Distributed Inference and Query Processing for RFID Tracking and Monitoring

Zhao Cao, Charles Sutton†, Yanlei Diao, Prashant Shenoy
Department of Computer Science †School of Informatics
University of Massachusetts, Amherst University of Edinburgh

ABSTRACT

In this paper, we present the design of a scalable, distributed stream processing system for RFID tracking and monitoring. Since RFID data lacks containment and location information, which is key to query processing, we propose to combine location and containment inference with stream query processing in a single architecture, with inference as an enabling mechanism for high-level query processing. We further consider challenges in instantiating such a system over a large supply chain and design techniques for distributed inference and query processing. Our technical contributions include (i) novel inference techniques that provide accurate estimates of object locations and inter-object relationships such as containment in noisy, dynamic environments, and (ii) distributed inference and query processing techniques that minimize the computation state transferred across warehouses while approximating the accuracy of centralized processing. Our experimental results, using both real-world data and large synthetic traces, demonstrate the accuracy, efficiency, and scalability of our proposed techniques.

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 [8]. 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 as well as while they are processed in the warehouse. Such an RFID-based infrastructure offers an organization with unprecedented visibility into its distributed supply chain, with the near-real-time ability to track and monitor objects and detect anomalies as they occur. To illustrate, consider the following types of continuous queries that may be posed on the RFID streams generated at the warehouses.

- *Tracking queries*, which include queries such as "report any pallet that has deviated from its intended path," or "list the path taken by an item through the supply chain." Tracking queries are essentially *location* queries that require object locations or location histories.
- *Containment queries,* which include queries such as "raise an alert if a flammable item is not packed in a fireproof case," or "verify that food containing peanuts is never exposed to other food cases for more than an hour." This class of queries involve inter-object relationships, e.g., containment between objects, cases, and pallets, and are useful for enforcing packaging and shipping regulations.
- *Hybrid queries,* which include "for any frozen food placed outside a freezer, raise an alert if it has been exposed to room temperature for 6 hours." This class of queries combine sensors streams (e.g., temperature) and RFID streams (e.g., object location and containment) to detect various conditions.

Unfortunately, the nature of RFID data makes these queries difficult to answer. The key challenge is that although queries typically involve object locations and inter-object relationships such as containment, the RFID data does not directly contain this information. Rather, the data contains only the observed tag id and the reader id; this is a fundamental limitation of RFID technology. To enable queries on the data that is not actually available, the key is to exploit statistical regularities in the tag id and reader information so that one can *estimate* object locations and object relationships. The estimation problem is complex, however, because RFID readings are inherently noisy. The read rates in actual deployments can be as low as 60%-70% [10] due to the sensitivity of radio frequency to environmental factors such as occluding metal objects and interference [6]. Objects can also be read by multiple readers in nearby locations due to the overlap of their read ranges.

A second key challenge is that large supply chains are distributed or even global in scope, for which a centralized approach may be limiting. In a centralized approach, RFID streams from various warehouses are sent to a central location for query processing. This approach can fail to scale because of the bandwidth overheads incurred due to high data volume and can also potentially increase the latency of detecting anomalous events, especially in geographically large supply chains. In contrast, a distributed approach processes data streams as they are generated, thereby reducing the delay of answering queries. However, as objects move from one warehouse to another, tracking and monitoring queries must also "move" with these objects. To do so, both the state of objects and the state of mornitoring queries relevant to these objects must be transferred to the new location to seed the computation there.

Research contributions: In this paper, we present the design of a scalable, distributed stream processing system for RFID tracking and monitoring. Our system combines location and containment inference with stream query processing into a single architecture, with inference as an enabling mechanism for high-level query processing for tracking and monitoring. Regarding inference, the key novelty in our work is to introduce the notion of *smoothing over object relations*, whereas all existing work on RFID data cleaning [7, 10] and location inference [13, 15] is limited to the traditional approach of smoothing over time. Our new notion of smoothing plays a pivotal role in designing a simple, effective statistical system that addresses the uncertainties about both individual objects (e.g., object locations) and inter-object relationships (e.g., containment) in

VLDB '10, September 13-17, 2010, Singapore

a dynamic world with changes over time. We further present techniques to scale such inference and thus enabled query processing to large supply chains that span multiple warehouses and numerous objects. More specifically, our contributions include:

Novel statistical framework (§3). Working within a principled statistical machine learning framework, we present a novel graphical model whose structure implicitly smooths over object relations. In contrast to prior work that uses temporal smoothing, in our work smoothing over containment leads to a much simpler graphical model, thereby allowing much more efficient inference techniques. At the same time, our model and inference techniques can still accurately estimate location and containment information, so that high-level query processing can return high-quality answers.

Our general approach is as follows: (i) Our probabilistic model describes a physical world comprising object locations, containment relationships, and noisy RFID readings. (ii) We devise an inference algorithm, called RFINFER, for our model, working within an expectation maximization (EM) framework. The design of our model allows us to derive a simple customized M-step, which is essential for working at scale but still offers provable optimality in terms of maximizing the likelihood of the data. Furthermore, our algorithm is developed in an *unsupervised learning* framework; that is, it does not use machine learning techniques that require access to any specially-generated training data. (iii) We finally extend our algorithm to also detect changes of containment using a statistical method called *change point detection*.

Distributed inference and query processing (§4). To suit the increasing scale of RFID tracking and monitoring, we develop a distributed approach that performs inference and query processing locally at each warehouse, but transfers the state of inference and state of query processing as objects move across warehouses. A naive inference algorithm would incur high transfer overhead by requiring the entire history of observations collected from multiple warehouses over a long period of time. Instead, we propose to truncate history by sifting out the observations most informative about the true containment, and further distill such useful history into a few numbers for each object to minimize the inference state transferred. In distributed processing of tracking and monitoring queries, the main issue is that we need to transfer one copy of query state for each object. Our work exploits the inference results, in particular, stable containment to share query state among objects.

Performance evaluation (§5). Our evaluation, using both real-world data and large synthetic traces, yields the following results: (i) Our inference algorithm is highly accurate, with less than 7% error on containment and 0.5% error on location, for noisy traces with stable containment. (ii) With containment changes, our algorithm can achieve 85% accuracy when read rates reach 0.7 while keeping up with stream speed, as demonstrated using both real lab traces with various noise factors and simulations. (iii) Our distributed inference method provides 3 orders of magnitude reduction in communication cost over a centralized approach without compromising accuracy. (iv) Our highly accurate inference results allow a query processor to produce high-quality answers and further allow effective sharing of query state across objects for state migration.

We finally note that on a more general level, the idea behind smoothing over object relationships—to exploit relationships among entities in data cleaning—has potential application in a wide range of problems, which we will explore in future work.

2. BACKGROUND

In this section, we provide background on RFID technology and RFID tracking and monitoring applications. Our system targets a distributed supply chain with multiple warehouses. Each item in the supply chain is assumed to be packed into a case, and multiple cases packed onto a pallet, which yields a containment relationship between items, cases and pallets. Items, cases, and pallets are assumed to be tagged; each tag has a unique identity. We assume the tags are passive RFID tags, which are battery-less and have a small amount of on-board memory, e.g., 1-4 KB in the current generation of tags and up to 64KB in the next generation. This memory is writable and can be exploited to store supply-chain-specific object state and enable "*querying anytime anywhere*".[1] We assume that each warehouse employs multiple RFID readers: at the entry and exit points as well as at the belt and shelves to scan resident objects. Each such reader periodically sends a radio signal to the tags in its read range; the tags, powered by the radio energy, send back their tag ids. The reader immediately returns the sensed data in the form of ($time$, tag_id, $reader\_id$). The local servers of a warehouse collect raw RFID data streams from all of the readers and process these streams. The data streams from different warehouses are further aggregated to support global tracking and monitoring.

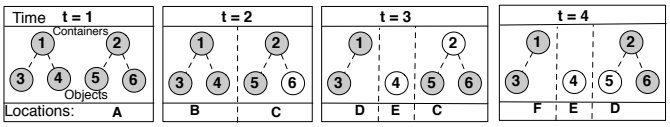


Figure 1: Example of noisy RFID readings and containment changes

We next illustrate the tracking and monitoring queries that our work aims to support. Such queries assume that events in the input stream contain attributes ($time$, tag_id, $location$, $container$) and optional attributes describing object properties, such as the type of food or type of container, which can be obtained from the manufacturer's database. Note the different schemas for raw RFID readings and events required for query processing—events in the latter schema are produced by an inference module as we discuss shortly.

