博士學位論文

대량 RFID 데이터 처리를 위한 부하 분산

濟州大學校 大學院

컴퓨터工學科

盧 永 湜

2011年 6月

대량 RFID 데이터 처리를 위한 부하 분산

指導敎授 邊 暎 哲

盧 永 湜

이 論文을 工學 博士學位 論文으로 提出함

2011年 6月

盧永湜의 工學 博士學位 論文을 認准함

審査委員長 _____ 印

委 員 _____ 印

委 員 _____ 印

委 員 _____ 印

委 員 _____ 印

濟州大學校 大學院

2011年 6月

Load Balancing for Large-Scale RFID Data Processing

Young-Sik Noh

(Supervised by professor Yung-Cheol Byun)

A thesis submitted in partial fulfillment of the requirement for the degree of Doctor of Computer Engineering

2011. 6.

This thesis has been examined and approved.

Thesis director, _____

Thesis director, _____

Thesis director, _____

Thesis director, _____

Thesis director, _____

June 2011

Department of Computer Engineering

Graduate School

Jeju National University

감사의 글

2005년 대학원 생활을 시작하여 어느덧 박사 학위과정을 마치고 학위를 받게 되었습니다. 석사과정 2년과 박사과정 5년 동안 열심히 했던 대학원 생활을 뒤돌아보니 저에게 도움을 주셨던 모든 분들이 머릿속을 스쳐갑니다.

제일 먼저 저에게 처음으로 컴퓨터 공부의 재미를 느끼게 해주셨던 제주산업정보대학 시절 선배님이신 현준이와 다희 아빠 재호형님께 너무 고맙다는 말을 하고 싶습니다. 또한 제주산업정보대학시절 철부지 없던 저를 잘 이끌어 주셨던 김대영 교수님, 고희준 교수님, 양창우 교수님, 박충희 교수님과 탐라대학교시절 현창문 교수님께도 고마움을 전하고 싶습니다. 그리고 저에게 대학원을 권유해 주시고 또 하나의 길을 열어주신 자바정보기술(주) 박상열 사장님께 감사드리며, 명호형, 호윤이형, 경민이형, 병섭이형, 대현이형, 홍림이형께도 감사합니다.

무엇보다 제가 지난 7년간 대학원 생활을 함에 있어 아낌없이 가르침을 주셨던 지도교수님이신 변영철 교수님께 감사합니다. 또한 논문 심사에 애써주신 김장형 교수님과 변상용 교수님, 이상준 교수님, 건국대학교 고명철 교수님께도 감사합니다. 그리고 대학원 생활동안 세심한 지도를 아끼지 않아주셨던 강창언 교수님, 안기중 교수님, 곽호영 교수님, 송왕철 교수님, 김도현 교수님께 감사합니다.

대학원 생활동안 연구실에서 힘이 되어주신 홍연미, 양문석, 변지웅, 차지윤, 한대오 연구원들과 요종, 정현, 장휴, 가영, 혜연, 영재, 진걸, 민지, 봉철이등 학부생들, 정은경, 이정하, 김남식 조교 선생님들, 많은 조언을 해주신 양동호 선배님, 변태보 선배님, 양덕성 선생님, 진용석 선생님, 논문에 대한 많은 방향을 제시 해주셨던 한경복 박사님, 오상현 박사님, 허지완 박사님, 권훈 박사님, 메카트로닉스 공학전공 강철웅 교수님, 제주관광대학 강민수 교수님께도 감사합니다.

마지막으로 고생했던 대학원 생활동안 뒤에서 응원을 해주었던 봉헌, 승종, 기철, 두식, 영석등 제주산업정보대학 99학번 모임, 순길, 은호, 정훈, 성호, 현태, 용수, 태양, 정인, 정호, 정환, 숭호, 정인이등 고등학교 우정회 친구들에게 감사하고, 나의 영원한 후원자이신 부모님께 감사하고 건강하시라는 말을 남깁니다.

