Architectural Considerations for Distributed RFID Tracking and Monitoring

Zhao Cao
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003, USA
caozhao@cs.umass.edu

Yanlei Diao
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003, USA
yanlei@cs.umass.edu

Prashant Shenoy
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003, USA
shenoy@cs.umass.edu

ABSTRACT

In this paper we discuss architectural challenges in designing a distributed, scalable system for RFID tracking and monitoring. We argue for the need to combine inference and query processing techniques into a single system and consider several architectural choices for building such a system. Key research challenges in designing our system include: (i) the design of inference techniques that span multiple sites, (ii) distributed maintenance of inference and query state, (iii) sharing of inference and query state for scalability, and (iv) the use of writeable RFID tags to transfer state information as objects move through the supply chain. We also present the status of our ongoing research and preliminary results from an early implementation.

1. INTRODUCTION

RFID is a promising electronic identification technology that enables a real-time information infrastructure to provide timely, high-value content to monitoring and tracking applications. An RFID-enabled information infrastructure is likely to revolutionize areas such as supply chain management, health-care and pharmaceuticals. Consider, for example, a distributed supply chain environment with multiple warehouses and millions of tagged objects that move through this supply chain. Each warehouse is equipped with RFID readers that scan objects and their associated cases and pallets upon arrival and departure and while they are processed in the warehouse. In order to track objects and monitor the supply chain for anomalies, several types of queries may be posed on the RFID streams generated at the warehouses.

- *Tracking queries:* Report any pallet that has deviated from its intended path. List the path taken by an item through the supply chain.
- *Containment queries:* Raise an alert if a flammable item is not packed in a fireproof case. Verify that food containing peanuts is never exposed to other food cases for more than an hour.
- *Hybrid queries:* Report if a drug has been exposed to a temperature of more than 80 degrees for 12 hours.

The first class of queries are location queries that require object locations or location history. The second class involves containment, i.e., relationships between objects, cases and pallets. The third kind involves processing of sensor streams (e.g., temperature readings) in conjunction with RFID streams to detect certain conditions. Typically raw RFID streams contain noisy data that lacks any location or containment information. Hence, such continuous queries require derivation of location and containment information from raw RFID data as well as processing of heterogeneous sensor streams along with RFID data streams.

In this paper, we discuss the architectural challenges in designing a scalable, distributed stream processing system for RFID tracking and monitoring. We propose to combine location and containment inference with scalable query processing into a single architecture, in contrast to prior approaches that dealt with these two problems separately. We present three architectural choices in instantiating such a system over large supply chains and present an analysis of their communication overheads. By doing so, we show that the choice between centralized and distributed approaches mainly depends on the read frequency of RFID readers and the number of active queries in the system. Furthermore, utilizing local storage of writeable RFID tags for inference and query processing makes the distributed approach a better solution with significantly reduced communication cost.

In this paper, we also describe key technical challenges in designing a distributed architecture, which include (i) the design of novel inference techniques that span multiple warehouses of the supply chain, (ii) distributed, consistent maintenance of inference and query state as objects move through the supply chain, and (iii) sharing of inference and query state for scalability. A novel aspect of our system is its ability to exploit writeable RFID tags, when available, and use the onboard tag storage to transfer query and inference state as the object moves from one location to another. We finally present the status of our ongoing research and preliminary results from an early implementation.

2. RELATED WORK

RFID stream processing. The HiFi system [9, 12] offers a declarative framework for RFID data cleaning and processing. It focuses on per-tag smoothing and multi-tag aggregation, but does not capture relationships between objects such as containment or estimate object locations via containment. Our system can produce a rich event stream with object location and containment information. It further offers distributed inference and event processing methods.

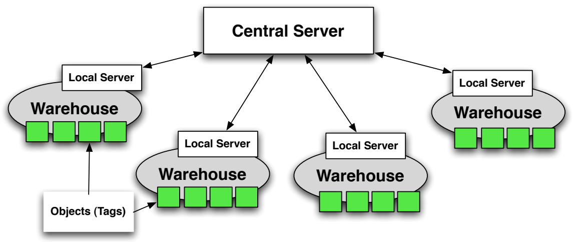

Figure 1: A distributed RFID data management system.

