Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2013, Article ID 980962, 13 pages
http://dx.doi.org/10.1155/2013/980962

Research Article

An Approach to Design and Implement RFID Middleware System over Cloud Computing

Wenhong Tian, Ruini Xue, Xu Dong, and Haoyan Wang

School of Computer Science, University of Electronics and Science Technology of China, Chengdu 610054, China

Correspondence should be addressed to Wenhong Tian; tian_wenhong@uestc.edu.cn

Received 22 December 2012; Revised 11 April 2013; Accepted 20 May 2013

Academic Editor: Xinrong Li

With the development of Internet of things, the number of radio frequency identification (RFID) network readers and tags increases very fast. Large-scale application of RFID networks requires that RFID middleware system can process a large amount of data efficiently, with load-balancing, efficient redundant data elimination, and Web service capabilities so that required information can be transmitted to applications with low overhead and transparency. In view of these objectives and taking especially advantages of virtualization and transparency of Cloud computing, this paper introduces an advanced RFID middleware management system (ARMMS) over Cloud computing, which provides equipment management, RFID tag information management, application level event (ALE) management, and Web service APIs and related functions. ARMMS has different focuses than existing RFID middleware in distributed design, data filtering, integrated load balance, and Web service APIs and designs all these over Cloud. The distributed architecture can support large-scale applications, integrated load-balancing strategy guarantees stability and high performance of the system, and layered and parallel redundancy data elimination scheme makes sure that needed information is transmitted to application level with low overhead; Web service APIs support cross-platform information processing with transparency to lower level RFID hardware.

1. Introduction

Radio frequency identification (RFID) middleware plays an intermediary role between systems. Therefore, the upper-layer applications can add and delete contents or even be replaced by other software without the need to make any changes to the middleware; similarly, the underlying types of RFID readers can increase and decrease its various hardware and software operations; the upper layer does not need to make any changes. Therefore, RFID middleware can eliminate the need of many to many connections and reduces operating costs.

Figure 1 provides an overview of EPCglobal standards architecture, EPC information service (EPCIS), application level event (ALE), discovery configuration and installation (DCI), and reader management (RM) that are very much related in this paper.

As shown in Figure 2, the EPC system has several important components: readers, middleware, EPCIS, and ONS. The reader identifies and reads tags. The middleware processes all the information in tags and manages the information. The information after middleware processing is sent to the EPCIS. The EPCIS can share the information over the Internet. Finally, the object naming service (ONS) sends other tag requests to EPCIS [1]. The RFID middleware also has several fundamental functions, including data filtering, counting and aggregation of tag data, and handling the huge quantity of data generated by the RFID system [1, 2]. When facing a huge amount of data, load balance among different middleware (their hosts) is very important to keep high performance and reliability of the system [3]. Additionally, providing Web service is a way to support large-scale applications and a convenient way for management.

Figure 3 shows a possible organization of a physical RFID network.

When designing an RFID middleware solution, the following issues need to be considered [1, 2].

(a) Multiple types of hardware and vendors support: the middleware must provide a common interface to

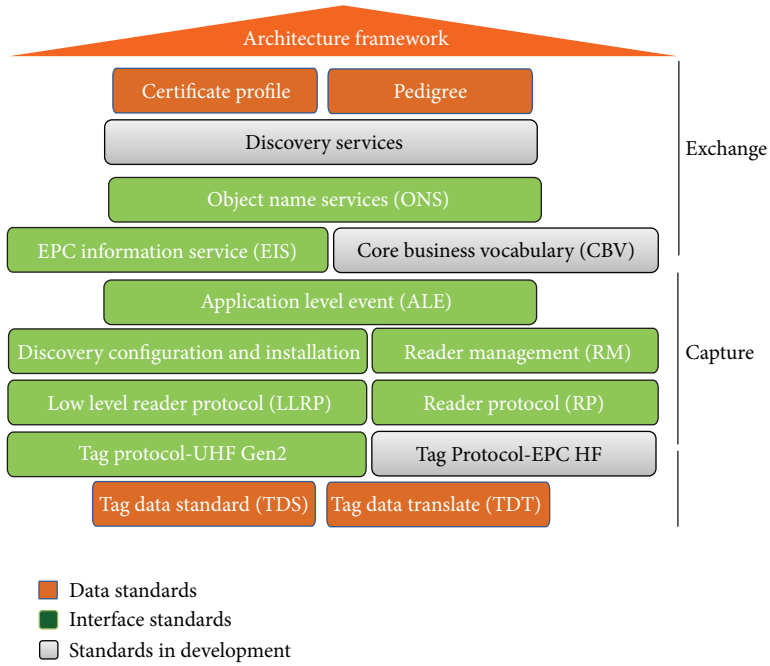

Figure 1: EPCglobal standards architecture [4].

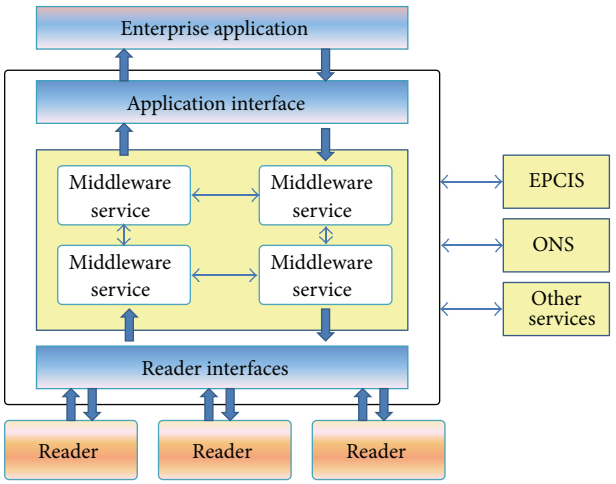

Figure 2: RFID middleware organization [1].

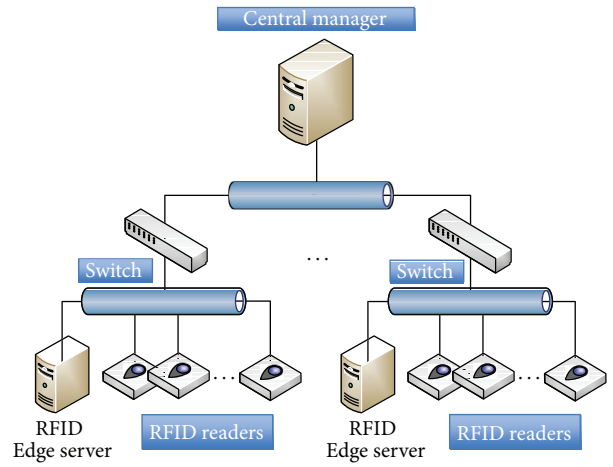

Figure 3: A networking example of RFID networks.

access different kinds of hardware by one or more vendors.

(b) Real-time handling of incoming data from the RFID readers: the middleware should handle the huge amount of data captured by the connected readers in real time without read misses.

(c) Interfacing with multiple applications: the middleware should be capable of interacting with multiple applications simultaneously, by catering to all the requirements of the applications with minimal latency.

(d) Device neutral interface to the applications: the application developer should only use the generic set of interfaces provided by the middleware independently of the type of hardware connected to the system.

(e) Scalability: the middleware design must allow easy integration of new hardware and data processing features. This needs distributed design and an open system to adapt to change.

(f) International standards compliant: the middleware design should follow international standards such as EPCglobal so that other related RFID hardware and software can also comply to the standards to make networking easier.