목 차

그림 목차

표 목차

국문초록

대량 RFID 데이터 처리를 위한 부하 분산

컴퓨터공학과 노영식
지 도 교 수 변영철

유비쿼터스 컴퓨팅을 선도하는 차세대 핵심요소로 RFID(Radio Frequency Identification) 기술이 주목을 받고 있다. 각종 태그 정보를 SCM, ERP등의 다양한 엔터프라이즈 애플리케이션에서 활용하기 위해서는 정확한 장소에 실시간으로 인증된 정보를 전달할 수 있는 RFID 미들웨어가 필요하다. RFID 미들웨어는 네트워크를 통하여 다수의 리더로부터 대량의 태그 데이터들을 수집하고 식별정보를 처리한다. RFID 사용 수요가 증가하고 있고, 유비쿼터스 서비스를 제공하기 위하여 대량 RFID 데이터를 처리할 수 있는 미들웨어에 대한 필요성이 증가하고 있다. 사실상 RFID 기술의 국제 표준을 이끌고 있는 EPCglobal에서는 EPC 네트워크 표준을 제안하였고, EPC 네트워크는 RFID 미들웨어와 관련하여 2004년에 인터페이스 중심의 ALE(Application Level Event)를 제안하였다. 최근 ALE 기반의 RFID 미들웨어 제품 및 솔루션은 EPC 코드 등과 같이 간단한 형식의 데이터를 처리할 수 있으나 대량의 데이터를 실시간으로 필터링, 그룹핑 등의 처리를 통하여 보다 정확하고 유익한 자료를 제공하기 위한 고려가 미진한 상태이다. 따라서, 다양한 유비쿼터스 응용에 RFID 미들웨어를 활용하기 위해, 국제 표준 스펙을 기반으로 구현된 RFID 미들웨어를 이용하여 대량의 RFID 데이터를 처리할 수 있도록 분산 환경을 구축하고, 효과적으로 클라이언트의 RFID 데이터 수집 및 가공 처리 요청을 분산 할 수 있는 방법이 필요하다.

본 논문에서는 EPCglobal의 ALE 스펙을 기반으로 구현된 RFID 미들웨어를 이용하여 보다 대량의 RFID 데이터를 수집하여 클라이언트 응용에게 가공된 RFID 데이터를 효율적으로 제공하기 위한 방법을 제안한다. 클라이언트의 RFID 데이터 수집 및 가공 처리 요청을 분산하기 위하여, 분산 환경에 구축된 여러 대

의 RFID 미들웨어의 자원 정보를 수집하고, 각 RFID 미들웨어의 자원 사항에 맞게 클라이언트의 요청을 처리할 수 있도록 한다. 이를 위해 기 구현된 ALE 스펙 기반 RFID 미들웨어의 자원 사항을 하드웨어 자원, 미들웨어 자원, ALE 자원으로 구분하여 개별 저장하며, 부하 분산 시스템에서 자원 정보를 요청했을 시에 개별 저장된 자원 정보를 SOAP 통신을 이용하여 XML 형태로 제공한다. 또한, 클라이언트의 요청 처리를 수행할 최적의 ALE 기반 RFID 미들웨어를 선정하기 위해 유전자 알고리즘을 이용하여 가중치를 생성하여 부하 분산 처리에 활용하였다. 본 논문의 ALE 스펙을 기반으로 구현된 RFID 미들웨어는 클라이언트의 RFID Tag 데이터 수집 및 가공처리 유형을 ECSpec 명세서 형태로 관리하며, 유전자 알고리즘을 이용하여 구한 50개 유형의 ECSpec 개별 가중치 보다, 50개 유형의 ECSpec 평균 가중치 값을 기반으로한 부하 분산 방법이 부하 분산 처리를 잘 수행할 수 있음을 확인할 수 있었고, 본 논문에서 제시한 50개 유형 이외의 ECSpec 처리 사항에도 평균 가중치 값을 사용할 수 있을 것으로 기대된다.

주요어 : RFID, ALE, 미들웨어, 유전자 알고리즘, 부하 분산

ABSTRACT

Load Balancing for Large-Scale RFID Data Processing

Young-Sik, Noh

Department of Computer Engineering

Graduate School

Jeju National University

RFID, which is the next generation technique leading the ubiquitous computing, has recently gained a lot of attention. RFID middleware should send certified information to a client in real-time to utilize tag information in various enterprise applications such as SCM, ERP, and etc. RFID middleware gathers large-scale tag data from many readers through a network and processes identification information. As the demand of RFID usage and ubiquitous services are increasing, the need for RFID middleware to process large-scale RFID data has been growing. EPCglobal leading the international standard in RFID technology suggested EPC network standards, and EPC network suggested Application Level Event(ALE) in 2004. Recently, RFID middleware products and solutions based on ALE can process the specific data such as EPC code, and etc. However, it is not considered to be able to efficiently provide correct and useful data by processing large-scale data in real-time through filtering, grouping, and other operations. Therefore, in order to utilize RFID middleware in various ubiquitous applications dealing with high-volume data, it is necessary to implement the distributed environment and to find ways to distribute requests to gather and process RFID data by client effectively. Large-scale RFID data can be processed by using RFID middleware that conforms to ALE specification and international standards.