Query 1 below is an example of a hybrid query that combines object locations, containment relationships, and other sensor readings (e.g., temperature). This query raises an alert for any frozen food that is outside a freezer and has been exposed to room temperature for 6 hours. The query is written using the Continuous Query Language [2] with an extension for event pattern matching [1]. The inner (nested) query checks for each frozen food if its container is not a freezer or does not exist, and if so retrieves the temperature based on the object's location. The outer query aggregates the retrieved temperatures for the object and checks if it has been exposed to room temperature for 6 hours. The query finally returns all the temperature readings in the 6 hour period and the tag id of the object—such information can assist a retail store in deciding whether to dispose of the food.

```
Query 1:
Select tag_id, A[].temp
From ( Select Rstream(R.tag_id, R.loc, T.temp)
       From   FrozenFood [Now] as R, Temperature
              [Partition By Sensor_id Rows 1] as T
       Where  (!(R.container Is_Type_Of 'freezer') or
                 R.container = NULL) and
              R.loc = T.loc and T.temp > 0 °C
     ) As S
     [ Pattern SEQ(A+)
       Where  A[i].tag_id = A[1].tag_id and
              A[A.len].time > A[1].time + 6 hrs]
```

3. INFERENCE ALGORITHM

In this section, we present our inference module that translates the low-level, noisy, and incomplete data from RFID readers into a

[1]This technology trend motivated us to minimize the computation state associated with a tag, as discussed in §4, so it can be held in a tag's local memory to enable *querying anytime anywhere* in the future.

high-level event stream with object location and containment information. More specifically, it translates raw readings, ($time$, tag_id, $reader\_id$), into events ($time$, tag_id, $location$, $container$) optionally with other attributes for object properties from the manufacturer. Our solution to this problem makes use of techniques from probabilistic reasoning, statistics, and machine learning.

Intuitively, the idea is that whenever an object is read, its container is likely to be read as well. Over time, we can use the *co-location history* of containers and objects to derive the containment relationships. To develop this intuition into a robust system, however, several design considerations must be addressed to effectively handle the noisy and incomplete input. To explain these considerations, we use the example in Figure 1: Each node represents a tag, and each edge a containment relation. The shaded nodes represent tags that were read (by the reader specified in the bottom row), and the unshaded nodes are tags that were missed by all readers.

A main design consideration is how to handle low read rates. If the read rate is in the 60%-70% range, it is difficult to accurately determine object locations, which makes it also difficult to tell when objects are co-located. If the containment relations were known for certain, then a powerful way to determine object locations would be to *smooth over containment relations*, meaning that whenever we read one object in a container, we know that all of the other objects must be in the same place. For example, in Figure 1, at time $t = 3$, we miss reading container 2, but we do read object 5. If we knew that container 2 contained object 5, then we could correctly infer that container 2 is also present at location C.

Unfortunately, the containment relationships are not known in advance, so instead we use an *iterative approach*. First, we start with the best available information about object locations and have a guess about containment relationships based on co-location. Then we can improve our understanding of object locations via smoothing over containment relationships. For example, in Figure 1, container 2 and object 5 are repeatedly co-located in the raw readings, so we can infer a containment relationship right away. Given the containment relationship, we can infer the location of container 2 at $t = 3$. The resulting better understanding of locations allows us to further improve our understanding of containment relationships. Revisit Figure 1. We did not have strong evidence about the container for object 6, but with the new location information about container 2, we see that it is consistently co-located with object 6.

A second main design consideration is how to detect changes in containment relationships. Consider an object and a container that have been consistently co-located, such as container 1 and object 4 in the first two time steps of Figure 1. If later on ($t = 3$ in the example), we fail to read the object, then following the idea of smoothing over containment, it is reasonable to infer than object 4 is still co-located with container 1. But at some point, if we repeatedly fail to read the object (as at $t = 4$), we may suspect that the object has actually been moved. To distinguish between these two competing explanations—either the object has been removed from the container, or it has not moved but its tag has been missed—we need a way to decide when there is enough recent evidence to conclude that the containment relationship has actually changed. How much evidence is enough should depend on the read rate: if the readers are less accurate, then we ought to demand more evidence.

To resolve these difficulties in a principled way, we propose a new algorithm RFINFER for inferring containment and location from RFID readings. To do so, we first construct a probabilistic model of the observed data that depends on the object locations and containment relationships (§ 3.1). We then infer the containment relationships using RFINFER (§ 3.2), which essentially computes the set of containment relationships that best explains the data and estimates object locations as a by-product. In statistics, this is called a *maximum likelihood* approach. Furthermore, containment changes can be detected by measuring how closely two different models fit the data: one that includes such changes, and one that does not (§ 3.3). This is a type of *change point detection* problem.

Finally, we require that inference keep up with stream speed so that, for example, processing five minutes of data takes no more than five minutes of computation time. This performance requirement poses a challenge for state-of-the-art machine learning techniques. The techniques we employ include optimizations (described in the appendix §9.1) and history truncation (detailed in § 4).

3.1 Graphical Model

In this section we describe a probabilistic model of container locations, object locations, and RFID readings. The model is a probability distribution over random variables that represent both the true state of the world, which we do not observe, and the RFID readings, which we do. For the purposes of describing the model, we assume that we know the containment relationships exactly; in fact, we infer them from RFID data, as explained in § 3.2.

We discretize both time and space: We divide time into a set of discrete *epochs* of, for example, one second in duration. All RFID readings that occur in the same epoch are treated as simultaneous. As for locations, given the set of tracking and monitoring queries we aim to support, such as those in the previous section, it suffices to localize objects to the nearest reader. Therefore, we model locations as a discrete set $\mathcal{R}$, which is the set of locations of all of the static readers. Finally, we assume that there are N containers, which we denote by integers $c \in [1, N]$, and that there are O objects, which we denote by integers $o \in [1, O]$.

The random variables in the model are as follows. For each epoch t, and each container c, let ℓ_{tc} be the true location of the container. This is a random variable which takes values from the set of locations $\mathcal{R}$. Similarly, let ℓ_{to} be the true location of each object o. As for the readings, let x_{trc} be a binary random variable that indicates whether the reader at location $r \in \mathcal{R}$ received a reading of the container c. Define y_{tro} similarly for each object o. To make the notation more compact, let $\ell = \{\ell_{tc}|\forall t, c\} \cup \{\ell_{to}|\forall t, o\}$ be the vector of all the true object and container locations over all time, and similarly define $\mathbf{x} = \{x_{trc}|\forall t, r, c\}$ for the container readings and $\mathbf{y} = \{y_{tro}|\forall t, r, o\}$ for the object readings. The model is a joint distribution $p(\ell, \mathbf{x}, \mathbf{y})$ over all of these random variables.

Our model is depicted graphically in Figure 4. The figure shows the model for a single epoch.

To describe the model, we explain how to sample from the probability distribution that describes the world, assuming that the world behaves exactly according to our model. At every epoch t, first the true location ℓ_{tc} is sampled for each container c. Because we do not assume any prior knowledge about the layout of the factory, we model this distribution as uniform over the set of all possible locations $\mathcal{R}$. Now there is no need to sample object locations, because each object must be in the same place as its container, so if object o is contained within container c, then $\ell_{to} = \ell_{tc}$.

Now we can generate the RFID readings. Each reader has a *read rate*, which we denote $\pi(r, \bar{r})$, which is the chance of the reader at location r reading an object which is actually at location $\bar{r}$. Typically, we expect a reader to detect an object only if both are at the same location. However, there is a small chance that a reader can detect an object which is actually closer to a nearby reader. In an actual deployment, it is possible to measure the read rates periodically by using reference tags fixed to known locations and listening for these tags' responses to a given number of interrogations [10, 15]. To sample the readings, each reader independently attempts to read the tag on every container and the tag on every object. Formally, each binary observation variable x_{trc} is sampled indepen-

dently with probability according to the read rate; that is, x_{trc} is true with probability $\pi(r, \ell_{tc})$. We write this probability as

$$p(x_{trc}|\ell_{tc}) = \begin{cases} \pi(r, \ell_{tc}) & \text{if } x_{trc} = 1 \text{ (tag read)} \\ 1 - \pi(r, \ell_{tc}) & \text{if } x_{trc} = 0 \text{ (otherwise),} \end{cases} \quad (1)$$

and similarly for y_{tro}.

Putting it together, this defines a *joint probability distribution* as

$$p(\ell, \mathbf{x}, \mathbf{y}) = \prod_{t=1}^{T} \prod_{c=1}^{C} p(\ell_{tc}) \prod_{r \in \mathcal{R}} p(x_{trc}|\ell_{tc}) \prod_{o|(o,c) \in \mathcal{C}} p(y_{tro}|\ell_{to}) \quad (2)$$

It can be seen that this model treats all time steps as independent and all containers as independent. For each epoch and container, it iterates over all readers and considers the probabilities of each reader observing the container as well as its contained objects. Because the model treats all epochs as independent, it does not perform any temporal smoothing over readings; however, it compensates for this by smoothing over containment relations instead. To smooth the readings over time as well would add significant complexity to the model, and significant computational cost to the inference procedure. In § 5, we verify experimentally that smoothing over containment relations is effective at inferring object locations.

An important quantity is the probability that the model assigns to the observed data, that is, $p(\mathbf{x}, \mathbf{y}) = \sum_{\ell} p(\ell, \mathbf{x}, \mathbf{y})$. This quantity is called the *likelihood* of the data. Note that the likelihood is a function of the containment relationships $\mathcal{C}$. To emphasize this, we define $L(\mathcal{C}) = \log p(\mathbf{x}, \mathbf{y})$. According to our model, this is

$$L(\mathcal{C}) = \sum_{t=1}^{T} \sum_{c=1}^{C} \log \sum_{a \in \mathcal{R}} p(\ell_{tc}) \prod_{r \in \mathcal{R}} p(x_{trc}|\ell_{tc}) \prod_{o|(o,c) \in \mathcal{C}} p(y_{tro}|\ell_{to}) \quad (3)$$

The log likelihood measures how probable the observed RFID readings are under the current set of containment relationships. This will be an important quantity for inferring the containment relationships, because a set of containment relationships that have higher log likelihood provide a better explanation of the readings.

