Literature Review on RFID and DSMS

1. RFID & RFID system

Recently, RFID has gained a lot of attention as a technology that facilitates ubiquitous computing. This section gives a brief introduction to RFID and RFID systems.

1.1 What is RFID?

RFID stands for “radio frequency identification”, and it's a technology that enables remotely storing and retrieving information by means of electromagnetic radiation. An RFID system consists of 2 main components [2]:

1. RFID tag (transponder), which is usually attached to the object to be identified and carries information in an electronic microchip;
2. RFID reader (interrogator), which detects tags and performs read/write operations on RFID tags. Normally readers are connected with an additional interface to forward tag information to another system, like a PC or robot control system. [1]

In the most common RFID system, the reader transmits a low-power radio signal to the tag, which receives the signal and uses it as a power source to activate the embedded integrated circuit, and then transmits the information stored in it back to the reader through the antenna. The information carried by the RFID tag is the identity of the tag and other relevant information.

1.2 RFID: Benefits and Limitations

RFID is not a new technology, but it was not widely deployed and used until 1990s. [3] Businesses across various industry sectors, for example, Wal-Mart, and US Department of Defense, also began to adopt RFID in their enterprise information systems. This enthusiasm is driven by the benefits brought by RFID system, which include:

Efficiency: data capturing can be done with less or without labor involved, data collection speed is greatly improved as multiple tags can be identified almost simultaneously;
Flexibility: there are not many restrictions on tag placements;
Improved data accuracy level: with RFID system, objects can be tracked on the item level;
Cost savings: despite all these benefits, RFID is not much more expensive than traditional data collection methods like the barcode system. Normally, a passive tag costs several cents, and its price will be even cheaper in near future.

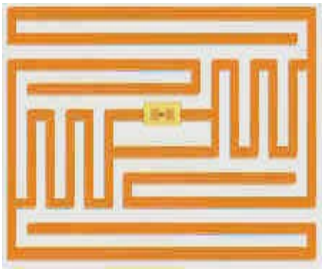

Illustration 1: RFID tags used in Wal-Mart

However, RFID system also has its limitations. The 2 main undesired effects are false negative reads, where RFID tags are not read at all, and false positive reads, where RFID tags are detected when they are not in the interrogation range of the reader. [2] These failures can be caused by collisions on the air interface, tag detuning, tag misalignment, metal and water in the vicinity of the RFID system. [4]

1.3 Types of RFID

Typically RFID systems are classified according to the functionality of the tags they use. Different systems can be distinguished by [2]:

1.types of power supply [1]:
Passive tag, which requires no power source or battery within the tag and uses the radio wave broadcast by the reader to energize its operation. This is the least expensive tag.
Semi-passive tag, which employs an embedded battery to achieve better performance. Normally, such tags are only activated when they are in the reading range of a reader.
Active tag, which uses batteries for the entire operation, and can generate radio waves in the absence of an RFID reader.
Some related statistics about these tags are listed in the following table [5]:

Name	***Power Source***	***Transmission Mode***	***Range (within)***	***Life***
passive	scavenging	reflective	3 meters	unlimited
semi-passive	battery	reflective	10 meters	5-10 years
active	battery	active	100 meters	1-5 years

Table 1: RFID tags classified by the power source

2.operating frequency:
The frequency of an RFID system defines the relationship between the tag and reader, and impacts both the transmission range and speed.
RFID systems can work in the low frequency (LF), high frequency (HF), ultra high frequency (UHF), microwave parts of the spectrum. Common frequencies for RFID systems are 125-134 KHz (LF), 13.56 MHz (HF), 860-930 MHz (UHF), and 2.45 GHz (microwave).
Some related statistics about tags of different frequencies are as follows: [6]