In view of these and taking especially advantages of virtualization and transparency of Cloud computing, this paper introduces an advanced RFID middleware management system (ARMMS) over Cloud computing, which provides equipment management, RFID tag information management, application level event (ALE) management, and Web service APIs and related functions. ARMMS has different focuses than existing RFID middleware in distributed design, data filtering, integrated load balance, and Web service APIs over Cloud.

Major contributions of this paper include

(1) proposing an RFID middleware architecture over Cloud computing including live migration;

(2) proposing a layered and parallel redundant data removal mechanism;

(3) introducing an integrated load-balance mechanism for RFID middleware;

(4) introducing Web service APIs to support cross-platform operation and information processing to improve transparency of RFID middleware to lower level hardware and software.

The rest of this work is organized as follows: Section 2 describes the related works on RFID networks and middleware design. Section 3 presents detailed features and implementation of our proposed RFID middleware system. Section 4 provides performance evaluation of proposed RFID middleware. Conclusions and future works are finally drawn in Section 5.

2. Related Research

In this section, we mainly introduce related researches on general introduction, related standards, redundancy removal, and load balance of RFID middleware design. There are several surveys in the literature [3, 5–8] that propose system taxonomies and major development in RFID middleware. General introduction of RFID technology and middleware is provided in [7–10]. Reference [5] presents a taxonomy for RFID system. There are many researches on standardization of the RFID middleware system. Massachusetts Institute of Technology (MIT) proposes EPCglobal standards. Many researchers have proposed various extensions based on the standards and implementation methods, such as in [11–13] which focus on an RFID middleware information addressing. Reference [14] is about Class 1 General 2 UHF air interface protocol. Reference [4, 15] introduces EPCglobal, EPCglobal ALE middleware design standards, ALE middleware, and messaging and device management standards.

There are many researches on RFID middleware design. Reference [16] studies RFID middleware for distributed large-scale systems. Reference [17] reviews the state-of-art RFID middleware. Reference [3] discusses the application requirements and RFID constraints for middleware. Reference [6] introduces the basics of RFID networks and middleware including Savant, WinRFID, IBM WebSphere RFID middleware, Sun Java RFID system, and FlexRFID with a focus on security, privacy, and business rules. Reference [18] presents a lightweight RFID middleware design (through temporary database implementation). Reference [19] introduces a middleware called WinRFID. Reference [20] shows the behavior and performance of message-oriented middleware system.

Some researches on redundant data removal are as follows. Reference [21] introduces a mechanism filtering redundant data in large-scale applications of RFID inventory. Reference [22] proposes an efficient filtering algorithm CLIF, for the detection and elimination of redundant data within a network. Reference [23] defines the EPC global standard of RFID tag data and proposed in-network phased filtering mechanisms (INPFM). Reference [24] proposes an adaptive RFID data-smoothing filter SMURF. Reference [25] introduces tuning approach in data filtering to reduce energy consumption in wireless sensor networks. Reference [26] presents an energy-efficient in-network RFID data filtering scheme in wireless sensor networks that has better performance regarding computational and communication costs than INPFM and CLIF. Tian et al. [27] present a parallel method to redundancy data elimination in RFID networks by applying MapReduce and Hadoop cluster in Cloud computing. INPFM and CLIF eliminate duplicated data during tags transmission phase. INPFM filters duplicated data at k hop distance. As pointed out in [26], several problems still exist by applying INPFM or CLIF. Problems include that computation cost is high, duplicate transmission cost is high, and inducing large delays when the total number of tags increase. EIFS [26] is claimed to have better performance than INPFM and CLIF regarding computational and communication cost; however, it can just eliminate about 80% of all redundancy, which may still be a problem for upper layer applications. Another issue for EIFS is that it just simulated a small number of tags (max 500 tags). In this paper, we propose a parallel method for redundancy elimination, which can remove 100% redundant data for large scale of data up to 100 million.

Many researches on load balance of RFID middleware have been conducted. Reference [28] introduces a number of RFID tag-based middleware load-balancing strategies which are good for all middleware and their hosting servers which have same configuration (homogenous case). Reference [29] introduces a distributed and agent-based design of load balance system for RFID middleware, which applies mobile and stationary agents to gather information and execute load balancing. Reference [1] proposes a grid-based load-balancing mechanism for RFID middleware applications, which incorporates functional modules buffer management and load balancing management over a grid networking platform, to buffer the read data and share the middleware loading, and compared the processing time and the packet lost ratio to the existing methods. Reference [30] introduces a centralized method for load-balancing method using connection pool in RFID middleware. Reference [31] introduces a dynamic load balancing approach based on the standard RFID middleware architecture by considering interdependencies due to RFID readers in contrast to most of the existing approaches where independencies are assumed among jobs. Reference [32] introduces a framework to

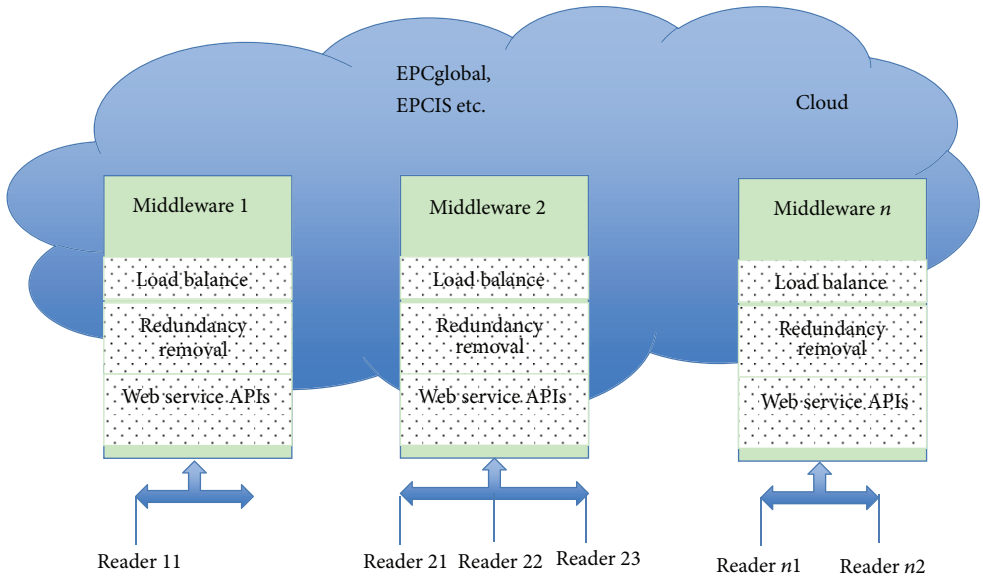

Figure 4: Proposed ARMMS middleware architecture.

dynamic and integrated load balancing for RFID middleware by considering heterogeneous configuration and multiple factors such as CPU and memory of middleware hosting servers.

3. Architecture and Features of Proposed RFID Middleware

Figure 4 shows proposed ARMMS middleware architecture based on Cloud computing. The ARMMS middleware system can be built based on Cloud computing. Especially for the load balance, hosting servers can be set up on virtual machines which can bring benefits to allocation, migration, and security; for redundancy removal, layered mechanism and parallel processing of MapReduce in Cloud computing can be applied; for Web service (APIs), dynamic web serving based on elastic Cloud computing can be applied. We will introduce these features in detail in the following section.

ARMMS has a very clear hierarchy from bottom to up as shown in Figure 5: Reader API Interface, Edge servers, RFID middleware manager/ALE server, middleware API, and RFID applications. In the following, these five layers are introduced briefly.

RFID Reader APIs. Different RFID readers should provide related APIs to middleware system to access so that functions such as equipment management, data filtering and aggregation, snd load-balance, can be conducted.

