논문 2008-45CI-3-14

RFID 미들웨어에서 질의 처리를 위한 리더 단계 여과

(Reader Level Filtering for Query Processing in an RFID Middleware)

카빌무하마드아샤드*, 류 우 석*, 홍 봉 희**

(Muhammad Ashad Kabir, Wooseok Ryu, and Bonghee Hong)

요 약

RFID 환경에서 미들웨어는 응용프로그램의 질의에 따라 리더로부터 끊임없이 들어오는 태그 데이터 스트림을 여과하고 수집하는 역할을 수행한다. 이때, 태그 데이터가 많을수록 미들웨어의 부하가 증가한다. 따라서 본 논문에서는 미들웨어의 부하를 줄이기 위해 리더 단계 여과 기법을 제안한다. 먼저 리더에서의 여과기능을 분석하고 응용프로그램에서 전달된 질의를 미들웨어에서 처리할 질의와 리더에서 처리할 질의로 분류한다. 또한 응용에서 전달되는 질의들을 분석하여 중복된 질의들을 단일 질의로 변환하여 리더로 전달되는 질의의 수를 감소시킴으로서, 리더로부터 전송되는 태그 데이터 수를 최소화한다. 본 논문에서 제안한 질의 계획을 토대로 RFID 미들웨어를 설계 구현하였으며, 제안한 질의 계획이 미들웨어에서의 처리 시간 및 리더에서 미들웨어로 전송되는 데이터 트래픽을 감소시키는 것을 실험을 통해 입증하였다.

Abstract

In RFID system, Middleware collects and filters streaming data gathered continuously from readers to process applications requests. The enormous amount of data makes middleware in highly overloaded. Hence, we propose reader level filtering in order to reduce overall middleware load. In this paper, we consider reader filtering capability and define query plan to minimize number of queries for processing into middleware and reader level. We design and implement middleware system based on proposed query plan. We perform several experiments on implemented system. Our experiments show that the proposed query plan considerably improves the performance of middleware by diminishing processing time and network traffic between reader and middleware.

Keywords : Reader Level Filtering, Query Processing, User Memory, RFID Tag, Middleware

Ⅰ. Introduction

Radio Frequency Identification (RFID) is a newly emerging wireless technology that uses radio waves to identify individual tagged objects without line of sight or contact between readers and tagged objects. RFID system has recently begun to find greater use in range of applications including industrial automation, supply chain management[1], aircraft maintenance, baggage handling, patient safety in hospital[2], monitoring in production management[3], foodstuffs traceability, etc. Nowadays tags contain additional memory (called "User memory") which used to store application specific information. Data stored on user memory can be define as non-EPC data, where Electronic Product Code (EPC) is an identification scheme for universally identifying physical objects, defined by EPCglobal[4]. According to survey by EPCglobal Tag Data Joint Research Group, 79% industries need to write data on user memory[4].

* 학생회원, ** 정회원, 부산대학교 컴퓨터공학과
(Department of Computer Engineering, Pusan National University)

※ "This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD)" (The Regional Research Universities Program/Research Center for Logistics Information Technology).
접수일자: 2008년4월26일, 수정완료일: 2008년5월6일

In RFID system, data are stream that are generated rapidly and automatically[5~6]. Middleware receives data from the readers connected to it. The data volume depends on the number of connected readers and the number of tags on the reader interrogation zone. Middleware should process the high volume data streams for real-time applications. Hence middleware is often in highly overloaded for enormous data.

To reduce middleware load we propose reader-level-filtering. Different makes and models of readers vary widely in the functionality they provide, from "dumb" readers that do little more than report what tags are currently within the reader's RF field, to "smart" readers that provide sophisticated filtering, smoothing, reporting, and other functionality. Reader Protocol (RP)[7] which is a standard interface between reader and middleware provides a uniform way for middleware to access and control the conforming readers manufactured by variety of vendors as well as functionality of reader level filtering.

One simple approach of reader level filtering is delegating all filtering conditions to reader. Obviously, it makes middleware an idle and reader in heavy load. Hence, we propose a smart query plan that will split filtering conditions to process in middleware and reader level.

In this paper, we point out the necessity of reader level filtering from the application and middleware perspective. After that we have analyzed data format and query patterns, and propose a smart query plan and overall middleware architecture with experimental evaluation.

The paper is organized as follows: In section Ⅱ, we discuss the related work. Section Ⅲ outlines the RFID system with related standards and defines the problem with significance of reader-level-filtering. In section Ⅳ, we have analyzed query, proposed query conversion policy and query extraction policy with example. We illustrate query processing module and present experimental results of performance evaluation in section Ⅴ. The paper concludes with a summary of contribution in section Ⅵ.

Ⅱ. Related Work

Several approaches to reduce middleware load have been proposed. Load balancing can be based on workload variation of RFID middlewares according to the location of connected readers[8]. The proposed approach considers workload variation of RFID middlewares according to the location of the connected readers. It focuses on job transfer policy, selection policy, and destination policy. In that approach a set of chosen readers should be disconnected from the source middleware and then reconnected to the target middleware during reader reallocation. It surely incurs the cost of disconnection and reconnection, and there is a possibility of losing tags information collected by the readers. Although it tries to minimize reader relocation overhead, it does not ensure the accuracy of tag information or recover of losing tag information. The problem of this approach is that it does not ensure reliability of tag information.

Connection pool based load balancing method for RFID middleware has been proposed in [9]. The Connection Pool distributes the tag data to connected several middleware by CPM (Connection Pool Manager). Tag data are distributed to middleware that has low load relatively among connected several middleware. Normally, the query define in middleware by application is continuous query, issued once and then logically run continuously over the RFID stream data. So, each query contains duration and repeat period. Moreover, it is defined for some specific readers. Hence, during the specific time duration, the middleware should collect tag data from the specific readers. The proposed method in that paper does not consider this constraint.