3.2 Inferring Containment Relationships

To infer containment relationships from RFID readings, we use a *maximum likelihood* framework, that is, we determine the containment relationships such that, according to the model, the observed readings are most likely. Formally, this amounts to maximizing the log likelihood $L(\mathcal{C})$ with respect to the set of containment relationships $\mathcal{C}$. In this section, we describe the algorithm that performs this maximization, which we call RFINFER.

The idea is that determining containment relationships would be simple if, besides the RFID data, we also observed the true locations of all containers. However, the true container locations are in fact unknown. To handle this, we propose an algorithm RFINFER for inferring the containment relations. RFINFER is developed in the EM framework, which offers a general approach for maximizing likelihood functions in the presence of missing data, in our case the container locations. The algorithm alternates between two steps. In the first step, the *expectation step* (or E-step), we infer a distribution over the locations of each container, given some current guess about the containment relations. In the second step, the *maximization step* (or M-step), we choose the best containment relations given our current guess of the container locations. We iterate these two steps until the containment relations do not change.

In the E-step, the distribution that we want to compute is the conditional distribution $p(\ell|\mathbf{x}, \mathbf{y})$ over the location of each container that results from the joint distribution of Eq (3)—this distribution is called the *posterior distribution* of the container location and sometimes denoted as $q_{tc}(\cdot)$ for simplicity. From the definition of conditional probability, it can be shown that

$$p(\ell_{tc}|\mathbf{x}, \mathbf{y}) \quad = \quad S \prod_{r \in \mathcal{R}} p(x_{trc}|\ell_{tc}) \prod_{o|(o,c) \in \mathcal{C}} p(y_{tro}|\ell_{to}), \quad (4)$$

where S is a constant that does not depend on ℓ_{tc}.

In the M-step, we update the current estimates of containment relationships based on the current belief about locations. We do so by defining a score w_{co} to measure the *strength of co-location* between object o and container c:

$$w_{co} = \sum_{t=1}^{T} \sum_{a \in \mathcal{R}} p(\ell_{tc} = a|\mathbf{x}, \mathbf{y}) \sum_{r \in \mathcal{R}} \log p(y_{tro}|\ell_{to} = a). \quad (5)$$

This score measures how likely are the readings of object o if it were always co-located with container c. To estimate the container for o, we simply pick the best container $C(o) = \arg\max w_{co}$.

Note that RFINFER also computes location information. When the algorithm has converged, the final values of $p(\ell_{tc}|\mathbf{x}, \mathbf{y})$ are our best estimates of the location of each container at each time step.

Finally, the following theorem states that our algorithm is guaranteed to converge to an optimum of the likelihood:

Theorem 1. *The* RFINFER *algorithm converges, and the resulting values $\mathcal{C}^*$ are a local maximum of the likelihood defined in Eq* (3)*.*

The proof is given in the appendix § 9. The key step is to show that our simple, custom M-step indeed maximizes the likelihood.

3.3 Change Point Detection

In this section, we describe how we infer changes in containment relationships. This type of problem, called *change-point detection*, is the subject of a large literature in statistics (see [3] for an overview). A *change point* is a time t at which the containment relationships change, that is, some object has either changed containers or been removed altogether. Finding change points is challenging because of the noise in RFID readings. For example, in Figure 1 at $t = 4$, it may be unclear if object 4 has actually been removed from container 1, or it has simply been "unlucky" enough to be missed twice in a row. To distinguish these two possibilities, we need to way to quantify the unluckiness of a set of readings.

We propose a statistical approach based on hypothesis testing. Suppose that we have received readings from epochs $[0, T]$. Then we define a *null hypothesis*, which is that the containment relationships have not changed at all during epochs $[0, T]$. Then, if under the null hypothesis, it turns out that the observed RFID readings are highly unlikely, we reject the null hypothesis, concluding that a change point has in fact occurred. To measure whether the observed readings are unlikely, we again use the likelihood Eq (3). Consider a single object o. Let $\mathcal{C}_{0:T}$ be the maximum likelihood containment relations based on the full data, so that $L(\mathcal{C}_{0:T})$ is the best possible likelihood if there is no change point. Alternatively, suppose there is a change point at some time t'. Then let $\mathcal{C}_{0:t'}$ and $\mathcal{C}_{t':T}$ be the best containment relations that allow object o to change locations at time t'. Maximizing over possible change points, the best possible likelihood if there is any change point for o is $\max_{t'} L(\mathcal{C}_{0:t'}) + L(\mathcal{C}_{t':T})$. We perform change point detection using the difference of these two log likelihoods, that is,

$$\Delta_o(T) = L(\mathcal{C}_{0:T}) - \max_{t' \in [0,T]} [L(\mathcal{C}_{0:t'}) + L(\mathcal{C}_{t':T})] \quad (6)$$

Essentially, this measures how much better we can explain the data if we use two different sets of containment relationships instead of

one. This is a type of *generalized likelihood ratio* statistic, which is a fundamental tool in statistics. The change point detection procedure will signal that there has been a change point whenever the value of $\Delta_o(T)$ is greater than a threshold δ.

Intuitively, to choose the threshold we would like to know what values of $\Delta_o(T)$ would be typical if there were no change point. Fortunately, we can obtain as much of this data as we want, simply by sampling hypothetical observation sequences from the model, exactly as described in § 3.1. Since none of the hypothetical sequences actually contain a change point, if our procedure signals a change point on one of them, it must be a false positive. In practice, all of the hypothetical $\Delta_o(T)$ values are quite small, so we choose δ to be their maximum. Further, all of this computation can be done in advance before any RFID data is observed.

In summary, the change point procedure is: First, before any data arrives, choose the threshold δ as described previously. Then, at every epoch T, for each object o, compute $\Delta_o(T)$ from Eq (6). If $\Delta_o(T) < \delta$, then there is no change point for o. Otherwise, if $\Delta_o(T) \geq \delta$, then we flag a change point at the time t' that achieved the maximum in Eq (6). Furthermore, we disregard the data from $0 \ldots t'$ in all subsequent calls to the change point algorithm, so that we do not flag the same change point more than once.

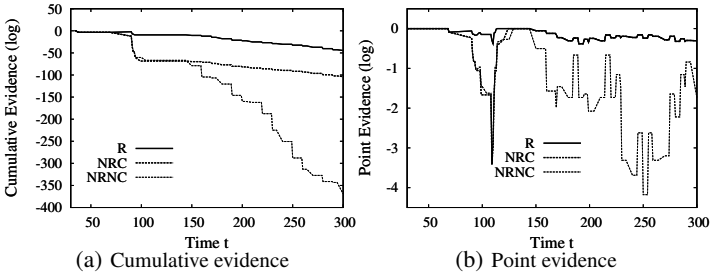
Analysis, Optimizations, and Extensions. We refer the reader to the appendix (§9) for the complexity analysis, implementation details, optimizations, and extensions of our algorithm.

4. DISTRIBUTED PROCESSING

As the sizes of supply chains grow into dozens of warehouses and millions of objects, the sheer volume of data poses a scalability challenge to an RFID tracking and monitoring system. A centralized approach, like centralized warehousing, requires all the data to be transferred to a single location for processing, but this incurs both a delay in returning answers to queries and a high communication cost. Instead, we use a distributed approach that performs local RFID stream processing at each warehouse. However, inference and query processing often require information from multiple warehouses. To solve that problem, we perform *state migration*, which transfers the state of inference and query processing for an object when it moves across warehouses. State migration raises new issues concerning how to minimize the state that is transferred without sacrificing accuracy at answering queries, which we address in § 4.1 for inference and in § 4.2 for query processing. Reducing the amount of state also opens up the possibility of writing the state to local storage of the RFID tags (once the technology of writable tags matures for large deployments), thereby enabling "*querying anytime anywhere*" when a tag is in sight.

4.1 Distributed Inference

In our distributed approach, we run inference separately at each warehouse on the locally-generated RFID readings. As an object leaves one warehouse for another, the history associated with this object (i.e., the history table described in §9.1) needs to be transferred to the new location for use by the inference procedure there. But transferring the complete history would incur both a high communication cost across warehouses and a high processing cost at the next warehouse, because inference scales linearly with the amount of history. In this section, we describe two techniques for avoiding these problems. First, we *truncate the history* by sifting out the observations that are most informative about the true containment relationships, and retaining only those observations for future processing. Second, when transferring inference state between warehouses, we *distill the history* into one number (a co-location strength) for each container-object pair, avoiding the overhead of transferring the entirety of the readings.



(a) Cumulative evidence (b) Point evidence

Figure 2: Evidence of co-location of three candidate containers.