RFID databases. Siemens RFID middleware[20] uses application rules to archive RFID data streams into databases. Cascadia [21] supports RFID-based pervasive computing with event specification, extraction and archival. Techniques are also available to integrate data cleansing with query processing [16], encode flow information [13] and recover high-level information from incomplete, noisy data by exploiting known constraints [23]. These techniques, however, are not designed for fast RFID stream processing and only for centralized processing.

Event query processing. Most event processing methods [1, 14, 22] use centralized processing which requires all RFID data to be transferred to a single site, incurring high communication cost. The system in [2] uses multi-step event acquisition and processing to minimize event transmission cost. In contrast, our system performs both inference and query processing, does so at a local site whenever possible, and then transmits computation state across sites for distributed inference and event pattern detection.

Probabilistic inference in sensor networks. There have been several recent techniques [15, 5, 18, 11] for inferring the true value of a phenomenon, such as temperature, light, or an object's position, that a sensor network is deployed to measure. Our inference problem differs because we aim to infer inter-object relationships, such as containment, that sensors cannot directly measure, hence the different design of our graph model for inference. Further, our system combines inference with query processing into a single architecture and explores advanced techniques to reduce the combined cost of inference/query state migration in a distributed system.

3. OVERVIEW OF THE SPIRE SYSTEM

In this section, we provide an architectural overview of a distributed RFID data management system, which we call SPIRE. This system is designed to support RFID-based tracking and monitoring in large-scale supply chains with multiple warehouses and millions of objects.

In a typical environment as depicted in Figure 1, each object is affixed with a tag that has a unique identity under the EPC standard [7]. Most tags are passive tags that have no individual power systems but small amounts of memory, e.g., 1-4 KB in the current generation of EPC tags [17] and up to 64KB [10] in the next generation. Such tag memory can be used to store object information and facilitate query processing as we describe in the next section. An RFID reader periodically sends a radio signal to the tags in its read range; the tags use the radio energy to send back their tag ids. The reader immediately returns the sensed data in the form of (`tag_id`, `reader_id`, `time`). The local servers of a warehouse collect raw RFID data streams from all of the readers, and filter, aggregate, and process these streams. The data streams from different warehouses are further aggregated to support global tracking and monitoring.

We next illustrate two tracking and monitoring queries using an extension of the Continuous Query Language [3] with additional constructs for event pattern matching [1, 22]. These queries assume that events in the input stream contain attributes (`tag_id`, `time`, `location`, `container`) and optional attributes describing object properties, such as the type of food and expiration date, which can be obtained from the manufacturer's database[1]. It is worth noting the difference in schema between raw RFID readings and events required for query processing. Such differences motivate our work on inference, which is discussed shortly.

Query 1 below is an example of containment queries. It sends an alert when peanut-free food has been contained in the same case as food containing peanuts for more than 1 hour. The inner (nested) query block performs a self-join over the input stream, where one join input retains only the events for peanut-free food, the other input retains only those for food containing peanuts, and the join is based on equality on container. Each join result represents an event that a container contains both food with peanuts and foot without peanuts. This event is published immediately into a new stream. The outer query block detects a sequence pattern over the new stream: each match of the pattern contains a sequence of events that refer to the same tag id of the peanut free-food and span a time period of more than 1 hour. For each pattern match, the query returns the tag id of the peanut free food and the length of the period—such information can assist a retail store in deciding whether to dispose of the food.

```
Query 1:
Select tag_id, A[A.len].time - A[1].time
From ( Select Rstream(R1.tag_id, R1.loc)
       From   Food [Range 3 minutes] As R1,
              Food [Range 3 minutes] As R2
       Where  R1.type = ‘peanut free’ and
              R2.type = ‘peanut’ and
              R1.container = R2.container
     ) As S
     [ Pattern SEQ(A+)
       Where  A[i].tag_id = A[1].tag_id and
              A[A.len].time > A[1].time + 1 hr]
```