Middleware load can be reduced by delegating some filter conditions to reader. This concept is similar to query dissemination in sensor network[10], where the system disseminate query into the network. But there are some significant differences between sensor network and RFID system. In sensor network, communication is basically wireless, nodes are fixed or mobile with limited power capacity and each node is responsible to communicate with its child node, collect and aggregate data and send to its parent node. Unlike sensor node, RFID reader has no power limitation, and communication with middleware is generally wire communication.

III. Problem Definition

In this section, we describe the basic components of RFID system with related standards. Here, we also define the problem by analyzing two application scenarios.

1. Target Environment

RFID systems have been around for decades and comprise of three main components: the RFID tag, or transponder, which is located on the object to be identified and is the data carrier in the RFID system; the RFID reader, or transceiver, which may be able to both read data from and write data to a transponder; the data processing subsystem often referred to as RFID middleware that is application-agnostic, manages readers, filters and aggregates data obtained from the transceiver and delivers these to the appropriate consumers.

Efforts to standardize the various system components for RFID were initiated by the AutoID center, an industry sponsored research program. This work has matured with the creation of the EPC Network standard, was designed to enable all objects in the world to be linked via the internet, administered by the EPCglobal organization[4]. The standards specify not only the tags and readers, but define methods, interface and protocols for data processing and connectivity to IT infrastructure. Figure 1 shows the basic components of RFID system with related standards.

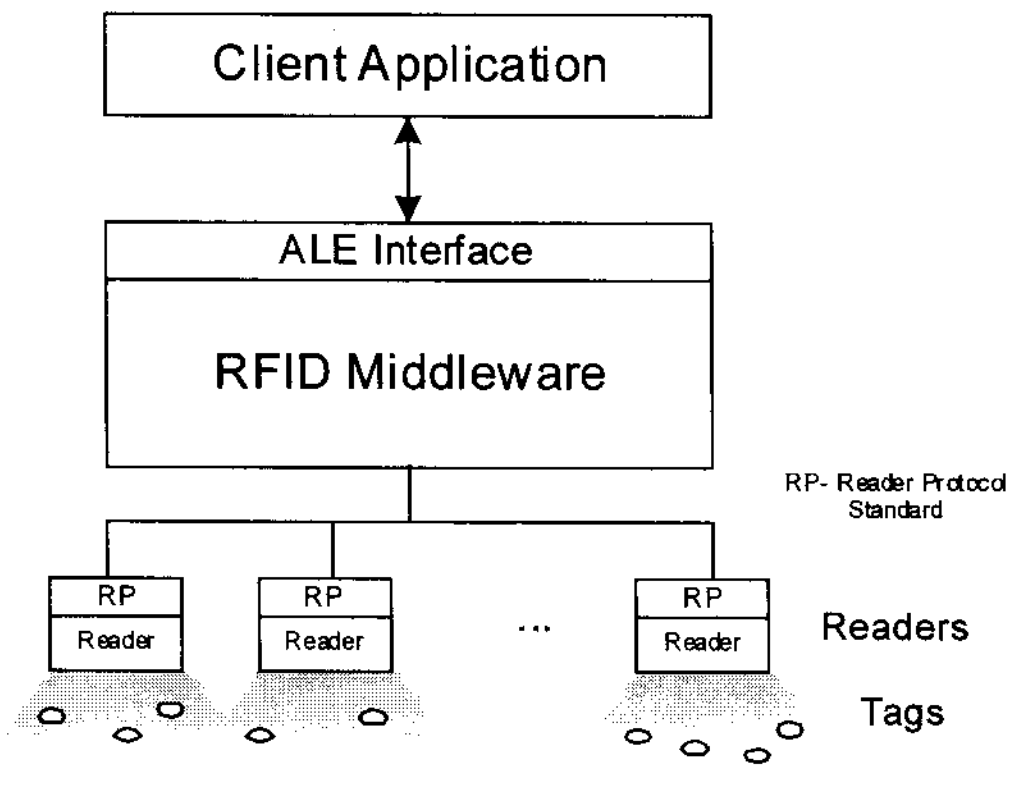


그림 1. RFID 시스템 개요
Fig. 1. Overview of RFID System.

Middleware Standard: The EPC Network Architecture defines Application Level Events (ALE) [11] as a standard interface through which clients may obtain filtered and consolidated data from various sources. The processing done at ALE typically involves: (1) receiving data from one or more sources such as readers; (2) accumulating data over intervals of time, filtering to eliminate duplicate data and that are not of interest, and counting and grouping data to reduce the volume; and (3) reporting in various forms.

Middleware to Reader Interface Standard: Reader Protocol (RP) defines standard interface by which readers interact with EPCglobal compliant software such as middleware. A goal of the RP is to insulate the upper layer (middleware) from knowing the details of how reader and tags interact.

2. Problem Definition

In RFID environment, applications want some specific data from tag memory which fulfill filter conditions defined by them. Checking filter conditions can be treat as query processing. To process query over user memory, we consider reader-level-filtering and reader-specific-data collection.

Let us assume that a pallet contains products and is passing through two interrogation zones, for example, dock_door#1 and dock_door#2, as shown in figure 2, and application interest only data from dock_door#2. Hence, data collected from readers belonging to dock_door#1 are redundant. Reader-specific data collection can reduce those redundant data and finally diminish middleware load. Note that ALE defines the concept of logical reader (i.e. dock_door#1, dock_door#2), for hiding from clients the details of exactly what physical devices were used to