Truncating History. The goal of *history truncation* is to determine the time steps whose observations are most informative for determining containment. This can be accomplished by monitoring the strength of co-location computed in our containment inference algorithm RFINFER. Recall that in the M-step of Algorithm 1, the co-location strength w_{co} for each object o and container c is a sum over all time steps of a quantity that we will call the *point evidence* of co-location, denoted

$$e_{co}(t) = \sum_{a \in \mathcal{R}} q_{tc}(a) \sum_{r \in \mathcal{R}} \log p(y_{tro} | \ell_{to} = a).$$

Then the *cumulative evidence* of co-location is $E_{co}(t) = \sum_{t'=1}^{t} e_{co}(t')$. To see how these quantites are used, suppose that an object started at the entry door at time 0, was scanned on the conveyor belt around time 100, and then placed on a shelf at time 150. Consider three candidate containers that were co-located with this object at the entry door: the real container (denoted by R) that always travels with the object, a second container (NRC) that was co-located at the door and the shelf but not the belt, and a third container ($NRNC$) that was never co-located after the door. Fig. 2(a) shows the cumulative evidence of co-location of three candidate containers with an object. Around time 100, the belt reader scanned the real container alone with the object, causing the cumulative evidence of the other two containers to dropped fast. This is exactly the kind of informative region that we want to find during history truncation. The information afterwards is less useful, beacuse the false container NRC becomes co-located the object again on the shelf, while the container $NRNC$ was already eliminated from contention by the belt reader. So, if we truncate the entire history associated with the object to those few readings from the belt scans, we may still have enough information to infer the true container.

Our history truncation method aims to find a subsequence of readings, called the *critical region*, that distinguish the true container from others, and discards other readings. While our intuition was explained using the cumulative evidence of co-location, our algorithm actually uses the point evidence of co-location, as shown in Fig. 2(b) (in log space). During the critical region around time 100, the real container has much higher point evidence than the two false containers; this is not true either before or after the region.

The history truncation algorithm, which runs after containment inference is complete, examines the point evidence for the candidate containers of each object. It searches through time by applying a sliding window $[t - w, t]$. For each window, it computes the sum of point evidence $\sum_{t'=t-w}^{t} e_{co}(t')$ for each container, and computes the difference in evidence between the best container and second best. If this difference is above a threshold, it includes all the readings in the current window into the critical region CR, overwriting the past set of readings in CR if existent. When the search reaches the end, all readings not in CR are simply discarded.

After running this algorithm, we have compressed the entire history from [0, T] to a small sequence CR. When the new readings

arrive in the time period $[T, T']$, rather than running inference over the entire period $[0, T']$, we run inference only over the data in the the critical region CR and in a period of recent history denoted by H. If containment is stable, it suffices to have H = [T, T'], i.e., including all new readings obtained since last inference. To support change point detection, however, we may need a somewhat larger recent history H. According to Eq (6), the change point can be any point in the entire history. In practice, it is more likely to be in the recent history since it was not detected last time. However, it may be imprudent to restrict the change point only to the most recent period $[T, T']$ because a change point before the time T might not get sufficient evidence in the previous change point detection. Our experimental results in §5.1 show that the sufficient size of H is within a factor of 2 of T'-T. As time elapses, the recent history H moves forwards and we can compression all readings falling behind H by applying the critical region algorithm.

Distilling History. When an object leaves a warehouse for the next, the history about the object that is transferred is called the *inference state* for the object. One option is to transfer the set of readings of the object and its candidate containers in the critical region CR and recent history H to the next warehouse. As such, we can seed the inference for this object in the new location. However, the inference state for an object may still not be small since each object can have dozens of candidate containers, and each container and object can have hundreds of readings in CR and H. If each RFID reading ($time$, tag_id, $reader\_id$) takes 20 bytes, the inference state for a single object can be up to, e.g., 200,000 bytes.

An alternative is to distill the history by transferring co-location weights w_{co} instead of the readings themselves. Then the inference algorithm simply adds the old transferred weights to the new weights that are computed from readings in the new warehouse. This dramatically reduces the inference state, because we only need to store a single number w_{co} for each candidate container of an object, rather than a large number of readings. This approximation can affect accuracy: if later evidence shows that the containment inference results from the old warehouse were incorrect, we can no longer revise the old location estimates as the readings have been discarded. Even in this case, however, inference at the new warehouse still has the potentially to get the containment right, because readings obtained there can eventually overrule the old weights.

4.2 Distributed Query Processing

Given an event stream with object location and containment information, the query processor processes this stream and other sensor streams to answer monitoring queries. Below, we discuss distributed query processing by assuming a relational stream processor [1, 2] (support for more complex queries is discussed in §10).

To enable distributed query processing, as an object leaves one warehouse for another, we need to identify the state of query processing relevant to the object and transfer it to the next warehouse to continue query processing there. Consider Query 1 in §2. While part of the query can be performed locally at each warehouse, such as joining the object stream and the temperature stream, other parts of the query can span sites, such as detecting the pattern of continuous exposure to room temperature for 6 hours. Recent work handles such complex pattern queries using automata-based evaluation [1], and compactly represents the query state using (i) the current automaton state, (ii) the minimum set of values needed for future automaton evaluation, e.g., the tag id and the time of the first exposure to room temperature for Query 1, and (iii) the values that the query returns, e.g., the tag id and the sequence of temperature readings for Query 1 (details are available in [1]).

Since tracking and monitoring queries are evaluated on a per-object basis, the key issue is that the query processor maintains a copy of query state for each object. To reduce the total amount of state transferred for numerous objects, we exploit *stable containment* to share query state across objects. In particular, at the exit door of a warehouse, we consider the objects in each container relevant to a query, e.g., all of those being frozen food. These objects have the same container and location at present, but potentially different history—for most queries, these objects often have commonalities in the query state. We propose a centroid-based sharing technique. Let Q_o denote the query state for object o. We choose the most representative query state (the centroid) of all Q_o's based on a distance function that counts the number of bytes that differ in the query state of two objects. The centroid selection problem has a $O(n^2)$ complexity, but since each case contains a small number of objects, e.g., 20, this computation cost is not high. Given the choice of the centroid, we compress the query state of every other object based on its distance to the centroid.

Table 1: Parameters used for generating RFID streams.

Parameter	Value(s) used
Number of warehouses (N)	1 - 10
Frequency of pallet injection (fixed)	1 every 60 seconds
Cases per pallet (fixed)	5
Items per case (fixed)	20
Number of items in steady state (I)	up to 32,000 / warehouse
Main read rate of readers (RR)	[0.6, 1]
Overlap rate for shelf readers (OR)	[0.2, 0.8]
Non-shelf reader frequency (fixed)	1 every second
Shelf reader frequency (fixed)	1 every 10 seconds
Frequency of anomalies (FA)	1 every 10 - 120 seconds

5. PERFORMANCE EVALUATION

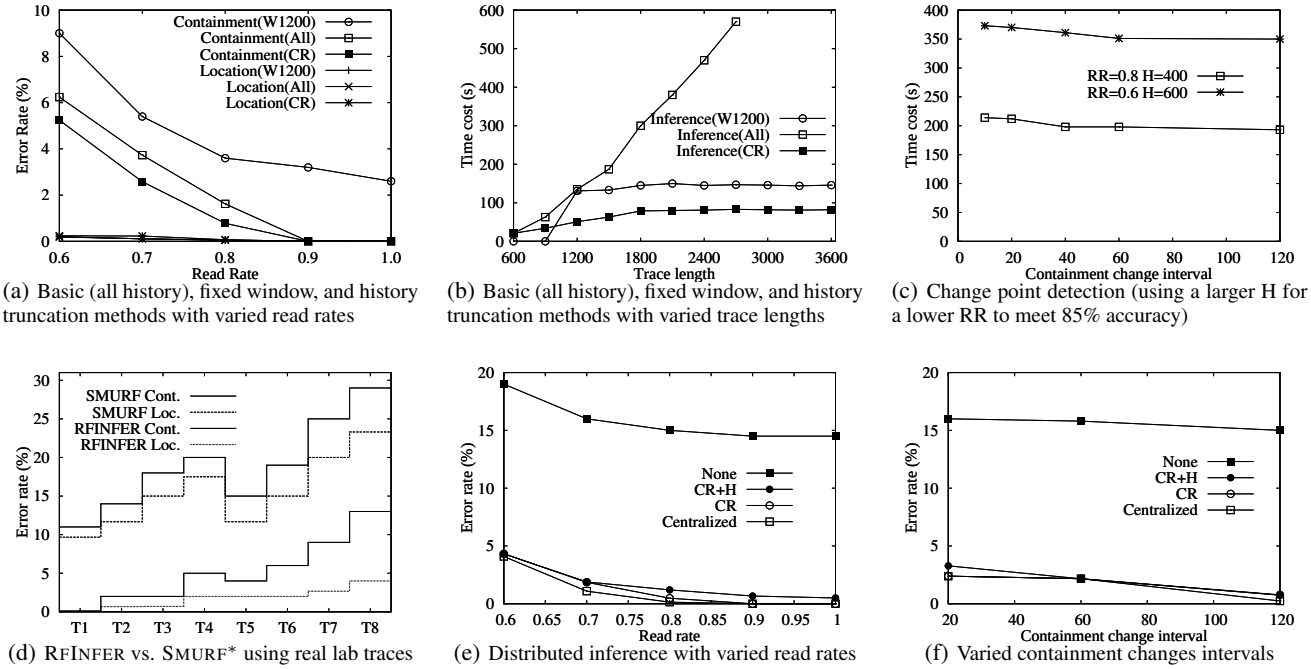
We have implemented a prototype of our inference approach, connected it to a stream query processor [1], and extended the combined system with state migration for distributed processing. We evaluate our system using both synthetic traces emulating RFID-based enterprise supply chains and real traces from a laboratory warehouse setup. The parameters for our synthetic traces are summarized in Table 1. Details of these parameters and our performance metrics are described in the appendix (§11.1).

