Distributed Continuous Range-Skyline Query Monitoring over the Internet of Mobile Things

Chuan-Chi Lai, *Member, IEEE*, Zulhaydar Fairozal Akbar, Chuan-Ming Liu, *Member, IEEE*, Van-Dai Ta, and Li-Chun Wang, *Fellow, IEEE*

Abstract—A Range-Skyline Query (RSQ) is the combination of range query and skyline query. It is one of the practical query types in multi-criteria decision services, which may include the spatial and non-spatial information as well as make the resulting information more useful than skyline search when the location is concerned. Furthermore, Continuous Range-Skyline Query (CRSQ) is an extension of Range-Skyline Query (RSQ) that the system continuously reports the skyline results to a query within a given search range. This work focuses on the RSQ and CRSQ within a specific range on Internet of Mobile Things (IoMT) applications. Many server-client approaches for CRSQ have been proposed but are sensitive to the number of moving objects. We propose an effective and non-centralized approach, Distributed Continuous Range-Skyline Query process (DCRSQ process), for supporting RSQ and CRSQ in mobile environments. By considering the mobility, the proposed approach can predict the time when an object falls in the query range and ignore more irrelevant information when deriving the results, thus saving the computation overhead. The proposed approach, DCRSQ process, is analyzed on cost and validated with extensive simulated experiments. The results show that DCRSQ process outperforms the existing approaches in different scenarios and aspects.

Index Terms—Internet of Mobile Things, Query processing, Range-skyline, Cooperative process

I. INTRODUCTION

IN recent years, skyline queries receive much attention in various applications such as multi-preference analysis and decision making. In such applications, a skyline set contains the most interesting objects or best objects and retrieves the objects that are not dominated by any other objects. In database systems, queries specialized to search for the non-dominated data objects are called *skyline queries* and their corresponding result sets are known as skyline sets. The data objects in a skyline set are known as skyline objects. In tradition, the skyline query is discussed in a static environment, where all the data objects and query are static. Now it is progressively extended for dynamic or distributed environments. If the user is moving or the query is issued from a dynamic environment, such a case addresses the skyline problem in dynamic environments. The skyline query is also used in spatial networks and all the considered data objects are highly distributed. Thus,

how to process skyline queries in distributed environments has become an important issue.

Conventional Location-Based Services (LBSs) focus on processing proximity-based queries, including range query [1, 2] and nearest neighbor (NN) query [3, 4]. However, these queries are not sufficient for providing high-quality services to mobile users without considering both spatial and non-spatial information simultaneously. A typical scenario is finding a nearby hotel with a cheap price, in which the distance is a spatial attribute and the price is non-spatial. Clearly, in this case, a multi-criteria query is more appealing than a conventional spatial query that considers the distance only. Among various multi-criteria queries, the skyline query is considered as one of the most classical ones and receives a great deal of attention in LBS research. However, the resulting skyline may contain many useless data objects since the resulting data objects (hotels) may be too far away from the query (user). Some other works [5, 6] tend to solve the range-skyline query to improve the QoS of LBSs by considering the dynamic data objects and supporting the *continuous range-skyline* query. The continuous range-skyline is a collection of range-skyline answers during a specific time interval that the query concerns. Such a query is applied in many LBSs whose environments are dynamic. For example, searching taxis is an application of the continuous range-skyline query. Users can use such a service to obtain some candidate taxis which are nearby, cheap, and high-ranked. Hence, this work focuses on processing the range-skyline query and the continuous case.

Most of the existing approaches [5–8] process the skyline or range-skyline in a centralized way under the assumption that data objects are stored in a centralized fashion. They provide some algorithms that focus on how to index the spatial or non-spatial data and how to efficiently process queries with a large number of data objects. Although some of them have discussed spatial queries and moving data objects in the distributed and mobile environments like the Internet of Mobile Things (IoMT) or Mobile Wireless Sensor Networks (MWSNs), the computing model is still centralized. The collected data objects are stored distributively and the process of data sensing (collection) phase is not discussed. They discussed the changes (or updates) of data and treated such cases as moving data objects. In these approaches, each mobile node needs to continuously obtain the location information of itself by GPS and sends the information back to the server(s) for updating the database(s). However, the overhead of the data collection process was not mentioned and addressed. If we use the conventional methods in a specific scenario, the position of

C.-C. Lai and L.-C. Wang are with the Department of Electrical and Computer Engineering, National Chiao Tung University, Hsinchu, Taiwan. (E-mail: cclai1109@nctu.edu.tw; lichun@g2.nctu.edu.tw)

Z. F. Akbar is with the Department of Informatics Engineering, Electronic Engineering Polytechnic Institute of Surabaya (PENS), Surabaya, Indonesia. (E-mail: fyrozal.akbar@gmail.com)

C.-C. Liu and V.-D. Ta are with the Department of Computer Science and Information Engineering, National Taipei University of Technology, Taipei, Taiwan. (E-mail: cmliu@csie.ntut.edu.tw; daitv88@gmail.com)

each data object changes frequently, the server(s) will receive a huge amount of information for updating the location of each data object in a short time. In this case, the system will be overloaded. Furthermore, a large number of messages for updates will occupy quite a lot of communication bandwidth.

In general, the IoMT applications based on MWSNs are self-configuring and infrastructure-less. IoMT consists of many mobile sensor nodes connected by wireless communication. In such environments, each node can move freely and independently in any direction, so the communication links between nodes will change frequently. Each node can forward the information unrelated to its owns and act as a router. In comparison with the client-server environment, IoMT applications may have no centralized server to handle the spatial queries. Accordingly, the information system for an infrastructure-less IoMT application must process queries in a distributed (or decentralized) way. Each mobile sensor node can cooperate and exchange data with each other and then derive the answers for spatial queries. Since most of the existing works consider multiple data sources but only a few works consider the fully distributed and dynamic computing environments, one of the main objectives of this work is to provide a fully distributed approach for processing *Range-Skyline Queries* (RSQ) and *Continuous Range-Skyline Queries* (CRSQ) in an infrastructure-less mobile environments, IoMT.

In this work, we propose a *Distributed Range-Skyline Query process* (DRSQ process) in IoMT whose computing model is decentralized. We further extend DRSQ to *Distributed Continuous Range-Skyline Query process* (DCRSQ process) for supporting continuous range-skyline query processing. Each mobile node can filter out the irrelevant data objects, derive a primary candidate answer set of the received query, and then report the candidate set to the query node. For validating the DRSQ process, we perform the simulation experiments with the following measurements: the *response time* and the *number of messages* (*I/O operations*). Furthermore, to validate the DCRSQ process, we consider four measurements: the number of accessed objects (the overhead on the query node), the number of messages, precision and recall. We also consider the effects on the number of sensor nodes, the number of queries, the transmission range of a node, and the query range in the DRSQ process. One additional effect, node speed, is considered in the DCRSQ process. Besides, we give the analysis on network cost for the proposed approach and compare the proposed approach with the centralized approach. As the results show, the proposed approach has a better performance in the simulation.

We address the distributed continuous range-skyline query (DCRSQ) processing over the IoMT and make the following contributions:

- We propose a distributed approach, DCRSQ, to make the process of range-skyline query appropriate to the Internet of Mobile Things environments.
- To the best of our knowledge, this is the first study for this problem simultaneously considering the computing process and information filtering in the data collection phase so that the performance of system is significantly improved in comparison with the conventional approach.

- With the mobility, DCRSQ can make each node predict the time when its neighboring mobile data objects fall in the query range and avoid periodically flooding messages for updating the information of neighbors.
- We give a formal analysis of the network cost on the proposed approach and conduct extensive simulations to evaluate the performance. The simulation results show that DCRSQ can save more than 15% network cost and achieve almost 90% accuracy in most of the scenarios.

The balance of this paper is organized as follows. In Section II, we introduce the background and review related research. Section III presents the overview of problem and the notations used in this work. Section IV introduces the proposed solution and a breakdown of the data structures and algorithms. Some analysis on network cost will be discussed in Section V. Simulation experiments are presented in Section VI. Finally, we make concluding remarks in Section VII.

II. RELATED WORK

The IoMT has triggered a lot of emerging applications and services [9, 10] in wireless communications, fog/edge computing, and (mobile) big sensor data processing. Ang et al. [9] identified some important research topics, like data analysis and processing, in smart city ecosystems which base on IoMT. They also investigate some use cases in IoMT (smart cities) such as real-time urban monitoring [11] and spatial decision support system for flood risk management [12]. These use cases are spatio-temporal applications classified in [10]. In spatio-temporal IoMT applications, spatial query processing plays a key role for the decision making. In our work, we focus on the range-skyline query for this kind of IoMT applications.

A range-skyline query is an extension of skyline query with a distance threshold in spatial databases. Borzsony et al. [13] introduced skyline operator into the database systems with algorithms *Block Nested Loop* (BNL) and *Divide-and-Conquer* (D&C). A great number of researchers also keep their eyes on skyline query processing from then on. Papadias et al. [7] proposed a *Branch-and-Bound Skyline* (BBS) method based on the best-first nearest neighbor algorithm [14]. Cheema et al. [15] proposed a *safe zone* based approach and combined it with Vonronoi cells to provide a better BBS algorithm for processing skyline queries. Hose and Vlachou [16] provided comprehensive analysis of previous skyline algorithms without indexing supports, and proposed a new hybrid method with improvement. Lin et al. [5] also discussed both the indexing and non-indexing algorithms, and then extended their work to process probabilistic RSQ. All these works discuss the issues in a centralized data storage.

To make the applications scalable and improve the performance of skyline query processing, many parallel or distributed algorithms have been proposed. Wu et al. [21] first attempted a progressive processing of skyline queries on a CAN-based P2P network [22]. By using the query range to recursively partition the data region involved and encoding each involved sub-region dynamically, their method can progressively report skyline objects without accessing the data sites not containing potential skyline objects, thus saving computation overhead. Chen et al. [18] proposed a parallel approach

TABLE I
COMPARISONS OF EXISTING WORKS FOR SKYLINE QUERY

Considered Issues	Methods						
	BBS [7]	LDSQ [17]	PadSkyline [18]	EDDS [19]	RSQ [5]	L-SQ [20]	**DRSQ & DCRSQ**
Distributed Databases	×	×	✓	✓	×	✓	✓
Distributed Computing	×	×	✓	✓	×	✓	✓
Moving Objects	×	×	×	×	✓	×	✓
Moving Queries	×	×	×	×	✓	✓	✓
Non-Index based	×	×	×	×	✓	×	✓

to filter data objects efficiently from distributed databases for processing constrained (or range) skyline queries. Zheng et al. [17] introduced a variant notion of the valid scope for skyline queries, that can save the re-computation if the next query is still inside the valid scope. Although the above existing works considered the distributed databases, they still used some high-performance servers to process skyline queries with the data objects from multiple data sources. Alternatively, we consider an infrastructure-less environment, IoMT, in this work.

Huang et al. [20] proposed techniques for skyline query processing in MANETs. Lightweight devices in MANETs are able to issue spatially constrained skyline queries that involve data stored on many mobile devices. Queries are forwarded through the whole MANET without routing information. However, they only considered the mobile distributed data sites over MANETs but did not consider the moving objects. Ahmed et al. [19] proposed an approach, Enhanced Distributed Dynamic Skyline (EDDS), to handle skyline queries over the IoMT. EDDS used disc track and sector to map the data locations. Such a way improves the performance of searching the new input data objects for computing and updating the skyline. Although EDDS considered the dynamically data input from the distributed sensor nodes, EDDS did not consider the mobile sensor nodes (moving objects).

In fact, the conventional works can be categorized into following models: (1) single data source with a centralized computing model, (2) multiple data sources with a centralized computing model, and (3) multiple data sources with a distributed/decentralized computing model. The third model is more popular in recent years. However, it is not easy to compare all the works in type (3) by simulation or experiments since the considered environments, assumptions, and requirements are quite different. To the best of our knowledge, most of works in type (3) consider the distributed computing model with multiple "powerful" computing servers for the query processing. Only very few works consider query processing over a lightweight mobile environment whose computing resource is limited. However, these few works only consider the static spatial data and do not support moving data objects. We therefore present a comparison summary of the existing methods related to skyline query and our work in Table I.