Name	***Frequency***	***Range***	***Data Rate***	***Ability to read in the vicinity of water and metal***	***Passive tag size***	***Typical Applications***
LF	125 KHz	< 0.5m (1.5ft)	slower |	better |	larger |	access control, animal tracking, vehicle immobilizers, POS applications
HF	13.56 MHz	1m (3ft)	| |	| |	| |	access control, smart cards, smart shelves, item level tracking
UHF	860-930 MHz	3m (10ft)	| | |	| | |	| | |	pallet tracking, electronic toll collection, baggage handling
Microwave	2.45/5.8 GHz	1m (3ft)	| | faster	| | worse	| | smaller	supply chain applications and electronic toll collection

Table 2: RFID tags with different frequencies

3.read/write ability:
Read-only tag, which is also called write-once, read-many tag. After the data is written to the tag, it can not be modified any more. Most tags belong to this category.
Writable tag, whose stored information can be changed by predefined operations.
4.other characteristics like the amount of data that are stored on the tags, or even the anti-collision algorithm used.

2. Data Stream Management System

A data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items. [7] Data generated by RFID system is a perfect example of data stream.
Data stream management system (DSMS) emerges with the purpose to deal with such data streams. This section first elaborates the motivations and challenges to develop such a new data management system, and then introduces several DSMS projects and summarizes the techniques they adopt to process data streams and execute continuous queries.

2.1 Motivations and Challenges

Currently, transient data streams are used in a wide variety of applications including financial transactional applications, network monitoring, road traffic monitoring, security, manufacturing, sensor networks, web applications, and others.

However, traditional DBMS is not suitable for processing such data streams. Traditional database systems employ a human-active, DBMS-passive (HADP) model, which assumes that DBMS is just a passive storage for a large collection of data, and human beings initiate queries and execute updates on this static storage. [8] And these assumptions are problematic when dealing with data streams, because data stream applications need a DBMS-active, human-passive (DAHP) model, in which data are acquired from external sources, and human beings are notified when abnormal activity is detected. Besides, traditional DBMS are not designed for rapid and continuous loading of infinite data, and they can not directly support continuous queries. Furthermore, query approximation, query adaptivity, and real-time requirements are quite essential in executing queries over data streams [9], which are not a feature of DBMS, either.

The differences between traditional DBMS and DSMS are summarized in the following table [10]:

Comparison	***DBMS***	***DSMS***
computational resources	resource rich	resource limited
query processing	reasonably complex, near real-time query processing	extremely sophisticated query processing
data model	set/ bag of tuples	tuple sequences
database update types	arbitrary data modifications	mainly appending operations
query types	one-time, transient queries	persistent, continuous queries

Comparison	*DBMS*	*DSMS*
query answer	exact answer	approximate query answers
query evaluation	arbitrary data access	one/ few pass algorithms
query plan	optimized at beginning	adaptive query plans

Table 3: Comparison between DBMS and DSMS

Another important issue is that data processing in DSMS does not preclude the presence of data relations stored in conventional databases. Traditional databases could also be used for static data storage in a DSMS. In fact, some DSMS projects like *TelegraphCQ* are developed based on *PostgreSQL*. [18] Besides, hybrid queries, which are an important type of query of DSMS systems, might also need to be partially executed on DBMS.

2.2 DSMS Projects

There are quite a few academic projects involved in the research of DSMS. We are going to give an overview of these projects in this subsection.

1. *Eddy* is a query processing mechanism, which allows fine-grained, adaptive, online optimization. [12] *Eddy* continuously reorders operators in a query plan as it runs according to the changing characteristics of incoming data, with the aim to achieve dynamic query optimization and execution. Although it's not designed specifically for data streaming processing, it is extensively studied by later DSMS projects, and its idea for query optimization is borrowed for later DSMS research.

2. *NiagaraCQ* is an Internet-scale continuous query system using group optimization based on the assumption that continuous queries on the Internet have some similarities. [20] It proposes a new optimization methodology called incremental group optimization strategy, which enables the scalability of query group optimization in large dynamic environment.