5.1 Single-Site Inference

We first evaluate the accuracy and efficiency of our RFINFER inference algorithm on the synthetic RFID streams from a single warehouse. By default, we run inference once every 300 seconds.

Inference with stable containment. We evaluate our inference algorithm first using traces with stable containment. To deal with traces of various lengths, we consider the Critical Region (CR) method that we proposed for history truncation in distributed processing (§4.1) as an optimization also for traces produced at a single warehouse. This method results in the use of the critical region and a short recent history H (by default, the most recent 600 seconds) for inference. For comparison, we also include a simple window-based truncation method that keeps the most recent W readings for inference (W=1200 seconds here).

We first test the sensitivity of these methods to the read rate RR. As Fig. 3(a) shows, while all three methods offer high accuracy for location inference, they differ widely for containment inference: The window method has the worse accuracy because when the useful observations, such as the belt readings, fall outside the window, the inference algorithm can no longer use them to infer containment. Using the entire history or the CR method gives better accuracy as expected. Interestingly, while the CR method was initially proposed for improving performance, it also improves over the basic algorithm in accuracy due to the removal of noise from infer-



(a) Basic (all history), fixed window, and history truncation methods with varied read rates

(b) Basic (all history), fixed window, and history truncation methods with varied trace lengths

(c) Change point detection (using a larger H for a lower RR to meet 85% accuracy)

(d) RFINFER vs. SMURF* using real lab traces

(e) Distributed inference with varied read rates

(f) Varied containment changes intervals

Figure 3: Experimental results for single-site inference (a-c), our lab warehouse deployment (d), and distributed inference (e and f).

ence. Further, its sensitivity to the read rate is comparable to that using the full history, which is the best that one would expect.

We next vary the trace length from 600 to 3600 seconds and compute the total inference time when using the entire history, the window, and the CR methods in Fig. 3(b). Here we see that using the entire history severely penalizes the performance, the window based truncation stays in the middle, and the CR method performs the best with its running time insensitive to the trace length.

Containment change detection. We next employ the change point detection algorithm presented in §3.3 to detect containment changes. We use a recent history size of H (600 seconds by default) in addition to the detected critical region for inference with change point detection. To generate events of interest, we inject anomalies that randomly choose an item and move it to a different case in the warehouse. The frequency of such anomalies is every 20 seconds by default but also varied over a wide range. Each run simulates a warehouse with 32,000 items in steady state over 4 hours.

Choice of threshold. We first examine the effect of the threshold δ for change point detection by considering fixed values in a range. We also evaluate the effectiveness of our offline method that chooses δ by (1) sampling hypothetical observation sequences from the graphical model, (2) computing a distribution of the Δ statistics (Eq. 6), and (3) selecting the largest Δ sample (rounded to integer) as the threshold. We created traces with varied read rates.

	Threshold δ									
	10	20	30	40	50	60	70	80	90	100
RR=0.6	64	71	75	85	89	**89**	87	87	87	87
RR=0.7	85	88	90	92	94	96	**94**	93	91	90
RR=0.8	92	97	98	97	**97**	97	96	95	95	93
RR=0.9	83	98	**97**	97	97	97	97	96	96	95

The table above shows the F-measures for these traces as the threshold takes various fixed values. The bold numbers indicate the F-measures of the threshold chosen by our offline sampling algorithm. As can be seen, the best fixed threshold that gives the optimal F-measure varies across traces. Our chosen threshold always approximates the optimal value within 2% across all read rates.

Tradeoff between accuracy and efficiency. Our change point detection algorithm offers a tradeoff between accuracy and efficiency depending on the read rate RR and the history length H. To depict this tradeoff, we varied H from 300 to 900 and the RR from 0.6 to 0.9 and computed the F-measure and the time cost of each combination of the two. The detailed results are in Table 3 in the appendix. We found that, for a given read rate, longer histories offer better accuracies but at the expense of higher time cost; for a given history, a higher read rate offers higher accuracy. Overall, our system needs $H \leq 600$ to achieve above 85% accuracy. Fig. 3(c) shows the time cost for two particular combinations of H and RR to achieve the 85% accuracy. For the low value RR=0.6, we need H=600 to meet the accuracy requirement; for the higher RR=0.8, we only need H=400. Hence, they have different time costs, but both show limited sensitivity to the containment change interval.

5.2 Evaluation of Lab RFID Deployment

To evaluate our system in real-world settings, we developed an RFID lab with 2 ThingMagic Mercury5 readers connected to 7 circularly-polarized antennas, 20 cases containing 5 items each, and Alien squiggle tags attached to all cases and items. We used the 7 antennas to implement 1 entry door reader, 1 belt reader, 4 shelf readers, and 1 exit reader. Cases with contained items transitioned through the antennas in that order, receiving 5 interrogations from each nonshelf antenna and dozens from a shelf antenna. The shelf antennas had overlapping read ranges as they were placed close to each other. We created 8 traces with distinct characteristics, by varying the environmental noise, overlap among antennas, and tag orientations: T_1 (RR=0.85, OR=0.25), high average read rates and limited overlap rates between antennas; T_2 (RR=0.85, OR=0.5), high read rates and significant overlap rates; T_3 (RR=0.7, OR=0.25), lower read rates and limited overlap rates; T_4 (RR=0.7, OR=0.5), lower read rates and significant overlap rates; T_5 to T_8 extend T_1 to T_4, respectively, with containment changes to 35% of the cases. Further details of these traces are given in the appendix (§11.2). Each trace is 15 minutes long with 15,000 to 22,000 readings. We ran inference every 5 minutes.

For comparison, we include an alternative method, called SMURF*, that extends the state-of-the-art SMURF[10] method for RFID cleaning and location inference, with heuristics for containment infer-

ence. This method is detailed in the appendix (§11.3).

Fig. 3(d) shows the inference error rates for RFINFER and SMURF*. As can be seen, RFINFER is much more accurate than SMURF* across all traces although they both use intuitions such as smoothing and co-location. The key reason is that RFINFER uses *smoothing over containment relations* and a principled approach for the iterative feedback between location and containment estimates. This is shown to be more effective than smoothing over time for individual objects and then combining such location evidence in a heuristic way to infer containment as in SMURF*. For RFINFER, the location error rates are low across all traces. The containment error rates are within 5% in traces T_1 to T_4 despite the heterogeneous read rates, added environmental noise, and significant overlap between antennas. Containment changes cause containment error rates to rise, especially given lower read rates or higher overlap rates, but with a maximum of 13% with all the noise factors combined in T_8.

5.3 Distributed Inference

We next compare centralized and distributed approaches to inference by simulating 10 warehouses for 4 hours. Each warehouse has 32,000 items in steady state, totally 0.32 million items. Our system runs inference at stream speed for each warehouse. Fig. 3(e) shows the error rates for varied read rates. The naive no state-transfer method (labeled "None") has a high error rate, while our critical region (CR) method and its variant that transfers the weight of containers that co-located with an item in recent history ("CR+H") both perform close to the centralized method. The CR method has a slightly lower error rate than CR+H, indicating that the weights from the recent history are mostly noise and affect subsequent inference. The communication costs (shown in the appendix) show that our CR methods offer 3 orders of magnitude reduction in communication cost over a centralized approach (which approximating its accuracy). Finally, Fig. 3(f) shows a similar result regarding these methods when the containment change frequency varies.

5.4 Distributed Inference and Querying

We finally extend our distributed inference experiment with query processing. We run two queries: Q1 from §2, and Q2 that reports the frozen food that has been exposed to temperature of higher than 10 degrees for 10 hours. The table below reports the F-measures for different read rates and the total size of query state with and without sharing the state using inferred containment (§4.2). We see that the overall accuracy of query result is high (89% or higher). Also, state sharing yields up to an order of magnitude reduction in query state size. Finally, the accuracy and query state reduction ratio of Q1 are lower than those of Q2. This is because Q1 combines inferred location and containment, but Q2 only uses the inferred location which is generally more accurate than inferred containment.

		RR=0.6	**RR=0.7**	**RR=0.8**	**RR=0.9**
Q1	F-m.(%)	89.2	94	95.1	96
	State w/o share(bytes)	65,500	66,000	67037	67,000
	State w. share(bytes)	6,986	5,737	5,589	5,156
Q2	F-m.(%)	93.5	96.1	97.3	97.5
	State w/o share(bytes)	80,248	85,510	87,029	87,000
	State w. share(bytes)	7,296	6,108	5,341	5,273

6. RELATED WORK

RFID stream processing. Recent research has addressed RFID data cleaning [7] and location inference for static readers [10, 13] and mobile readers [15]. However, containment inference is more challenging since inter-object relationships cannot be directly observed. Our work is the first to employ *smoothing over object relations* in RFID inference, with demonstrated performance. Our work further supports distributed inference and querying.