Query 2 combines RFID readings with temperature sensor readings and alerts if an object has been exposed to a temperature of more than 80℃ for 12 hours. It has a similar structure as Query 1, but returns all the sensor readings in the period when the temperature regulation was violated.

```
Query 2:
Select tag_id, A[].temp
From ( Select Rstream(R.tag_id, R.loc, T.temp)
       From   Object [Now] as R,
              Temperature [Rows 1 minute] as T
       Where  R.type = ‘drug’ and
              T.temp > 80 ℃ and
              R.loc ≃ T.loc
     ) As S
     [ Pattern SEQ(A+)
       Where  A[i].tag_id = A[1].tag_id and
              A[A.len].time > A[1].time + 12 hrs]
```

The SPIRE system we design has two main functionalities: inference and query processing. Both of them can be

[1]How to obtain the object properties to the site of query processing is an architectural issue, which we discuss in the next section.

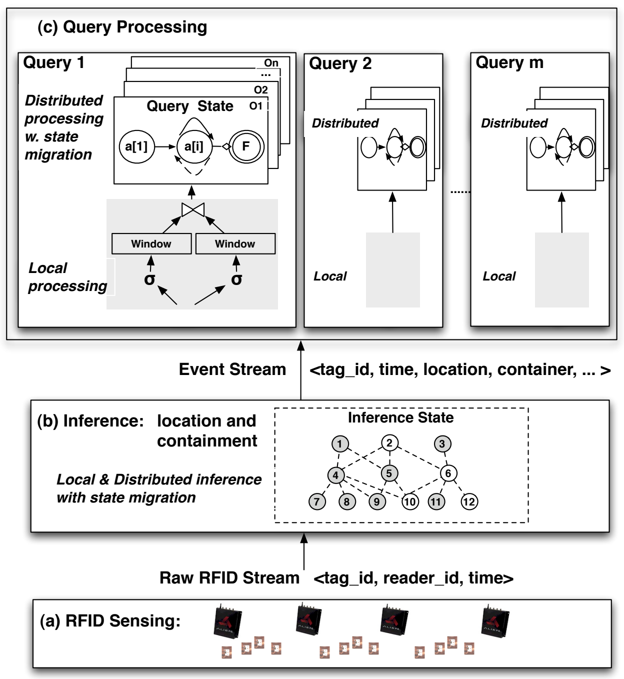
(c) Query Processing
Query 1
Distributed processing w. state migration
Query State
a[1]
a[i]
F
O1
O2
On
Window
Window
Local processing
σ
σ
Query 2
Distributed
Local
Query m
Distributed
Local
Event Stream
<tag_id, time, location, container, ... >
(b) Inference: location and containment
Local & Distributed inference with state migration
Inference State
Raw RFID Stream
<tag_id, reader_id, time>
(a) RFID Sensing:

Figure 2: Architectural overview of the SPIRE system.

implemented in a centralized or distributed fashion. The tradeoffs between these implementation choices are the focus of the next section.

Inference. While data stream processing has been a topic of intensive recent research, RFID data stream processing presents several new challenges:

- *Insufficient information.* As the above examples show, query processing often requires information about object locations and inter-object relationships such as containment. However, raw RFID data contains only the observed tag id and its reader id due to the limitations of being an identification technology.
- *Incomplete, noisy data.* The problem of deriving location and containment information from raw data is compounded by the fact that RFID readings are inherently noisy, with read rates in actual deployments often in the 60%-70% range [12]. This is largely due to the sensitivity of radio frequencies to environmental factors such as occluding metal objects and interference [8]. Mobile RFID readers may read objects from arbitrary angles and distances, hence more susceptible to variable read rates.