3. *CACQ* is a continuously adaptive continuous query implementation motivated by the need to handle data streams from sensor networks. [21] It bases its query adaptivity on the *eddy* query processing framework. Furthermore, it maximizes the sharing of work and state across queries to improve overall query performance.

4. *PSoup* is a main memory system that further exploits the mechanisms of *CACQ* and *eddy*. [19] It also proposes solutions to some problems that *NiagaraCQ* and *CACQ* do not address such as queries over historical data and disconnected operations. The main idea behind PSoup is that both data and queries are streaming, and more importantly, are duals of each other: multi-query processing is viewed as a join of query and data streams. *PSoup* continuously runs the data/query join, and materializing the results in the Results Structure.

5. *TelegraphCQ* focused on handling large streams of continuous queries over high-volume, highly variable data streams. [18] It builds on the ideas developed in *PSoup* and *CACQ*, and makes some improvements like processing out-of-memory data, supporting scheduling and resource management for groups of queries, using the Qos to adapt the

query engine to resource limitations, etc. All these features make the *TelegraphCQ* more like a fully functioned data stream processing system.

6. The *Aurora* project builds a data processing system for the purpose of serving stream monitoring applications. [8] It supports continuous queries, views, and ad hoc queries over data streams using the same mechanisms. The *Aurora* system consists of a large network of triggers, which could be created and controlled by the *administrator*. Each trigger is in fact a data flow across various self-defined operators. The processing of data flows and the query optimization of *Aurora* are all based on Qos specifications. Aurora uses a network of operators to share the computation of different standing queries, which is similar to that in NiagaraCQ [20].

7. STREAM is a DSMS dealing with the issues of memory requirements for executing various types of queries by considering the cost of self-maintenance of different materialized views. It also considers resource management issues including approximate query answering in the face of limited resources. [9] Besides, some solutions to general DSMS problems, such as continuous query semantics, translating declarative queries into physical query plans, query optimization, load shedding, are proposed. [14]

Recently, the focus of DSMS research has been shifted and extended from a single-site basis to a distributed environment, which makes the results of previous research more applicable for data stream processing in real-world applications.

8. For example, *Medusa* extends the single site processing of data streaming to a distributed environment. [13]
9. *Borealis,* which is based on the streaming processing of *Aurora* and the distribution logic of *Medusa,* is said to be a “second-generation SPE”. It inherits *Aurora’*s Qos-based techniques for query optimization, and provides further functionalities like dynamic query results revision, dynamic query modifications, flexible and highly-scalable optimization, for certain application domains. [11]

10. Another project called *HiFi* intends to build a prototype for general high fan-in system also in a distributed environment, which are “widely distributed systems whose edges consist of numerous receptors like sensor networks, RFID readers, or probes, and whose interior nodes are traditional host computers organized using the principles of cascading streams and successive aggregation”. [15] The researchers of *HiFi* projects look at streaming database from the perspective of information systems, and present an architecture for database design and processing. The prototype system is based on previous work in federated data management, data stream processing, sensor network query processing, and distributed data management. For example, the initial design of the *HiFi* system was implemented using *TelegraphCQ* and *TinyDB* [16] code bases. The *HiFi* system differs with the projects like *Medusa* and *Borealis* in that it focuses on tackling problems that arise in a system with a high fan-in topology, while the latter two aim to distribute stream processing for load balancing and high availability.

Among all these research projects on DSMS, projects 1, 2, 3, and 4 could be considered as continuous query engines, while 5, 6, and 7 can be regarded as general DSMS except that 6’s (*Aurora* project) primary focus is monitoring applications. Projects 8, 9, and 10 extend the previous CQ engines or DSMS systems to general real-world environments.

In [7], a set of requirements for DSMS are summarized, including designing data models and query semantics which allow order-based and time-based operations, query optimization and execution, load shedding, and etc. We are going to investigate how the aforementioned projects address these problems in following sub-sections.