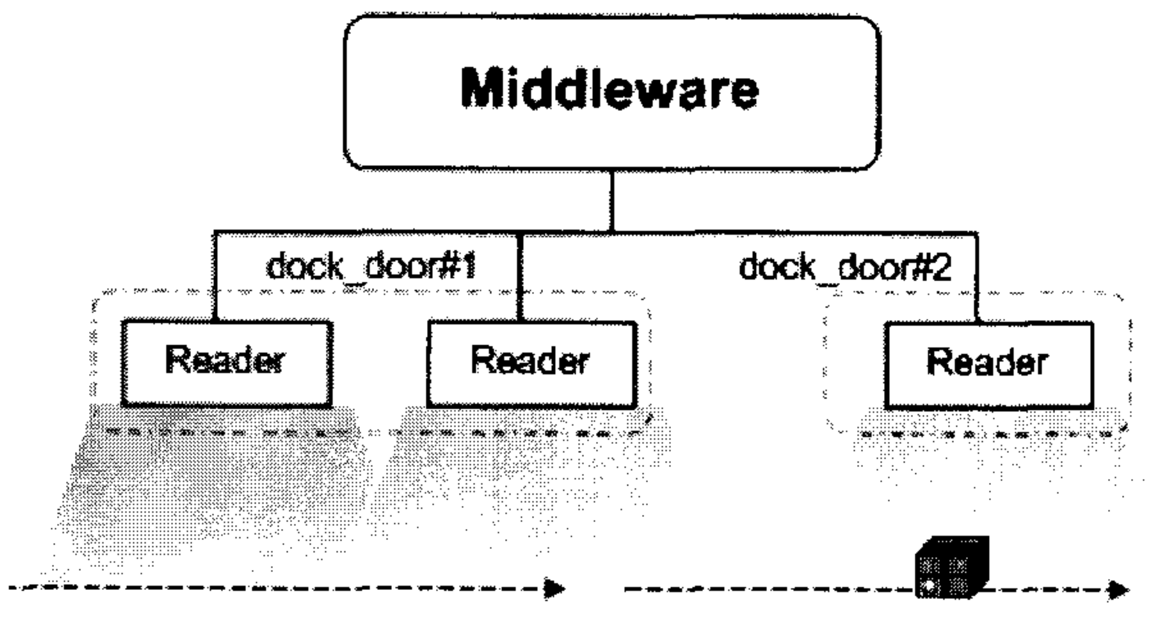


그림 2. RFID 리더를 통한 데이터 수집
Fig. 2. Reader-specific data collection.

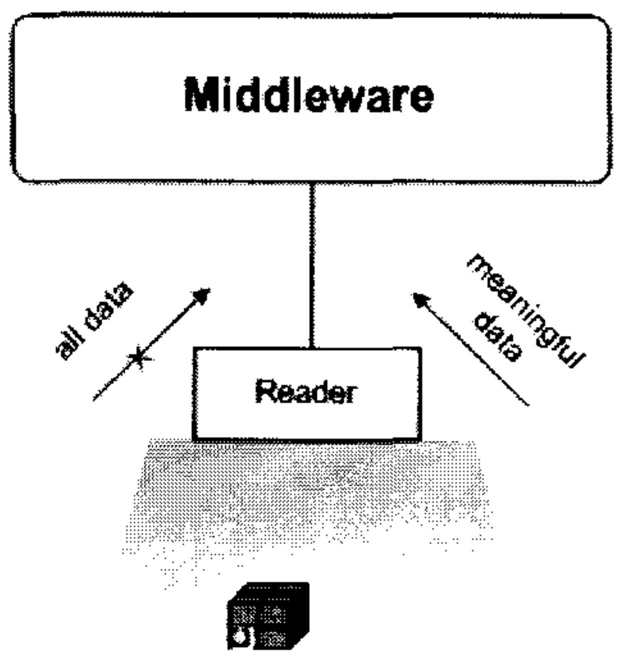


그림 3. 리더 단계 여과
Fig. 3. Reader-level filtering.

gather data relevant to a particular logical location.

Let us consider another scenario, a pallet contains several types of products and is passing through an interrogation zone, as shown in figure 3, and application needs some specific product information. In this case, data related to other products is not meaningful to application. If we use reader only for data collection purpose then middleware should process huge amount of redundant data which drastically affect middleware performance. To solve this problem we should consider reader filtering capability and delegate some filtering conditions for reader level processing.

Moreover, in a certain time, middleware may contain several queries from various clients. There is high possibility to exist some queries with overlap filter condition and common reader specification. If middleware delegate that condition separately to reader, after each read cycle middleware will receive huge amount of duplicate data. Also it requires more *TagSelectors* for reader, which will increase reader load and decrease performance. However, delegating all filtering conditions to reader makes middleware in idle and reader in heavy load. Hence, we need an query extraction policy (called query plan) which will select appropriate query patterns for reader level processing.

Ⅳ. Reader Level Filtering

1. Query Analysis

ALE proposed a standard query interface for RFID applications. The interface specifies *ECSpec* (Event Cycle Specification) as a query for middleware, which executes during a given time. It has predicates, composed of filtering conditions regarding readers and tags. The condition related to tags might be non-EPC patterns that represents set of ranges or ranges and fixed values.

Table 1 shows the data type, format, and corresponding filter patterns for non-EPC data according to ALE specification. *Hex* is a valid data format and *, *[lo-hi], fixValue*, and *&mask=value* are valid patterns for non-EPC data. If a pattern is single hex value ("*fixValue*"), the pattern matches a value equal to the pattern. If a pattern is in the form

표 1. 비 EPC 데이터에 대한 여과 스펙
Table 1. Filter specification for non-EPC data.

Data type	Format	Filter patterns
uint	hex	*, [lo-hi], fixValue, &mask=value
	decimal	*, [lo-hi], fixValue

```
<logicalReaders>
     <logicalReader> dock_door#1 </logicalReader>
</logicalReaders>
<filterSpec>
   <filter>
     <includeExclude>INCLUDE</includeExclude>
     <fieldspec><fieldname>afi</fieldname></fieldspec>
     <patList><pat>xC1</pat><pat>&xFO=x20</pat></patList>
   </filter>
   <filter>
     <includeExclude>INCLUDE</includeExclude>
<fieldspec><fieldname>countryOfOrigin</fieldname></fieldspec>
     <patList><pat>*</pat></patList>
   </filter>
   <filter>
     <includeExclude>EXCLUDE</includeExclude>
     <fieldspec><fieldname>lotCode</fieldname></fieldspec>
     <patList><pat>[x9-xB]</pat></patList>
   </filter>
</filterSpec>
```

그림 4. 비 EPC 데이터의 여과를 위한 ECSpec 예제
Fig. 4. Example of application level filtering conditions for non-EPC data.