In SPIRE, we design an inference module that derives object locations and containment relationships despite missed readings. This module resides between the RFID sensing module and the query processing module, as shown in Figure 2. We focus on containment in the following discussion (inference for object locations alone is detailed in our recent publication [19]). First, an RFID reader can read several containers and all of their contained items simultaneously, which makes it difficult to infer the exact container of each item. In SPIRE, we explore the correlations of observations obtained at different locations at different times to infer containment. For instance, while the containment information at the loading dock of a warehouse is ambiguous due to the readings of two containers simultaneously, true containment may be revealed at the receiving belt where containers are read one at a time. Such containment remains unchanged as containers are placed on shelves but may change later in the repackaging area. The inference algorithm needs to adapt to such changes in a timely fashion.

However, missed readings significantly complicate the inference problem. Consider a scenario that an item was last seen in location A with its container. Now its container is observed in B but the item is not observed in any location. There are a few possible locations for the item: it was left behind in location A but the reading in A was missed; it moved to location B with its container and its reading in B was missed; it disappeared unexpectedly (e.g., stolen). The inference algorithm needs to account for all these possibilities when deriving containment and location information.

In SPIRE, we employ a time-varying graph model to infer object location and containment information, as depicted in Figure 3 (the example is taken from our ICDE poster paper [6]). Our graph model $G = (V, E)$ encodes the current view of the objects in the physical world, including their reported locations and (unreported) possible containment relationships. In addition, the model incorporates statistical history about co-occurrences of objects.

The node set V of the graph denotes all RFID-tagged objects in the physical world. These nodes are arranged into layers, with one layer for each packaging level, e.g., an *item*, *case* or a *pallet*, which is encoded in each tag id. Nodes are assigned colors to denote their locations. The node colors are updated from the RFID readings in each epoch using the color of the location where each tag is observed. If an object is not read in an epoch, its node becomes uncolored but retains memory of the most recent observation.

The directed edge set E encodes possible containment relationships between objects. We allow multiple outgoing and incoming edges to and from each node, indicating an object such as a case may contain multiple items, and conversely, an item may have multiple potential cases (our probabilistic inference will subsequently chose only one of these possibilities). To enable inference, the graph also encodes additional statistics. Each edge maintains a bit vector to record recent positive and negative evidence for the co-location of the two objects. For example, the recent co-location bit vector for nodes 3 and 7 in Figure 3 is 01 at time t=2 and and 011 at t=3. Further, each node remembers the last confirmed containment by a special reader such as a belt reader that scans cases one at a time. For example, at time t=2 in the figure, the belt reader confirms that the edge from node 3 to node 7 represents the true containment. This information stays valid for a while but then becomes obsolete when containment changes.

In summary, the graph model encodes the following information about each object for inference, which we call the **inference state**: (i) the most recent observation of the object, (ii) all of its possible containers, (iii) its recent co-location history with each of the containers, and (iv) its confirmed container in the past by a special reader.

After the graph is updated from the RFID readings in each epoch, an inference algorithm runs on the graph to estimate the most likely container and location of each object. Our algorithm combines node inference, which derives the most likely location of an object, with edge inference, which derives the most likely container of an object, in an iterative fashion through the graph. In particular, the algorithm

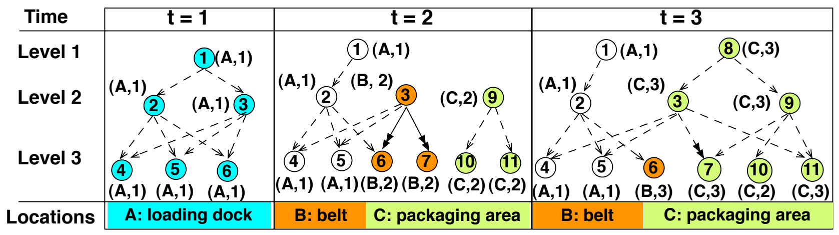

Figure 3: Examples of the time-varying colored graph model for containment and location inference.

runs (1) from the colored nodes (with known locations), (2) through the edges linked to the colored nodes, where edge inference determines the container of a color node, (3) to the uncolored nodes incident to these edges, where node inference determines the location of an uncolored node given its recent color and the colors of the processed neighboring nodes, (4) to the edges linked to these nodes, and so on. As such, inference sweeps through the graph in increasing distance from the colored nodes.