Edge Server. The main function of an Edge server is to run a single middleware software, directly manage readers by calling their APIs, acquire tag data sent by readers, and execute preliminary tag filtering algorithm to filter out redundant tag data generated by a reader.

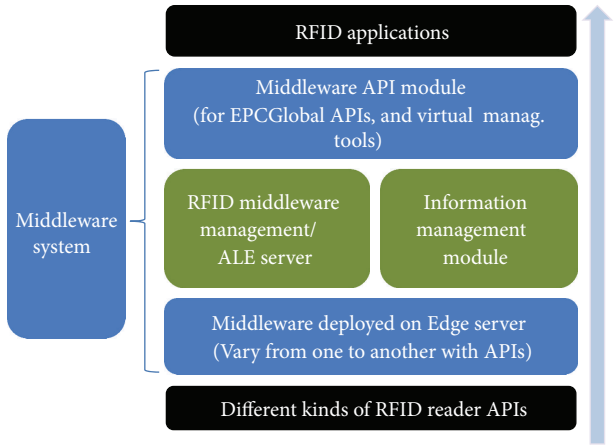

Figure 5: The structure of ARMMS.

Middleware Manager/ALE Server. Middleware manager/ALE server is primarily responsible for a single middleware software management on an Edge server according to the host load.

Information Management Module. Providing related distributed messaging queues for several applications to achieve asynchronous communication. The application may not be only on one machine but can be on different machines in a local area network. Figure 6 shows messaging mechanism in distributed ARMMS.

Middleware API Module. According to EPCglobal APIs, this module provides basic API operations including ALE events manipulation and information checking operations. Meanwhile, the visual management tools may also include a hierarchy of the visual management tools, mainly for

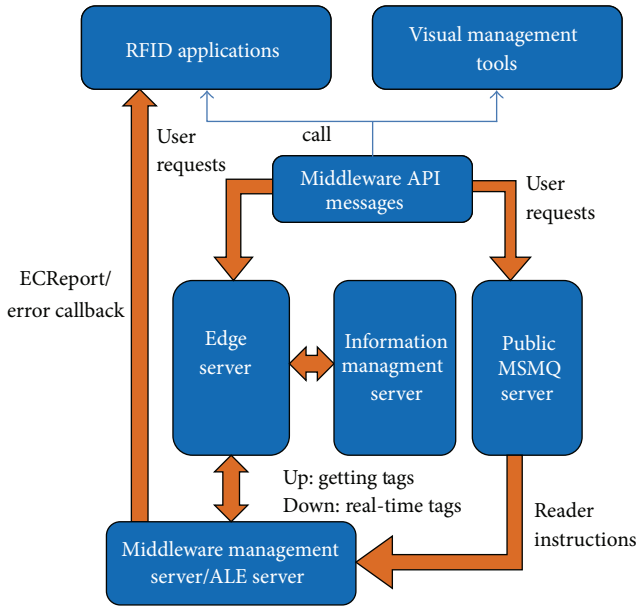

Figure 6: Messaging in distributed ARMMS.

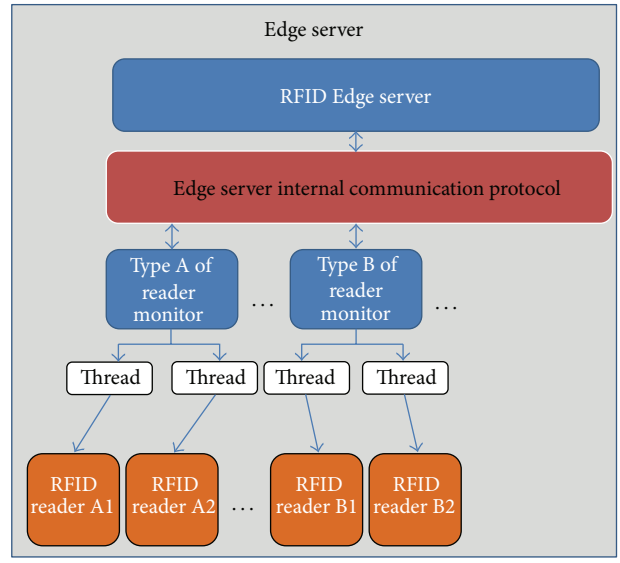

Figure 7: Structure of proposed Edge server.

providing more convenient modification and configuration of the middleware.

RFID Applications. Upper layer applications can interface with ARMMS middleware system to provide different service.

3.1. Edge Server Design. Edge server is made up of two types of backend applications, RFID Edge server and Monitor process. RFIDEdgeserver is responsible for providing message queues and collecting and distributing messages. An Edge server runs only one RFID Edge server process. Monitor backend process is the only object calls a specific RFID device API, one per type of RFID device. It uses an internal communication protocol to communicate with RFID Edge server. For a specific type of RFID device, a monitor process should be implemented. So, if we want to extend this middleware system to support a new type of RFID device, we should implement a new monitor process with the new device's API and internal communication protocol. Figure 7 shows the structure of proposed Edge server.

The main function of the monitor backend process includes the following.

(1) Calling the corresponding RFID devices' API and obtaining the tag data in the reading range.

(2) Monitoring the RFID devices connected with Edge server in real time and sending the error reports to Center Node (management center).

(3) Sending TagDetected message to the RFID Edge server process when a tag data is arrived.

RFID devices from different manufactures apply different API calling mechanisms. If middleware system calls the API directly, the scaling ability will undoubtedly become very poor. Applying support for new type of RFID device will lead to recompiling the source code. Despite operation mechanisms for reading and writing varying from one type of device to another, it is easy to find that they obtain tag data in a similar way. Edge server real-time tag algorithm is designed for the hardware abstract layer and implementing first level (reader level) redundant data elimination. We allocate a real-time tag cache for each RFID device object in the monitor thread, the real-time algorithm is responsible for assuring the tag data in the cache is in the device reading/writing range.

3.2. ALE Server Design. The requirements of central management node are in ECSpec and formatted as xml document. ALE server will transform the xml document to ECSpec object in memory when it gets the ALE requirement from central management node and initial an ECSpecUnit object with data in ECSpec to handle the requirement. The ECBoundarySpec part in ECSpec defines the ALE executing arguments, like recycle reading time, reading trigger, and so forth. The ECReportSpec part defines the report feeding back, like tag filtering pattern, grouping pattern, and so forth. So, the ECSpec is the only input in the ALE middleware system, it includes all the arguments, which are needed in ALE executing mechanism. There is a timer in the ECSpecUnit object. ALE mechanism is all depending on this clock. The ALE server flowchart is provided in Figure 8.

3.3. Central Management Node. The center node includes backend process and soap process. Soap process is responsible for communication with the RFID application who calling the middleware API. The relevant standard is "ale_1_1_1-standard-XML and SOAP bindings-20090313." Backend process is responsible for the load balance of the RFID device connection to Edge server. The Edge server load balance is shown in Figure 9 and will be explained in detail in Section 3.5.

