PeerTrack: A Platform for Tracking and Tracing Objects in Large-Scale Traceability Networks

Yanbo Wu, Quan Z. Sheng, Damith Ranasinghe and Lina Yao
School of Computer Science
The University of Adelaide, SA, 5005, Australia
{yanbo, qsheng, damith, lina.yao}@cs.adelaide.edu.au

ABSTRACT

The ability to track and trace individual items, especially through large-scale and distributed networks, is the key to realizing many important business applications such as supply chain management, asset tracking, and counterfeit detection. Unfortunately, enabling traceability across independent organizations still poses significant challenges in dealing with large volume of data and sovereignty of the participants. This paper describes PeerTrack, a scalable platform for efficiently and effectively tracking and tracing objects in large-scale traceability networks. With a novel data model, a DHT-based indexer, and a distributed query processor, PeerTrack provides an environment where traceability applications can share data across independent organizations in a peer-to-peer fashion. This paper presents the motivation, system design, implementation, and a proof-of-concept system of the PeerTrack platform.

1. INTRODUCTION

Traceability refers to the capability of an application to track and trace the state (e.g., location) of an object, including discovering the information of its current and past state, as well as estimating the information of its future state. Traceability is essential to a wide range of important business applications such as manufacturing control, logistics of distribution, product recalls, and anti-counterfeiting [6].

Recent advances in sensor and RFID (radio-frequency identification) technologies make automatic tracking and tracing possible in large-scale applications (e.g., nation-wide supply chain management across companies). An emerging technique that targets large-scale traceability is the so called "Networked RFID" [5, 6]. The basic idea behind the Networked RFID is to realize a "data-on-network" system, where RFID tags contain an unambiguous ID and other data pertaining to the objects are stored and accessed over the Internet. With "Networked RFID", traceability applications analyze automatically recorded identification events to discover the current location of an individual item. They can also retrieve historical information, such as previous locations, transportation time between locations, and time spent in storage.

EDBT 2012, March 26–30, 2012, Berlin, Germany.

However, enabling traceability is not a single layer problem [9]. Large-scale global networks have the potential to generate unprecedented amounts of data related to individual objects. An important challenge centers on the efficient management and sharing of this data in traceability applications. An obvious solution is to publish all data collected within each organization to a central data warehouse. Unfortunately, this approach has several severe drawbacks. Firstly, object movement and related data are valuable business information that companies may be very reluctant to put in a shared central warehouse. Secondly, such an approach has very limited scalability and is not feasible for large-scale applications where the amount of data collected could be enormous [1, 2, 6, 8]. The system architecture for data gathering, processing and sharing must be scalable in order to deal with the data collected from networked systems. For efficient processing and storage, data models must be carefully designed. To allow business users making useful decisions and analysis in a timely manner, different types of traceability queries as well as event-driven notification services must be conveniently supported.

Motivated by these concerns, we have developed the PeerTrack platform for efficiently tracking and tracing objects in large-scale traceability networks. PeerTrack features a pure P2P architecture for data and query processing. In particular, we have designed a novel data model for traceability networks, which eliminates the data dependencies between organizations. Important tracking and tracing queries have been implemented as built-in features, meanwhile we provide the flexibility of developing add-in queries. In the following sections, we will overview the design and implementation of the platform, and sketch the proposed demonstration. For the details of the data model, architecture and data models, interested readers are referred to [10].

2. SYSTEM OVERVIEW

PeerTrack (see Figure 1) exploits the distributed hash table (DHT) infrastructure Chord [7] to manage peers and route messages. Each organization is viewed as an equal peer of the network. Within a peer, applications can access local data, as well as remote data of other peers using the *tracking engine* via the Internet. The tracking engine includes a traceability data model, an indexer, and a distributed query processor. Developers can implement various kinds of traceability applications using the generic APIs built on top of these modules. To support event-driven services (e.g., notifications), a rule engine has been developed. It monitors an event queue which is public to other modules. A visual tool, namely the *rule editor*, facilitates an easy definition of rules.