III. PRELIMINARIES

In this section we give some preliminaries of the problem, including some fundamental notations and definitions. We consider a data set S of sensor nodes. Each mobile sensor node $s \in S$ is associated with a spatial (i.e., location or distance) attributes and several other non-spatial attributes (e.g., temperature, trust rank, and possibility). Note that we use Euclidean distance in the spatial attribute and the non-spatial dominance relation between the mobile objects is described as Definition 1.

Definition 1 (**Non-spatial Dominance**). *Given two sensor nodes s and s', if s' is no worse than s in all non-spatial attributes, then we say s' non-spatially dominates s. We say that s' is a non-spatial dominator object of s, and s is a non-spatial dominance object of s'. Formally, it is denoted as $s' \lhd s$. The set of s's non-spatial dominator objects is denoted as $Dom(s)$, i.e., s is dominated by any object in $Dom(s)$ on non-spatial attributes.*

If $s' \lhd s$ and $s \lhd s'$ are hold, it means that non-spatial attributes of s' and s are equivalent. In this case, the system is going to check the dominance relation between the spatial attributes of s' and s. So the complete dominance relation can be described as

Definition 2. *(Dominance)*
Given a query node q and two sensor nodes s and s', if 1) s' non-spatially dominates s, and 2) $dist(q, s') \leq dist(q, s)$ (i.e., s' also spatially dominates s), then we say s' dominates s w.r.t. the query node q. Formally, it is denoted as $s' \lhd_q s$.

Note that if $s' \lhd_q s$ and $s' \lhd_q s$, it means that both spatial and non-spatial attributes of s' and s are equivalent with respect to the query node q.

With the above definitions, point-skyline query (PSQ) can be defined as

Definition 3. *(Point-Skyline Query (PSQ))*
Given a data set S, the skyline of a query node q is a subset of S in which each object (sensor node) is not dominated by any other object in S w.r.t. q. We call this subset skyline set and denote it as $PSQ(S, q) = \{s | s \in S \wedge \forall s' \in S - \{s\} : s' \not\lhd_q s\}$.

In accordance with the above definitions, the range-skyline query can be defined in Definition 4 and such a definition comes from a global view of system.

Definition 4. *(Range-Skyline Query (RSQ))*
Given a data set S with a range R, the range-skyline query with respect to q returns the skyline set of the subset of objects (sensor nodes) that locate in R. Formally, it is denoted as $RSQ(R, S, q)$, and $RSQ(R, S, q) = \{s \in S \wedge s$ locates in $R | \forall s' \in S - \{s\} \wedge s'$ locates in $R : s' \not\lhd_q s\}$.

Fig. 1 is an example of a range-skyline w.r.t. query node q, where the mobile user wants to search a nearby taxi around the query node with a high rank of service quality. The system firstly uses q's range value, R, to prune the irrelevant moving objects which are out of the range. Then the system examines dominance relations between the remaining data objects. We assume that the mobile objects with smaller weights have a higher priority in this example. Since s_1 is the nearest neighbor of q and spatially dominates all the other sensor nodes, s_1 must be in the resulting range skyline. Sensor node s_4 non-spatially dominates the other nodes in the range R, except s_3 and s_5. So the possible RSQ is $\{s_1, s_3, s_4, s_5\}$. However, s_3 is dominated by s_5 in the non-spatially attribute. Thus, $RSQ(R, S, q)$ will be $\{s_1, s_4, s_5\}$. It means that the system returns taxi s_1, s_4 and s_5 to the mobile user.

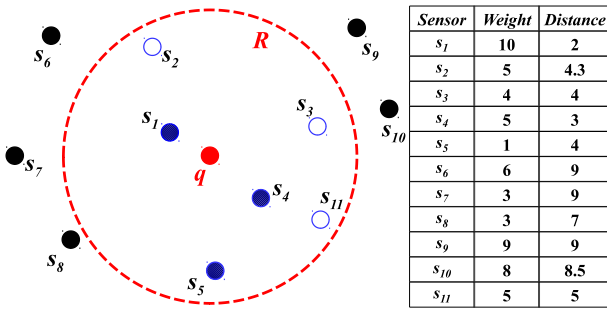| Sensor	Weight	Distance
s_1	10	2
s_2	5	4.3
s_3	4	4
s_4	5	3
s_5	1	4
s_6	6	9
s_7	3	9
s_8	3	7
s_9	9	9
s_{10}	8	8.5
s_{11}	5	5

Fig. 1. An example of an $RSQ(R, S, q)$ where S is the data set, R is the range with the query node q as the center, and the output is $\{s_1, s_4, s_5\}$

A continuous range-skyline query (CRSQ) is an extension of RSQ. CRSQ will monitor the environmental information within a given range and continuously produce the skyline for a period of time. It means that the system monitors each continuous range-skyline query q within a specific range R for a time period $\Delta t = [t_0, t_{end}]$. Since each sensor node can move in the considered environment, such a phenomenon will make the answer of an RSQ change during the monitoring time Δt. We formally define the continuous range-skyline query as below.

Definition 5. (Continuous Range-Skyline Query (CRSQ))
Suppose that the notations are defined as above. Given a query q with a query range R and a time period $\Delta t = [t_0, t_{end}]$, the continuous range-skyline query *returns a collection of range-skyline sets $RSQ_{t_i}(R, S, q)$, where $t_i \in \Delta t$ and i is the number of updated results. Formally, it is denoted as $CRSQ(R, S, q, \Delta t) = \{RSQ_{t_i}(R, S, q) | t_i \in \Delta t, i \in N\}$.*

An example of a continuous range-skyline query q is shown in Fig. 2. In this example, the system monitors the range-skyline of q for a time period Δt and the result may be a collection of answers that contains different RSQ answers at different time since the answer may change. The answer collection contains 3 RSQ results at time t_0, t_1, and t_2 and these results are respectively shown in Fig. 2(a), Fig. 2(b), and Fig. 2(c). The result of q is $CRSQ(R, S, q, \Delta t) = \{RSQ_{t_0}(R, S, q), RSQ_{t_1}(R, S, q), RSQ_{t_2}(R, S, q)\}$, where $RSQ_{t_0}(R, S, q) = \{s_1, s_4, s_5\}$, $RSQ_{t_1}(R, S, q) = $

$\{s_1, s_8, s_{11}\}$, and $RSQ_{t_2}(R, S, q) = \{s_8\}$. The final output therefore will be $\{< \{s_1, s_4, s_5\}, [t_0, t_1) >, < \{s_1, s_8, s_{11}\}, [t_1, t_2) >, < \{s_8\}, [t_2, t_{end}] >\}$.

IV. THE DISTRIBUTED CONTINUOUS RANGE-SKYLINE QUERY PROCESS

This section describes in detail the proposed distributed range-skyline query process over the IoMT. The proposed approach includes two parts: distributed range-skyline query process (DRSQ process) and distributed continuous range-skyline query process (DCRSQ process). The fundamental distributed approach for processing snapshot RSQs will be introduced in the DRSQ process. In second part, the DCRSQ process is extended from the DRSQ process with the consideration of node mobility to support CRSQ. Thus, the system can predict the change of RSQ result when monitoring a CRSQ during a period of time in IoMT environments.

A. Distributed Range-Skyline Query

In general, the query processing in mobile and distributed environments like IoMT based on MWSNs or MANETS, contains three steps. The first step is the local process that computes the skyline set based on local data and filters information received along with the query. The second step is query routing by which the query message can be forwarded to some of the neighboring nodes in order to retrieve their partial results. Thus, the node decides whether a neighbor can contribute to the skyline set based on available routing information. The last step is to merge the results, where the node receives the local result sets from queried neighbors and merges all the partial results by checking for dominated objects. Then, each node sends the merged result to the query node hop by hop.

1) Description of DRSQ: We assume that each mobile sensor node can hold a small database to store the collected sensing data and the query's information for the distributed query process. The collected sensing data set is called *local data set* and all of the data are collected from the sensor node's one-hop neighbors and itself. Hence, each mobile sensor node can derive the result of *local range-skyline query process* (LRSQ process) which may be the subset of range-skyline and return the result to the query node for computing the final (global) range-skyline answer. The result of LRSQ is defined in Definition 6.

Definition 6. (Local Range-Skyline Query)
Suppose that the notations are defined as above and a query node q with a query range R is given. After a mobile sensor node s_j receives the query q, s_j will return a subset of the local data set S_{s_j} and each object s in S_{s_j} is s_j's neighbor and not dominated by any other object s' in S_{s_j} w.r.t. q, where $S_{s_j} \subseteq S$. We refer to this result as a local range-skyline set and denote it as $LRSQ_{s_j}(R, S, q) = \{s$ locates in $R \wedge s \in S_{s_j} | \forall s' \in S_{s_j} - \{s\} \wedge s'$ locates in $R : s' \not\preceq_q s\}$.

According to Definition 6, the query node q will receive the results of $LRSQ_{s_j}(R, S, q)$, where s_j is q's one-hop neighbor $(1 \leq j \leq k)$ and k is the maximum number of neighbors.
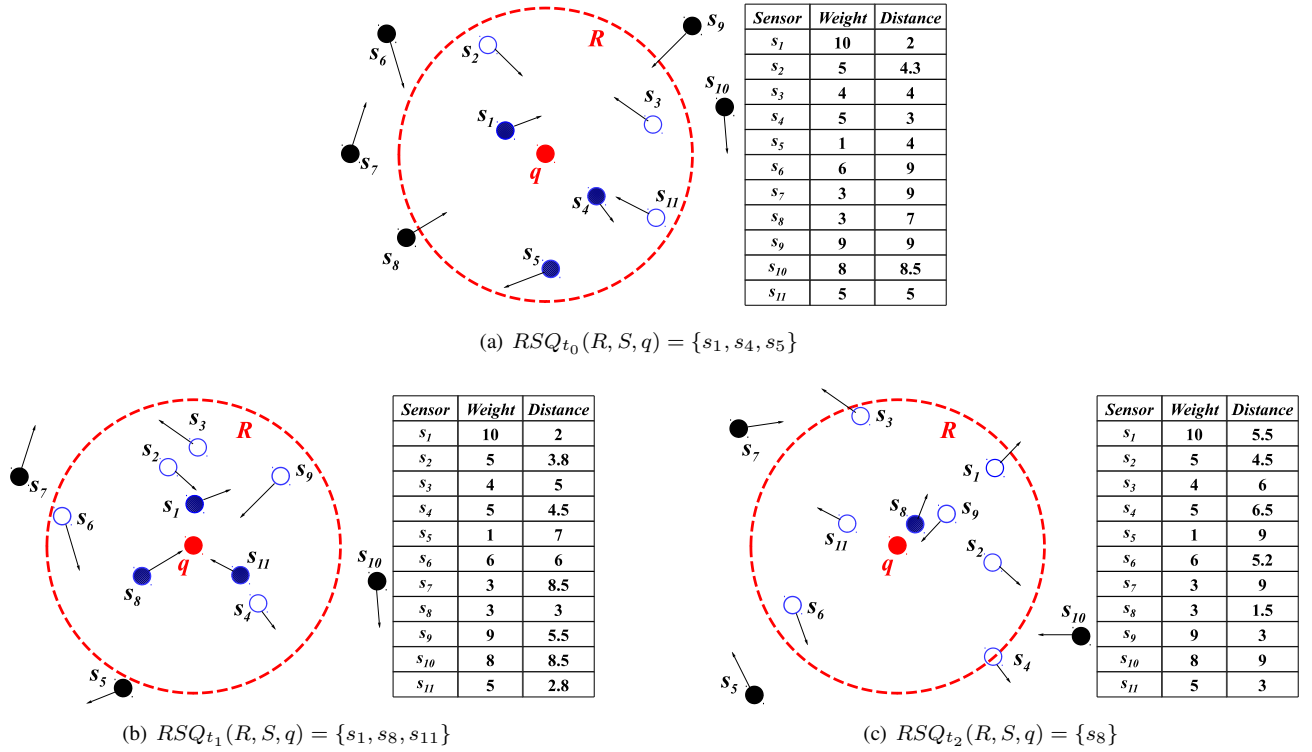
(a) $RSQ_{t_0}(R, S, q) = \{s_1, s_4, s_5\}$

(b) $RSQ_{t_1}(R, S, q) = \{s_1, s_8, s_{11}\}$

(c) $RSQ_{t_2}(R, S, q) = \{s_8\}$

Fig. 2. An example of a $CRSQ(R, S, q, \Delta t)$ and the output is $\{< \{s_1, s_4, s_5\}, [t_0, t_1) >, < \{s_1, s_8, s_{11}\}, [t_1, t_2) >, < \{s_8\}, [t_2, t_{end}] >\}$ where t_0, t_1, and t_2 are different times during $\Delta t = [t_0, t_{end}]$

Then the query node takes the union of these results as the candidate set, $RSQ_{candidate} = \bigcup_{j=1}^{k} LRSQ_{s_j}(R, S, q)$. After $RSQ_{candidate}$ is derived, the query node will use Definition 1 and Definition 2 to examine the dominance relations of all the mobile objects in $RSQ_{candidate}$ again and then save the final result in $RSQ_{distributed}$ set. Such a cooperative and distributed process is refer to *distributed range-skyline query process* (DRSQ process). As a result, DRSQ can be defined as Definition 7.