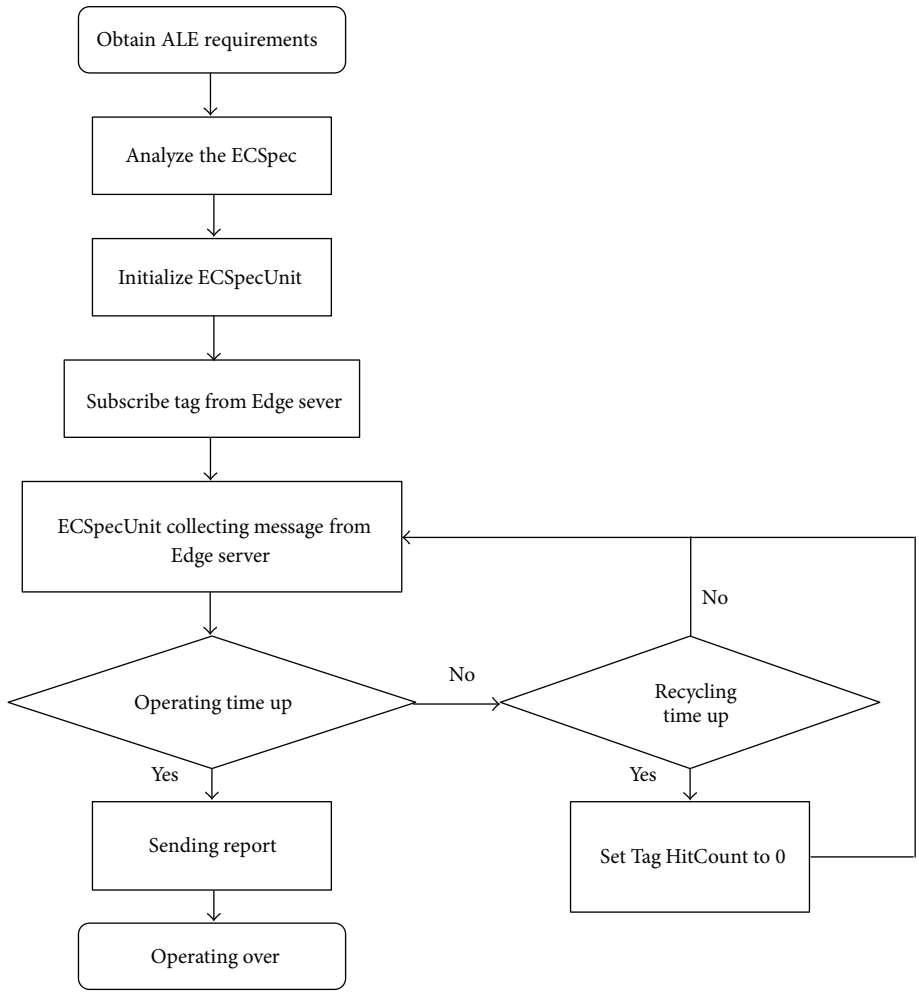

Figure 8: Flowchart of ALE server process executing.

3.4. Layered Redundant Elimination Mechanism. Figure 10 shows the data redundancy in RFID system, which can be divided into the following categories.

3.4.1. Redundancy within a Reader. When the reader gets an accurate reading of data to obtain object information on a tag, multiple times of same tag information can be read within a short period, so that redundant tags are generated.

3.4.2. Redundancy between Readers. Many readers are often densely installed to cover the entire region, and these readers' reading range may overlap with nearby readers (as shown in Figure 10); when two or more adjacent readers read the same tag information in the overlap region, redundant (repeating) tags information can be obtained.

3.4.3. Redundancy between Different Logical Groups. Different logical groups can be formed by functionality or location differences. When two or more adjacent logical groups read the same tag information in the overlap region, redundant (repeating) tags information can be obtained. Redundancy of a reader and between readers cannot be completely removed because of locality view, this is shown as the redundancy elimination percentage in [22, 23]. It needs a global elimination by collecting all tag data after reader layer and logical layer removal. To the best of our knowledge, there is no approach proposed using MapReduce [33] for redundancy elimination in RFID networks. This paper proposes a parallel MapReduce method to the redundancy data elimination. MapReduce is a programming model and can be applied to a cluster that consists of a large number of machines. Parallel MapReduce method can greatly increase the speed and efficiency of eliminating redundancy. Figure 11 presents our proposed MapReduce flowchart for redundancy elimination.

3.5. Dynamic and Integrated Load-Balancing Mechanism. Workloads of RFID middleware can change from location to location and can vary at different times. There is an urgent need to provide dynamic and integrated load-balancing solution to manage RFID network and serve upper layer applications. In our proposed model, the Edge server load-balancing module is responsible for monitoring the load on the Edge server and dynamic adjusting connections between Edge servers and RFID reader devices.

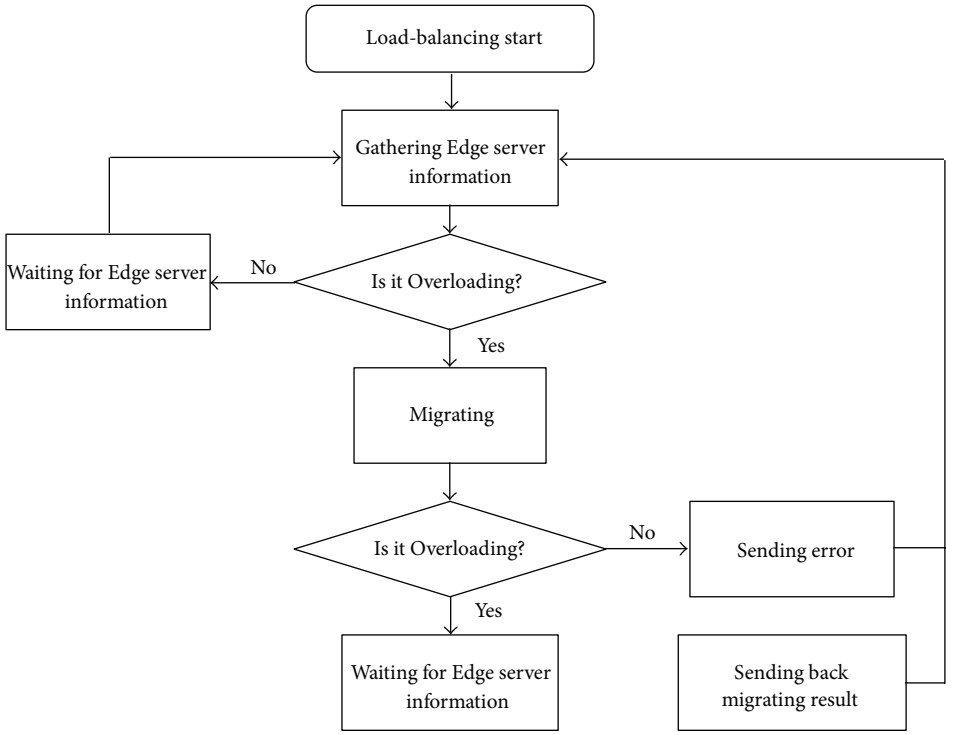

Figure 9: Edge server load balance mechanism.

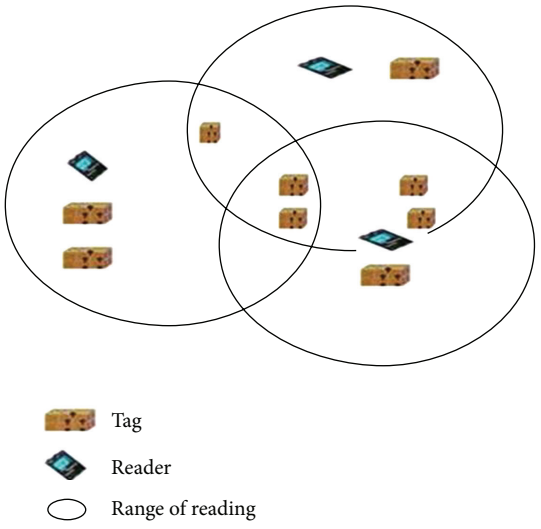

Figure 10: Data redundancy in RFID networks.