[lo-hi], the pattern matches any value between *lo* and *hi*, inclusive. If a pattern is in the form *&mask=value*, the pattern matches any value that is equal to *value* after being bitwise and-ed with *mask*. Figure 4 shows an example of filtering conditions for non-EPC data.

An *ECSpec* comprise of *ECFilterSpec* and a list of logical reader names. *ECFilterSpec* is a set of filter condition. Each logical reader represents one or more physical reader. For reader-specific data collection we should delegate filtering conditions to some specific physical readers defined by *ECSpec*, not all physical readers connected to middleware. Each filter condition of *ECSpec* comprise of three parameters: *includeExclude*, *fieldSpec* and *patList* as shown in figure 4. The *fieldspec* specifies which field of the tag is considered in evaluating this filter, and the format for patterns in the *patList*.

Table 2 (called pattern combination table) shows the all possible combinations of patterns in a pattern list. The value of the *includeExclude* is INCLUDE or EXCLUDE. If value is INCLUDE, a tag is considered to pass the filter if the value in the specified field matches any of the patterns in *patList*. If this parameter is EXCLUDE, a tag is considered to pass the filter if the value in the specified field doesn't match any of the patterns in *patList*. The *patList* specifies the patterns against which the value of the specified tag field is to be compared. Each member of this list is a pattern value conforming to the format implied by *fieldSpec*. Example in figure 4 specifies

표 2. 가능한 패턴 조합의 목록
Table 2. All possible pattern combinations in a pattern list.

No.	Pattern Types			
	fixValue	*	&M=V	Range
1	O	X	X	O
2	O	X	O	O
3	X	X	X	O
4	X	X	O	O
5	O	X	X	X
6	O	X	O	X
7	X	X	O	X
8	X	O	X	X

filter conditions using three field-names. We assume that the field-size of *afi*, *countryOfOrigin* and *lotCode* are 8, 8 and 4 bits respectively and hex is their default data format.

The Reader-layer of RP describes the features of reader in terms of a conceptual pipeline processing, which comprise of four subsystems. Read-subsystem is one of them. It has read filtering stage, which maintains a list of filtering patterns configured by the middleware (defined in terms of *TagSelector* objects), and uses those patterns to filter out tags of no interest. The purpose of the filtering stage is to reduce the volume of data by only including tags of interest to the client.

A *TagSelector* encapsulates the logic in a reader that eliminates tags from being reported according to the conditions specified by the *TagSelector* parameters. This filtering logic is simple schema based on bit-wise patterns. A *TagSelector* is specified by using two hexadecimal strings, a filter value and a filter mask. In the filter mask M, all bit positions where the value is important for the filtering should be set to 1. A specific field of tag matches the filter if and only if the result of applying the filter mask on the filter value using a bit-wise AND operation is the same as when applying the filter mask on the tag field. This can define as follows:

```
IF ( ( V bitand M) == (F bitand M) ) THEN
      FieldMatchesTheFilter()
```

For example,

Filter mask $\boldsymbol{M}$ = $1C_H$ (00011100 in binary, meaning we are only interested in the bit values at positions 4, 5, and 6)
Filter value $\boldsymbol{V}$= 10_H (00010000 in binary. Because of the setting of M, only the values of bit positions 4-6 are important. Therefore in this case a filter value of "$F3_H$" / 11110011 would have the same result)
Actual tag field data $\boldsymbol{F}$ = 55_H (01010101 in binary)
In this case:
$\boldsymbol{V}$ bitand $\boldsymbol{M}$ = 10_H bitand $1C_H$ = 00010000 bitand 00011100 = 00010000 = 10_H
$\boldsymbol{F}$ bitand $\boldsymbol{M}$ = 55_H bitand $1C_H$ = 01010101 bitand 00011100 = 00010100 = 14_H
The two values are different, so there is no match.

In addition to this filtering definition, a *TagSelector* contains a *TagField* object. The *TagField* defines the tag data for which the *TagSelector* applies (i.e., the *TagField* specifies where to find the data field on the tag that should be processed by the filter). Multiple *TagSelector* objects can be associated with any given reader. Each of those filter objects is specified to be either inclusive (meaning that only tags matching the filter should be reported) or exclusive (meaning that tag should be only reported in it does not match the filter). In case of multiple *TagSelectors* are used, a tag should be reported if the following two conditions hold:

- The tag matches at least one of the inclusive patterns; and
- The tag does not match any of the exclusive patterns

As a special case, if zero inclusive patterns are defined, the first check should be omitted.

2. Query Conversion Policy

In previous section, we have described all possible patterns of non-EPC data for *hex* format with an example (figure 4) and also reader level filtering format (*TagSelector* object) according to RP standard. In this section, we describe how to map those patterns to reader level filtering format.