RFID databases. Existing work has addressed RFID data archival [16], event specification and extraction [17], integrating data cleansing with query processing [12], and exploiting known constraints to derive high-level information [18]. Our system addresses a different problem: it processes raw data streams to infer object location and containment, thereby enabling stream query processing, and scales inference and query processing to distributed environments.

Inference in sensor networks. Various techniques [11, 5, 14, 9] have been used to infer true values of temperature, light, object positions, etc., that a sensor network is deployed to measure. Our inference problem differs because the inter-object relationships, such as containment, cannot be directly measured, hence requiring different statistical models and inference techniques. We further address distributed inference and query processing for scalability.

7. CONCLUSIONS

In this paper, we presented the design of a scalable, distributed stream processing system for RFID tracking and monitoring. Our technical contributions include (i) novel inference techniques that provide accurate estimates of object locations and containment relationships in noisy, dynamic environments, and (ii) distributed inference and query processing techniques that minimize the computation state transferred across warehouses while approximating the accuracy of centralized processing. Our experimental results demonstrated the accuracy, efficiency, and scalability of our techniques. In future work, we plan to extend our work to include probabilistic query processing, exploit on-board tag memory to hold object state and enable anytime anywhere querying, and explore smoothing over object relations for other data cleaning problems.

8. REFERENCES

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

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

[3] M. Basseville and I. V. Nikiforov. *Detection of Abrupt Changes: Theory and Application*. Prentice-Hall, 1993.

[4] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4):523–544, 2007.

[5] M. Cetin, L. Chen, et al. Distributed fusion in sensor networks. *IEEE Signal Processing Mag*, 23:42–55, 2006.

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

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

[8] S. Garfinkel and B. Rosenberg, editors. *RFID: Applications, Security, and Privacy*. Addison-Wesley, 2005.

[9] A. Ihler, J. Fisher, et al. Nonparametric belief propagation for self-calibration in sensor networks. In *In IPSN*, 225–233, 2004.

[10] S. R. Jeffery, et al. An adaptive RFID middleware for supporting metaphysical data independence. *VLDB Journal*, 17(2), 2007.

[11] M. Paskin, C. Guestrin, et al. A robust architecture for distributed inference in sensor networks. In *IPSN*, 55–62, 2005.

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

[13] C. Ré, J. Letchner, et al. Event queries on correlated probabilistic streams. In *SIGMOD*, 715–728, 2008.

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

[15] T. Tran, C. Sutton, et al. Probabilistic inference over RFID streams in mobile environments. In *ICDE*, 2009.

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

[17] E. Welbourne, et al. Cascadia: a system for specifying, detecting, and managing RFID events. In *MobiSys*, 281–294, 2008.

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

R	Number of reader locations
N	Number of containers
O	Number of objects
o	Index of a single object; $o \in [1, O]$
c	Index of a single container; $c \in [1, N]$
t	Index of time epoch (e.g., 1 second long)
$\mathcal{R}$	Set of possible reader locations
ℓ_{tc}	True location of container c at time t.
ℓ_{to}	True location of object o at time t
$\pi(r, \bar{r})$	Read rate. Probability that reader at location $r \in \mathcal{R}$ detects an object at location $\bar{r} \in \mathcal{R}$
y_{tro}	Binary variable indicating whether object o was read by reader at location r at time t
x_{trc}	Binary variable indicating whether container c was read by reader at location r at time t
$\mathbf{x}$	Binary vector of all container readings
$\mathbf{y}$	Binary vector of all object readings
w_{co}	Strength of co-location between container c and object o
$\mathcal{C}$	Containment relations; set of pairs (`object_id`, `container_id`)
$L(\mathcal{C})$	Likelihood of the observed readings, given containment relations $\mathcal{C}$
$\Delta_o(T)$	Change-point statistic for epoch T

Table 2: Notation used in this paper

Appendix

This appendix gives additional details of our algorithms and performance results. The notation used in our algorithms is summarized in Table 2.

9. ENHANCEMENTS OF RFINFER

In this section, we present additional details about our model, the RFINFER inference algorithm, implementation and optimizations, and finally two extensions of the inference algorithm.

Graphical Model. The graphical representation of our model is given in Figure 4. This figure shows the structure of the probabilistic model over a single epoch. Each node in the graph represents a random variable, and the arcs are drawn between variables that are most "closely" dependent (in a sense that is formalized in the theory of graphical models). This example includes two containers and four objects. The unshaded variables at top are the true container locations ℓ_{tc} and true object locations ℓ_{to}. The shaded variables are the RFID readings x_{trc} and y_{tro} for all readers r, containers c, and objects o. A variable x_{trc} exists whether or not container c was read; if c was not read, then $x_{trc} = 0$. The containment relations are reflected in the structure of the graph.

Algorithm and Proof. The pseudocode for RFINFER is shown in Algorithm 1. We next prove Theorem 1 about the optimality of the RFINFER algorithm.

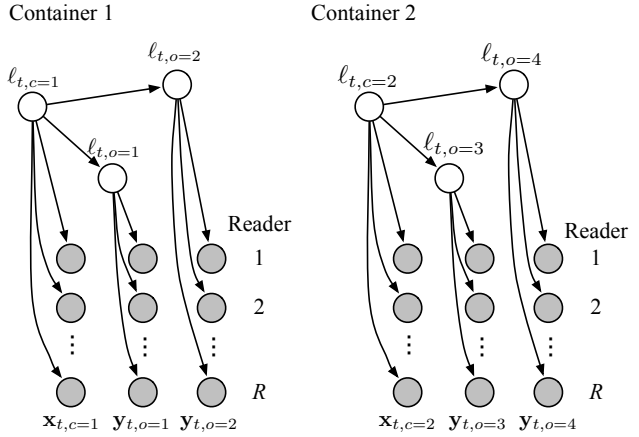


Figure 4: Graphical model of locations and RFID readings.

Algorithm 1 Pseudocode of RFINFER for inferring containment

```
   while not converged do
     // E step: compute new q
     for t = 0 to T do
       for c = 1 to N do
5        for all a ∈ R do
           q_tc(a) ← ∏_{r∈R} p(x_trc|ℓ_tc = a) ∏_{o|(o,c)∈C} p(y_tro|ℓ_to = a)
         // Now q_tc(a) = S^{-1} p(ℓ_tc = a|x, y)
         S ← Σ_{a∈R} q_tc(a)
         for all a ∈ R do
10         q_tc(a) ← q_tc(a)/S
         // Now q_tc(a) = p(ℓ_tc = a|x, y)
     // M step: compute new w
     for o = 1 to O do
       for c = 1 to N do
15       w_co ← Σ_{t=1}^{T} Σ_{a∈R} q_tc(a) Σ_{r∈R} log p(y_tro|ℓ_to = a)
     // M step: compute new containment set
     C ← ∅
     for o = 1 to O do
       c* ← arg max_{c∈[1,N]} w_co
20     C ← C ∪ {(o, c*)}
```

Proof. We show that RFINFER (Algorithm 1) is guaranteed to converge to a local maximum of the likelihood $L(\mathcal{C})$ in (3). Following the EM theory, we can interpret both the E-step and the M-step as maximizing a lower bound on the likelihood, which is

$$L(\mathcal{C}) \geq \sum_{t=1}^{T} \sum_{c=1}^{C} \sum_{a \in \mathcal{R}} q_{tc}(a) \log \frac{p(\ell_{tc} = a, \mathbf{x}, \mathbf{y})}{q_{tc}(a)} = \mathcal{O}(\mathcal{C})$$

The fact that this is a lower bound can be proven by Jensen's inequality. The E-step maximizes this bound with respect to q_{tc}, and the M-step with respect to $\mathcal{C}$. In RFINFER, the E-step is identical to the standard E-step of EM, but we use a custom M-step that is specific to our model. So it suffices to prove that the M-step in RFINFER indeed maximizes $\mathcal{O}(\mathcal{C})$.