The above description assumes that all the data is available at a central server for inference. If inference is to be made in a distributed fashion, as objects move from site to site we need to transfer inference state with the objects so that information is available for subsequent inference—this process is called **inference state migration**. Revisit the example in Figure 3. Suppose that after time t=3, cases 2, 3 and items 4, 5, 7 move to a new warehouse and are observed at the loading dock of the new warehouse. Now we want to infer the containers of items 4, 5, 7. The information in the previous inference state, such as case 3 being the confirmed container of item 7 and the co-location history of items 4, 5 with case 2, 3, will be very useful to the inference in the new location. This indicates that inference state needs to be maintained across sites on a global scale. We detail several architectural choices for doing so in the next section.

Query Processing. As the inference module streams out events with inferred location and containment information, the query processor, as shown in Figure 2(c), processes these events to answer continuous monitoring queries like the two examples above. As the figure shows, part of query processing can be performed at each warehouse, such as filtering of events based on object properties (Queries 1 and 2), a self-join over the input stream (Query 1), and a join between an object stream and a sensor stream (Query 2).

A more challenging issue is with the part of query processing that spans sites, such as the detection of a complex pattern over a large period of time (see the `pattern` clause in Queries 1 and 2). Our discussion first assumes that queries run at a central server with all the information needed for query processing transfered to the server. Our query processing module employs a new type of automaton to govern the pattern matching process. Each query automaton comprises a nondeterministic finite automaton and a match buffer that stores each match of the pattern. The automaton for Query 1 is depicted in Figure 2(c). The start state, $a[1]$, is where the Kleene plus operator starts to select the first relevant event into the match buffer. At the next state $a[i]$, it attempts to select zero, one, or more events into the buffer. The final state, F, represents the completion of the matching process. Each state is associated with a number of edges, representing the possible actions. Each edge has a formula expressing the condition on taking the edge. Edge formulas are evaluated using the values from the current event as well as the previous events. (Details of this query automaton model are reported in our recent publication [1].)

In summary, the following information, called the **query state**, is maintained to evaluate a pattern query using our automaton model: (*i*) the current automaton state, (*ii*) the minimum set of values extracted from the input events that future automaton evaluation requires, e.g., $A[1]$.tag_id and $A[1]$.time for Query 1 (details on extracting these values are available in [1]), and (*iii*) the set of values that the query intends to return, which can be simple values as in Query 1, or a long sequence of readings as in Query 2. If the pattern query is defined on a per-object basis, as in both of our example queries, the system needs to maintain a copy of query state for each object, as depicted by the copies labeled $O_1, \cdots, O_n$ in Figure 2(c). Finally, if a monitoring system supports multiple queries, the size of query state is further multiplied by the number of concurrent queries.

Similar to inference, if we evaluate pattern queries in a distributed fashion, we need to perform **query state migration** across sites so that we can resume the automaton execution in a new location and continue to expand the set of values that the query intends to return.

4. ARCHITECTURAL CHOICES: BENEFITS AND DRAWBACKS

There are three possible choices for instantiating the system architecture presented in the previous section.

Centralized warehouse. This simplest approach is to employ a centralized architecture—similar to a centralized warehouse—where all RFID data is sent to a central location for stream processing, and possibly archival. The advantage of such a centralized approach is that the system has a global view of the entire supply chain, which simplifies stream processing. Inference is also simpler since all of the object state is maintained at a single location. In this case, the local servers depicted in Figure 1 only need to perform simple processing tasks such as cleaning and/or compression. The primary disadvantage of the approach is the high communication cost of transmitting RFID streams to the central location; since RFID data can be voluminous, the network bandwidth costs can be substantial.