PeerTrack relies on a generic data model for moving objects in

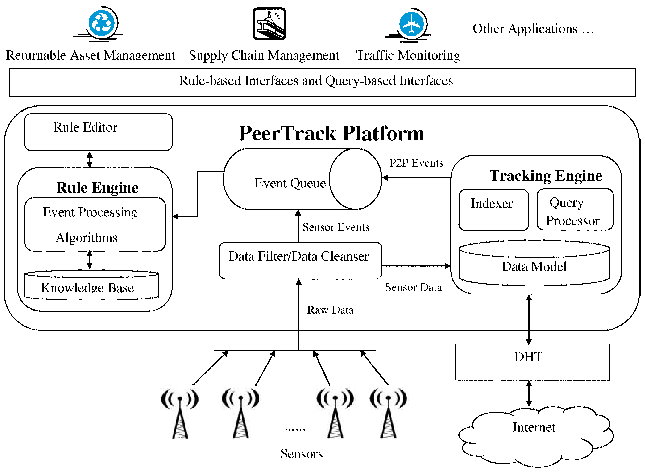

Figure 1: The PeerTrack Architecture

large-scale networks, namely "a Model for mOving Objects in Discrete Space" (MOODS). A discrete space refers to a finite set of nodes which represents all the organizations in the network. MOODS eliminates the data dependencies between organizations by storing the information about object movements at the nodes where the object has been transported. In particular, PeerTrack introduces the *Information of Object Path* (IOP), which includes properties that indicate the departure and arriving information of objects. With IOP, each node maintains segments of objects' moving paths and uses this information to expedite P2P queries in the network.

The indexer is the key to acquiring IOP. It indexes an observed object and its latest location (i.e., the last node where the object is observed) at a deterministic node called the *gateway node*. The gateway node is solely determined by the id of the object thanks to the determinism of the DHT. When the object is observed at a new node $\mathcal{N}_d$, the node sends the object's id to its gateway node via the indexer. The indexer at the gateway node uses this information to update its index and sends a message back to $\mathcal{N}_d$, notifying it the node where the object comes from (i.e., $\mathcal{N}_s$). Meanwhile, the indexer also sends a message to $\mathcal{N}_s$, informing the node that the object has arrived at $\mathcal{N}_d$. As a result, the IOP is established, which is essentially a distributed double linked list. To reduce the indexing overhead for large-volume objects, we enhance the indexing algorithm by grouping the objects according to the prefixes of their hashed ids. The scheme to determine the optimal length of prefixes for grouping is carefully chosen in regard to both scalability and load balancing. Our detailed design of the indexer can be found in [10].

The fundamental design principle of the query processor is to process a query locally to the extent possible and, if necessary, enhance it using locally available information before forwarding it to appropriate remote organizations. Due to the introduction of IOP in MOODS, we do not have to flood the query to all the nodes in the network. Instead, the query is processed following the IOP link. As a result, the performance of query processing can be significantly improved. To answer a tracking query, i.e., finding the current location of an object, the query processor simply contacts the gateway node for the object via Chord. A tracing query, i.e., finding the pedigree of an object, can be answered by first tracking the object (i.e., finding its current location) and then simply tracing back the list using the IOP information.

3. IMPLEMENTATION

The PeerTrack platform provides an environment for supporting the development of distributed and large-scale traceability applications. It offers two interfaces to application developers, namely the *query interface* and the *rule interface*, in the form of Java API. The query interface accepts queries and executes them either locally and/or remotely at other nodes. The rule interface can be used to specify business rules for event-driven services (e.g., out-of-stock alerts). These two interfaces are built on top of PeerTrack's main modules, namely the *tracking engine* and the *rule engine*. We adopted the open source Java project OpenChord[1] as the implementation of Chord [7]. It supports storing all serializable Java objects within the DHT.

The event filterer and cleanser have been implemented to filter and clean the data stream collected from various data sources, e.g., RFID readers and barcode readers. It mainly implements the algorithms proposed in [3].

The indexer is transparent to the application developers. It implements the group indexing and IOP acquisition algorithm described in Section 2. The indexer reads data from the stream processor periodically. Objects observed within a cycle are classified into groups according to the prefixes of their hashed ids. We use SHA-1 as the hash function for its uniformity. The length of the prefixes is determined by the function $\log_2 \mathcal{S} + \log_2 \log_2 \mathcal{S}$, where $\mathcal{S}$ is the size of the network, represented by the number of nodes within the network. We have conducted extensive experiments to show that the function can guarantee both scalability and good load balancing. The anti-entropy aggregation protocol proposed in [4] has been implemented to estimate the value of $\mathcal{S}$.