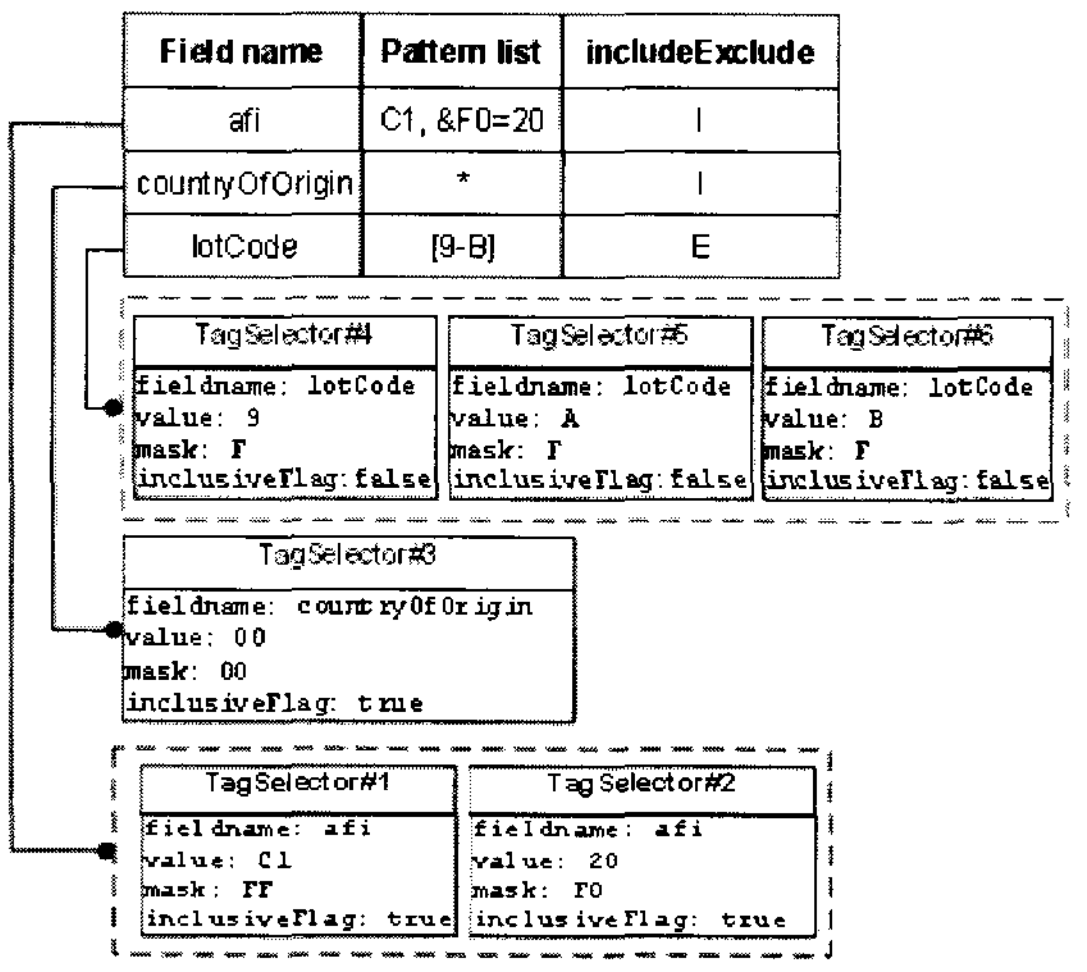
Reader level filtering format (*TagSelector*) consists of *Field-name, value, mask* and *inclusiveflag*. It requires one *TagSelector* for each pattern in pattern list in the form of *fixValue*, *, or *&M=V*. For *fixValue*, *value* field of *TagSelector* is *fixValue* and *mask* is sequence of 'F'. Here, number of 'F' equals ceil(field-size/4). For *, both *mask* and *value* are '0'. For *&M=V, mask* equals *M* and *value* equals *V*.

But to map *range* pattern, it requires more *TagSelector* object. Simple technique is to define one *TagSelector* for each value within range. Hence, no. of *TagSelector* equals (*hi-lo+1*) for range *[hi-lo]*.

Figure 5 shows an example of mapping application-level filtering conditions (figure 4) to reader-level filtering format. For example, one of the filter-patterns of *afi* field is $C1_H$. As we mentioned before, the *afi* field-size is 8 bits and data format is *hex*. So, the *mask* and *value* of *TagSelector* object should be FF_H and $C1_H$ respectively. In case of *fixValue*, * and *&mask=value* format, it is easy to map and need only one *TagSelector* for each pattern. But in case of *range* pattern, it is needed more than one *TagSelector* and the number of *TagSelector* depends on the range. If the field size is large and the corresponding range pattern cover large space then the number of *TagSelector* will increase dramatically. Also the large number of *TagSelector* should be burden for reader. Hence, we define query splitting policy to balance the load between reader and middleware.

3. Query Extraction Policy

For queries with overlap filter condition, we propose splitting and merging techniques. Each filter condition specifies either an inclusive or exclusive test based on the value of *includeExclude*. If the *includeExclude* parameter of a common field name is INCLUDE, merging technique computes the pattern list which cover all patterns specified by each member in the common field name list. Instead of delegate query conditions separately, middleware delegates one query condition with merging pattern list. In the same way, splitting technique computes intersection of all patterns specified by each member in the common field name list with *includeExclude* equals EXCLUDE. Instead of delegate query conditions separately, middleware delegates one query condition with splitting pattern list.



Field name	Pattern list	includeExclude
afi	C1, &F0=20	I
countryOfOrigin	*	I
lotCode	[9-B]	E

```
TagSelector#4                TagSelector#5                TagSelector#6
fieldname: lotCode           fieldname: lotCode           fieldname: lotCode
value: 9                     value: A                     value: B
mask: F                      mask: F                      mask: F
inclusiveFlag: false         inclusiveFlag: false         inclusiveFlag: false

TagSelector#3
fieldname: countryOfOrigin
value: 00
mask: 00
inclusiveFlag: true

TagSelector#1                TagSelector#2
fieldname: afi               fieldname: afi
value: C1                    value: 20
mask: FF                     mask: F0
inclusiveFlag: true          inclusiveFlag: true
```

그림 5. 미들웨어 질의에서 리더 질의로의 변환 예제
Fig. 5. Mapping example of filter condition (middleware to reader level).

```
Algorithm QuerySplitting (Query Q)
/* Split filter-condition for middleware and reader level processing */

Given: pattern-combination-table (Table 2)
Input: Q= {fc_1, fc_2, fc_3, ..} a set of filter-condition define by application

for each filter-condition f_c in Q
  if (includeExclude(f_c) = INCLUDE)
    if (pattern combination of pattern-list(f_c) is one of the
types 5-8 in pattern-combination-table)
      Consider f_c for reader level processing
    else
      Consider f_c for middleware level processing
  else if (includeExclude(f_c) = EXCLUDE)
    for each pattern p in pattern-list(f_c)
      if (pattern-type(p) is Range )
        Consider p for middleware level processing
      else
        Consider p for reader level processing
    end for
end for
```

그림 6. 질의 분할 알고리즘
Fig. 6. Splitting algorithm.