Analysis: Consider a supply chain with 1000 warehouses, where each warehouse stores 10,000 cases, and each case contains 10 items. Both the number of readers in a warehouse and the read frequency of each reader will vary in different supply chains. More readers and a higher read frequency yield greater monitoring accuracy, but can also lead to higher deployment and data processing costs. The choice of these parameters depends on the system budget and the monitoring requirement. Assume that, on average, there are between 500 to 5000 RFID readings per object every day depending on the actual deployment. Further, assume that the temperature sensors report temperature readings every 5 minutes and that there are 100 temperature sensors in each warehouse. Let each RFID reading tuple be 20 bytes, and temperature reading tuple be 9 bytes. A simple calculation shows that, in this scenario, approximately 1.1 to 11 TB of data will be sent to the central location every day. Even if a compression scheme offers a factor of 20 reduction in data volume, a figure that we have observed in our recent research, this will still yield between 55 GB to 550 GB of data each day. However, such compression requires that location inference be done locally before transferring location tuples to the central server. (Details of the above analysis and other analysis in the rest of the section are available in our technical report [4].)

Distributed processing with state migration. An alternative to the centralized approach is to employ distributed stream processing. In this approach, each warehouse employs local stream processing—continuous queries on objects that reside in the warehouse are processed locally. As objects move through the supply chain from one warehouse to another, queries on those objects also "move" from one warehouse site to another in order to ensure local processing. The advantage of such an approach is that communication costs are significantly lowered since local processing implies that RFID streams are processed on-site.

However, the approach is not without drawbacks. First, the approach requires state migration to transfer the inference state and query state associated with an object whenever it moves from one warehouse to another. Such state overhead is high whenever historical information is involved (for either inference or query processing). Second, inference techniques become more complex. The inference state must be distributed across multiple sites while presenting the same logically unified view of the global state of the system as the centralized approach. The design of such distributed inference techniques is still an open research question and a focus of our work. Finally, since the distributed approach has no global view of the supply chain, queries that involve objects residing in different warehouse locations are more complex. For instance, queries might be tracking different parts required to assemble a product to ensure that they arrive at a factory within a short time of one another; since each part can take a different path through the supply chain, local query processing is either infeasible or may require substantial state exchange between different warehouse sites and the central location.

Analysis: Assuming the same scenario as before. Based on the time-varying graph model in Section 3, if 10 cases are read by the exit reader of the warehouse each time, the number of possible containers for each item is approximated by 10 in this analysis (it varies in practice with the co-location history and actual read rates of readers). Then the inference state of each object can be (roughly) estimated to be 184 bytes. Hence, the size of the inference state migrated every day across all warehouses is round 18.4 GB. As for query processing, object properties such as type of food are required for processing and need to be fetched from the manufacturer's database to each warehouse. Assuming 100 bytes for each object, the total cost of fetching object properties is 11 GB every day. We estimate the size of query state using techniques in [1], e.g., 17 and 37 bytes (including a 12 byte tag id) per object for Query 1 and Query 2, respectively. If there are 10 queries of type Query 1 and 10 queries of type Query 2, the query state migrated every day is 54GB. Then the total communication cost is 83.4 GB every day, and the ratio of communication cost between the centralized and distributed approach varies from 0.65 to 6.7, depending on the RFID read frequency. Thus, there is a cross-over point between these two approaches—if an object is read more than 750 times a day, the distributed approach has better performance; otherwise, the centralized approach is better.

Another important observation is that the ratio of communication cost between centralized and distributed approaches is independent of the size of the supply chain (the number of warehouses and number of objects in each warehouse). It only depends on (1) the RFID read frequency for each object, as discussed above, and (2) the number of queries in the system. To understand the effect of the latter, consider a setting where there there are 100 queries of type Query 1 and 100 queries of type Query 2. In this setting, the query state migrated every day is 540 GB. Then the total communication cost of the distributed approach is 569.4 GB every day, even larger than the centralized approach. On the other hand, if the system has fewer queries and higher read frequency, the distributed approach is a better choice.