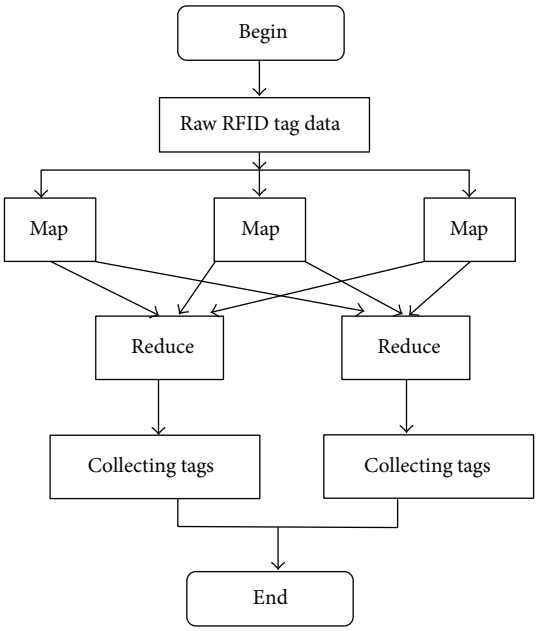

Figure 11: Proposed MapReduce flowchart.

Reference [28] defines an RFID network middleware system, reader collection, middleware collection, reader load, and middleware load as shown in Figure 12.

(a) $M = \{m_1, m_2, \ldots, m_n\}$, where M is a middleware collection and m_n represents middleware n; $CR_k = \{r_1^k, r_2^k, \ldots, r_p^k\}$ is a reader collection connected to the middleware k; one reader is connected to only one middleware during a period of time.

(b) $WL_R[r]$: the amount of tags handled by the reader, representing the load of a reader.

(c) $WL_M[m_i]$: the amount of tags handled by middleware m_i.

(d) $WL_M^U[m_i]$ and $WL_M^L[m_i]$ and the upper and lower load limits of middleware m_i, respectively.

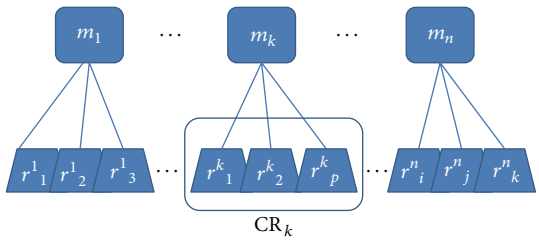

FIGURE 12: Definition of RFID middleware load [28].

Load-balancing strategy in [28] is to always choose the most overloaded middleware to migrate. During a certain period, relocation always migrates one or more readers on the most loaded middleware to least loaded middleware. This strategy is suitable for the situation that all middleware have the same configuration (homogenous case). This does not work for the case that middleware is not configured as the same. The configuration of middleware depends on CPU, memory, network bandwidth, and so forth, of their hosts. Therefore, we design a new strategy to consider the case that all middleware may not have the same configuration (heterogeneous case), with focus on dynamic and integrated load balancing.

(1) Average utilization of CPU and memory of each middleware:

$$\begin{aligned}
\text{CPU_}m_i &= \frac{WL_M[m_i]}{WL_M^U[m_i]} \times \text{CPU_}m_i^U + \text{CPU_}b,\\
\text{Mem_}m_i &= \frac{WL_M[m_i]}{WL_M^U[m_i]} \times \text{Mem_}m_i^U + \text{Mem_}b,\\
\text{AVG_}c &= \frac{\sum(\text{CPU_}m_i \times \text{SpeC_}m_i)}{\sum \text{SpeC_}m_i},\\
\text{AVG_}m &= \frac{\sum(\text{Mem_}m_i \times \text{SpeM_}m_i)}{\sum \text{SpeM_}m_i}.
\end{aligned} \tag{1}$$

(2) Integrated load imbalance level of middleware m_i is defined as

$$\begin{aligned}
&L_m[m_i]\\
&= \frac{1}{2}\left((\text{CPU_}m_i - \text{AVG_}c)^2 + (\text{Mem_}m_i - \text{AVG_}m)^2\right).
\end{aligned} \tag{2}$$

(3) Average imbalance level of host servers equals to the sum of imbalance levels of all servers divided by the number of hosting servers:

$$L\_M = \frac{1}{n}\sum L\_m[m_i]. \tag{3}$$

In (1)–(3), $\text{CPU\_}m_i$ is CPU utilization of host server m_i, $\text{Mem\_}m_i$ is memory utilization of host server m_i, AVG_c is average CPU utilization of all host servers, AVG_m is memory utilization of all host servers, $\text{SpeC\_}m_i$ is CPU configuration of host server m_i, for example, the number of CPU cores and CPU frequencies, and $\text{SpeM\_}m_i$ is memory configuration of host server m_i. $\text{CPU\_}m_i^U$ is CPU utilization of host server m_i with full load, and $\text{Mem\_}m_i^U$ is memory utilization of host server m_i with full load. CPU_b is CPU utilization of the middleware itself, and Mem_b is memory utilization of the middleware itself.

Integrated load balance considers factors including CPU utilization, memory utilization, and network bandwidth utilization, which can be expanded.

Allocation Policy. When there are new readers to add, it always chooses the middleware with lowest integrated load L_M to connect.

Reallocation (Migration) Policy. This mainly considers overloaded circumstance and chooses an overloaded middleware for migration. It needs to quantify how many readers to migration, at the same time to reduce the number of migrations to avoid system oscillation. Therefore, always migrate readers in the middleware with highest integrated load to the middleware with lowest integrated load until all load-balance thresholds (metrics) are met. In the migration, we take full advantages of live virtualization technology in Cloud computing. Migration of a virtual machine is simply moving the VM running on a physical machine (let us call it host Edge server) to another physical machine (let us call it target node). The key of live migration is that it does not disrupt any active network connections even after the VM is moved to the target node. Virtualization has other advantages such as running multiple applications on one Edge server, providing security and isolation to different applications, rapidly adjust resource allocation, and be transparent to applications without downtime. Figure 13 shows live migration of middleware by virtual machines.

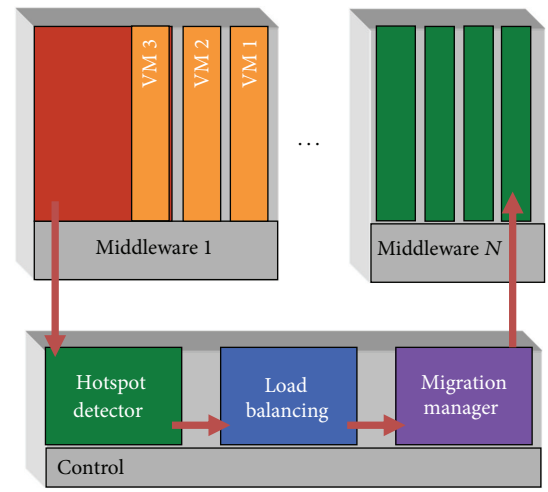

FIGURE 13: Live migration of middleware by virtual machines.

Through extensive numerical examples, we find that this dynamic and integrated load-balancing mechanism achieves

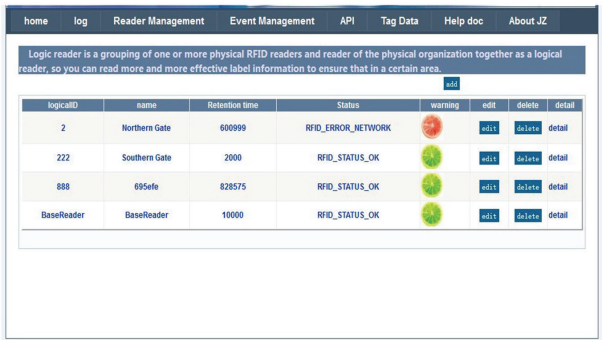

Figure 14: Logical view of RFID readers.

