code of the mapping becomes 10. Then, we concatenate this bit-code with the least-significant bit-code of the original address, which is 10, to form the address of the storage-sensor of sensor 0110, which is 1010.

One problem with the above spatial mapping is assuming that the bit-length of both the original sensor and the storage-sensor addresses are equal. This should not always be the case. The bit-length of the storage-sensor may be less than that of the original sensor. As

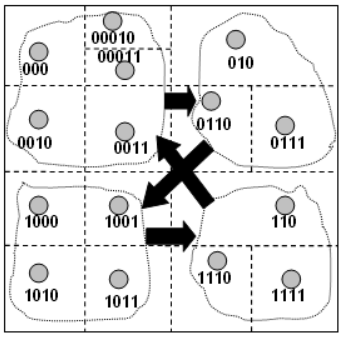

Figure 2. Spatial mapping in a cluster

an example for this case from Figure 2, sensors 1000 and 1001 are mapped to sensor 110. On the other hand, the problem becomes more tricky when the bit-length of the address of the original sensor is longer than that of the storage-sensor. For example, if we consider sensor 010 in the Figure, we find that two sensors are candidates to be its storage-sensor, namely sensors 1000 and 1001. For this case, any *arbitration rule* can be used to select one of the two sensors. For example, a 1 bit could be trivially appended to the mapping result, thus mapping 010 to 1001. A slightly more load-balance oriented rule may be to append the most-significant bit in the original sensor address, which is 0 for sensor 010, thus mapping sensor 010 to sensor 1000. It is worth mentioning that the arbitration rule is not applied except at the final hop of the routing path of the reading of sensor 010 in order to determine its exact final destination. Additionally, only one arbitration rule should be used by all sensors in the cluster.

We now move on to add the temporal dimension to our spatial data indexing. We define the *switching_time* to be the time duration after which the mapping function changes. Repeatedly applying this mapping function results in partitioning the day into *slots*, where the length of each slot is equal to the *switching_time*. At the start of each slot, all sensors change the mapping function (Recall that sensors are assumed to be synchronized). To do so, the value of $offset$ is incremented. This means that, instead of cluster 01 being mapped to cluster 10, it will be mapped to cluster 11. This temporal change of the mapping function has the effect of changing the mappings among subclusters at the start of each time slot. Note that the value of the $offset$ increment may be more than one. However, it should be constant throughout the cluster.

Our proposal of spatio-temporal data indexing scheme has its unique advantages. To understand the advantage of the spatial indexing scheme, we compare it with the *local storage* scheme, where each sensor stores its own readings. When both schemes face data skewness or query hotspots, we find that the major advantage of the spatial scheme is to reduce the load imposed on sensors falling in the hotspot area by relieving them from answering queries addressing this area and imposing the responsibility of answering those queries on other sensors, possibly less overloaded. However, the cost of this achievement is incurred in forwarding all the readings away from their originating sensors. The problem right here is that this cost is paid by all sensors in the cluster, while the achievement is only gained by a small set of sensors in the cluster. Furthermore, this achievement is not enough when we consider traffic skewness problems and specifically query hotspots as the same location where the hotspot is mapped to will be overloaded by queries after a very short time. Assuming hotspots are recurrent, or spanning long time intervals, both regions, the original one and the mapped-to one, will die after a short time period.

To complement the shortcoming of the pure spatial indexing when confronting hotspots, our proposal uses the time dimension in the mapping. Our temporal scheme changes the mapping function at the start of every time slot. The intuition behind this temporal change is that hot areas/subclusters will be mapped to different areas throughout time. Thus, it is more likely to achieve load-balancing between sensors as the high load will be decomposed among a larger set of sensors. Furthermore, in case a cluster c_2 is responsible of storing the readings of a hot cluster in one slot, it is less likely to be responsible of storing readings of other hot areas in the following time slots.

2.3 Point-to-Point Delivery of Readings

Data delivery includes delivering readings to their storage-sensors and delivering query results to the query issuer mobile users. We first focus on the the reading delivery in this subsection, then we cover the query processing in the next subsection. For both cases, our techniques are based on using an underlying point-to-point routing protocol.