Definition 7. *(Distributed Range-Skyline Query)*
Suppose the candidate set $RSQ_{candidate}$ of query node q has been computed. Then, the query node q uses $RSQ_{candidate}$ to derive the skyline set of the data objects in R and the result of distributed range-skyline query can be denoted as $DRSQ(R, S, q) = \{s \text{ locates in } R \wedge s \in RSQ_{candidate} | \forall s' \in RSQ_{candidate} - \{s\} \wedge s' \text{ locates in } R : s' \not\vartriangleleft_q s\}$.

The above distributed process for LRSQ derivation has a benefit that many irrelevant data objects are also pruned during the process. Thus, the computation overhead of the query node can be reduced.

2) Overview of DRSQ process: Before introducing the DRSQ processes on a query node and a sensor node with pseudo-codes in detail, we use a running example in Fig. 3 to depict the overview of DRSQ process and explain it step by step. Note that the spatial and non-spatial attributes of each sensor node (data object) are shown in Fig. 1.

As shown in Fig. 3(a) and Fig. 3(b), a query node spreads the query message m_q $(TTL = 2)$ to its one hop and two-hop neighbors. Each message m_q contains the information of query node, such as query range R, location, and speed

of q. Fig. 3(c) then shows that two-hop neighbors of q, s_2, s_3, s_5, and s_{11}, use their own local information to derive their local range-skyline results and return these local range-skyline results to q's one-hop neighbors, s_1 and s_4. After s_1 and s_4 receive the local range-skyline results from the two-hop neighbors of q, they will merge the received local range-skyline results and do the dominance check with the information of themselves. As Fig. 3(c) shows, the local skyline candidate sets of s_1 and s_4 are $LRSQ_{s_2} = \{s_1, s_2\}$ and $LRSQ_{s_3} \cup LRSQ_{s_5} \cup LRSQ_{s_{11}} = \{s_3, s_4, s_5\}$ respectively. Sensor nodes s_1 and s_4 then check the dominance relations between all the candidate data objects and obtain their local range-skyline results, $LRSQ_{s_1} = \{s_1, s_2\}$ and $LRSQ_{s_4} = \{s_4, s_5\}$ because of $s_5 \vartriangleleft_q s_3$ respectively, as Fig. 3(d) shows. After aggregating the local range-skyline sets, $LRSQ_{s_1}$ and $LRSQ_{s_4}$, s_1 and s_4 return them to the query node q respectively.

After receiving the local range-skyline sets from s_1 and s_4, query node q merges these sets to derive a candidate set $RSQ_{candidate} = LRSQ_{s_1} \cup LRSQ_{s_4} = \{s_1, s_4, s_5\}$. Fig. 3(e) presents the step of obtaining a candidate set of q. Finally, in Fig. 3(f), query node q checks the dominance relations between all the data objects in local candidate set and derives the final range-skyline, $DRSQ(R, S, q) = \{s_1, s_4, s_5\}$.

3) The DRSQ Process: In this subsection, we introduce DRSQ process with pseudo-codes in detail. The whole DRSQ process includes two parts: LRSQ and GRSQ processes. The pseudo-codes of Algorithm 1 and Algorithm 2 respectively show the LRSQ process on a mobile sensor node and the GRSQ process on the query node as well as present the ideas and frameworks of our proposed approaches. Note that each sensor
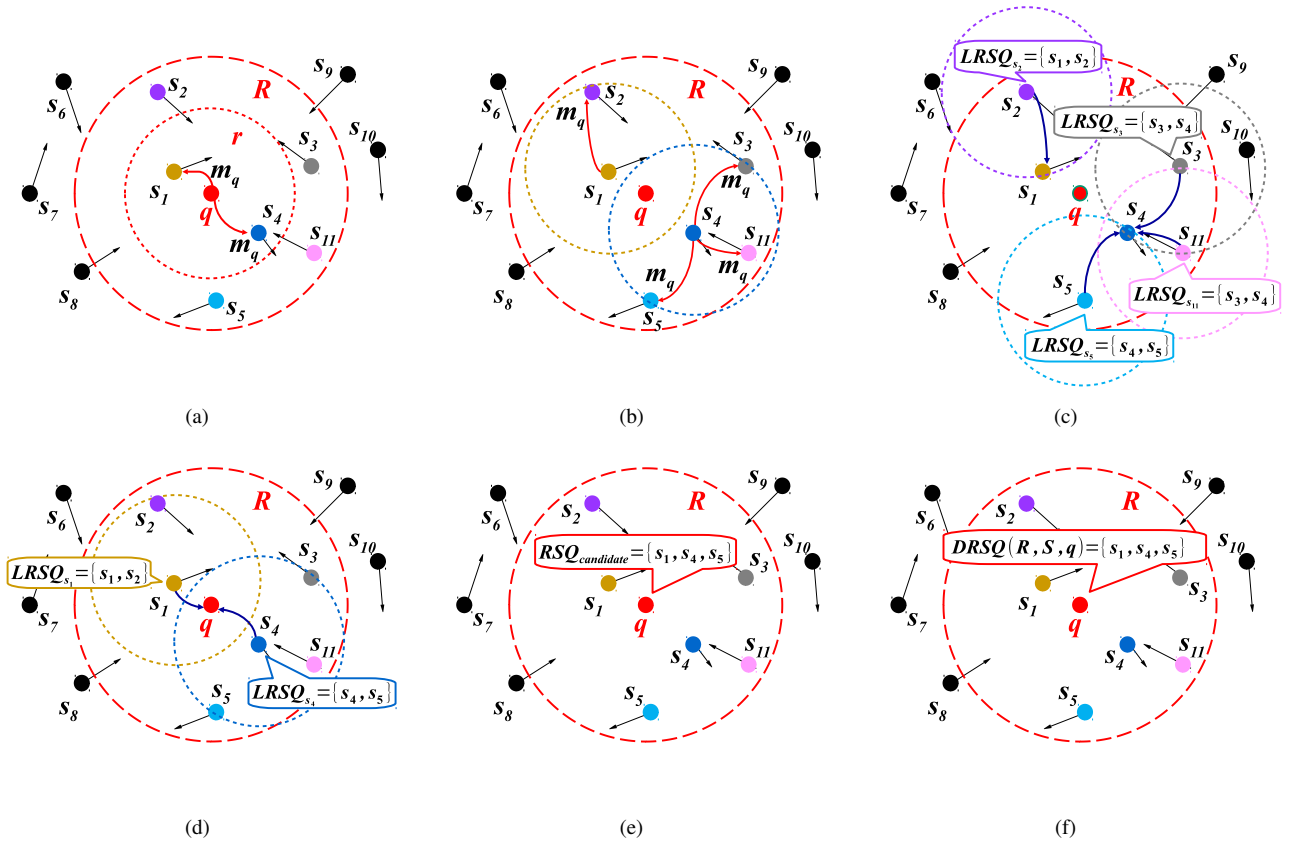
Fig. 3. A running example of DRSQ process where the data set is $S = \{s_1, ..., s_{11}\}$, m_q is a query message, and the TTL of m_q is 2: (a) one-hop neighbors s_1 and s_4; (b) two-hop neighbors s_2, s_3, s_5, and s_{11}; (c) two-hop neighbors derive and return their LRSQ results to the one-hop neighbors; (d) one-hop neighbors merge all the received information, prune the irrelevant information, and then return LRSQ of themselves to q; (e) q unites all the received information and obtains a candidate set; (e) q derives the final result after checking the dominance relations between all the data objects in the candidate set.

node repeatedly runs the LRSQ process in Algorithm 1 for a query and thus continuously receives messages from the network. When a user (mobile device) issues a query q, the device floods query messages to its one-hop neighbors with a maximum hop count, *Time-To-Live* (TTL). After flooding the query messages, the query node starts GRSQ process (in Algorithm 2) to collect the local skyline sets from its one-hop neighbors and then derives the final result, $RSQ_{distributed}$, for the query.

When each one-hop neighbor of q receives the query messages, it will do the operations from Line 5 to Line 15 of LRSQ process (Algorithm 1). If the TTL value in the received query message is larger than 0, it means that the mobile sensor node is an intermediate node in the routing path of the query and the sensor node will forward the query to its neighbors at Line 9. Otherwise, the mobile sensor node is an end node in the routing path of the query. The sensor node will stop forwarding the query message, add the data objects of itself to the local skyline set RSQ_{local} at Line 13, and then start to return the RSQ_{local} to the query node (at Line 33) through the reversed routing path of the query. Note that RSQ_{local} is equal to the term, $LRSQ_{si}(R, S, q)$, and we may use both interchangeably afterward in this paper.

When an intermediate sensor node receives a response message for the query, it will perform the operations from Line 16 to Line 31 in Algorithm 1 to compute the latest local range-skyline RSQ_{local}. Since the received response message contains a local range-skyline set w.r.t. the neighboring node which sent this message, the intermediate mobile sensor node will save the received local range-skyline in a temporary set $RSQ_{neighbor}$. Note that all the data objects in $RSQ_{neighbor}$ do not dominate each other, so the sensor node will check the dominance relations between each data object o' in $RSQ_{neighbor}$ and the data object o of itself. If data object o' in $RSQ_{neighbor}$ is not dominated by data object o, the data object o' is still a local range-skyline member. The operations of dominance relation checking are presented from Line 20 to Line 27 of Algorithm 1. The mobile sensor node executes the operation at Line 29 if the data object o of itself is not dominated by any other data objects in $RSQ_{neighbor}$. It indicates that the sensed data object o of the intermediate sensor node becomes one of the local range-skyline member. After the dominance validation, the intermediate sensor node keeps forwarding the response message, including the latest local range-skyline, back to the query node. All the intermediate sensor nodes do the above operations and update the local skyline set which is saved in the response message until the response message is received by the query node.