2.3 Data Models and Query Languages

Stream data are infinite sequences of data, which could only be seen once when received. Thus, the design of data models and query languages should be able to represent these characteristics and execute query over such kind of data.

Usually data stream queries are executed on a finite set of data although the data sequences are unbounded. In order to extract a limited set of data items from data streams, the *window* concept needs to be defined.
According to the variability of both endpoints of a window, we can identify 3 types of *windows*: fixed window with 2 fixed endpoints, sliding window with 2 variable endpoints, and landmark window with 1 fixed endpoint and 1 moving endpoint. Besides, a *window* could be defined by either the time interval (time-based) or the number of tuples (count-based or tuple-based).

At a tiny time point, individual data item in a stream could be modeled as a relational database tuple, or an instantiation of an object. In *Aurora* and *STREAM*, single data items both take the form of relational tuples, while in *Cougar*, a system with the focus on dealing with sensor data [22], item types are modeled as data types with associated methods.

Query languages can also be categorized according to whether it's relation-based or not.
An example of relation-based language is CQL [23], Continuous Query Language used in STREAM project. Besides some relational operators like *select, join, union*, CQL also defines operators to convert streams to relations and vice versa, such as *seq-window, i-stream, d-stream, and r-stream*. Here's an example of windowed binary join of stream S1 and S2:

*Select * From S1 [Rows 1000], S2 [Range 2 Minutes] Where S1.A=S2.A And S1.A>10*

It can be seen that the *join* operation looks very similar to what's in SQL.

As an object-based language, *Cougar*'s query is a little different from what's in CQL. For instance, we issue the following query to get all temperature data on 3rd floor on a minute-basis. [22]

Select R.s.getTemp() From R Where R.floor=3 And $every(60)

In this query, *R* is a relation which has the schema *R(location point, floor int, s sensorNode),* and s is the sensor Node, which is modeled as an ADT that supports methods like getTemp().

Besides the above 2 types, there's another type for data stream query languages called procedural language. [7] Procedural language does not resemble SQL any more (not

like CQL or any object-based stream query language). It needs the user to specify how the data should flow through various defined operators, which makes it more like low-level query language. The *Aurora* system adopts such kind of query language. [8] *Aurora* defines 7 operators including *Filter, Map, Union, BSort, Aggregate, Join, and Resample*. A typical query in the *Aurora* system would look like:

Join (P, Size s, Assuming Left O, Assuming Right O) (S1, S2)

In the above query, *P* is a predicate over which the join operation is executed, *s* is an integer which represents the size of a window, and *O* is the specifications about the ordering of S1 ans S2 tuples in the query results.

2.4 Query Optimization

According to optimization objectives, query optimization techniques can be placed into 3 categories: stream rate-based optimization, resource-based optimization, and Qos-based optimization. [10]

1. Rate-based optimization approach is proposed with the objective of maximizing the output rate of query evaluation plans. [25] The fundamental statistics used in this approach are estimates of the rates of the streams in the query evaluation tree, such as output rate, operator selectivity. Output rate expressions of each operator can be represented and furthermore the expression of the query plan can be derived, which is a function of time and can be used to choose a plan with the greatest output rate.
Aurora uses the similar technique when interchanging 2 commutative operators. By iteratively reordering operator boxes, the query is optimized. [8]

2. The goal of resource-based approach is to optimize memory consumption. In a streaming processing engine *(SPE),* normally operators in a query plan will keep a temporal storage in memory for data stream that passes by. Thus a scheduling policy is required for efficient memory utilization, especially in the case when data streams flow into the SPE in a high speed.
The *STREAM* project [14] presents 2 scheduling strategies. One is *FIFO scheduling* which processes data stream through operators according to the arrival order. The other is g*reedy scheduling,* which aims to achieve the greatest reduction rate in total queue size. In *STREAM's chain scheduling algorithm*, these 2 strategies are applied to intra-chain scheduling and inter-chain scheduling respectively.