For queries with no overlap filter conditions, we propose query split algorithm as shown in figure 6. The key concept of the algorithm is as follows:

Query for tags can be classified into two types according to the value of query predicate *include-Exclude*. The possible value of *includeExclude* is INCLUDE or EXCLUDE. The significance of those values in query processing have already discussed in previous section.

For INCLUDE, to pass filter condition a tag should satisfy at least one of the filter patterns in pattern list. That is, there is logical OR relationships among patterns. Hence, in this case, it is not possible to split patterns and process in middleware and reader level. However, it requires more *TagSelector* for range pattern than others. Hence, our basic approach is that pattern list with absence of range pattern is processed by reader. So, pattern list with pattern combination no. 5 to 8 (Table 2) are processed by reader and middleware processes remaining combinations.

For EXCLUDE, to pass filtering condition a tag should satisfy all filter patterns in pattern list. That is, there is logical AND relationships among patterns. Hence, it is possible to split patterns and process in middleware & reader level. Our basic approach is that pattern in the form of *fixValue*, *, or *&M=V* are processed in reader level and Range pattern is processed in middleware level.

표 3. 질의 분할 예제
Table 3. Query Splitting example.

Query Predicates			Processing level
Field name	Pattern list	IncludeExclude	
afi	C1, &F0=20	I	Reader
countryOfOrigin	*	I	Reader
lotCode	[9-B]	E	Middleware

Table 3 illustrates the query plan according to splitting algorithm for example in figure 4.

V. Design and Implementation

1. System Design

This section presents an RFID middleware architecture that addresses query processing over user memory considering reader level filtering. Figure 7 shows the middleware architecture.

Our proposed middleware comprises of modules, where each module provides different functionality of middleware. *Event cycle controller* controls start and stop point of event cycle according to boundary condition specified by *ECSpec*. *Reports synthesizer* collects filtered data and makes *ECReports*. *Reader Manager* stores and manages physical reader, base reader & logical reader information. *Query Manager* stores user define query specification, separates query conditions and deliver each condition to related modules.

Query Extractor splits filtering conditions for reader-level and middleware-level execution. *Query*

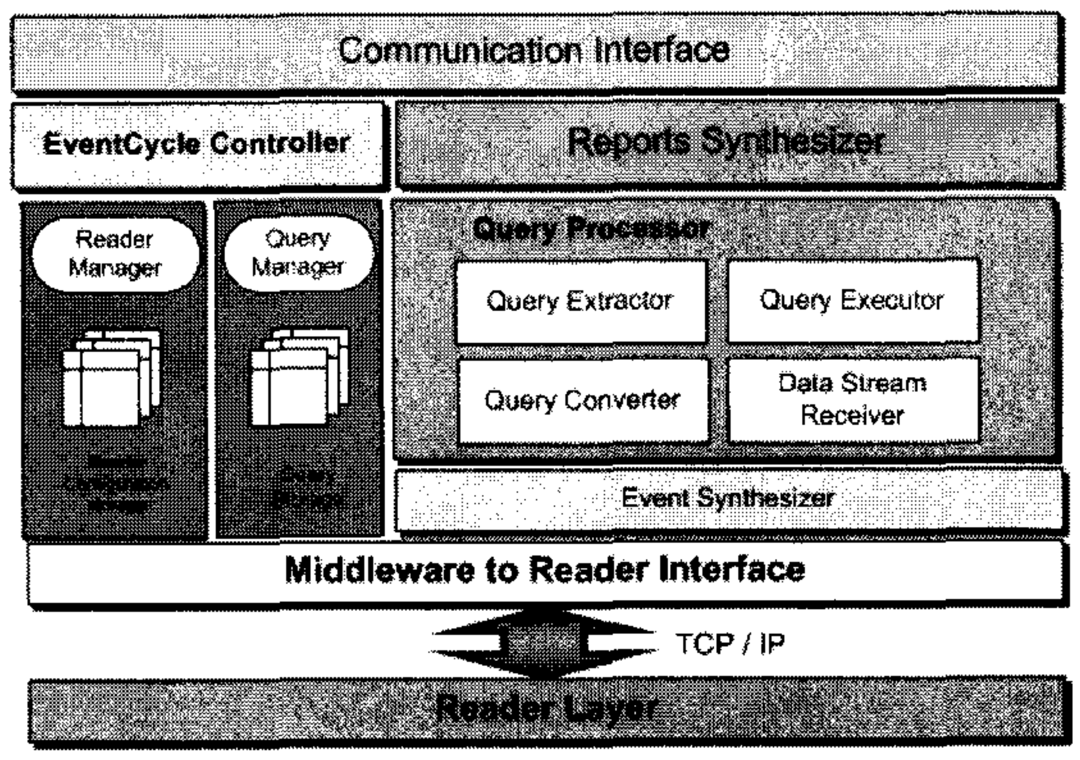


그림 7. RFID 미들웨어 아키텍처
Fig. 7. RFID Middleware Architecture.

Executor executes middleware-level filter-conditions and sends result to Reports Synthesizer. *Query Converter* converts middleware-level filter-condition to reader-level filter pattern. *Event Synthesizer* converts reader protocol based event to tag event. *Middleware to Reader Interface* communicates with reader, sends filter condition to reader, receives filtered tag events, and sends to appropriate module. *Data Stream Receiver* collects tag events, transforms to executable format and sends to *Query Executor*.