Algorithm 2 describes the operations executed by the query node q after it floods the query messages. From Line 4 to Line 11, the query node collects all the local range-skyline sets

---

Algorithm 1: LRSQ process on a mobile sensor node

Input: received message m and neighbor list $list_{neighbor}$
Output: local range-skyline RSQ_{local}

1   $q \leftarrow$ new query object;   /* create a temporary empty query object */
2   $s \leftarrow$ new node;   /* create a temporary empty source node */
3   $RSQ_{local} \leftarrow \emptyset$;   /* create a local range-skyline set */
4   $o \leftarrow$ this.sense();   /* save self's environmental data in a temporary object */
5   **if** $m.type == RSQ\_TYPE$ **then**
6     $q \leftarrow$ this.parse(m, RSQ_TYPE);   /* save the query information to q */
7     $s.address \leftarrow m.source\_address$;   /* record the previous node s */
8   **if** $m.TTL > 0$ **then**
9   this.flood(m, $m.TTL - 1$);   /* forward message m to all the neighbors */
10   **return**;
11   **else if** $m.TTL == 0$ **then**
12   **if** o locates in q's range **then**
13   add o into RSQ_{local};
14   **end**
15   **end**
16   **else if** $m.type == RSQ\_REPLY\_TYPE$ **then**   /* LRSQ */
17    $RSQ_{neighbor} \leftarrow \emptyset$;   /* create a temporary range-skyline set */
     /* get the neighbor's local range-skyline set */
18    $RSQ_{neighbor} \leftarrow$ this.parse(m, RSQ_REPLY_TYPE);
     /* check dominance relations between the recieved objects and itself */
19   int $isDominated = 0$;
20   **foreach** data object o' in $RSQ_{neighbor}$ **do**
21   **if** $o \lhd_q o'$ **then**
22   **continue**;
23   **else if** $o' \lhd_q o$ **then**
24      $isDominated = 1$;
25   **end**
26     $RSQ_{local} \leftarrow RSQ_{local} \cup \{o'\}$;
27   **end**
28   **if** $isDominated == 0$ **then**
29     $RSQ_{local} \leftarrow RSQ_{local} \cup \{o\}$;
30   **end**
31   **end**
     /* return m' to q through the previous sensor node s in q's routing path */
32   $m' \leftarrow$ this.createMessage(RSQ_{local}, RSQ_REPLY_TYPE); /* only in DRSQ approach */
33   this.forward(m', s);   /* only in DRSQ approach */
34   **return** RSQ_{local};

Algorithm 2: GRSQ process on the query node

Input: received message m and neighbor list $list_{neighbor}$
Output: distributed range-skyline $RSQ_{distributed}$

1   $RSQ_{distributed} \leftarrow \emptyset$;   /* create a set to save the distributed range-skyline */
2   $RSQ_{candidate} \leftarrow \emptyset$;   /* create a set to save the candidate range-skyline */
3   int $i = 0$;
4   **repeat**
5   **if** $m.type == RSQ\_REPLY\_TYPE$ **then**
6     $RSQ_{neighbor} \leftarrow \emptyset$;   /* create a temporary range-skyline set */
     /* obtain the neighbor's local range-skyline set */
7     $RSQ_{neighbor} \leftarrow$ this.parse(m);
8     $RSQ_{candidate} \leftarrow RSQ_{candidate} \cup RSQ_{neighbor}$;
9     $i++$;
10   **end**
11   **until** $i == list_{neighbor}.length$;
     /* check dominance relations between the recieved candidiates and itself */
12   **foreach** data object o in $RSQ_{candidate}$ **do**
13   int $isDominated = 0$;
14   **foreach** data object o' in $(RSQ_{candidate} - \{o\})$ **do**
15   **if** $o' \lhd_q o$ **then**
16      $isDominated = 1$;
17   **break**;
18   **end**
19   **end**
20   **if** $isDominated == 0$ **then**
21     $RSQ_{distributed} \leftarrow RSQ_{distributed} \cup \{o\}$;
22   **end**
23   **end**
24   **return** $RSQ_{distributed}$;

---

dominance relations between all the candidate points and then saves the non-dominated data objects in a set, $RSQ_{distributed}$. Finally, the query node (the user's device) returns the final range-skyline, $RSQ_{distributed}$, to the user.

B. Distributed Continuous Range-Skyline Query

This subsection is organized as follows. We will first present some important notations and assumptions for the DCRSQ process. Second, we will introduce the proposed DCRSQ approach with a running example extended from the CRSQ example in Fig. 2. Last, the proposed DCRSQ process will be explained in details with some definitions.

1) System Assumptions: In order to make the DRSQ process able to support CRSQ in the DCRSQ process, some additional and modified assumptions of the system are needed and we will describe them in detail before introducing the DCRSQ process. Details of the system assumptions are given as follows:

- Each mobile node can always obtain the spatial (location and mobility) and non-spatial (sensed data) informations of its one-hop neighbors and itself with its GPS equipment. We call such collected informations as *local information*.
- Each mobile node has a limited buffer to store the received CRSQ queries and each stored query will be continuously processed with local information until the query is expired.

from its one-hop neighbors and merges them into the candidate range-skyline set $RSQ_{candidate}$. Such a union operation is done to avoid recording the same data objects multiple times. Note that the Line 5 is a operation to check whether the received message is a reply message or not. Such a check can avoid the routing loop of a query message. In the considered environment, the query node q is a sensor node and may be an i-hop neighbor of a node s, so q may receive the query message from s if $TTL > 2i - 1$, where the appropriate value of TTL will be discussed in Section V. Such a scenario may only occur when the query range R is much larger than the transmission range r.

However, $RSQ_{candidate}$ is not the final result for the query because the data objects in all the received local range-skyline sets may dominate each other. So the query node executes the operations from Line 12 to Line 23 for checking the

- Each node also collects the information of existing queries from its neighbors while generating local information of itself.
- The size of a network packet (message) is fixed and one packet only can store one data object.

We here show all the important notations used in this paper in Table II.

TABLE II
IMPORTANT NOTATIONS

Notation	Description
$\mathcal{A}$	The sensing area
S	The set of all the mobile nodes
s	Sensor node (mobile object)
q	Query node
R	Query range
r	Transmission range (Sensing range)
N	Number of mobile sensor nodes
N_R	Number of mobile sensor nodes in the query range
N_r	Number of neighbors in the transmission range
Δt	A period of time for monitoring a CRSQ
T	A time interval for each sensor node return its information periodically
T_{safe}	A predicted time period that the answer may change
t_{M_s}	The monitoring time of node s w.r.t. q
m_q	Query message
m_r	Reply message
TTL	Time-To-Live (Hop count) of a message
$dist$	Euclidean distance

2) Overview of DCRSQ: To support CRSQ, the DCRSQ process should take node mobility into account because the movement of nodes may cause frequent change of the answer. We thus adapt the idea of [23], *safe-time*, for considering the mobility of nodes. The safe time is derived for a node to predict when a neighbor node enters and leaves the query q's range. If a neighboring object leaves the range of q, this object will not to a point of range-skyline and thus it will not be processed on the sensor node. It means that the node can determine that processing this neighboring object is necessary or not with the safe-time information.

For deriving the precise safe time, we consider the movement of mobile nodes simultaneously and use Fig. 4 to illustrate. Initially, $dist(q, s)$ is the distance from the query node q to sensor node s, where $dist(q, s) \leq R$. In the following, we present the equation to calculate the safe time $t_{\overline{qs}}$ when $dist(q, s) = R$. Suppose the initial location of object $q(s)$ is $(x_q, y_q)((x_s, y_s))$ with speed $(v_{x_q}, v_{y_q})((v_{x_s}, v_{y_s}))$. Since $dist(q, s) = R$, we can have

$$
\begin{aligned}
R^2 &= [(x_q + v_{x_q} * t_{\overline{qs}}) - (x_s + v_{x_s} * t_{\overline{qs}})]^2 \\
&+ [(y_q + v_{y_q} * t_{\overline{qs}}) - (y_s + v_{y_s} * t_{\overline{qs}})]^2. \quad (1)
\end{aligned}
$$

After transposing each term, (1) can be a quadratic equation. Then, we can simply use the discriminant of quadratic equation to get two values of the safe time. We select the minimum positive value as the safe time $t_{\overline{qs}}$. Note that the above example shows the case of a leaving node. The other scenario is that a node may enter the range of query q. If the node is out of the q's range and receives the query message in advance, the time periods of its entering and leaving can also be obtained by the same way.

For example, a query q issues a CRSQ with $\Delta t = [3, 10]$ at time t_0. Fig 5 shows the relative locations of s and q at each
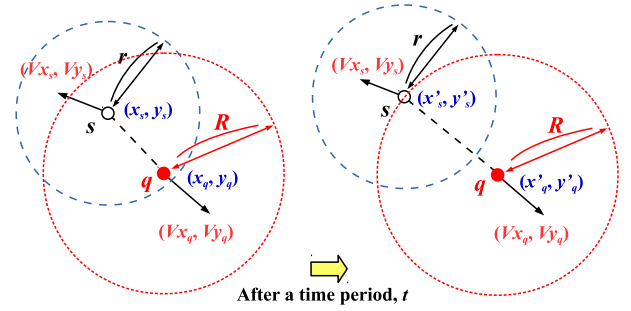

Fig. 4. An example for deriving the safe time $t_{\overline{qs}}$ when $dist(q, s) = R$

time step t_i where $i \geq 0$. The query q can use (1) to obtain the safe time of node s, $t_{\overline{qs}} = [t_{enter}, t_{leave}] = [1, 6]$. So the exact monitoring time of node s w.r.t q, t_{M_s}, is $[3, 6]$, since the q only concerns the results during the time $\Delta t = [3, 10]$ and the node s will leave the range of q after time t_6.
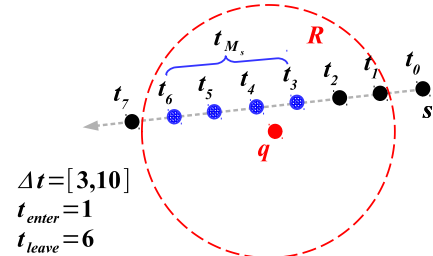

Fig. 5. The monitoring time ($t_{M_s} = [3, 6]$) of node s w.r.t. q where $\Delta t = [3, 10]$,

Combine the prediction of monitoring time with the LRSQ process, the continuous local range-skyline candidate sets also can be obtained. We refer such a process to *continuous local range-skyline query process* (CLRSQ process). Consider the query $CRSQ(R, S, q, \Delta t = [0, 3])$ in Fig. 2, the nodes s_2, s_3, s_5, and s_{11} in CLRSQ process (modified from the LRSQ process in Fig. 3), respectively return information of their local range-skyline candidate sets during the time $\Delta t = [0, 3]$. Node s_2 returns $CLRSQ_{s_2} = \{< \{s_1, s_2\}, [t_0, t_1) >, < \{s_1, s_2\}, [t_1, t_2) >, < \{s_2\}, [t_2, t_3] >\}$ back to the intermediate node s_1. Nodes s_3, s_5, and s_{11} respectively return $CLRSQ_{s_3} = \{< \{s_3, s_4\}, [t_0, t_1) >, < \{s_3, s_{11}\}, [t_1, t_2) >, < \{s_3, s_{11}\}, [t_2, t_3] >\}$, $CLRSQ_{s_5} = \{< \{s_4, s_5\}, [t_0, t_1) >, < \{s_4\}, [t_1, t_2) >, < \{s_4\}, [t_2, t_3] >\}$, and $CLRSQ_{s_{11}} = \{< \{s_3, s_4\}, [t_0, t_1) >, < \{s_3, s_{11}\}, [t_1, t_2) >, < \{s_3, s_{11}\}, [t_2, t_3] >\}$ to the intermediate node s_4. In such a case, node s_1 will use the received $CLRSQ_{s_2}$ and the local information of itself to derive the $CLRSQ_{s_1} = \{< \{s_1, s_2\}, [t_0, t_1) >, < \{s_1, s_2\}, [t_1, t_2) >, < \{s_2\}, [t_2, t_3] >\}$; and node s_4 will use the received $CLRSQ_{s_3}$, $CLRSQ_{s_5}$, $CLRSQ_{s_{11}}$, and the local information of itself to calculate the $CLRSQ_{s_4} = \{< \{s_4, s_5\}, [t_0, t_1) >, < \{s_3, s_{11}\}, [t_1, t_2) >, < \{s_3, s_{11}\}, [t_2, t_3] >\}$. Finally, with received $CLRSQ_{s_1}$ and $CLRSQ_{s_4}$, the query node q can obtain a predicted final result of CRSQ, $DCRSQ(R, S, q, \Delta t) = \{< \{s_1, s_4, s_5\}, [t_0, t_1) >, < \{s_1, s_3, s_{11}\}, [t_1, t_2) >, < \{s_3, s_{11}\}, [t_2, t_3] >\}$, at time t_0.