Data delivery can be implemented in a *data-centric hop-by-hop* manner using the sensor virtual addresses, exactly like in DIM [10] and KDDCS [3]. In such a case, the packet destination field is set to the virtual address of the destination sensor (i.e., the result of the mapping function). Each intermediate node receiving this packet computes the Least Common Ancestor (LCA) in the k-d tree between itself and this destination. This is defined as the most-significant non-matching bit between their bit-codes. The node then determines the direction to which the packet should be sent based on the value of this bit, e.g. if the LCA bit is 0 for the destination and

the bisector of this bit is horizontal, then the packet should be forwarded up. The process continues till the packet reaches its destination.

However, the geo-centric nature of our scheme can give room to the following optimization in the reading delivery process. Whenever a sensor s computes the virtual address of the destination t, s can inversely map the virtual address of t into a zone z, which is a box formed by an $[x_1, x_2]$ range and a $[y_1, y_2]$ range, where x's and y's are relative coordinates as described in 3.1. Once zone z is formed, it can be used as a packet destination. Note that in case the underlying geographic routing protocol does not support routing to a geographic range, the packet can easily be routed to the center point of z. Each intermediate sensor compares the destination virtual address to its virtual address. Once the packet reaches the sibling-node/child-node/parent-node of z in the k-d tree, routing switches to the data-centric scheme described above to route it to its exact destination. The mix between geographic routing and data-centric routing takes benefit from the fact that our virtual addresses are purely based on geography to optimize energy consumption (as geographic routing tends to be follow a shorter path when routing a packet to its destination).

2.4 Query Processing

Our query processing scheme is based on using the data indexing scheme to *distributively* deduce the set of sensors targeted by any query and contacting those sensors to get the all readings stored by these sensors and matching the issued query.

When a mobile user sends an ad-hoc query Q, it is picked by a nearby sensor s_i, the *query issuer* sensor. Based on the parameters of Q, s_i starts the query processing of Q by first forming the *query area*, represented by the ranges $[x_1, x_2]$ and $[y_1, y_2]$ that the query involves, based on knowing its relative address and the boundaries of the cluster. Note that the query box would be truncated if it goes beyond the boundaries of the cluster. As a second step, s_i uses the query area and the boundaries of the cluster to form a bit code for the query with *don't care (d)* bits in the bit locations where the query area intersects with the uniform bisector corresponding to this bit location. As an example from Figure 2, a query issued in sensor 0111 may have a query bit-code of $01dd$ when the query spans all the 01 cluster sensors. The query bit-code is then mapped to get its destination bit-code, which is $10dd$ for our example query.

To process the query, the data-centric approach is used to process the query using the destination bit-code. Each intermediate sensor (including the query issuer) processes the bit code from left to right. The sensor first compares the bit code with its bit code to check whether it is the destination of the query. If not, the sensor gets the LCA in the k-d tree between itself and the destination. Whenever a node determines the LCA bisector, its determines the direction and the neighbor to which the query should be forwarded.

Whenever a sensor encounters that its LCA with the query is a d bit, this is an indication that the query should be splitted to two locations (up and down the horizontal bisector or left and right of the vertical bisector). Thus, the node splits the query into two queries replacing the d bit with 0 and 1, respectively. The issuer of each of the queries will be s_i, i.e., exactly like the original query. Running this process in a hop-by-hop manner will end up by splitting the query Q into a set of point-to-point requests to each of the storage-sensors storing readings of sensors falling in the query area. Whenever a storage-sensor receives its corresponding request, it processes the query against the readings it is currently storing and forms its (possibly empty) result set of Q. If not empty, this result set is sent as one or more point-to-point requests to s_i. Whenever s_i aggregates all the result sets coming from all storage-sensors (possibly after waiting for a long enough or a user-defined deadline), it forwards the resulting set to the mobile user that issued the query.