The query processor accepts queries from both the application layer and the DHT layer (rewritten from other nodes). A `Dispatcher` class has been implemented to pick up a corresponding registered processor instance for an incoming query. All queries and their processor instances are implemented as plug-ins. For example, for a tracking query `TrackQuery` that locates an object, a `TrackQueryProcessor` is implemented. Both classes are registered in the query processor via the `register(Query q, Processor p)` interface at runtime, where `Query` and `Processor` are the Java classes for all queries and processors. This mechanism ensures high flexibility and makes it possible to dynamically upgrade the system without rebooting. We have implemented some important traceability queries. A special implementation is the `RewritableQuery` class, which represents queries to be rewritten for executing at other nodes via DHT interface. Tracking and tracing queries are subclasses of the `RewritableQuery`. The query objects are all serializable for seamless integration with OpenChord.

The rule engine is built on top of *JBoss Drools*[2], which includes three main modules, namely *Drools Guvnor*, *Drools Fusion*, *Drools Expert*. We used Drools Fusion and Drools Expert to build our rule engine. *Drools Guvnor* is used as the knowledge base for storing and organization business contexts and the defined rules. The rule engine monitors an *event queue* which receives events from other

[1]http://open-chord.sourceforge.net/
[2]http://www.jboss.org/drools

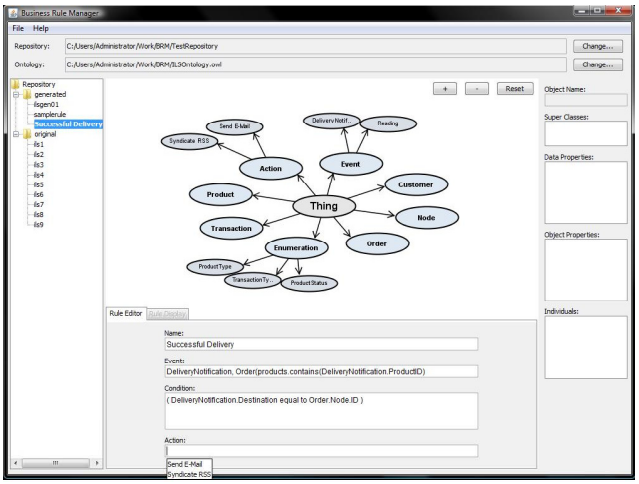

Figure 2: Screenshot of the Rule Editor

modules in PeerTrack and triggers corresponding actions if the conditions are satisfied. The rule editor (Figure 2) is developed to provide a visual interface for efficiently defining and editing business rules. It also contains a knowledge base that assists users who do not have specialized knowledge to define business rules with a semi-structured natural language. The rule editor supports the semantic specification of rules and automatically translates the visual representation of rules to a specified rule language (in this case, Drools Rule Language).

4. DEMONSTRATION SCENARIO

We have developed several traceability applications on top of the PeerTrack platform, including mobile asset management and supply chain management. In this paper, we will focus on presenting a mobile asset management system. This system has been deployed at the International Linen Service PTY LTD. (ILS), a company that provides a suite of linen services for over 200 customers (e.g., hotels, hospitals, and aged-care homes) in South Australia. Each customer has one or more delivery locations (nodes). In this system, trolleys (objects) are reusable containers for linens and they are attached with RFID tags. They are transported among nodes and can be detected by RFID readers when they arrive at a delivery location. An order is simply an aggregation of several trolleys for the same customer. The mobile asset management system developed from the PeerTrack platform offers an automated tracking and tracing service with the capability to monitor and control ILS logistical operations in real-time over 300 different locations.

We also developed a visual monitoring tool that has been deployed at each customer's site, together with the P2P services. Figure 3 shows the screenshot of the visualization tool deployed at ILS. We use this application as the demo to show the advantages of our PeerTrack platform.

Tracing and Tracking. Tracing a trolley can be initiated using the "Order/Object Search Tool" panel. After a user inputs the trolley id and clicks the "Trace Trolley" button, the visualization tool creates `TraceTrolleyQuery`, which is a subclass of the `TraceQuery`, and sends it to the PeerTrack platform. PeerTrack traces it to the original location of the trolley following the IOP stored in each node along its moving path. The search result is shown as highlighted lines on the map of the query initiator.