Unfortunately, the predicted result may not be correct. In

the above example, $DCRSQ(R, S, q, \Delta t)$ is not equal to $CRSQ(R, S, q, \Delta t)$ since the query node q cannot obtain the information of node s_8 at time t_0. So the result needs to be updated continuously. In the DCRSQ process, three cases may happen if a node enters the range of q. First, if s_8 received a query message from q, s_8 would return its local range-skyline while entering the range of q. It means that at least one node locates in the range of q at time t_0 and it is the intermediate (relay) node between s_8 and q. In such a case, the intermediate node has the information of s_8 and uses that to derive the predicted CLRSQ result. Then the predicted DCRSQ result should be a correct answer, $DCRSQ(R, S, q, \Delta t) = \{< \{s_1, s_4, s_5\}, [t_0, t_1) > , < \{s_1, s_8, s_{11}\}, [t_1, t_2) >, < \{s_8\}, [t_2, t_3] >\}$. However, the mentioned example is not in this case since there is no intermediate node between s_8 and q. It is the second case that s_8 does not receive any information of q at time t_0. According the assumptions of the DCRSQ process, s_8 will obtain the information of q from its neighbors when s_8 enters the range of q after time t_1. Hence, the RSQ results at time t_1 and t_2, $< \{s_1, s_3, s_{11}\}, [t_1, t_2) >$ and $< \{s_3, s_{11}\}, [t_2, t_3] >$, will be updated to $< \{s_1, s_8, s_{11}\}, [t_1, t_2) >$ and $< \{s_8\}, [t_2, t_3] >$, since $s_8 \lhd_q \{s_3, s_{11}\}$. The last case is that s_8 cannot successfully obtain the information of q when it enters the range of q. Such a case will be recognized as an incorrect result and it only occurs when the mobile environment is too sparse.

3) Description of DCRSQ: According to Definition 5, the system will process the CRSQ for a period of time Δt and derive the collection of possible answers. However, such a definition comes from the global and centralized view of system. In the previous subsection, the distributed method for processing RSQ has been introduced with Definition 6 and Definition 7. In the DCRSQ process, we use a mechanism to make each node able to predict the change of LRSQ answer with the node mobility. With above definitions and examples, the formal descriptions of CLRSQ and can be defined as Definition 8 and Definition 9 respectively.

Definition 8. *(Continuous Local Range-Skyline Query)*
Suppose that the notations are defined as above and a query $CRSQ(R, S, q, \Delta t)$ is issued by the query node q. After a mobile sensor node s_j receives the query message from q, s_j will return a collection of local range-skyline sets $LRSQ_{s_j}(R, S, q, t_i)$, where $t_i \in \Delta t$ and i is the number of local answer change. Formally, it is denoted as $CLRSQ_{s_j}(R, S, q, \Delta t) = \{LRSQ_{s_j}(R, S, q, t_i) | t_i \in \Delta t, i \in N\}$.

According to Definition 8, the query node q will receive the results of $CLRSQ_{s_j}(R, S, q, \Delta t)$, where s_j is one-hop neighbor of q, $1 \leq j \leq k$, and k is the maximum number of neighbors. Then the query node q will take the union of received results, which are the local range-skyline sets for time t_i, as $RSQ_{candidate}(R, S, q, t_i) = \bigcup_{j=1}^{k} LRSQ_{s_j}(R, S, q, t_i)$. Thus, the candidate collection can be denoted as $CRSQ_{candidate} = \{< RSQ_{candidate}(R, S, q, t_0), [t_0, t_1) > , < RSQ_{candidate}(R, S, q, t_1), [t_1, t_2) >, \ldots, < RSQ_{candidate}(R, S, q, t_i), [t_i, t_{end}] >\}$. After deriving $CRSQ_{candidate}$, q will check the dominance relations of all

objects in each $RSQ_{candidate}(R, S, q, t_i)$ set again and then obtain the final result $DRSQ(R, S, q, t_i)$ at each time t_i. We call such a process *distributed continuous range-skyline query process* (DCRSQ process) and the definition is given in Definition 9.

Definition 9. *(Distributed Continuous Range-Skyline Query)*
Suppose the candidate collection $CRSQ_{candidate}$ of query node q has been computed. Query node q uses $CRSQ_{candidate}$ to derive a collection of the range-skyline sets for different time t_i and we use $DCRSQ(R, S, q, \Delta t)$ to represent such a collection of continuous range-skyline sets, where $DCRSQ(R, S, q, \Delta t) = \{DRSQ(R, S, q, t_i) | t_i \in \Delta t, i \in N\}$.

Note that the fundamental process of DCRSQ is similar to DRSQ process mentioned in section IV-A3. The main difference is that each mobile node s_j generates a continuous local range-skyline set $CLRSQ_{s_j}(R, S, q, \Delta t)$ with the safe-time information of candidate nodes. Thus the DCRSQ process can provide sufficient information to the query node q for deriving, predicting, and updating the answer as time continuously goes on. Algorithm 3 gives the high-level description of DCRSQ process. To implement Line 12 and Line 13 of Algorithm 3, we use the idea of *sliding window*, which is already a widely used design in many domains. Since it is out of the scope of this paper, we will not address it. In addition, if the Δt is the specific time of the query issuing, $[t_i, t_i]$, $t_i \geq 0$, the query will be a snapshot RSQ and the DCRSQ will do the same process as the DRSQ process does.

V. COST ANALYSIS AND DISCUSSION

Suppose that N mobile data objects are distributed independently and uniformly in the sensing area, $\mathcal{A}$, and each data object has d attributes. If all the attributes of each object are in a uniform distribution, the skyline search problem can be treated as the problem of finding the maxima [24] in an $N \times d$ matrix. Hence, the expected size of skyline will be $n_{sky} = O((\ln N)^{d-1})$. In the considered environment, the query node does not need to process all the mobile sensor nodes (or data objects) for the range-skyline query and thus the expected size of range-skyline will be $n_{range-sky} = O((\ln N_R)^{d-1}) \leq O((\ln N)^{d-1})$, where N_R is the number of mobile sensor nodes in the query range R and $0 \leq N_R \leq N$. Note that the value of N_R is influenced by the value of N, sensing area $|\mathcal{A}|$ and the query range R and $N_R = \lfloor \frac{\pi R^2 N}{|\mathcal{A}|} \rfloor$. According to the above notations, the average number of data objects in the transmission range of a mobile sensor node will be $N_r = \lfloor \frac{\pi r^2 N}{|\mathcal{A}|} \rfloor$, where r is the transmission range of a mobile sensor node. If $N_r \leq 1$, the density of mobile sensor nodes is too sparse and thus it is too hard to route messages. In such a case, none of the conventional centralized and proposed approaches can perform well in the CRSQ processing. Hence, we only discuss the case, $N_r > 1$, in this work. Note that we do not discuss the case here $N_R \leq 1$ since none of mobile sensor nodes can serve this query.

In the considered IoMT, the mobile sensor nodes in the query range have to return information to the query node

---

Algorithm 3: DCRSQ process on a mobile node (both query and sensor node)

Input: received message m and neighbor list $list_{neighbor}$

1   $RSQ_{distributed} \leftarrow \emptyset$;   /* create a set to save the distributed range-skyline */

2   $RSQ_{local} \leftarrow \emptyset$;   /* create a set to save the latest local range-skyline */

3   $RSQ_{current\_local} \leftarrow \emptyset$;   /* create a set to save the previous local range-skyline */

4   $list_{safe\_time} \leftarrow \emptyset$;   /* create a list to record the safe time of neighbors */

5   **if** $this.nodeType == QUERY\_NODE$ **then**

6   **repeat**

      /* call the Algorithm 2 to update the final distributed range-skyline */

7       $RSQ_{distributed} \leftarrow$ GRSQ$(m, list_{neighbor})$;

8   **until** $m.isExpired()$;

9   **else if** $this.nodeType == SENSOR\_NODE$ **then**

10   **if** $m.type == RSQ\_REPLY\_TYPE$ **then**

      /* derive the safe time of each neighbor */

11      $list_{safe\_time} \leftarrow$ UpdateSafeTime$(list_{neighbor})$;

      /* call the Algorithm 1 to update the local range-skyline */

12      $RSQ_{local} \leftarrow$ LRSQ$(m, list_{neighbor})$;

      /* update the local range-skyline with the safe time values of neighbors */

13      $RSQ_{local} \leftarrow$ SafeTimeCheck$(RSQ_{local}, list_{safe\_time})$;

      /* return the update message when the local range-skyline changes */

14   **if** $RSQ_{current\_local}$ is not equal to RSQ_{local} **then**

15       $s \leftarrow$ new node;

16       $s.address \leftarrow m.source\_address$;

17       $m' \leftarrow this.createMessage(RSQ_{local}, RSQ\_REPLY\_TYPE)$;

18       $this.forward(m', s)$;

19   **end**

20   **end**

21   **end**

22   **return** $RSQ_{distributed}$;

---

in hop-by-hop manner. The possibility distribution function of each hop in a multi-hop wireless environment has been discussed in [25] and we use that to obtain the possibility P_i of the ith hop transmission. To obtain sufficient information for deriving the accurate result of a RSQ, the system must guarantee that more than N_R neighboring nodes of the query node can receive the query message. Then we can denote such an expected network cost for spreading the query message as

$$E[q_{spread}] = \sum_{i=1}^{k} N_r^i \prod_{j=1}^{i} P_j, \qquad (2)$$

where N_r^i is the average number of ith-hop neighbors with respect to the query node q. Since all sensor nodes in the query range should be notified with the query messages from q, we can find a minimum value of $k \in \mathbb{N}$ that $E[q_{spread}] \geq N_R$. Hence, the expected hop count TTL_q can be derived by (2) and $TTL_q = k$.

The process of data collection in the centralized approach is straightforward and each of the mobile sensor node which receives the query message will return the information of itself to the query node. Since the ith-hop neighbor needs to return an i-hop response message to the query node, the network cost of the reply messages for the ith-hop neighbors will be $N_r^i \times i \prod_{j=1}^{i} P_j$ without the cooperative pruning.

Hence, the expected network cost for returning messages in the centralized approach can be denoted as

$$E_{centralized}[q_{response}] = \sum_{i=1}^{k} N_r^i \times i \prod_{j=1}^{i} P_j, \qquad (3)$$

where $k = TTL_q$ is determined by (2) with the constraint $E[q_{spread}] \geq N_R$. In summary, the total network cost of the centralized approach for a RSQ, q, can be denoted as

$$E_{centralized}[q] = E[q_{spread}] + E_{centralized}[q_{response}]. \quad (4)$$

In the proposed approach, DRSQ process, each node derives the local range-skyline and the expected size of result is $O(\ln N_r)^{d-1}$. The reason is that DRSQ process combines the information filtering into the data collection, thus reducing a large number of irrelevant response messages. Hence, the network cost for replying the information can be denoted as

$$E_{DRSQ}[q_{response}] = \sum_{i=1}^{k} N_r^i (\ln N_r^i)^{d-1} P_{k-i}, \qquad (5)$$

and $E_{DRSQ}[q_{response}] < E_{centralized}[q_{response}]$ in normal cases. So the total network cost of DRSQ process can be estimated as

$$E_{DRSQ}[q] = E[q_{spread}] + E_{DRSQ}[q_{response}]. \qquad (6)$$

For monitoring a CRSQ query in the centralized approach, the query node has to spread the query message periodically during the time period Δt and each neighboring node also has to periodically return the information of itself. So the network cost of the centralized approach can be denoted as

$$E_{centralized}^{CRSQ}[q] = \frac{|\Delta t|}{T} \times E_{centralized}[q], \qquad (7)$$

where T is the time interval that each mobile sensor node periodically reports the updated information to the query node and the default value of T is 1 second. In DCRSQ process, the query node does not have to periodically spread query messages since each mobile sensor node can buffered the information of the query. So the network cost is mainly influenced by the frequency of the answer changes and it can be derived by

$$E_{DCRSQ}^{CRSQ}[q] = E[q_{spread}] + \frac{|\Delta t|}{\overline{T_{safe}}} \times E_{DRSQ}[q_{response}], \quad (8)$$

where $\overline{T_{safe}}$ is the average safe-time that the result needs to be updated.

VI. SIMULATION RESULTS

All of the simulations are implemented as custom programs using C++ and executed on a Windows 7 system with an Intel i5-4460 3.20GHz CPU and 8GB memory. In all the simulation scenarios, the mobile sensor nodes are distributed uniformly and the results are reported with the average of 200 executions. The used mobility model is Random Way Point (RWP) and the network routing protocol is AODV [26]. Since none of existing works provides distributed RSQ process over IoMT environments, we thereby use a centralized method [7] as

the compared centralized approach and it is executed on the query node for calculating the query results. In the centralized approach, the query node directly uses the flooding scheme to spread query messages and then collects information from moving data objects.