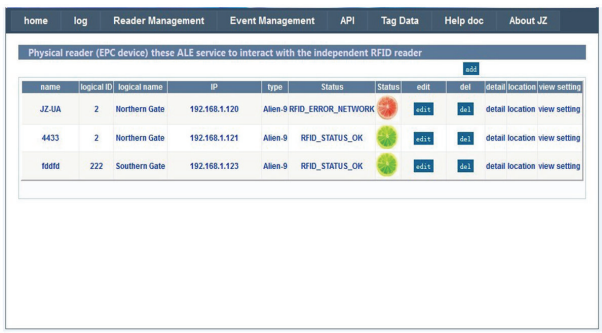

Figure 15: Physical RFID readers.

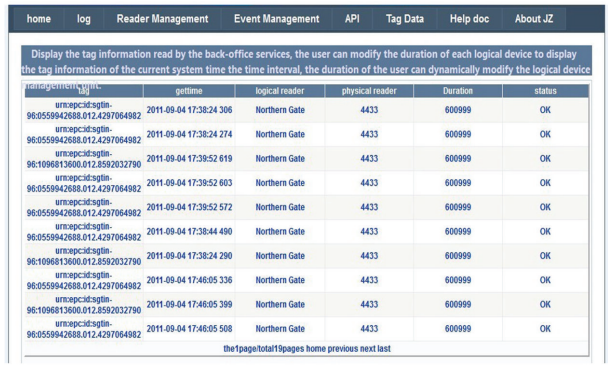

Figure 16: Tag data information.

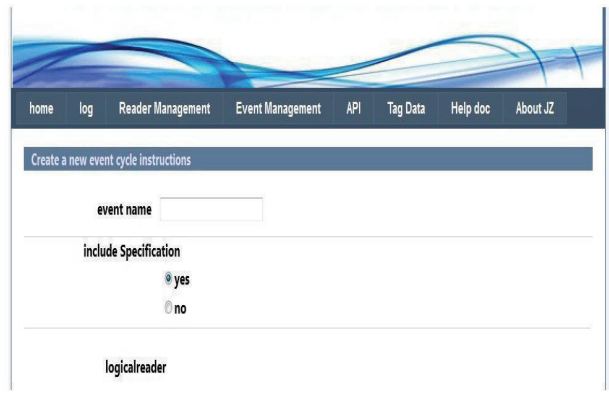

Figure 17: ALE management.

lower average and a total imbalance level than traditional load-balancing strategies such as Round-robin and the one introduced in [1, 28].

3.6. Web Service APIs. In order to provide convenient management and service, Web service APIs are necessary. It has an interface described in a machine-processing format (e.g., WSDL format). Other systems can interact with the Web service APIs in a manner prescribed by description using messages, conveyed using HTTP with an XML serialization.

4. Performance Evaluation

4.1. General Information and Web APIs. Figures 14 and 15 show the logical view and physical view of readers, respectively; Figure 16 presents tag data information management, Figure 17 for ALE management, and Figure 18 for Web service APIs.

4.2. Load Balance. For the performance evaluation, we have the following configuration: Edge server A has one reader with process rate of 1500 tags/second; Edge server B has two readers, both with process rate 2000 tags/second; Edge server C has three readers with process rate 5000 tags/second; Edge server D has one reader with process rate 6000 tags/second. Table 1 also provides the configuration of host Edge servers including CPU in Ghz, memory (MEM) in GB; the upper bound (UB) of utilization of CPU (UB) and MEM (UB); and $WL_M^U[m_i]$ for each host Edge servers. Table 2 is the CPU and MEM utilization information of four host Edge servers before load balancing. Table 3 provides CPU and MEM utilization results comparison between ARRMS and the method in [28]. It can be observed that ARRMS has better integrated load balance for both CPU and memory in host Edge servers since it considers CPU and memory integrated.

4.3. Redundancy Elimination and Others. Table 4 shows the data format of an RFID tag: Device ID (2 bits) represents the identification of the device, EPC code (28 bits) is for tag content, and Timestamp (14 bits) records date and time information of a tag.

Taking a tag "003000C2001602200001457FFC000420110627172133" as an example:

(i) bit 0-1-00 is the device number of a tag, which has length 2;

(ii) bit 2-29-3000C2001602200001457FFC0004 are tag's EPC code, whose length is 28 bits;

(iii) bit 30-43-20110627172133 is the information of timestamp of a tag, 20110627 is the date, 172133 is the time, and the length of the time is 14 bits.

First, we define data redundancy based on the following RFID data model [25]: when a reader reads a tag, the following data can be read: EPC (EPC code), reader (reader ID), and timestamp (timestamp). When the reader reads two

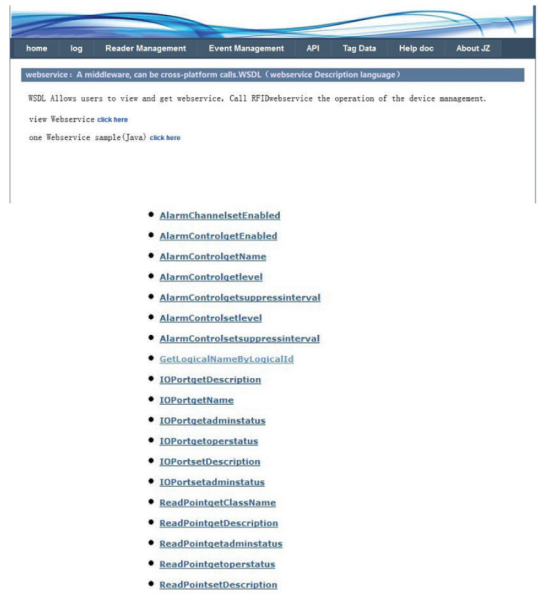

Figure 18: Some Web service APIs.

Table 1: The configuration of Edge servers.

Edge server	CPU (GHz)	MEM (GB)	CPU (UB)	MEM (UB)	$WL_M^U[m_i]$
A	1.5	1	0.7	0.7	5000
B	2	1.5	0.7	0.7	10000
C	2.5	2	0.7	0.7	20000
D	3	2.5	0.7	0.7	30000

Table 2: Utilization information before load balancing.

Edge server	CPU utilization (%)	MEM utilization (%)
A	34	37
B	42	46
C	70	78
D	26	28

labels (Label A and Label B), if all of the following three conditions are satisfied, it can be said that these two labels are repeated data (tags), that is redundant data [4]:

Label A: EPCA, reader A, and timestamp A;

Label B: EPCB, reader B, and timestamp B.

(1) EPC codes are the same, that is EPCA = EPCB.

(2) Reader A and Reader B are the same.

(3) The difference between Timestamp A and Timestamp B is less than constant T, which may be 10 milliseconds, for example. Figure 19 provides an example using MapReduce.

The process of parallel MapReduce redundancy data elimination can be summarized as follows (see [27] for more details).

(1) In Map phase, the data structure Tag consists of three variables: Reader ID, EPC, and Timestamp. According to the definition of tag redundancy, if two tags are redundant, their Reader IDs and EPC codes must be the same, and if one of them is different, the two tags are not redundant tags. Input RFID tags are divided into different blocks and assigned to hosts of Hadoop clusters. Reader ID and EPC codes are Keys and timestamps are outputs as Values to form (Key, Value) pairs in MapReduce model. In this case, that is, Key is (Reader ID + EPC), while Value is Timestamp.