When maximizing with respect to $\mathcal{C}$, we can ignore terms that do not depend on $\mathcal{C}$. Expanding $\mathcal{O}(\mathcal{C})$ using Eq (3) and removing irrelevant terms yields

$$\max_{\mathcal{C}} \mathcal{O}(\mathcal{C}) = \max_{\mathcal{C}} \sum_{t=1}^{T} \sum_{c=1}^{C} \sum_{a \in \mathcal{R}} q_{tc}(a) \sum_{o|(o,c) \in \mathcal{C}} \log p(\mathbf{y}_{tc} | \ell_{to} = a)$$

$$= \max_{\{c(o), \forall o\}} \sum_{o=1}^{O} w_{c(o),o},$$

where $c(o)$ denotes the container of object o, and $\mathbf{y}_{tc} = \{y_{trc} | \forall r\}$. In this last equation, notice that each containment decision $c(o)$ that we are maximizing over appears in only one term of the summation. This means that we can find the global maximum by maximizing each term independently, i.e., $\max_{\mathcal{C}} \mathcal{O}(\mathcal{C}) = \sum_o \max_{c'} w_{c',o}$. This is exactly what is computed in lines 12–20 of RFINFER. □

Computation Complexity. Each iteration of RFINFER requires $O(TNOR^2)$ time, where by *iteration* we mean a single execution of lines 2–20. This is due to two reasons. First, the computation of $q_{tc}(a)$ in line 6 requires $O(OR)$ time, and is executed $O(TNR)$ times by the outer loops. Second, the computation of w_{co} in line 15 requires $O(TR^2)$ time, and is executed $O(NO)$ times by its outer loops. This is the running time of a naive implementation of the

algorithm; in Section 9.1, we describe several optimizations that improve the performance significantly. Also, note that this is the computational complexity per iteration; unforunately, it is difficult to characterize the number of iterations required for EM to converge, because this depends strongly on characteristics of the unknown true distribution.

9.1 Implementation and Optimizations

In this section, we sketch the main data structures and optimizations that we use to implement the RFINFER algorithm together with the change point detection extension.

Data structures. We use a series of tables: (1) The read rate table (size: $R \times R$) stores the read rates $\pi(r, \bar{r})$. (2) Two history tables: one for container readings $\mathbf{x}$ (size: $T \times R \times N$), and one for object readings $\mathbf{y}$ (size: $T \times R \times O$) (3) The posterior probability table ($T \times N \times R$) stores the posterior distribution $q_{tc}(a)$ over container locations. (4) The weight table ($N \times O$) stores the co-location strengths w_{co}. (5) Finally, the containment table (vector of length O) stores the container inferred for each object. Although the history and posterior probability tables grow with time, in the next section we describe a history truncation method that reduces the memory requirement without sacrificing accuracy. Finally, many of these tables, especially the history tables, are sparse, i.e., most cells are 0, and so can be easily compressed to save memory.

Frequency of inference. We run RFINFER periodically as new data arrives, for example, once every 300 epochs.

Optimizations. We further employ several optimizations to improve inference efficiency. Recall from Section 3.2 that both the E-step and M-step of our algorithm have the complexity $O(TNOR^2)$. The E-step can be easily improved as each object is typically read in only a small number of locations and each container contains a small number of items. If both of those are bounded independently of R and O, then the E-step requires only $O(TNR)$ time. In a similar way, the M-step also be improved by a factor of R.

Second, we propose an optimization, called *candidate pruning*, to improve the M-step further to $O(TOR)$, eliminating the factor of N. The idea is that in line 15, when computing the container that is most strongly co-located with a given object, it is probably safe to consider only containers that have been observed frequently with the object. So as a heuristic, we restrict the set of candidate containers to those that were most frequently co-located during the first several epochs. When testing for change points, we also include as candidates the most frequently co-located containers from recent epochs. Our experimental results show that candidate pruning is effective at reducing the time cost without affecting the accuracy.

We further propose a *memorization* technique that avoids unnecessary computation: If the set of objects in a container did not change in the previous EM iteration, then the location probabilities and co-location strengths for that container cannot change at the current iteration, so we can simply re-use the old values without any extra work. This optimization does not introduce any error.

9.2 Extensions

We briefly discuss possible extensions to RFINFER. (1) *Hierarchical containment:* Just as objects are grouped into containers, containers may themselves be stored in larger containers, such as pallets. It is easy to extend our model and algorithms to arbitrarily nested containment hierarchies. (2) *Probabilistic containment output:* If probabilistic query processing is desired, then we need to produce a confidence value on the containment tuples in the output. A natural confidence value is the co-location strength w_{co}, but this is not a probability because it is always negative (it is an average of log probabilities). Fortunately, a standard trick can be used to convert w_{co} into a distribution over containers c_o for object o, by $p(c_o) = \exp\{w_{c_o}\} / \sum_{c'} \exp\{w_{c'o}\}$. This choice can be theoretically justified by appealing to an approximation algorithm for graphical models called variational message passing.

10. DISCUSSION OF QUERY PROCESSING

We discuss how our system can provide support for relational and probabilistic query processing. For query processing, our system can use either data stream processing based on the traditional relational model [1, 2], or probabilistic data stream processing based on an uncertainty model [4, 13]. In the former approach, the events are treated as deterministic, so the query processing results may contain errors if inference is not entirely correct. However, this approach does not require any change of existing data stream systems and is shown to yield only a small error given the high accuracy of our inference algorithms (detailed in §5). The latter approach, on the other hand, exposes the uncertainty in inference to the query processor by explicitly including location and containment distributions in the events. Probabilistic query processing then produces answers with confidence values. The computation, however, can be much more expensive than in the first approach. Our system supports this approach by extending the inference algorithms to output both the location distribution and the containment distribution, as described in §9.2. We leave the choice of approach to the discretion of the application, but assume the first approach when discussing the issues in distributed query processing for simplicity.

11. EXPERIMENTAL EVALUATION

In this section, we describe additional experimental results beyond the key results presented in the main body of the paper.

11.1 Experimental Setup

We first describe the methodology employed to generate the synthetic traces summarized in Table 1. We developed a simulator using CSIM to emulate an RFID-based enterprise supply chain. Each supply chain has N warehouses organized in a single-source directed acyclic graph (DAG). Pallets of cases are injected at the single source, and then move through a sequence of warehouses with a scheduled delay in each warehouse and scheduled transit time between two warehouses, until they reach final destinations. Our simulator guarantees that in a period of time, the numbers of pallets arriving at a warehouse and departing from it are about the same, so there is no backlog in the system. For distributed processing and scalability tests, our simulator can "bulk-load" a large number (I) of items into the supply chain before starting to inject pallets at the source and consume pallets at the destinations.

Within a warehouse, pallets first arrive at the entry door and are read by the reader there. They are then unpacked. By default, each warehouse has a reader at the conveyor belt that scans the cases one at a time. The cases are then placed onto shelves and scanned by the shelf readers. After a period of stay, cases are removed from the shelves and repackaged. The newly assembled pallets are finally read at the exit door and dispatched to subsequent warehouses in a round-robin fashion. In the simulation, all readers have a read rate RR for its location, uniformly sampled from [0.6, 1] unless stated otherwise. There is significant overlap between adjacent shelf readers: a shelf reader can read objects in a nearby location with probability OR uniformly sampled from [0.2, 0.8]. Two separate parameters are used to control the read frequencies of shelf readers and non-shelf readers as in many real-world deployments. Finally, to stress test our containment change detection algorithm, our simulator can inject anomalies that randomly pick an item and place it in a different case, with the frequency specified by the parameter FA.

Our evaluation uses the following metrics: **Error rate** (%): To measure accuracy, we compare the inference results with the ground truth and compute the error rate. **F-measure**: For change point

detection, we evaluate the accuracy of the reported changes. We use *precision* to capture the percentage of reported changes that are consistent with the ground truth, and *recall* to capture the percentage of changes in the ground truth that are reported by our algorithm. We combine them into $F\text{-}measure = 2 * precision * recall/(precision + recall)$. **Running cost**: We report the time taken to evaluate a trace using a single-threaded implementation running on a server with an Intel Xeon 3GHz CPU and running Java HotSpot 64-bit server VM 1.6 with maximum heap size 1.5GB.

11.2 Lab RFID Deployment

Using our lab setup, we created 8 traces with distinct characteristics, by varying the environmental noise, overlap among antennas, and tag orientations:

- T_1 (RR=0.85, OR=0.25) represents the case of high read rates, an average of 0.85 across antennas, and limited overlap rates, an average of 0.25 for shelf antennas using low power.
- T_2 (RR=0.85, OR=0.5) is case of high read rates and significant overlap rates (using high power), an average of 0.5.
- T_3 (RR=0.7, OR=0.25) involves lower read rates due to added environmental noise, i.e., a metal bar placed on each shelf that is 1/3 the length of the shelf.
- T_4 (RR=0.7, OR=0.5) further has higher overlap rates.
- T_5 to T_8 extend T_1 to T_4, respectively, with containment changes. When all 20 cases were placed on shelves, 3 items were moved from one case to another and 1 item was simply removed, causing containment changes in 35% of the cases.

We also obtained traces with varied tag orientations but observed little impact of this factor, verifying that squiggle tags are orientation-insensitive when used with circularly-polarized antennas.

11.3 Alternative Method for Comparison

We describe the design of the SMURF* method used in both the lab experiment and in simulations using synthetic traces. This method first uses SMURF to smooth raw readings of objects to estimate their locations separately. It then uses heuristics to infer containment changes: Given an item and a time t, if the most frequently co-located case before time t is the same as that after time t, then there is no containment change. Otherwise, it further checks if none of the top-k co-located cases before time t is in the top-k co-located case after time t. If so, it reports a containment change for this item at time t. Note that the second check is needed because due to the missing readings, the real container may not be the most frequently co-located case. Once a containment change is signaled, it picks the best co-located container from t to the present.

11.4 Single-Site Inference

We describe several additional results for our inference methods when run on a single warehouse.