Similarly, for tracking a trolley, the `TrackTrolleyQuery` is sent to the corresponding gateway node via the underlying structured overlay. The gateway node which maintains its latest location then sends back the node id of the current location of the queried trolley. The result is shown at the query initiator as an informational bubble (see the one at the left top of the "Global Delivery Network" map in Figure 3).

With this architecture, the tracking and tracing can be done with minimum number of network calls. In the implementation, to further reduce the network cost, we cache the IP address of customers so that in most cases the underlying P2P routing cost is eliminated.

Inventory monitoring. The query processor offers facilities for developer to implement various kinds of queries and register them into the system. To realize quasi-real-time inventory monitoring, an `InventoryQuery` class (which is subclass of `RewritableQuery`) and its processor `InventoryQueryProcessor` are implemented. The former defines the query parameters including the node to monitor and the refreshing interval. When the query is issued to the query processor, an `InventoryQueryProcessor` instance is initialized for query processing. Specifically, the `InventoryQueryProcessor` periodically sends the `InventoryQuery` object via OpenChord to the monitored node.

When the node icon is clicked, a context menu is popped up with the option to start monitoring the node. The inventory is displayed near the node (e.g., the number "50", "100") and refreshed at a certain interval.

Real-time Monitoring. The users can define various kinds of rules in PeerTrack and get notified when an event of interest occurs. The rules for successful and incorrect deliveries have been created with the rule editor (Figure 2 shows the definition for the "Successful Delivery" rule). When the successful delivery rule is triggered, i.e., a trolley is delivered to the correct destination, the "Real-time Order/Object Status Monitoring" area in Figure 3 is updated with the latest information. Meanwhile, the "Global Delivery Network" map is updated by showing an icon at the destination of the delivery.

This is realized by the indexer and the rule engine. As part of the IOP acquisition process, the indexer will be notified by the gateway node after an object has arrived at another node. This notification, as an event, is also sent to the event queue which is monitored by the rule engine. In the application, we also implemented the "Internal Workflow Monitoring" sub-system that monitors the internal movements of the trolleys.

5. CONCLUSION

In this paper, we have presented PeerTrack, a comprehensive platform for efficiently and effectively tracking and tracing objects in large-scale and distributed networks. PeerTrack relies on a novel data model, a DHT-based indexer and a distributed query processor. The platform has been validated by successfully creating a number of traceability applications. Currently we are extending the platform to support queries on predicting future status of objects. This typically involves overcoming uncertainty issues introduced by traceability applications using statistical and probabilistic techniques. Interested readers are referred to the project website[3] for