2.5 Adaptive Hotspot Decomposition

Query hotspots are mainly characterized by being dynamically changing as time progresses. For example, a lot of mobile users may be interested in restaurant occupancies in downtown Manhattan during the lunch hour. However, a much smaller set of users would be interested in the same type of data 2 hours later. To deal with such a case, our *static* spatio-temporal data indexing scheme, where the *switching_time* is set once at the start of the network operation, may be either too optimistic or pessimistic. In the first case, a large *switching_time* is used at the expense of not being able to fully load-balance hotspots if they occur. At the second case, a small *switching_time* is adopted to be able to cope with query hotspots of different sizes. However, this introduces an additional overhead in terms of the average number of sensors involved in answering any query issued to the system, especially when query distributions are mostly uniform or very lowly skewed.

To increase the efficiency of our data-indexing, we extend it with an *adaptive* query hotspot decomposition scheme. Our scheme is based on continuously keeping track of the hotspot distribution in the cluster and dynamically changing the value of the *switching_time* parameter to recapture the load-balancing effect among

the sensors as time progresses. The is based on an collecting feedback about query loads encountered by all sensors as follows. Each sensor keeps track of its Average Querying Frequency (AQF), which is the number of queries whose result set involve readings stored in that sensor. We define the variable *phase* to be a system parameter denoting a time duration. At the start of each phase, one sensor, which is either selected after a coordination between all sensors or may be constant throughout the network operation, acts like the *central authority* in the cluster. Suppose this central authority node to be B, it initiates a breadth-first search query involving all the cluster sensors and collecting the AQFs of all sensors. Results are *aggregately* sent to B. Thus, each sensor sends and receives exactly 2 messages in this process. Based on the max AQF, the collective AQF, the median AQF, and the minimum AQF of the network, B determines the approximate distribution of the AQF across all sensors. Based on this distribution, B easily determines whether a hotspot exists or no, as well as how severe (in terms of skewness) is the hotspot.

In case of a query hotspot, our scheme deduces that the current system parameters are not able to deal with the degree of skewness in the network. Thus, a parameter update needs to take place depending on the expected hotspot degree. In case the AQF distribution is highly skewed, this is an indication that a hotspot takes place in a small geographic area. For this case, it is most likely that a hotspot does not span more than one subcluster, assuming the initial setup of the system parameter values, especially the $prefix$, used a rough expectation of the query load distribution based on the popularity of the geographic areas spanned by the cluster. Therefore, decreasing the value of the $switching\_time$ would be enough to rotate the mapping of the hot subcluster around the other subclusters, thus, reachieve load-balancing in the cluster. In decreasing the value of the $switching\_time$, B follows well-defined steps with equal length. For example, assuming a user-query does not span more than 1 hour of reading generation time, a typical starting switching time would be 1 hour, while a typical decrement value would be 10 or 15 minutes.

On the other hand, in case the AQF follows a normal distribution with high tails, this indicates that a hotspot may be spanning more than one subcluster. In such a case, a good decomposition would be to increase the value of the prefix, thus, increase the number of the subclusters in the network. Additionally, the value of the increment of the offset should be changed from 1 to another random value falling to the range $[0, 2^{prefix} - 1]$. This results in mapping the subclusters of the hotspot to far apart subclusters every time slot. It should be noted that the above changes increase the number of sensors involved in any query. Thus, assuming a query hotspot occurred at some time during the network operation then was decomposed, it is not beneficial to keep the parameters set to the same values forever. Instead, B resets the system parameters when it would find the AQF distribution uniform and close to be so.

3 Experimental Evaluation

We implemented STDCS on top of GPSR [7] using the Glomosim wireless network simulator [8]. We simulated a typical cluster of stationary sensors with multiple mobile users. In our simulation setup, Sensors are randomly distributed in a service area A covering a $200m$x$200m$ square. Each sensor has an equal starting energy amount of $e = 30K$ units. It consumes 1 unit for either receiving or sending one packet. Each sensor had a communication range of $25m$. Whenever a sensor s sends a packet p to a neighbor t, only s and t consume energy for sending and receiving p. The wireless medium is assumed to be reliable and not contributing to any packet loss.