Distributed processing with local tag storage. Our final architecture is an optimization of the distributed architecture where writeable RFID tags with onboard memory are exploited for further reducing the communication overheads. In this case, the state of all queries associated with an object as well as inference state is written onto the tag memory prior to departure from a warehouse. Upon arrival at a new warehouse, the tag memory is read and is used to "seed" the queries and the inference graph at the new location. It is important to note that the tag memory is used as a cache and that query/inference state continues to be stored at the prior location as before — in the event the tag memory can not be read for any reason, the approach reverts to state migration, where this state is fetched from the previous warehouse as before. Thus, writeable tags can be exploited to optimize overheads and the system correctness is not impacted even when tag memory is not available (or becomes corrupt).

Analysis: Since each tag already carries its id, the 12-byte tag id can be saved from any of the state regarding the object when stored in the tag's memory. For each object, the size of object properties then becomes 88 bytes, the inference state becomes 172 bytes, and with the same number of queries, the size of query state becomes 300 bytes. All the object properties and inference/query state that need to be stored on each tag total 560 bytes.

In this approach, the amount of data that needs to be stored is independent of the size of supply chain and the reader settings. The only factor that impacts the storage size is the number of queries. To illustrate, if there are 100 queries of each type, the size of query state becomes 3000 bytes. Then the object properties and inference/query state that needs to be stored on each tag totols 3.2 KB.

The above analysis shows that query state dominates the storage cost. Larger numbers of queries may challenge the scalability of this approach. Advanced techniques that share query state to save storage are outlined in the next section.

5. STATUS AND ONGOING WORK

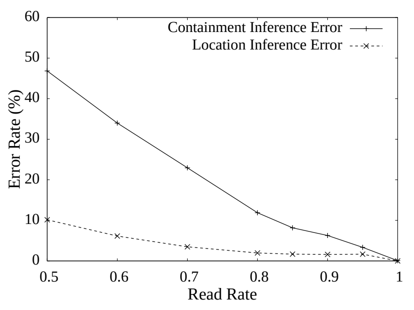

Figure 4: Results of location and containment inference.

As of August 2009, we have implemented the inference module and the query processor in a centralized fashion as well as a simulator for enterprise supply chains.

Our inference module includes a time-varying graph model that captures recent observations of objects and possible object containment relationships, and an online probabilistic algorithm that estimates the most-likely container and location of each object as new data arrives. Using RFID streams emulating a large warehouse, our results as shown in Figure 4 indicate that location inference can achieve 90% accuracy even for low read rates such as 50%. Our containment inference can achieve 90% accuracy when the read rates reach 85%. It has reduced accuracy for lower read rates due to both the loss of containment confirmation from special readers like belt readers and lack of consistent observations in the recent history. We are currently investigating advanced machine learning techniques to improve inference accuracy for the range of low read rates. Our initial results also show the inference techniques to be efficient: they can scale to 100,000 objects while running at stream speed.

Our ongoing work further explores a host of research issues for distributed inference and event query processing.

Migrating inference state. When a set of of objects leave a warehouse, we need to split the graph constructed for inference, and transfer a subgraph relevant to these objects to the next warehouse (state migration) or store a subgraph relevant to each object in its tag memory (using local storage). These objects may be connected to other objects in the graph due to ambiguous containment, and only some of them may be sensed by the exit reader. Hence, it is a nontrivial issue to identify a subgraph that is large enough for accurate inference later but does not include unnecessary data. To further save bandwidth or storage, we also consider methods to truncate the historical information included in the graph without affecting accuracy.

Sharing query and inference state. To support multiple queries over millions of objects, we need to maintain a copy of query state per query per object. Consider the approach that stores query state in local tag memory. The tag memory then needs to hold the state of all the queries relevant to this object. This requires intelligent packing schemes given limited tag memory. One of our schemes exploits multi-query optimization. For all queries relevant to an object, we construct a shared query plan (e.g., a combined automaton) and further merge their query states once they become equivalent. As such, we can store merged query state using less tag memory. Another scheme exploits stable containment: all the objects that have been contained in the same container are likely to have the same state for each active query as well as the same inference state. Therefore, all the objects can share a single copy of the state in their aggregate memory, hence reducing the amortized memory usage in each tag. These schemes can also be used to reduce communication cost during state migration.