2. Experimental Evaluation

There are no well-known and widely accepted data sets for experimental purpose. Therefore, we carried out experiments using uniformly distributed data sets, generated by the Tag Data Generator (TDG). Tag data sets are randomly generated unbiased data in the space and consist of 10K entries. We generated queries using Query Generator (QG). The QG generates filter condition based on the data model of *ECSpec*.

To reflect the real RFID environment, the QG allows the user to configure its specific variables such as field-size, number of patterns in pattern list, number of filter conditions in an *ECSpec*, maximum number of readers in a reader specification. We have generated several unbiased QG filter-condition sets for experiments and consist of 100, 500, 1K, 5K, and 10K entries. All data are kept in the main memory to support real-time processing. The time we measured was the wall clock time. Our performance measurements were made on a standard personal computer with an Intel Pentium IV 2.6 GHz processor, 1 GB of main memory and the Microsoft Windows XP operating system.

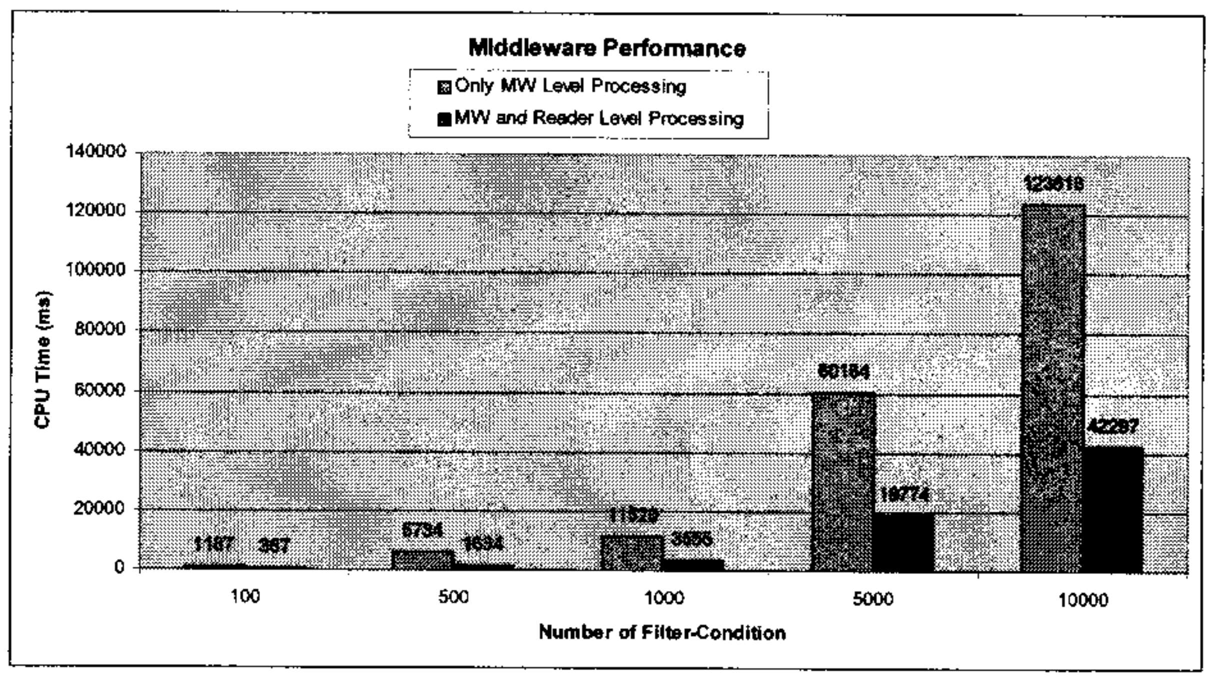


그림 8. 미들웨어의 수행시간 평가
Fig. 8. Middleware performance evaluation.

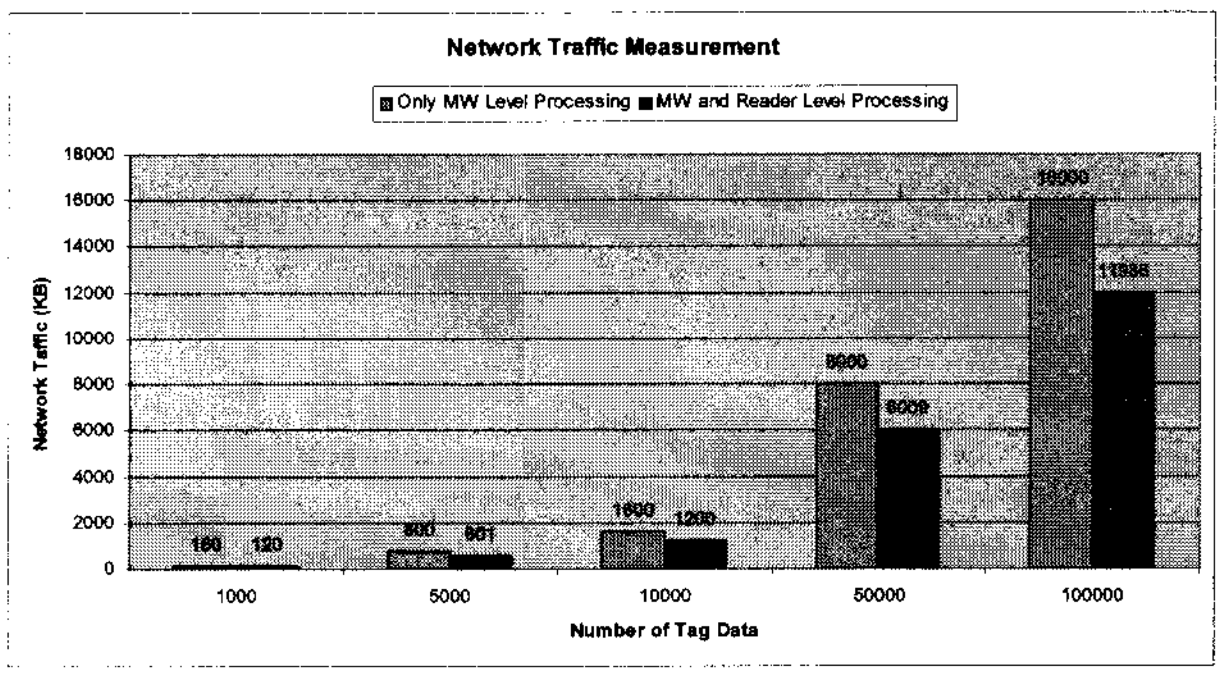


그림 9. 리더와 미들웨어간 데이터 통신량 비교
Fig. 9. Middleware to Reader network traffic.