We ran simulations for networks of sizes varying between 100 and 500 sensors. At the start of every simulation, node locations are picked at random. Initially, each node broadcasts one message to know its neighbors' locations and it receives as many messages as the number of its direct neighbors. No maintenance messages are further sent during the simulation. Each sensor was programmed to sense a reading every $10min$. For simplicity, a sensor reading was assumed to be sent in one packet. Each sensor was mounted with 20 memory locations, each being able to hold one reading. We assumed 10 reading types. At the start of the network, each sensor node picked a type at random (unless a hotspot is to be simulated).

Our simulation consisted of two phases: an *initialization* phase and a *running* phase. The initialization phase consisted of running the network for 3 hours in order to achieve a steady state distribution on the sensors' storage loads. In the running phase, readings continued to be generated with the same rate. Additionally, queries started to be generated. To form a query, a sensor is drawn at random from a uniform distribution (or normal when simulating hotspots). Once a sensor is picked, a radius length is picked at random from the range $[10m, 50m]$. Then, a type is drawn at random from a uniform distribution (normal for query hotspots). Then, the query is issued by the sensor, processed by the network, and the results are sent back to the issuer sensor. For uniform loads, 1 query is generated every 5 min, while for hotspots, x queries are generated every

minute, where $x \in [10, 50]$. We simulated each run for a duration of 1 day.

We analyze the performance using the following three metrics: *throughput*, *energy level*, and *node deaths*. At the end of each run, throughput measures the number of successfully sent packets by all sensors while node deaths indicate the percentage of sensors with depleted batteries. The remaining energy level of a sensor represents the energy left at this sensor node at the end of the simulation. Both the energy level and the node deaths are important metrics to indicate how our scheme balances the work load, thus the energy consumption, among the different sensors. Throughput on the other hand shows how load-balancing query hotspots increases the network performance in terms of the number of successfully sent packets.

To provide a comparison baseline for STDCS, we also implemented two other schemes: A *local storage* scheme, where each sensors stores its generated reading, and a pure *spatial indexing* scheme, where the mapping from sensor to its storage sensor is solely determined by the location field of the reading, i.e., mapping does not change during the simulation. In all graphs below, a data point represents the average of 10 runs. It is worth mentioning that we were aware of the standard deviation in all simulation runs and we did not encounter a large variance in our simulations.

3.1 STDCS vs Query Hotspots

Our first set of experiments studied the load-balancing ability of STDCS when facing query hotspots. We first compared STDCS with local storage and spatial indexing when facing query hotspots. We set the *switching_time* to be 1 hour (worst case STDCS performance). To simulate query hotspots, 10 queries were issued per minute (i.e., lowest hotspot skewness level). Then, we ran a performance study of STDCS for different values of *switching_time* to monitor how the change in the temporal dimension of the mapping affects the overall STDCS performance. Additionally, we compared one version of STDCS for different querying frequencies to study how STDCS deals with different skewness levels.

Figures 3 and 4 compare the three schemes in terms of node death percentages and energy levels, respectively. Figure 3 shows that STDCS outperforms local storage, which in its turn outperforms spatial indexing. The difference between STDCS and local storage is almost constant and does not depend on the network size. The main reason for that is that the geographic hotspot size does not basically change with the network size, thus, the percentage of sensors affected by the hotspot