The proposed approach, DCRSQ process, can support (snapshot) range-skyline query and continuous range-skyline query. If $\Delta t = [t_0, t_{end}] = 0$, where $t_0 = t_{end}$, the DCRSQ process will perform the DRSQ process for deriving the results of the (snapshot) RSQ at time t_0. We thus organize the simulation section as two scenarios. In the first scenario, we discuss the performance of DRSQ process in terms of *response time* and *number of messages*. The response time is the period of time from issuing a RSQ to obtaining the result during the DRSQ process. The number of messages represents the necessary network cost on data collection.

In the second scenario, the performance of DCRSQ process is discussed in terms of *number of accessed objects* and *number of messages*. Additionally, the correctness of DCRSQ result is discussed in terms of *precision* and *recall*. In both scenarios, the following important factors are discussed: *density (number of sensor nodes)*, *number of queries*, *query range*, and *transmission range*. For validating the DCRSQ process, an additional factor, *node speed*, is also in the discussion.

A. Scenario I: Performance of DRSQ Process

In the first scenario, we discuss the performance of DRSQ process. There are 100 mobile sensor nodes in a $400m \times 400m$ square sensing area. The default transmission range is $75m$ and the node speed is $5m/s$. Initially, mobile sensor nodes and queries are placed randomly in the area. The basic simulation settings for the first scenario are shown in TABLE III and, we execute the simulation 200 times to get the average results and the 95% confidence intervals under each scenario.

TABLE III
SIMULATION PARAMETERS FOR SCENARIO I

Parameter	Default Value	Range (type)
Sensing Area ($m \times m$)	400×400	–
Number of Sensor Nodes	100	50, 100, 150 ,200
Number of Queries	1	1, 2, 3, 4, 5
Query Range, R (m)	80	60, 80, 100, 120
Transmission Range, r (m)	75	50, 75, 100, 125
Node Speed (m/s)	2	1, 2, 3, 4, 5
TTL of Messages (centralized approach)	5	–
Bandwidth (Mb/s)	2	–

To the best of our knowledge, none of existing works proposed a method for processing range-skyline queries in such an environment, where the databases, CPUs, and memory are fully-distributed. We hence compare the proposed approach, DRSQ process, with a centralized approach which is a baseline. Note that the centralized approach does not use a powerful server. In the centralized approach, we assume that the query node is a sink node and can process the query with received information. The other mobile nodes only just forward the query and response messages without processing the local range-skyline.
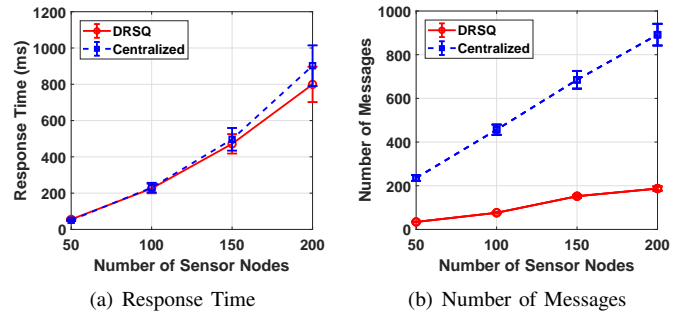

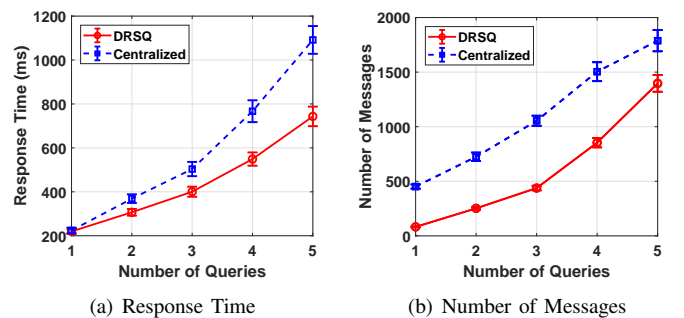Fig. 6. Impact of the number of sensor nodes on (a) response time and (b) number of messages

Fig. 7. Impact of the number of queries on (a) response time and (b) number of messages

1) DRSQ: Density: Fig. 6(a) shows that the response time of our proposed distributed approach, DRSQ process, is 10% better than the centralized approach when the density becomes more dense (the number of sensor nodes increases). Although both centralized approach and DRSQ process need to collect the information from the other sensor nodes, DRSQ process spends less time on data collection. There are two reasons. One is that DRSQ process only needs to access the sensor nodes around the query range. Conversely, the centralized approach asks all the sensor nodes in the considered environment for data collection. The other reason is that the query node using the centralized approach needs to process many data objects and the computation overhead is thus heavy.

Fig. 6(b) presents that DRSQ process is almost 75% better than the centralized approach in term of number of messages. In the DRSQ process, each mobile sensor node collects the information of its neighbors and derives a local RSQ result before sending a response message to the query node. Such a process can effectively prune a lot of irrelevant information from data objects (sensor nodes) and thus cost less number of messages on returning the local range-skyline. On the contrary, the centralized approach just floods query messages and collects the information from all the neighboring mobile sensor nodes to derive the range-skyline. It thus wastes more network cost on data collection.

2) DRSQ: Number of Queries: In this subsection, we discuss the impact of the number of queries. The number of queries indicates the maximum number of queries concurrently processed in the system. Fig. 7(a) shows that the response time of DRSQ process is much better than the centralized approach as the number of queries increases. When the
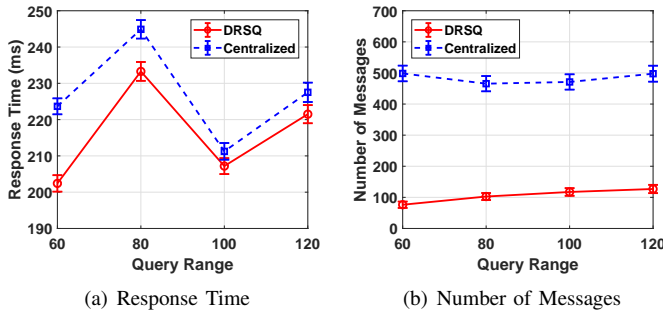
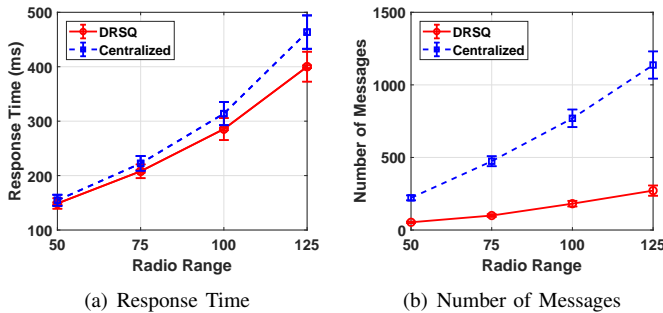Fig. 8. Impact of query range on (a) response time and (b) number of messages

Fig. 9. Impact of transmission range on (a) response time and (b) number of messages

number of queries is 5, the DRSQ process performs almost 30% faster than the centralized approach does. The DRSQ process only needs to access the sensor nodes which are around the query range. In contrast, the centralized approach floods query messages to asks all the sensor nodes in the considered environment for data collection. As the number of queries increases, a large amount of flooding messages harms the network routing performance and thus increases the response time. In addition, the query node using the centralized approach needs to process more information from data objects, so the overhead of RSQ computing on the query node becomes heavy. This can be verified by the experimental results shown in Fig. 7(b). As the result indicates, the DRSQ process costs 20% to 50% fewer number of messages than the centralized approach does since it can avoid irrelevant data objects during data collection, and thus reducing a large amount of duplicated messages for data transmissions.

3) DRSQ: Query Range: Different values on the query range also influence the performance of query processing. Fig. 8(a) shows that the DRSQ process outperforms the centralized approach by 2% to 10% in term of response time with all different range values. When the query range R is smaller than the transmission range r, the probability that the whole query range falls in the transmission range is high and it thus is easier for the query node to obtain the RSQ result by collecting sufficient information from its one-hop neighbors. When the query range R is larger than the transmission range r, the DRSQ process only needs to access the sensor nodes which are around the query range instead of accessing all the sensor nodes as the centralized approach does. So, the DRSQ process can outperform the centralized approach on

response time. Fig. 8(b) shows that DRSQ saves almost 80% transmission cost in comparison with the centralized approach. The reason is that the DRSQ process can effectively prune the irrelevant information from data objects during data (local skyline) collection.

4) DRSQ: Transmission Range: The last important impact is the transmission range r of a sensor node. As Fig. 9(a) indicates, the distributed approach outperforms the centralized approach by 5% to 10% with different transmission ranges in terms of the response time. Unlike the dramatic increasing response time of the centralized approach, the response time of DRSQ process increases more gently. The centralized approach has to do the dominance checks after it receives a large amount of information from the neighboring sensor nodes. So, it needs more computation overhead. Instead, the DRSQ process can avoid the irrelevant data objects in a distributed way during the information collection. The query node only processes the one-hop neighbors' local range-skyline sets whose sizes are much smaller than the sizes of the data sets in the centralized approach on the query node. Effectively pruning irrelevant data objects in a distributed way also reduces a lot of required messages for returning the local range-skyline sets to the query node and this trend is demonstrated in Fig. 9(b). DRSQ can save 60% to 70% transmission cost on the data collection.

B. Scenario II: Performance of DCRSQ Process

In the second scenario, we present the performance results of DCRSQ process. The duration of each query Δt is 10 seconds and the total duration of the simulation is 60 seconds. Initially, the mobile sensor nodes and query nodes are placed randomly in a $500m \times 500m$ square area. For each simulation set, we execute the simulation 200 times to get the average results and the 95% confidence intervals. The t_0 of each query's Δt is randomly generated from second 1 to 50. The other important settings are shown in Table IV.

The system will continuously return results for a CRSQ query within the time period Δt, so it is difficult to measure the response time precisely. Instead, we observe the number of accessed objects (collected data objects) on each query node. If the number of accessed objects on the query node is small, it means that the efficiency of DCRSQ process is better since a large number of irrelevant data objects are skipped during message routing. For the DCRSQ process in a mobile environment, the node speed is one of the important factors. If the node speed becomes fast, it may lead the answer changing more frequently and the overhead of processing CRSQ queries also becomes heavier. We thus discuss the impact of node speed on the performance of DCRSQ process. In addition, we use a server to check the correctness of results, generated by the DCRSQ process and the centralized approaches respectively in terms of precision and recall. Note that the server has a global knowledge of all the data objects and always generates the correct answer for a query.

1) DCRSQ: Density: Fig. 10(a) shows that DCRSQ process is better than the centralized approach in terms of number of accessed objects. When the number of mobile sensors

TABLE IV
SIMULATION PARAMETERS FOR SCENARIO II

Parameter	Default Value	Range (type)
Sensing Area ($m \times m$)	500×500	–
Simulation time (seconds)	60	–
Number of Sensor Nodes	60	30, 60, 90, 120
Number of Queries	1	1 to 10
Query Range, R (m)	100	50, 100, 150, 200
Δt of a Query (seconds)	10	–
Transmission Range, r (m)	75	50, 75, 100, 125
Maximum of Node Speed (m/s)	10	5, 10, 15, 20, 25, 30
TTL of Messages (centralized approach)	5	–
Bandwidth (Mb/s)	2	–

increases, the total number of accessed objects in the proposed method remains constant that is no more 1100 nodes. In contrast, using the centralized way, a query point needs about 2 to 5.5 times more nodes to derive the final range-skyline. This is due to no process of discarding irrelevant moving objects during data collection (local range-skyline processing) in the centralized approach. In summary, DCRSQ can save 50% to 80% computational cost in average for a query.

Similarly, Fig. 10(b) shows that DCRSQ process is better than the centralized approach in terms of number of messages. In DCRSQ process, each mobile sensor node collects the information of its neighbors and derives a local RSQ result before sending a response message to the query node. Such a process can prune a lot of moving objects which will not be the candidates and thus cost less number of messages on returning local range-skyline. On the other hand, the centralized approach just floods query messages and collects the information of all neighboring mobile sensor nodes for deriving the final range-skyline. So, the centralized approach wastes 10% to 20% more network cost on data collection.

For the accuracy, each mobile sensor node in the centralized approach does not consider the prediction location of the neighbor nodes. Each sensor node only forwards the collected information to the query point. The result of final range-skyline may be inaccurate, so we compare the results of DCRSQ process and centralized approach with the answer in a server to measure the precision and recall. Fig. 10(c) and Fig. 10(d) show that both precision and recall of the centralized approach are worse than DCRSQ process by 10% to 20%. Moreover, precision and recall of DCRSQ process are almost 100% correct when the number of sensor nodes is large.