3. The last type of optimization is based on performance and Qos requirements, which is also an important feature of the *Aurora* system. *Aurora* not only aims to maximize overall Qos but also intends to reduce overall execution costs. It employs *train scheduling* and *superbox scheduling* for single box and multi-box tuple processing. [8]

Besides these 3 kinds of optimization techniques, query adaptivity is emphasized by many DSMS researchers. Adaptive query processing means efficient query execution in unpredictable and dynamic environments.
The *eddy* system [12] presents a novel strategy to achieve query adaptivity, which is also adopted by some following researches. An *eddy* is actually an n-ary tuple router interposed between n data sources and a set of query processing operators, which are dynamically pipelined in a query plan. During query execution, the *eddy* keeps routing every tuple, which is associated with its processing state, to operator threads for partial

processing until the tuple has been handled by all the operators. The order for routing tuples is adaptively changed by *eddy* based on its continuous observation of tuples entering and exiting the operators. By separately routing each tuple, the query plan is dynamically revised according to system conditions, and tuples are delivered to those operators that are fast and selective. However, there's a trade-off between adaptivity and the overhead during executing single-tuple routing.
Adaptivity in *STREAM* is accomplished by a monitoring infrastructure called *StreaMon*. [26]

Even query optimization is going well in a DSMS, load shedding mechanism is still necessary, as overload is always possible in a dynamic streaming environment. In *Aurora,* load shedding is done in two ways: by dropping tuples or by filtering tuples. [8] Besides, there are some other issues related to query optimization and execution. For instance, overall system performance can be improved if many queries can share some intermittent results, which is studied in *STREAM* and *Aurora*. *Medusa* [13] and *Borealis* [11] extends query optimization to a distributed environment, and some interesting problems like latency and fault tolerance are also addressed. OSCAR, a subproject of TelegraphCQ, aims at efficiently dealing with hybrid queries – queries executed on both historical data and data streams. [17]

2.5 Other issues

We summarized the data model, query language, and query optimization issues (including adaptivity and load shedding) in previous subsections. Besides these topics that have been extensively studied, there are also some other issues related to DSMS research.

1. Load shedding brings the problem of inaccuracy of data. Approximate query answering techniques, such as histogram-based approach, and building summary for data streams, could help to handle this issue. CPU-limited approximation and memory-limited approximation are provided in *STREAM*. [14]

2. Data storage is also an important part of a DSMS. In projects like *Aurora* [8], and *OSCAR* of *TelegraphCQ* [17], data storage management strategies have been proposed.

3. For data streaming systems, especially those using RFID as the data source, how to clean the received data could be a problem that needs to be solved. The *Extensible receptor Stream Processing (ESP)* framework is presented as a part of *HiFi* project to deal with this issue. [24]

References:
[1] Lara Srivastava, Ubiquitous Network Societies: The Case of Radio Frequency Identification. *ITU Workshop on Ubiquitous Network Societies*, April 2005.
[2] James Brusey, Christian Floerkemeier, Mark Harrison, and Martyn Fletcher. Reasoning about Uncertainty in Location Identification with RFID. *Workshop on Reasoning with Uncertainty in Robotics at IJCAI,* Acapulco, Mexico, August 2003.
[3] Jeremy Landt, Shrouds of Time The History of RFID. *AIM Publications,* 2001.