(2) After simple sorting based on the Key, Map (mapping) outputs of each host are transferred to Reduce.

(3) Reduce stage: based on outputs of Map stage, it compares values of each group of Key Value (calling time comparison function), and if they are less than constants T (predefined), they are redundant and should be removed, then Reduce removes the redundancies and obtains an output and repeats doing this until all inputs are proceeded. Notice that there may be multiple workers (servers) that work paralleled in Map and Reduce stages.

(4) Working nodes (hosts) in MapReduce cluster summarize *Key*, *Value* pairs of other hosts processed by Reduce and output them according to the original tag format.

Table 3: Utilization information after applying our load-balancing mechanism and method in [28].

Edge server	CPU (%) [28]	MEM (%) [28]	CPU (%) ARRMS	MEM (%) ARRMS
A	34.0	37.0	34.0	37.0
B	82.0	91.0	42.0	46.0
C	50.0	55.0	50.0	55.0
D	26.0	28.0	39.0	43.0
L_M	0.12		0.01	

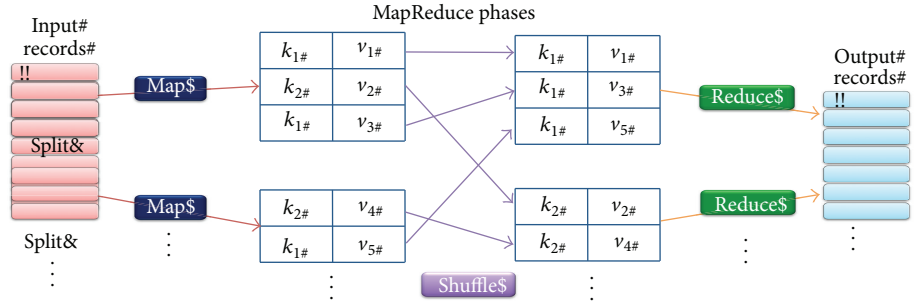

Figure 19: Redundancy elimination example using MapReduce.

Table 4: Data format of an RFID tag.

Field	Device ID	EPC code	Timestamp
Bits	2	28	14

As for evaluation, set the total number of RFID tags is S, the redundancy ratio is P_d and redundant tags are randomly generated following EPC data format (32 bits currently), as shown in Table 4 where total number of tags varying from 30 to 45 million and Cluster size means the number of host servers used in Hadoop MapReduce. From Table 5, we can see that the larger the number of tags and the MapReduce cluster size is, the bigger the difference of time spent on redundancy elimination is. As the number of tags increases (above 50000), MapReduce begins to show the advantages of scalability. The proposed method can be applied to large scale RFID applications such as supermarkets and other areas conveniently. The reason that Hadoop cluster can remove 100% redundant data for large-scale of data up to 100 million lies in that it is a centralized redundancy elimination mechanism and it can take all redundancy information into consideration.

Figure 20 also shows tag lost ratio comparison from simulation results, where connection pool is obtained using the method in [30] and Agent is obtained, applying by method in [29]. In the simulation, the total number of readers is fixed as 30 but with varying the total number of middleware: 2, 4, 6, 8, and 10. The tag information generation rate is 50 tags/sec, and the time for a middleware to process a tag information is 2 microseconds, and the total test length of each run is 15 minutes, and our results are the average of six runs. Figure 21 also provides average process time comparison for the same total amount of tags. Similar results are obtained in other cases, because of page limit, those results are not provided here.

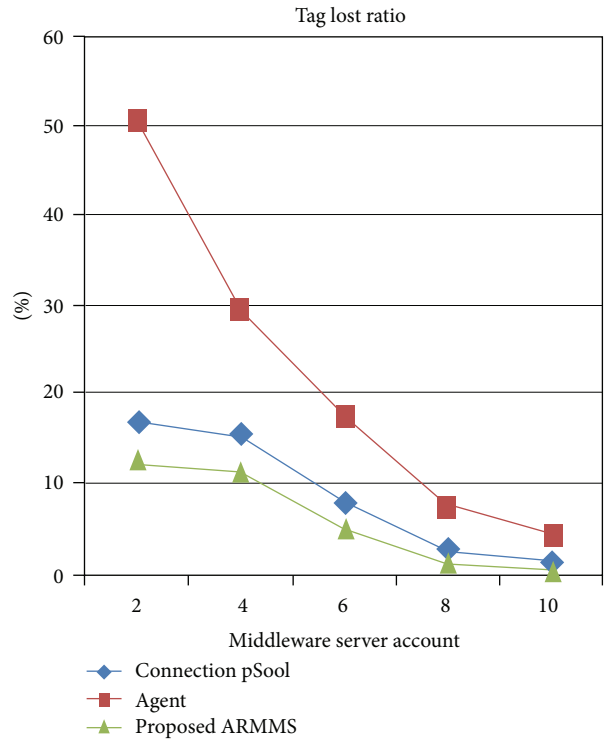

Figure 20: Tag lost ratio comparison.

5. Conclusions and Future Work

In this paper, a distributed and messaging RFID middleware design is proposed. ARMMS provides equipment management, RFID tag information management, application level event (ALE) management, and Web service APIs, and so forth, functions. For the equipment management, ARMMS

Table 5: Times spent for Hadoop cluster to eliminate redundancy.

Number of tags	Cluster size						
	1 server	2 servers	3 servers	4 servers	10 servers	15 servers	20 servers
30	4 sec	7 sec	7 sec	7 sec	7 sec	7 sec	7 sec
50000	5 sec	8 sec	7 sec	7 sec	4 sec	2 sec	1 sec
500000	16 sec	18 sec	14 sec	12 sec	6 sec	3 sec	2 sec
1.2 million	33 sec	24 sec	22 sec	19 sec	10 sec	6 sec	3 sec
45 million	52 min	48 min	42 min	41 min	21 min	11 min	5 min

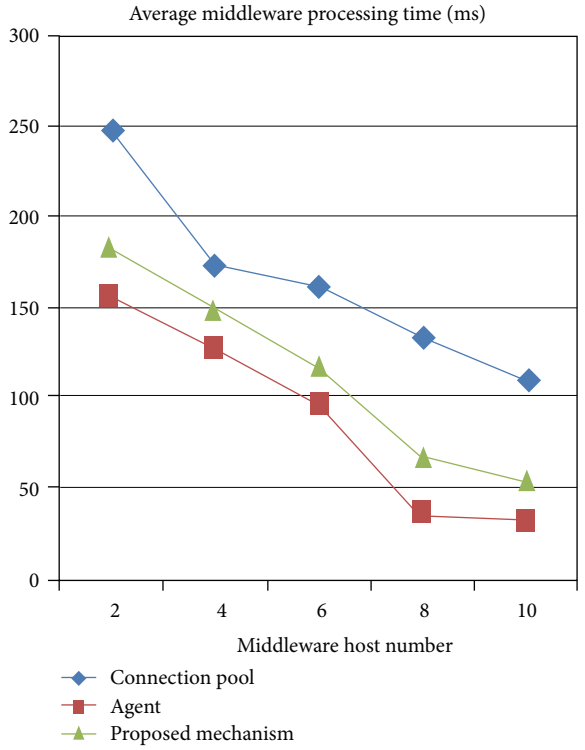

Figure 21: Average process time comparison.

provides real-time monitoring and load balancing of multi-site RFID devices also logical grouping of RFID readers. For tag information management, it provides filtering and statistical reports. ARMMS also provides Web service APIs based on EPCglobal standards. We are conducting more experimental tests and comparing different load-balancing strategies with proposed one. There are still several research directions awaiting further investigation.