2) DCRSQ: Number of Queries: In this simulation experiment, we are interested in evaluating the performance of different number of queries issued simultaneously. We set the number of queries from 1 to 10, which means that there are at most 10 queries within the time period Δt in the simulation. Fig. 11(a) and Fig. 11(b) show that as the number of queries increases, the number of messages and the number of accessed objects grow up respectively for both approaches. However, DCRSQ process only needs to access 50% to 70% amount of data objects in the derivation comparing to the centralized approach. DCRSQ outperforms the centralized approach and saves about 30% on network cost when the number of queries is smaller then 7. If the number of queries

exceeds 7, DCRSQ will cost more network messages. The reason is that the neighboring nodes cooperatively process the local result of CRSQ and some of them store duplicated local results (objects). That is, the query node may receive many duplicated reply messages.

Fig. 11(c) and Fig. 11(d) show that DCRSQ process is better than the centralized approach in terms of precision and recall. As the number of queries increases, DCRSQ process still can achieve almost 90% correctness and outperforms the centralized approach by 12% to 25% in terms of precision and recall, respectively. The reason is that DCRSQ process does not compute the irrelevant data objects anymore since they have already been filtered by the neighboring mobile nodes. Thus, a query point only checks the dominance objects that can guarantee to be in the final range-skyline. The other reason is that each mobile sensor node in the centralized way just gathers the information of its neighbors and sends the information back to the query node for deriving final range-skyline. Therefore, there is a possibility to compute a large number of irrelevant data objects for the query node. Thus, it makes the precision and recall of the centralized approach worse than the DCRSQ process. Although DCRSQ still has many redundant transmissions we mentioned above (in Fig. 11(b)), such duplicate local results significantly recover transmission failures and thus increase the precision and recall of the query result.

3) DCRSQ: Query Range: In this simulation set, we discuss the results of varying the query range. In Fig. 12(a), the number of accessed objects in the centralized approach remains around 3000 nodes for the final range-skyline in query point. However, the number of accessed node increases slightly when the query range becomes larger. It means that the query node in DCRSQ can save almost 30% to 80% computational cost in comparison with the centralized approach. As shown in Fig. 12(b), when the query range increases, the number of messages in the centralized approach is always about 1.7×10^5 because it always floods messages to the whole sensing area. DCRSQ process needs much less network cost than the centralized approach when the query range is smaller than 150 meters and only performs slightly worse when the query range is 200 meters. The possible reason is that the DCRSQ still costs too much network messages on exchanging information between the irrelevant nodes which are very far away from the query node.

As for the accuracy shown in Fig. 12(c) and Fig. 12(d), the trends in centralized approach, in comparison with the previous two measurements, are different. With the wider range of a query, the percentage of recall drops rapidly down about 40% in the centralized approach. Recall that all the neighboring sensor nodes have to report their information to the query node continuously for CRSQ queries in the centralized approach. As the query range becomes larger, more neighboring sensor nodes need to continuously report their information without data pruning. In such a scenario, the query node will be a bottleneck of the system and thus many messages may be dropped. Due to the above reason, the precision and recall of the centralized approach decrease significantly. On the contrary, the DCRSQ process already
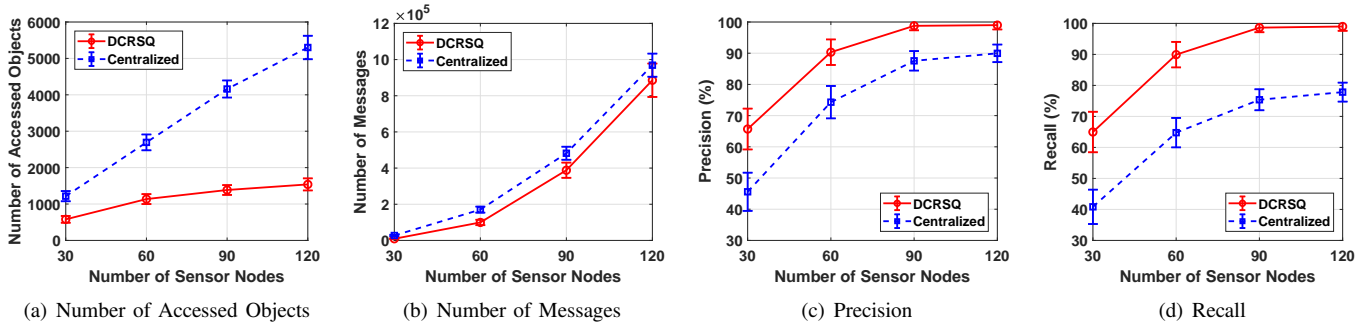
(a) Number of Accessed Objects   (b) Number of Messages   (c) Precision   (d) Recall

Fig. 10. Impact of the number of sensor nodes on (a) number of accessed objects, (b) number of messages, (c) precision, and (d) recall
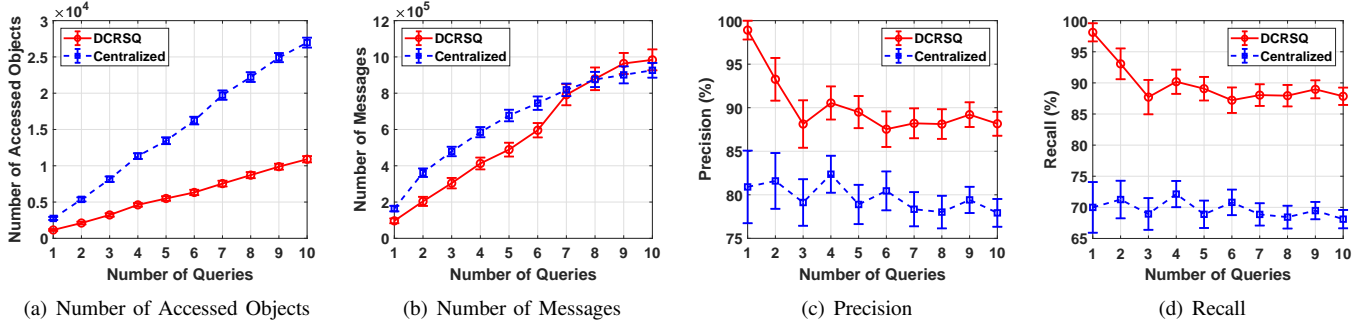

(a) Number of Accessed Objects   (b) Number of Messages   (c) Precision   (d) Recall

Fig. 11. Impact of the number of queries on (a) number of accessed objects, (b) number of messages, (c) precision, and (d) recall

computes the range-skyline locally and the local range-skyline candidate sets are continuously sent back to the query node for deriving the final range-skyline. Such a way can reduce large amount of network cost and avoid the bottleneck problem on the query node. Thus, the accuracy of our approach can achieve over 92% better.

4) DCRSQ: Transmission Range: Transmission range (or sensing range) of each object is also an important impact factor. We therefore measure the performance on different values of transmission range. As shown in Fig. 13(a), the number of accessed objects in DCRSQ process is much less than the centralized approach by up to 80% if the transmission range is 125 meters. Filtering unnecessary data objects in a distributed way can reduce a lot of required messages for returning the local range-skyline sets to the query point. Fig. 13(b) shows that DCRSQ process outperforms the centralized approach by up to 25% with different settings of transmission range in terms of the number of messages. Unlike the dramatic increasing of the number of messages in the centralized approach, the number of messages in DCRSQ process increases more slowly. The reason is that each mobile node in the centralized approach brings a large number of data objects from neighboring sensor nodes before forwarding them back to the query node.

Fig. 13(c) and Fig. 13(d) show the accuracy of both approaches. For the centralized approach, the precision and recall are under 50% when $r = 50$, and jump to more than 90% if r becomes larger than 100 meters. The reason is that each mobile sensor node may not successfully transmit information to others while the transmission range becomes too small. In contrast, DCRSQ process can achieve 98% precision and recall since DCRSQ process uses safe time to predict the locations

of its neighbors and thus provides more accurate results in the final range-skyline sets.

5) DCRSQ: Node Speed: The last simulation experiment investigates the effect of mobile sensor node's speed. We vary the maximum value of node speed from 5 to 30 m/s in the simulation. Fig. 14 indicates that all the trends are gradually decreasing and our proposed method, DCRSQ process, always has a better performance than the centralized approach. Fig. 14(a) and Fig. 14(b) show that DCRSQ outperforms the centralized approach in terms of the number of accessed objects and the number of messages. Note that each mobile sensor node in DCRSQ process knows when the neighboring nodes enter and leave the query range in advance. It thus can save almost 20% to 90% message cost on query processing and reduce 60% amount of accessed objects for deriving the final skyline result on the query node.

In addition, when each mobile node moves faster, the neighbors will change more frequently and wireless connections between mobile sensor nodes become more unstable. Hence, in high-speed scenarios, it is more difficult for the query node to collect sufficient information to derive the accurate result. Fig. 14(c) and Fig. 14(d) show that both DCRSQ and the centralized approach has the similar trends of performance on precision and recall. Comparing to the centralized approach, DCRSQ has 20% improvement in average on the precision and recall.

VII. CONCLUSION

In IoMT, very few works discuss the RSQ and CRSQ queries while simultaneously considering the following constraints: distributed computing units and databases on different mobile sensor nodes databases, moving data objects, and the
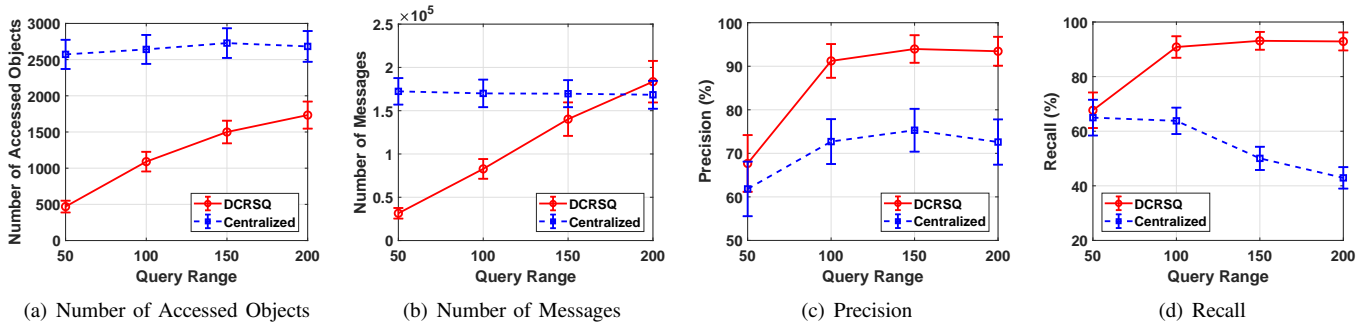
Fig. 12. Impact of query range on (a) number of accessed objects, (b) number of messages, (c) precision, and (d) recall
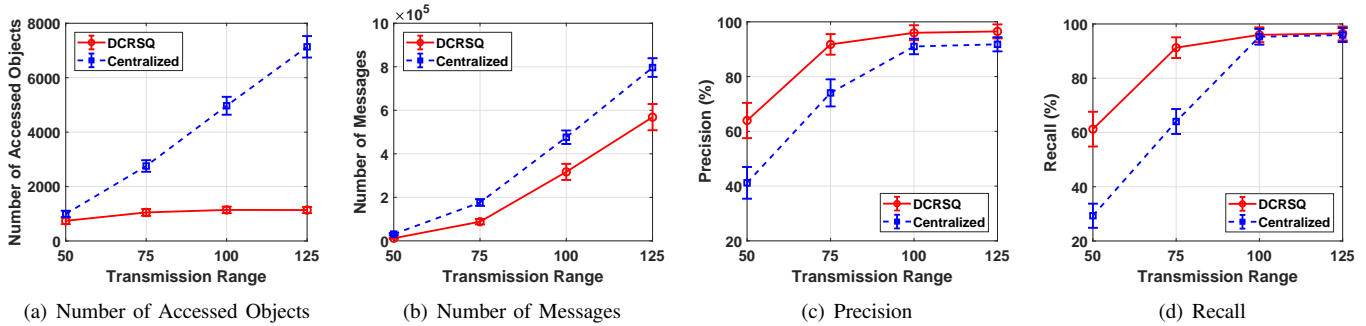