Basic inference algorithm. In the first experiment, we evaluate the basic algorithm presented in §3.2 for its sensitivity to various noise factors. We began with short 1500-second traces and ran inference each time with all the readings obtained thus far. We first varied the read rate RR from 0.6 to 1. Fig. 5(a) shows the inference error rates. In particular, location inference is highly accurate, with the error rate less than 0.5% for all read rates. Containment inference is more sensitive to the read rate but still achieves an error rate less than 7% for the 0.6 read rate. The sensitivity of containment inference to the read rate is due to its use of co-location information: the chance of reading both an item and its container reduces quadratically with the read rate. Fortunately, the use of history alleviates the problem and keeps the error rate low. The high accuracy of location inference is due to the effect of "smoothing over containment": once we understand containment correctly, the location of an object can be revealed by the readings of any other object(s) in the container.

We also varied the overlap rate OR from 0.2 to 0.8 while fixing the read rate RR at 0.7. Results show that neither location nor containment inference is sensitive to the overlap rate: the error rate of containment inference is flat at 2.3% and that of location inference is flat at 0.08%.

History truncation method. In addition to the total inference time that we presented in Fig. 3(b), we also computed the errors rates of the three methods. Fig. 5(b) shows the error rates of containment inference using the above methods (the error rates of location inference are always low in the 0.05% to 1% range, hence omitted). Again, we observe the naive window-based truncation to be inaccurate. Its error rate increases for longer traces because our simulation generates the belt readings useful for containment inference in the first half of the warehouse setup. In contrast, using the full history or the CR method makes inference not very sensitive to the trace length, with the CR method being somewhat better.

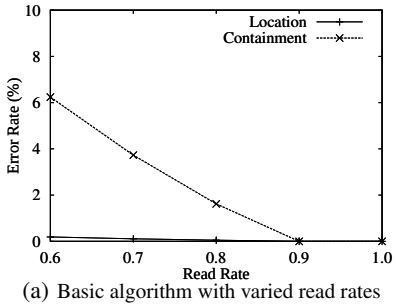
Containment change detection. We present additional results to further analyze the tradeoff between accuracy and efficiency. Recall that the change point detection algorithm requires a recent history (whose size is H), besides the critical region in the past, to detect containment changes. Since we run inference with the default frequency of once every 300 seconds, H has to be at least 300 seconds to include all new observations for inference. However, a longer recent history may be needed to ensure accuracy of change point detection while it may also increase inference cost. We next vary H to study such tradeoff between accuracy and efficiency. We created traces with varied read rates and for each trace, measured F-measures and time costs as different H values were used.

Table 3: F-measures (%) and time costs (sec) of change point detection with different recent history sizes (H) for different read rates (RR).

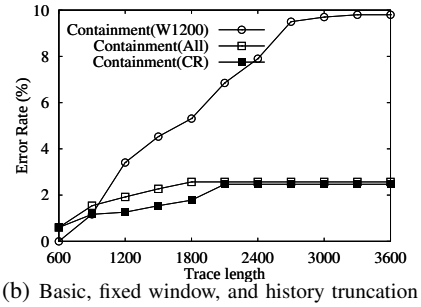
		Recent history size H						
		300	400	500	600	700	800	900
RR=0.6	*F-m.* (%)	46	67	81	**87**	88	86	92
	Time (s)	205	235	**293**	385	418	497	556
RR=0.7	*F-m.* (%)	72	**91**	90	90	93	93	94
	Time (s)	182	229	**288**	344	403	469	523
RR=0.8	*F-m.* (%)	73	**92**	94	97	93	95	96
	Time (s)	182	220	**283**	341	395	446	490
RR=0.9	*F-m* (%)	82	**94**	95	98	93	95	96
	Time (s)	172	207	**258**	322	381	436	458

Table 3 shows the results. The overall trend with all traces is that as H increases, the F-measure improves but the time cost also increases. Among different read rates, we see that achieving high accuracy for lower read rates such as 0.6 requires larger sizes of H, while the time cost varies with H consistently across all read rates. This implies two ways to trade off accuracy and efficiency: (1) If the application requirement is to achieve best accuracy while keeping up with stream speed, we should use the H sizes that yield the time costs in bold in the table because they allow inference to complete within 300 seconds, the interval before the next inference starts. This way, our algorithm offers above 90% accuracy for read rates $\geq$ 0.7 and above 80% for the read rate = 0.6. (2) If the requirement is to optimize performance while satisfying certain accuracy, say, 85%, we should use the H sizes that yield the F-measures in bold in the table. In particular, when the read rate is high, we can use a smaller size of H to reduce the time cost and enable more frequent inference (e.g., for read rate=0.8, running inference every 212 seconds can keep up with the stream speed).

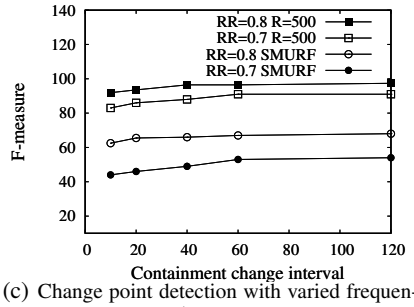
We finally show more results about the sensitivity of our algorithm to the frequency of containment changes. The parameter varied was the interval between two containment changes, from 10 to



(a) Basic algorithm with varied read rates



(b) Basic, fixed window, and history truncation methods with varied trace lengths



(c) Change point detection with varied frequencies (against SMURF*)

Figure 5: Experimental results for single-site inference

120 seconds. For comparison, we include the SMURF* algorithm described in §11.3. Fig. 5(c) shows the F-measures of our algorithm and SMURF*. For our algorithm, we chose the H size to keep up with stream speed based on Table 3, i.e., H=500 for both RR=0.7 and RR=0.8. As can be seen, our algorithm has much better accuracy than SMURF* and our accuracy is not so sensitive to the containment change interval. SMURF* lacks of the iterative feedback between location and containment estimates and a principled approach to doing so. Hence, it has very low F-measures, e.g., around 50% for read rate 0.7 and just above 60% for read rate 0.8, hence of limited practice use.

Summary: The above results show that our inference algorithm is highly accurate for various noisy traces with stable containment ($\leq$7% error rate for containment inference and around 0.5% for containment inference). The critical region method can significantly reduce inference cost for long traces while further improving accuracy. With change point detection, our algorithm can achieve 85% accuracy in most cases when the read rate $\geq$ 0.7 while keeping up with stream speed, as demonstrated using both real lab traces with various noise factors and simulations with different containment change frequencies.

11.5 Distributed Inference

We provide additional details of our comparison of distributed and centralized inference. For the centralized approach, we assume that all raw data is shipped to a central location for inference with simple gzip compression of data. For the distributed approach, we consider several methods for inference state management: a naive method ("None") that does not transfer any state for an item, the critical region (CR) method that transfers only the weight of the container that dominated others in the detected critical region, and a variant of the CR method (CR+H) that also transfers the weights of containers that co-located with an item in the recent history. We report on the error rate (for inference results about both stable containment and changes) and communication costs of these methods.

We first varied the number of warehouses with a belt reader from 1 to 6, assuming that the source warehouse always has a belt reader. By a belt reader, we mean a special reader that reads containers one at a time, offering valuable information about the true container. Fig. 6 shows the error rates. The centralized approach is always the most accurate because it has all the data. As long as the source warehouse with a belt reader, the two CR-based methods have almost the same accuracy as the centralized approach. In contrast, the None method loses accuracy as the number of warehouses with belt readers reduces. This implies that discarding inference state leads to poor performance if the supply chain does not offer repeated special setups for containment inference.

Finally the communication overheads of the distributed and centralized approaches are shown in the following table. The figure

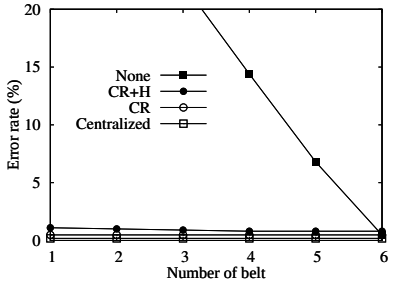


Figure 6: Accuracy for varying number of belt readers

shows that our CR methods offer an order of magnitude reduction in communication costs.

Table 4: Communication costs (bytes) of a centralized approach and three state migration methods for distributed inference.

	Centralized	None	CR	CR+H
RR=0.6	125,895,500	0	225,890	231,180
RR=0.7	145,858,950	0	223,790	224,480
RR=0.8	166,746,235	0	225,890	224,000
RR=0.9	187,589,810	0	225,890	224,000

Summary: Our results show that distributed inference using the CR methods have accuracy that are close to the centralized approach, while incurring significantly lower communication costs.

Comment on scalability. In our simulation, our combined inference and query processing system scaled to 32,000 items per warehouse while keeping up with stream speed, totally 0.32 million objects over 10 warehouses. While our simulation was limited by a a single threaded implementation, a real-world implementation of our approach would employ a multi-threaded approach that could exploit multi-cores on servers. This will allow us to easily scale to over a million objects in the 10-warehouse supply chain and even larger supply chains with more warehouses, while our reported results on accuracy, performance, and communication costs stay true.