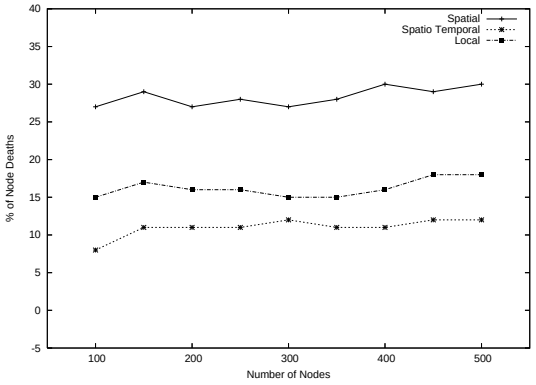

Figure 3. Node deaths for query hotspots

Energy Range	0			(0-25]			(25-50]			(50-75]			(75-100]		
Size/Scheme	STDCS	Local	Spatial	STDCS	Local	Spatial	STDCS	Local	Spatial	STDCS	Local	Spatial	STDCS	Local	Spatial
100	5	14	27	4	4	4	4	13	4	6	12	3	81	57	62
150	6	18	29	2	7	3	4	6	5	5	12	11	83	57	52
200	5	17	27	3	5	4	3	8	6	5	9	10	84	59	53
250	6	15	28	1	5	6	2	9	7	6	17	9	85	54	50
300	6	14	27	2	6	7	1	12	6	5	18	13	86	50	47
350	6	16	28	3	7	5	2	9	5	5	13	13	84	55	49
400	7	18	30	2	6	5	1	9	7	3	15	13	87	52	45
450	7	18	29	2	7	7	3	9	11	3	14	12	85	50	41
500	7	17	30	2	8	5	1	11	9	4	15	12	86	49	44

Figure 4. Energy levels for query hotspots. Values represent the percentages of sensors for each network size/energy level range/scheme triplet.

is almost the same in all cases. Considering the fact that STDCS already adds an additional burden on each sensor in the cluster for forwarding readings to storage-sensors, we realize that the load-balancing gain achieved by STDCS is fairly large when compared to local storage. Of course, the difference is expected to increase when the querying frequency increases, or or when the *switching_time* value decreases. The same conclusions follow for throughput.

Figure 4 classifies the sensors in the cluster based on their (remaining) energy levels at the end of the simulation run, for all three schemes with the different network sizes. STDCS is the scheme having the most percentage of sensors in the range $(75, 100]$, usually above 85%. Furthermore, we can see that the performance of STDCS in terms of the percentage of sensors in the the range $(75, 100]$ is almost constant (if not slightly increasing) with the increase in the network size as opposed to the decreasing performance of the other two schemes. For all network sizes, we find that STDCS is consistent in terms of the percentage of nodes falling in each range, a property that is not achieved by the other schemes. STDCS also tends to reduce the number of nodes highly affected by the hotspot. This is clear when analyzing the fairly small percentages of sensors falling in ranges $(25, 50]$ and $(50, 75]$. This shows that STDCS achieves a

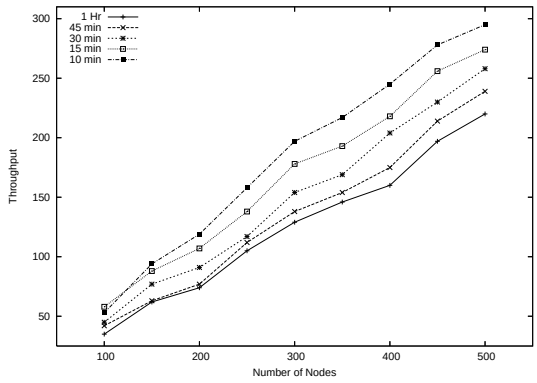

Figure 5. Switching time effect (hotspots)

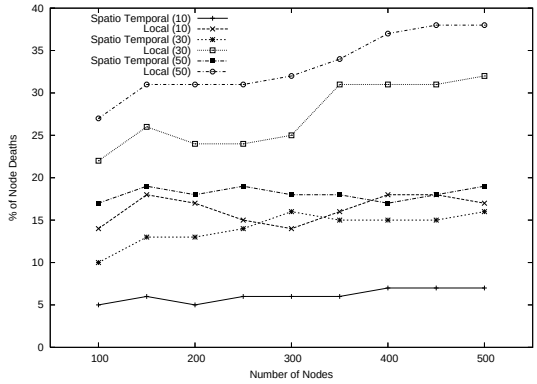

Figure 6. Node deaths vs hotspot levels

better load-balancing among sensors in terms of energy consumption. The decrease in percentages of these two ranges with the increase in the network size gives an idea about the *scalability* of STDCS, in terms of load-balancing, when compared to other schemes.