[4] Christian Floerkemeier, and Matthias Lampe, Issues with RFID Usage in Ubiquitous Computing Applications, *Second International Conference on Pervasive Computing,* Linz/ Vienna, Austruia, April, 2004.
[5] Sudarshan S. Chawathe, Venkat Krishnamurthy, Sridhar Ramachandran, and Sanjay Sarma. Managing RFID Data. *Proceedings of the 30th VLDB Conference,* Toronto, Canada, 2004.
[6] SAMSys, http://www.samsys.com/default.php?alpha=company&beta=tech .
[7] Lukasz Golab, and M. Tamer Ozsu, Data Stream Management Issues – A Survey, *Technical Report CS-2003-08,* University of Waterloo, Canada, April 2003.
[8] Daniel J. Abadi, Don Carney, Ugur Cetintemel, and etc. Aurora: A New Model and Architecture for Data Stream Management, *The VLDB Journal*, 2003.
[9] Brian Babcock, Shivnath Babu, Mayur Datar, and ean adaptive repartitioning operator continuous query systemstc., Models and Issues in Data Stream Systems. *Proceedings of 21st ACM Symposium on Principles of Database Systems,* March 2002.
[10] Nick Koudas, and Divesh Srivastava, Data Stream Query Processing, *AT&T Labs-Research,* May 2005.
[11] Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, and etc., The Design of the Borealis Stream Processing Engine. *Proceedings of the 2005 CIDR Conference,* January 2005.
[12] Ron Avnur, Joseph M. Hellerstein, Eddies: Continuously Adaptive Query Processing, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data,* Dallas, 2000.
[13] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, and etc., Scalable Distributed Stream Processing, *CIDR 2003, First Biennial Conference on Innovative Data Systems Research,* Asilomar, California, January 2003.
[14] Arvind Arasu, Brian Babcock, Shivnath Babu, and etc., STREAM: The Standford Data Stream Management System. *Overview paper in STREAM Group,* 2004.
[15] Michael J. Franklin, Shawn R. Jeffery, Sailesh Krishnamurthy, and etc., Design Considerations for High Fan-in Systems: The *HiFi* Approach, *Proceedings of the 2nd CIDR Conference,* Asilomar, California, 2005.
[16] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: An Acquisitional Query Processing System for Sensor Networks. *ACM Transactions on Database Systems,* vol. 30, issue. 1, pp 122-173, 2005.
[17] Sirish Chandrasekaran, and Michael Franklin, Remembrance of Streams Past: Overload-Sensitive Management of Archived Streams, *Proceedings of the 30th VLDB Conference,* Toronto, Canada 2004.
[18] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, and etc., TelegraphCQ: Continuous Dataflow Processing for An Uncertain World, *Proceedings of the 1st Biennial Conference on Innovative Data Systems Researches,* Asilomar, CA, January 2003.
[19] Sirish Chandrasekaran, and Michael J. Franklin, PSoup: A System for Streaming Queries over Streaming Data, *The VLDB Journal The International Journal on Very Large Databases,* vol12, no. 2, pp 140-156, August 2003.
[20] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang, NiagaraCQ: A Scalable Continous Query System for Internet Databases, *In Proceedings of the ACM SIGMOD Conference on Management of Data,* 2000.
[21] Samuel Madden, Mehul Shah, Joseph M. Hellerstein, Vijayshankar Raman, Continuously Adaptive Continuous Queries over Streams, *ACM SIGMOD,* Madison, Wisconsin, USA, June 2002.
[22] Philippe Bonnet, J. E. Gehrke, and Praveen Seshadri, Towards Sensor Database Systems, *In Proceedings of 2nd International Conference on Mobile Data Management,* pp 3-14, 2001.

[23] Arvind Arasu, Shivnath Babu, and Jennifer Widom, An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations, *IEEE Data Engineering Bulletin,* vol. 26, no. 1, 2003.
[24] Shawn R. Jeffery, Gustavo Alonso, Michael J. Franklin, and etc., A Pipelined Framework for Online Cleaning of Sensor Data Streams, *22nd International Conference on Data Engineering,* Atlanta, Georgia, USA, 2006.
[25] Stratis D. Viglas, and Jeffery F. Naughton, Rate-Based Query Optimization for Streaming Information Sources, *ACM SIGMOD Conference,* Madison, Wisconsin, USA, June 2002.
[26] Shivnath Babu, and Jennifer Widom, StreaMon: An Adaptive Engine for Stream Query Processing. *In Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, June 2004*.