Acknowledgements: This work has been supported in part by the National Science Foundation under the grants IIS-0746939, IIS-0812347, and CNS-0626873.

6. REFERENCES

[1] J. Agrawal, Y. Diao, D. Gyllstrom, et al. Efficient pattern matching over event streams. In *SIGMOD*, 147–160, 2008.

[2] M. Akdere, U. Çetintemel, and N. Tatbul. Plan-based complex event detection across distributed sources. *PVLDB*, 1(1):66–77, 2008.

[3] A. Arasu, S. Babu, and J. Widom. CQL: A language for continuous queries over streams and relations. In *DBPL*, 1–19, 2003.

[4] Z. Cao, Y. Diao, P. Shenoy. Architectural considerations for distributed RFID tracking and monitoring. `http://www.cs.umass.edu/~yanlei/spire.pdf`, UMass Tech. Report, 2009.

[5] M. Cetin, L. Chen, et al. Distributed fusion in sensor networks. In *IEEE Signal Processing Mag*, 42–55,2006.

[6] R. Cocci, T. Tran, Y. Diao, and P. Shenoy. Efficient Data Interpretation and Compression over RFID Streams. In *ICDE*, 2008. Poster.

[7] EPCglobal Inc. EPCglobal tag data standards version 1.3. `http://www.epcglobalinc.org/`, Mar 2006.

[8] K. Finkenzeller. *RFID handbook: radio frequency identification fundamentals and applications*. John Wiley and Sons, 1999.

[9] M. J. Franklin, S. R. Jeffery, et al. Design considerations for high fan-in systems: The HiFi approach. In *CIDR*, 290–304, 2005.

[10] Fujitsu. `http://www.fujitsu.com/global/news/pr/archives/month/2008/20080109-01.html`.

[11] A. Ihler, J. Fisher and et.al. Nonparametric Belief Propagation for Self-Calibration in Sensor Networks. In *IPSN*, 225–233,2004.

[12] S. R. Jeffery, M. J. Franklin, et al. An adaptive RFID middleware for supporting metaphysical data independence. *VLDB Journal*, 17(2):265 – 289, 2007.

[13] C.-H. Lee and C.-W. Chung. Efficient storage scheme and query processing for supply chain management using RFID. In *SIGMOD*, 291–302, 2008.

[14] Y. Mei and S. Madden. Zstream: A cost-based query processor for adaptively detecting composite events. In *SIGMOD*, 2009.

[15] M. Paskin, C. Guestrin and J. Mcfadden. A robust architecture for distributed inference in sensor networks. In *IPSN*, 55–62,2005.

[16] J. Rao, S. Doraiswamy, et al. A deferred cleansing method for RFID data analytics. In *VLDB*, 175–186, 2006.

[17] Smartcard focus. `http://www.smartcardfocus.com/shop/ilp/se~72/p/index.shtml`.

[18] J. Schiff, D. Antonelli, et al. Robust message-passing for statistical inference in sensor networks. In *IPSN*, 109–118,2007.

[19] T. Tran, C. Sutton, R. Cocci, et al. Probabilistic inference over rfid streams in mobile environments. In *ICDE*, 2009.

[20] F. Wang and P. Liu. Temporal management of RFID data. In *VLDB*, 1128–1139, 2005.

[21] E. Welbourne, N. Khoussainova, et al. Cascadia: a system for specifying, detecting, and managing rfid events. In *MobiSys*, 2008.

[22] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *SIGMOD*, 407–418, 2006.

[23] J. Xie, J. Yang, Y. Chen, et al. A sampling-based approach to information recovery. In *ICDE*, 476–485, 2008.