[3]http://www.cs.adelaide.edu.au/peertrack

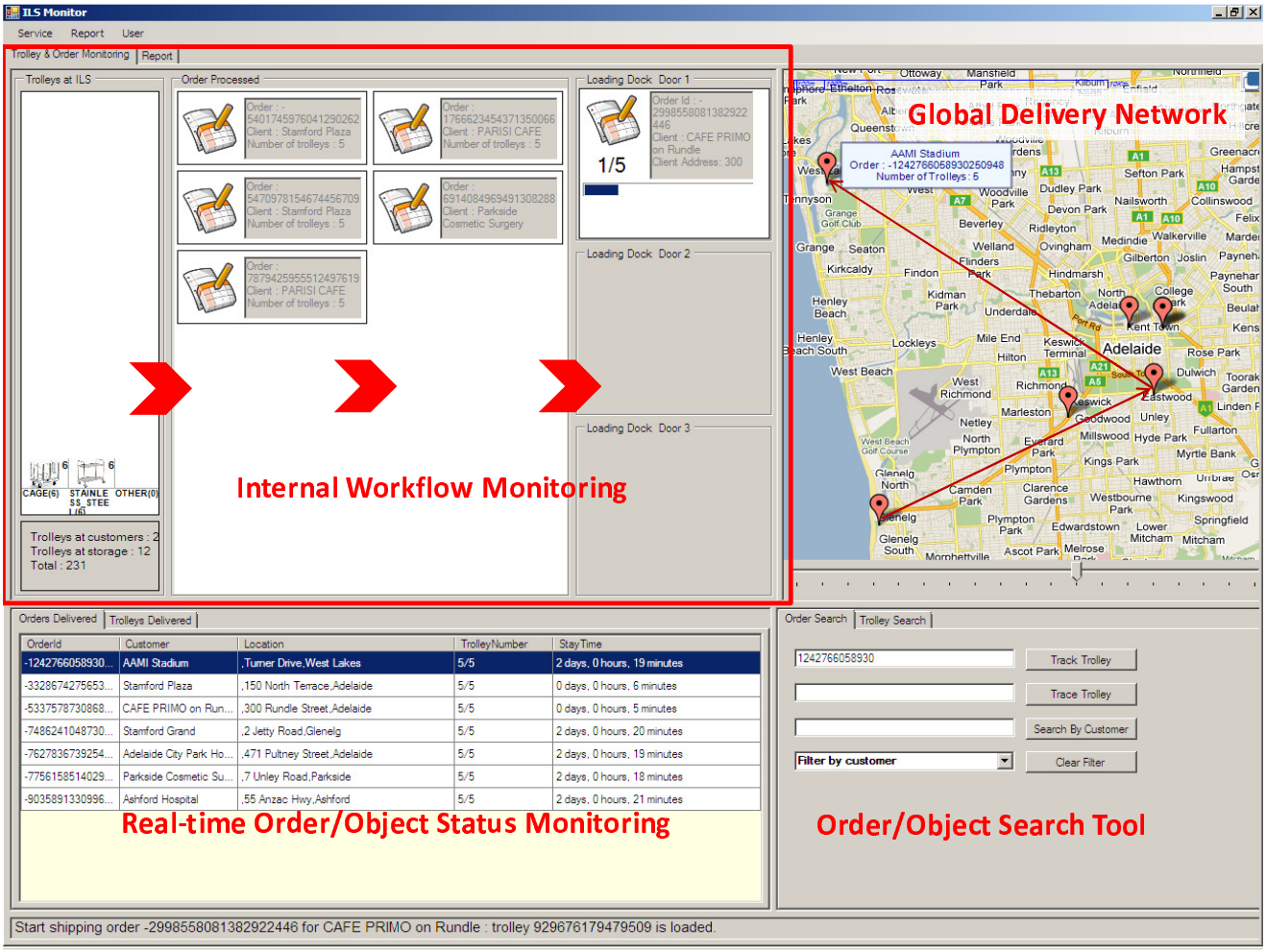

Figure 3: Visualization Tool

more details.

6. REFERENCES

[1] R. Agrawal, A. Cheung, K. Kailing, and S. Schönauer. Towards Traceability Across Sovereign, Distributed RFID Databases. In *Proc. of the 10th Intl. Database Engineering and Applications Symposium (IDEAS'06)*, Delhi, India, December 2006.

[2] M. J. Franklin, S. R. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, E. Wu, O. Cooper, A. Edakkunni, and W. Hong. Design Considerations for High Fan-in Systems: The HiFi Approach. In *Proc. of the Second Biennial Conf. on Innovative Data Systems Research (CIDR'05)*, Asilomar, CA, USA, January 2005.

[3] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. Declarative Support for Sensor Data Cleaning. In *Proc. of the 4th Intl. Conf. on Pervasive Computing (Pervasive'06)*, Dublin, Ireland, May 2006.

[4] M. Jelasity and A. Montresor. Epidemic-Style Proactive Aggregation in Large Overlay Networks. In *Proc. of the 24th Intl. Conf. on Distributed Computing Systems (ICDCS'04)*, Washington, DC, USA, 2004.

[5] G. Roussos, S. S. Duri, and C. W. Thompson. RFID Meets The Internet. *IEEE Internet Computing*, 13(1):11–13, 2009.

[6] Q. Z. Sheng, X. Li, and S. Zeadally. Enabling Next-Generation RFID Applications: Solutions and Challenges. *IEEE Computer*, 41(9):21–28, September 2008.

[7] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proc. of the 2001 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*, New York, NY, USA, 2001.

[8] S. Wu, S. Jiang, B. C. Ooi, and K.-L. Tan. Distributed Online Aggregations. *PVLDB*, 2(1):443–454, 2009.

[9] Y. Wu, D. C. Ranasinghe, Q. Z. Sheng, S. Zeadally, and J. Yu. RFID Enabled Traceability Networks: a Survey. *Distributed and Parallel Databases*, 29(5-6):397–443.

[10] Y. Wu, Q. Z. Sheng, and D. Ranasinghe. Peer-to-Peer Objects Tracking in the Internet of Things. In *Proceedings of the 40th International Conference on Parallel Processing (ICPP'11)*, Taipei, Taiwan, 2011.