(1) Considering distributed redundancy data elimination: when facing large-scale distributed applications, distributed redundancy data elimination other than parallel method should be investigated further, and elimination efficiency should be quantitatively evaluated.

(2) Considering real-time data allocation and evaluating further live migration costs and load balancing: real-time data allocation among different middleware (host Edge servers) causes a new challenge for load balance and should be considered further. Also, the live migration costs in time and other respects should be included to provide a complete view for load balance so that decision makers can have comprehensive information.

(3) Providing more comparative results against existing methods: currently, there is still difficulty to repeat some of the existing methods such as in [1], where experimental details are not provided. We need repeating their algorithms and configuration so that more comparative scienarios and results can be obtained.

Acknowledgments

This research is supported by the National Natural Science Foundation of China (NSFC) (Grants nos. 61150110486, 61034005 and 61272528) and by China Postdoc Funding (2011-2012).

References

[1] Y. Ma, H. Chao, J. Chen, and C. Wu, "Load-balancing mechanism for the RFID middleware applications over grid networking," *Journal of Network and Computer Applications*, vol. 34, no. 3, pp. 811–820, 2011.

[2] I. Abad, C. Cerrada, J. A. Cerrada, R. Heradio, and E. Valero, "Managing RFID sensors networks with a general purpose RFID middleware," *Sensors*, vol. 12, no. 6, pp. 7719–7737, 2012.

[3] C. Floerkemeier and M. Lampe, "RFID middleware design: addressing application requirements and RFID constraints," in *Proceedings of the Smart Objects Conference (SOC '05)*, pp. 219–224, Grenoble, France, October 2005.

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

[5] X. Huang, S. Le, and D. Sharma, "A taxonomy for RFID systems," in *Proceedings of the 1st International Conference on Signal Processing and Communication System*, pp. 1–8, Gold Coast, Australia, December 2007.

[6] M. A. E. Khaddar, M. Boulmalf, H. Harroud, and M. Elkoutbi, "RFID middleware design and architecture, book chapter, design and deploying RFID applications".

[7] R. Want, "An introduction to RFID technology," *IEEE Pervasive Computing*, vol. 5, no. 1, pp. 25–33, 2006.

[8] R. Weinstein, "RFID: a technical overview and its application to the enterprise," *IT Professional*, vol. 7, no. 3, pp. 27–33, 2005.

[9] K. Finkenzeller, *RFID Handbook: Radio-Frequency Identification Fundamentals and Applications*, John Wiley & Sons, New York, NY, USA, 2000.

[10] B. Nath, F. Reynolds, and R. Want, "RFID technology and applications," *IEEE Pervasive Computing*, vol. 5, no. 1, pp. 22–24, 2006.

[11] "ALE_1_1_1-standard-core-20090313[EB/OL]," http://wwwepcglobalinc.org, 2009.

[12] "ALE_1_1_1-standard-XML_and_SOAP_bindings-20090313[EB/OL]," http://www.epcglobalinc.org, 2009.

[13] Auto-ID Labs, http://www.autoidlabs.org/S .

[14] "Class 1 Generation 2 UHF Air Interface Protocol Standard Version 1. 0. 9[EB/OL]," http://www.epcglobalinc.org/.

[15] "EPCglobal-ReaderManagementrm_1_0_1-standard-20070531 [EB/OL]," http://www.epcglobalinc.org/, 2007.

[16] B. Feng, J. T. Li, P. Zhang, and J. B. Guo, "Study of RFID middleware for distributed large-scale systems," in *Proceedings of the Information and Communication Technologies (ICTTA '06)*, vol. 2, pp. 2754–2759, 2006.

[17] M. Cezon, G. Vaudaux-Ruth, L. Laurens, and J. Soldatos, "Review of state-of-the-art middleware," ASPIRE Project Public Deliverable D2. 1., 2008.

[18] F. Lin and B. Chen, "The design of a lightweight RFID middleware," *International Journal of Engineering Business Management*, vol. 1, no. 2, pp. 25–30, 2009.

[19] B. S. Prabhu, X. Su, H. Ramamurthy, C. Chu, and R. Gadh, "WinRFID: a middleware for the enablement of radio frequency identification (RFID) based applications," in *Proceedings of the Wireless Internet for the Mobile Enterprise Consortium (WINMEC '05)*, Los Angeles, Calif, USA, December 2005.

[20] P. Tran, P. Greenfield, and I. Gorton, "Behavior and performance of message-oriented middleware systems," in *Proceedings of the 22nd International Conference on Distributed Computing Systems*, 2002.

[21] Y. Bai, F. Wang, and P. Liu, "Efficiently filtering RFID data streams," in *Proceedings of the 1st International VLDB Workshop on Clean Databases (CleanDB '06)*, September 2006.

[22] C. Yingwen, V. L. Hong, X. Ming, C. Jiannong, K. C. C. Chan, and A. T. S. Chan, "In-network data processing for wireless sensor networks," in *Proceedings of the 7th International Conference on Mobile Data-Management (MDM '06)*, Nara, Japan, May 2006.

[23] W. Choi and M. Park, "In-network phased filtering mechanism for a large-scale RFID inventory application," in *Proceedings of the 4th International Conference on Information Technology and Applications (ICITA '07)*, pp. 401–405, Harbin, China, January 2007.

[24] S. R. Jeffery, M. Garofalakis, and M. J. Franklin, "Adaptive cleaning for RFID data streams," in *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB '06)*, September 2006.

[25] I. Kadayif and M. Kandemir, "Tuning in-sensor data filtering to reduce energy consumption in wireless sensor networks," in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE '04)*, pp. 1530–1539, Paris, France, February 2004.

[26] A. K. Bashir, S. Lim, C. S. Hussain, and M. Park, "Energy efficient in-network RFID data filtering scheme in wireless sensor networks," *Sensors*, vol. 11, no. 7, pp. 7004–7021, 2011.

[27] W. H. Tian, Y. P. Yang, K. She, X. Dong, and H. Y. Wang, "A parallel method to redundancy data elimination in RFID network," in *Proceedings of the International Conference on Instrumentation, Measurement, Circuits and Systems (ICIMCS '11)*, pp. 399–402, Hongkong, China, December 2011.

[28] H. S. Chae and J. Park, "An approach to adaptive load balancing for RFID middlewares," *International Journal of Mathematical and Computer Sciences*, vol. 2, no. 2, 2006.

[29] F. C. Jian and S. C. Heung, "Agent-based design of load balancing system for RFID middlewares," in *Proceedings of the 11th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS '07)*, pp. 21–28, March 2007.

[30] S. Park, J. Song, C. Kim, and J. Kim, "Load balancing method using connection pool in RFID middleware," in *Proceedings of the 5th ACIS International Conference on Software Engineering Research, Management, and Applications (SERA '07)*, pp. 132–137, August 2007.

[31] J. G. Park, H. S. Chae, and E. S. So, "A dynamic load balancing approach based on the standard RFID middleware architecture," in *Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE '07)*, pp. 337–340, October 2007.

[32] W. H. Tian, K. She, Y. P. Yang, and X. Dong, "An approach to dynamic and integrated load-balancing of distributed and messaging RFID middleware," in *Proceedings of the International Conference on Instrumentation, Measurement, Circuits and Systems (ICIMCS '11)*, pp. 403–406, Hongkong, China, December 2011.

[33] J. Dean and S. Ghemawat, "MapReduce: simplied data processing on large clusters," Google Paper, 2004.