Our next step is to study STDCS's performance when the *switching_time* changes. Figure 5 compares the throughput STDCS for different *switching_time* values. For this experiment, the querying frequency was 30 queries per minute. Thus, generated query hotspots are of medium skewness levels. The Figure shows that decreasing the *switching_time* value improves the STDCS performance by decreasing increasing the network throughput (and decreasing node deaths). The difference between the different versions of STDCS tend to be constant and proportional to the *switching_time* difference. This result shows the power of the temporal indexing. It improves the hotspot decomposition and balances the load of query processing among the different sensors in the cluster. This consequently results in much more balanced energy consumption among sensors, thus reduces node deaths and increases throughput.

Our final step is to study the effect of changing the querying frequency, i.e., the skewness level of the hotspot, on the performance of our scheme. Figure 6

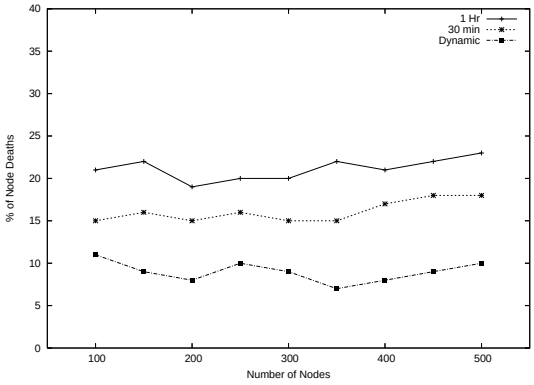

Figure 7. Adaptive STDCS performance

compares STDCS with local storage in terms of node death percentages for different skewness levels, namely 10, 30, and 50 queries per minute. To measure the worst case STDCS performance, we set the *switching_time* to be 1 hour. The main observation from the Figure is that the performance gap between both schemes considerably increase when the hotspot skewness increases. This is valid for both, node death percentages and throughput. Focusing on STDCS, we observe that its performance relatively improves when the hotspot becomes more skewed. To see that, note that STDCS node deaths are in the range of 5% when 10 queries are issued per minute and 18% when queries jump to 50 per minute. This shows how STDCS's load-balancing excels when traffic skewness increases.

3.2 Adaptive Hotspot Decomposition

In our second set of experiments, we focused on testing STDCS's adaptive query hotspot decomposition functionality and comparing its effect on STDCS with static STDCS for different *switching_time*. Our main goal was to show how much gain in performance the adaptive STDCS can reach compared to static STDCS. Figure 7 compares adaptive STDCS with static versions of STDCS of *switching_time* = 30 $min, 1\ hr$, respectively. To test a worst case performance for the adaptive STDCS, we set the querying frequency to be 10 queries per minute. The Figure shows that adaptive STDCS achieves around 5% performance gain over STDCS (30 min) and 10% over STDCS (1 hr). This performance gain shows the advantage of adaptively setting the STDCS system parameters based on the skewness level. It is worth mentioning that adaptive STDCS pays an additional cost for collecting AQFs from sensors at the start of each phase. However, this cost is compensated by the gains achieved by the adaptive query hotspot decomposition scheme. Also, the performance gains of adaptive STDCS are expected to increase when the querying frequency increases.

4 Related Work

Many approaches have been presented in literature defining how to store the data generated by a sensornet. One category of such storage solutions is to send all the data to be stored in base stations, lying within, or outside, the network. However, such approaches may be more appropriate to answer *continuous queries*, which are queries running on servers and mostly processing events generated by all the sensor nodes over a large period of time [5, 11, 17].

In order to improve the lifetime of the sensor network, as well as the *QoD* of ad-hoc queries, in-network storage techniques have been proposed. These schemes are mainly based on the *DCS* concept [14]. In-network DCS schemes differ from each other based on the events-to-sensors mapping used. The mapping was done using hash tables in DHT [14] and GHT [13], or using k-d trees in DIM [10] and KDDCS [3]. To the best of our knowledge, no spatial or temporal mapping schemes have been presented in literature.