Fig. 13. Impact of transmission range on (a) number of accessed objects, (b) number of messages, (c) precision, and (d) recall
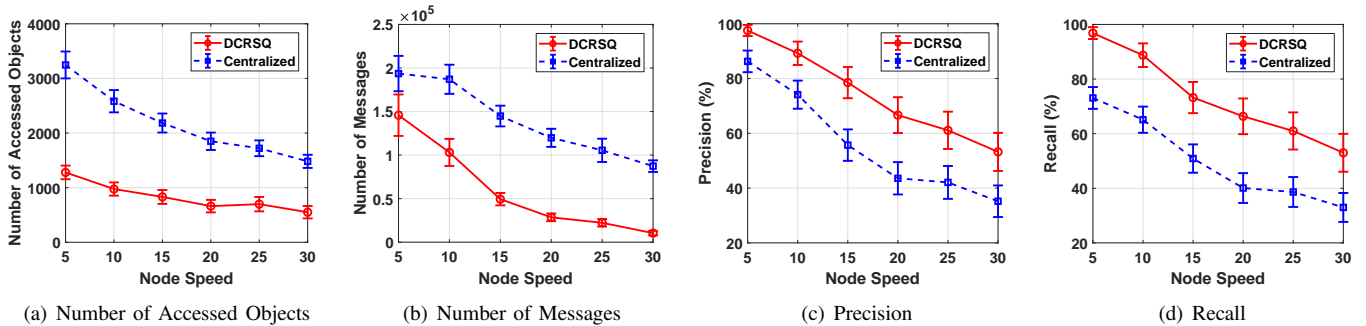

Fig. 14. Impact of node speed on (a) number of accessed objects, (b) number of messages, (c) precision, and (d) recall

mobile query. We hence propose a Distributed Continuous Range-Skyline Query process (DCRSQ process), for driving the results of RSQ and CRSQ queries. The main idea of the proposed DCRSQ process is to predict the appropriate time, safe-time, that the answer of a query changes. We apply such a prediction to each mobile sensor node and query node, so each mobile sensor node can compute the local result more precisely and then provide the local result to the query node for computing the final result. Instead of processing the information of all the neighboring mobile sensor nodes on the query node, the proposed distributed and cooperative approach with safe-time prediction can effectively reduce the computation overhead of the query node. The performance of DCRSQ process is also validated by the extensive simulation experiments. In some scenarios, the performance of DCRSQ process is almost 80% better than the performance of the centralized approach in terms of the number of accessed objects. The DCRSQ process saves more than 15% network cost in terms of the number of messages in general. In most scenarios, the DCRSQ process outperforms the centralized

approach by more than 10% to 25% accuracy (precision and recall).

In this work, we propose a prototype of distributed query process for CRSQ and simply use 2 dimensional data objects (distance and sensing value) to valid the performance in the simulation. For each sensor node, it needs to buffer the received query information and then help the data filtering and local computation until the query time is expired. Hence, there is one possible future research direction to find the relation between the minimum requirement (CPU and memory/storage) of each sensor and a parameterized (report/sense rate, number of sensor nodes and number of queries, etc.) IoT environment. Another possible future research work is to implement the distributed multi-criteria decision services on different modern open source IoT constrained platform [27] or simulators [28]. Such a way can help the research and open source communities for evaluating. In the future, we are going to develop distributed approaches for monitoring different spatial queries in practical drone-assisted IoMT applications [29].

References

[1] S. Im, M. Song, S.-W. K. Jongwan Kim, C.-S. Hwang, and S. Lee, "Cell-based distributed index for range query processing in wireless data broadcast systems," in *Knowledge-Based Intelligent Information and Engineering Systems Lecture Notes in Computer Science*, vol. 4251, pp. 1139–1146, Springer Berlin Heidelberg, 2006.

[2] J. Zhang and L. Gruenwald, "Optimizing data placement over wireless broadcast channel for multi-dimensional range query processing," in *Mobile Data Management*, pp. 256–265, IEEE, 2004.

[3] C. Li, Y. Gu, J. Qi, R. Zhang, and G. Yu, "A safe region based approach to moving *k*nn queries in obstructed space," *Knowledge and Information Systems*, vol. 45, no. 2, pp. 417–451, 2015.

[4] C.-M. Liu and S.-Y. Fu, "Effective protocols for knn search on broadcast multi-dimensional index trees," *Information Systems*, vol. 33, pp. 18–35, 2008.

[5] X. Lin, J. Xu, and H. Hu, "Range-based skyline queries in mobile environments," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, pp. 835–849, April 2013.

[6] S. Rahul and R. Janardan, "Algorithms for range-skyline queries," in *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '12, (New York, NY, USA), pp. 526–529, ACM, 2012.

[7] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," in *ACM Transactions on Database Systems*, TODS, (New York, NY, USA), pp. 41–82, ACM, 2005.

[8] L. Tian, L. Wang, P. Zou, Y. Jia, and A. Li, "Continuous monitoring of skyline query over highly dynamic moving objects," in *Proceedings of the 6th ACM International Workshop on Data Engineering for Wireless and Mobile Access*, MobiDE '07, (New York, NY, USA), pp. 59–66, ACM, 2007.

[9] L. Ang, K. P. Seng, A. M. Zungeru, and G. K. Ijemaru, "Big sensor data systems for smart cities," *IEEE Internet of Things Journal*, vol. 4, pp. 1259–1271, Oct 2017.

[10] X. Cheng, L. Fang, L. Yang, and S. Cui, "Mobile big data: The fuel for data-driven wireless," *IEEE Internet of Things Journal*, vol. 4, pp. 1489–1516, Oct 2017.

[11] F. Calabrese, M. Colonna, P. Lovisolo, D. Parata, and C. Ratti, "Real-time urban monitoring using cell phones: A case study in rome," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, pp. 141–151, March 2011.

[12] F. E. Horita, J. ao Porto de Albuquerque, L. C. Degrossi, E. M. Mendiondo, and J. Ueyama, "Development of a spatial decision support system for flood risk management in brazil that combines volunteered geographic information with wireless sensor networks," *Computers & Geosciences*, vol. 80, pp. 84–94, 2015.

[13] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Proceedings of 17th International Conference on Data Engineering*, pp. 421–430, 2001.

[14] G. R. Hjaltason and H. Samet, "Distance browsing in spatial databases," *ACM Trans. Database Syst.*, vol. 24, pp. 265–318, June 1999.

[15] M. A. Cheema, X. Lin, W. Zhang, and Y. Zhang, "A safe zone based approach for monitoring moving skyline queries," in *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, (New York, NY, USA), pp. 275–286, ACM, 2013.

[16] K. Hose and A. Vlachou, "A survey of skyline processing in highly distributed environments," *The VLDB Journal*, vol. 21, no. 3, pp. 359–384, 2012.

[17] B. Zheng, K. C. K. Lee, and W. C. Lee, "Location-dependent skyline query," in *The Ninth International Conference on Mobile Data Management (mdm 2008)*, pp. 148–155, April 2008.

[18] L. Chen, B. Cui, and H. Lu, "Constrained skyline query processing against distributed data sites," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 2, pp. 204–217, 2011.

[19] K. Ahmed, N. S. Nafi, and M. A. Gregory, "Enhanced distributed dynamic skyline query for wireless sensor networks," *Journal of Sensor and Actuator Networks*, vol. 5, no. 1, 2016.

[20] Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi, "Skyline queries against mobile lightweight devices in manets," in *22nd International Conference on Data Engineering (ICDE'06)*, pp. 66–66, April 2006.

[21] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, and A. El Abbadi, "Parallelizing skyline queries for scalable distribution," in *Proceedings of the 10th International Conference on Advances in Database Technology*, EDBT'06, (Berlin, Heidelberg), pp. 112–130, Springer-Verlag, 2006.

[22] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, (New York, NY, USA), pp. 161–172, ACM, 2001.

[23] K. Kim, Y. Cai, and W. Tavanapong, "Safe-time: Distributed real-time monitoring of cknn in mobile peer-to-peer networks.," in *Proceedings of the 9th IEEE International Conference on Mobile Data Management (MDM)*, pp. 124–131, 2008.

[24] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson, "On the average number of maxima in a set of vectors and applications," *J. ACM*, vol. 25, pp. 536–543, Oct. 1978.

[25] J. C. Kuo and W. Liao, "Hop count distribution of multihop paths in wireless networks with arbitrary node density: Modeling and its applications," *IEEE Transactions on Vehicular Technology*, vol. 56, pp. 2321–2331, July 2007.

[26] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*, pp. 90–100, Feb 1999.

[27] R. A. Light, "Mosquitto: server and client implementation of the mqtt protocol," *Journal of Open Source Software*, vol. 2, May 2017.

[28] I. Minakov, R. Passerone, A. Rizzardi, and S. Sicari, "Routing behavior across wsn simulators: The aodv case study," in *2016 IEEE World Conference on Factory Communication Systems (WFCS)*, pp. 1–8, May 2016.

[29] C. Lai, C. Chen, and L. Wang, "On-demand density-aware uav base station 3d placement for arbitrarily distributed users with guaranteed data rates," *IEEE Wireless Communications Letters*, pp. 1–1, 2019.

Chuan-Chi Lai is currently holding a post-doctoral position in the Department of Electrical and Computer Engineering at National Chiao Tung University, Taiwan, R.O.C. He received his Ph. D. in Computer Science and Information Engineering from National Taipei University of Technology (Taipei Tech), Taiwan in 2017. He won Excellent Paper Award and Best Paper Award in ICUFN 2015 and WOCC 2018 conferences, respectively. His current research interests are in the areas of data management and dissemination techniques in mobile wireless environments, mobile ad-hoc and sensor networks, distributed query processing over moving objects, and analysis and design of distributed algorithms.

Zulhaydar Fairozal Akbar received the MSc. degree in Electrical Engineering and Computer Science from National Taipei University of Technology (NTUT), in 2016. Now, he is a Junior Lecturer in Informatics Engineering Department, Electronic Engineering Polytechnic Institute of Surabaya (PENS). His research interests include mobile computing, data mining and machine learning.

Chuan-Ming Liu is an associate professor in the Department of Computer Science and Information Engineering, National Taipei University of Technology (NTUT), TAIWAN. He received his Ph. D. in Computer Sciences from Purdue University in 2002 and B.S. and M.S. degrees both in Applied Mathematics from National Chung-Hsing University, Taiwan, in 1992 and 1994, respectively. In the summer of 2010 and 2011, he has held visiting appointments at Auburn University and Beijing Institute of Technology, respectively. Dr. Liu's research interests include data management and data dissemination in various emerging computing environments, query processing in moving objects, location-based services, ad-hoc and sensor networks, parallel and distributed computation, and analysis and design of algorithms.

Van-Dai Ta received the MSc. degree in computer science from National Formosa University, Taiwan, in 2015. Now, he is currently a PhD student of National Taipei University of Technology, Taiwan. His current research interests are computer networks, wireless sensor networks, Data Mining.

Li-Chun Wang (M'96 – SM'06 – F'11) received Ph. D. degree from the Georgia Institute of Technology, Atlanta, in 1996. From 1996 to 2000, he was with AT&T Laboratories, where he was a Senior Technical Staff Member in the Wireless Communications Research Department. Since August 2000, he has joined the Department of Electrical and Computer Engineering of National Chiao Tung University in Taiwan and is jointly appointed by Department of Computer Science and Information Engineering of the same university. Dr. Wang was elected to the IEEE Fellow in 2011 for his contributions to cellular architectures and radio resource management in wireless networks. He won two Distinguished Research Awards of National Science Council, Taiwan in 2012 and 2017, respectively. He was the co-recipients of IEEE Communications Society Asia-Pacific Board Best Award (2015), Y. Z. Hsu Scientific Paper Award (2013), and IEEE Jack Neubauer Best Paper Award (1997). His current research interests are in the areas of software-defined mobile networks, heterogeneous networks, and data-driven intelligent wireless communications. He holds 19 US patents, and have published over 200 journal and conference papers, and co-edited a book, "Key Technologies for 5G Wireless Systems," (Cambridge University Press 2017).