The performance of middleware is measured by the CPU time required to process queries. Figure 8 shows the CPU time to process 100, 500, 1K, 5K, and 10K query sets for randomly generated 10K tags. Each query sets are processed over 5 times randomly generated 10K tags. We consider average of 5 processing times. Experimental result shows that for several filter-conditions Middleware level processing requires more CPU time than combine processing (Middleware and Reader level processing).

We also carried out experiments of middleware load. The middleware load is measured by the network traffic between middleware and readers. Figure 9 shows the network traffic results from 1K, 5K, 10K, 50K, and 100K tag data sets for randomly generated 100 filter-conditions. We assume that each tag memory size is 272 bits (e.g. Alien Tag) and reader sends 96 bits EPC and 64 bits user memory data, i.e. total 160 bits to middleware. The result of experiment shows that reader level filtering able to reduce network traffic compare to middleware level processing.

VI. Conclusion

In this paper, we introduce the concept of reader level filtering for reduce middleware load. Our approach of data filtering is essential to provide correct RFID data by considering middleware load and reader filtering capability. The approach that we formulate can minimize processing time of middleware and also middleware to reader network traffic.

We first analyzed the related standards to understand the key concept of query processing both in middleware and reader level. After that we propose query plan and the middleware architecture with role of each module. Then we perform experiments to validate our approach through simulated RFID data and query generator, and demonstrate that our approach is effective and efficient.

References

[1] R. Angles, "RFID Technologies: Supply-Chain Application and Implementation Issues," Information System Management, Vol. 22, no. 1, pp. 51-65, December 2005.

[2] M. T. Egan and W. S. Sandberg, "Auto Identification Technology and Its Impact on Patient Safety in the Operating Room of the Future," Surgical Innovation, Vol. 14, no. 1, pp. 41-50, March 2007.

[3] S. Zhou, W. Ling and Z. Peng, "An RFID-based remote monitoring system for enterprise internal production management," The International Journal of Advanced Manufacturing Technology, Vol. 33, no. 7-8, pp. 837-844, July 2007.

[4] EPCglobal Inc. http://www.epcglobalinc.org.

[5] Y. Bai, F. Wang and P. Liu, "Efficiently Filtering RFID Data Stream," in Proc. of First International VLDB Workshop on Clean Database, pp. 50-57, Seoul, Korea, September 2006.

[6] F. Wang and P. Liu, "Temporal Management of RFID Data," in Proc. of 31st International Conf. of Very Large Data Bases, pp. 1128-1139, Trondheim, Norway, September 2005.

[7] EPCglobal Inc. Reader Protocol (RP) Standard, version 1.1, Ratified Standard, June 21, 2006.

[8] H. S. Chae and J. Park, "An Approach to Adaptive Load Balancing for RFID Middlewares," International Journal of Applied Mathematics and Computer Sciences, Vol. 2, no. 2, pp. 76-80, 2006.

[9] S. M. Park, J. H. Song, C. S. Kim and J. J. Kim, "Load Balancing Method Using Connection Pool in RFID Middleware," in Proc. of 5th ACIS International Conf. on Software Engineering Research, Management and Applications, pp. 132-137, Busan, Korea, August 2007.

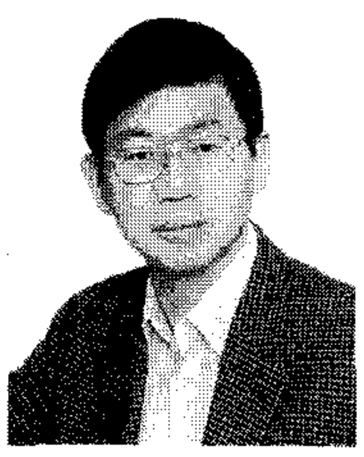
[10] J. Gehrke and S. Madden, "Query Processing in Sensor Networks," IEEE Pervasive Computing, Vol. 3, no. 1, pp. 46-55, March 2004.

[11] EPCglobal Inc. The Application Level Events (ALE) Specification, version 1.1, Ratified Specification, February 27 2008.

저 자 소 개

Muhammad Ashad Kabir(학생회원)
2004년 B. Sc. in Computer Science & Engineering, CUET (Bangladesh)
2006년~현재 부산대학교 대학원 컴퓨터공학과 석사과정
<주관심분야 : RFID Middleware, Reader Level Filtering, Query Processing in Middleware>

류 우 석(학생회원)
1997년 부산대학교 컴퓨터공학과 학사
1999년 부산대학교 대학원 컴퓨터공학과 석사
2002년~현재 부산대학교 대학원 컴퓨터공학과 박사과정
<주관심분야 : RFID 미들웨어, ALE, 태그 데이타 모델링, RFID 트랜잭션, RTLS 위치 측위>


홍 봉 희(정회원)-교신저자
1982년 서울대학교 전자계산기공학과 학사
1984년 서울대학교 대학원 전자계산기공학과 석사
1988년 서울대학교 대학원 전자계산기공학과 박사
1987년~현재 부산대학교 컴퓨터공학과 교수
<주관심분야 : RFID 미들웨어 데이터베이스, 실시간 위치정보 시스템, 유비쿼터스 미들웨어>