Irregularities have been classified as a vital issue in DCS techniques [6], e.g. irregular *sensor deployments* or skewed *events distribution*. However, the problems of load-balancing traffic skewness and data storage in DCS schemes has not been thoroughly addressed in literature. Migrating data among sensors to cope with data skewness was the main idea explored. Aly et al. [1] used this idea to decompose *Query Hot-Spots* in DIM via two algorithms, Zone Partitioning (ZP) and Zone Partial Replication (ZPR). Using data migration to load-balance storage hotspots arising because of the *irregular events distribution* problem was adopted by [2, 9, 3]. These techniques vary, based on their sophistication, from decomposing storage hotspots [2, 9] to avoiding them [3]. To our knowledge, no temporal load-balancing schemes have been presented in literature.

5 Conclusions and Future Work

In this paper, we presented the Spatio-Temporal Data-Centric Storage (STDCS) scheme, a novel load-balanced in-network storage scheme for real-time geo-centric sensornet applications. Our scheme introduces the novel idea of using both the generation time and the sensor location of the sensor readings to achieve load-balancing, in terms of energy consumption, among sensors. Through extensive simulation, we showed our scheme's ability to excel versus query hotspots of different sizes when compared to local storage and plain unbalanced spatial indexing. In the future, we would like to consider the effect of skewed sensor deployments and that of heterogeneous sensor reading frequencies on the STDCS performance.

References

[1] M. Aly, P. K. Chrysanthis, and K. Pruhs. Decomposing data-centric storage query hot-spots in sensor networks. In *Proc. of MOBIQUITOUS*, 2006.

[2] M. Aly, N. Morsillo, P. K. Chrysanthis, and K. Pruhs. Zone Sharing: A hot-spots decomposition scheme for data-centric storage in sensor networks. In *Proc. of DMSN*, 2005.

[3] M. Aly, K. Pruhs, and P. K. Chrysanthis. KDDCS: A load-balanced in-network data-centric storage scheme in sensor network. In *Proc. of CIKM*, 2006.

[4] J. L. Bentley. Multidimensional binary search trees used for associative searching. In *CACM, 18(9)*, 1975.

[5] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proc. MDM*, 2001.

[6] D. Ganesan, S. Ratnasamy, H. Wang, and D. Estrin. Coping with irregular spatio-temporal sampling in sensor networks. In *Proc. of HotNets-II*, 2003.

[7] B. Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless sensor networks. In *Proc. of ACM Mobicom*, 2000.

[8] L. Bajaj, M. Takai, R. Ahuja, R. Bagrodia and M. Gerla. Glomosim: A scalable network simulation environment. Technical Report 990027, UCLA, May, 1999.

[9] X. Li, F. Bian, R. Govidan, and W. Hong. Rebalancing distributed data storage in sensor networks. Technical report, University of Southern California, 2005.

[10] X. Li, Y. J. Kim, R. Govidan, and W. Hong. Multidimensional range queries in sensor networks. In *Proc. of ACM SenSys*, 2003.

[11] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. volume 36, pages 131–146, New York, NY, USA, 2002. ACM Press.

[12] S. Nath, J. Liu, J. Miller, F. Zhao, and A. Santanche. Sensormap: a web site for sensors world-wide. In *Proc. of SenSys*, 2006.

[13] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govidan, and S. Shenker. GHT: A geographic hash table for data-centric storage. In *Proc. of WSNA*, 2002.

[14] S. Shenker, S. Ratnasamy, B. Karp, R. Govidan, and D. Estrin. Data-centric storage in sensornets. In *Proc. of HotNets-I*, 2002.

[15] Z. Vincze, D. Vass, R. Vida, A. Vidacs, and A. Telcs. Adaptive sink mobility in event-driven multi-hop wireless sensor networks. In *Proc. of InterSense*, 2006.

[16] C. Westphal. Scaling properties of routing protocols in sensor networks with mobile access. Technical report, Nokia, July 2006.

[17] Y. Yao and J. Gehrke. Query processing for sensor networks. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, 2003.