In this paper, we propose the method to effectively provide client applications with processed RFID data by gathering large-scale RFID data by utilizing RFID middleware implemented with ALE specification. With the proposed approach, we can collect the resource information of RFID middleware used to implement distributed environment and process client requests according to the state of resources to distribute the collection of RFID data and the requests to process it from clients.

For this purpose, the resource state of existing RFID middleware implemented based on ALE specification is separately stored according to the resource information for each hardware, middleware, ALE. If the proposed load balancing system requests resource information, the information is provided as XML type by using Simple Object Access Protocol(SOAP) communication. Meanwhile, in order to select the right ALE-based RFID middleware to respond for client's requests, genetic algorism(GA) was used to decide proper weight of middleware. In the proposed middleware, the information for tag data acquisition and processing is represented and managed by ECSpec object. The experiments showed that load balancing method based on the weight calculated by using GA works well.

Keywords : RFID, ALE, Middleware, Genetic Algorithm, Load Balancing

약어표

RFID	Radio Frequency IDentification
SCM	Supply Chain Management
ERP	Enterprise Resource Planning
ISO	International Organization for Standardization
EPC	Electronic Product Code
ALE	Application Level Event
URN	Uniform Resource Name
SOAP	Simple Object Access Protocol
XML	eXtensible Markup Language
RR	Round Robin
WRR	Weighted Round Robin
EPCIS	Electronic Product Code Information Service
ONS	Object Name Service
PDA	Personal Digital Assistants
RC	Read Cycle
EC	Event Cycle
API	Application Program Interface
URI	Uniform Resource Identifier
JMS	Java Message Service
J2EE	Java 2 platform, Enterprise Edition
CARU	Context-Aware RFID Middleware System for Ubiquitous
DNS	Domain Name System
TCP	Transmission Control Protocol
LC	Least Connection
WLC	Weighted Least Connection
LBLC	Locality Based Least Connection

LBLCR	Locality Based Least Connection with Replication
DH	Destination Hashing
SH	Source Hashing
GA	Genetic Algorithm
PLC	Peak Load Control
VOD	Video on Demand
CPU	Central Processing Unit
CC	Cluster Controller
DHT	Distributed Hash Table
P2P	Peer to Peer
RDS	Rendezvous Directory Strategy
ISS	Independent Searching Strategy
GBS	Gossip Based Strategy
LR	Logical Reader
DB	Data Base
RVGA	Real Variable Genetic Algorithm
LB	Load Balancer
DAO	Data Access Objects
WSDL	Web Services Description Language
GAW	Genetic Algorithm Weighted
GAAW	Genetic Algorithm Average Weighted

I. 서 론

1. 연구 배경 및 목적

유비쿼터스 컴퓨팅을 선도하는 차세대 핵심요소로 RFID(Radio Frequency Identification) 기술이 주목을 받고 있다. 각종 태그 정보를 SCM(Supply Chain Management), ERP(Enterprise Resource Planning)등의 다양한 엔터프라이즈 애플리케이션에서 활용하기 위해서는 정확한 장소에 실시간으로 인증된 정보를 전달할 수 있는 RFID 미들웨어가 필요하다[1].

RFID 미들웨어는 다수의 리더로부터 대량의 태그 데이터들을 수집하며, 네트워크를 통하여 대량의 식별정보를 처리한다. 또한, 유비쿼터스 서비스를 제공하기 위하여 RFID 사용 수요가 증가하고 있어, 대량 RFID 데이터를 처리할 수 있는 미들웨어에 대한 필요성이 증가하고 있다.

사실상 RFID 기술의 국제 표준을 이끌고 있는 EPCglobal에서 제안하고 있는 EPCglobal Class 1 Gen. 2는 900MHz 대역의 사실상의 표준일 뿐만 아니라 ISO 18000-6C로서 국제 표준으로 채택되었다. 또한 EPCglobal은 EPC 네트워크 표준안을 제안하였고, RFID 미들웨어 플랫폼을 개발하였거나 개발하려는 많은 업체들이 이를 표준으로 수용하고 있다[2]. EPC 네트워크는 RFID 미들웨어와 관련하여 2004년에 인터페이스 중심의 ALE(Application Level Event)를 제안하였다[3].

ALE 기반 RFID 미들웨어는 리더 장치로부터 EPC(Electronic Product Code) 코드 데이터를 입력받을 경우 이를 처리하기 위하여 내부적으로 URN(Uniform Resource Name) 형태의 데이터로 변환하고, 이를 이용하여 필터링과 그룹핑 등을 수행한 후 그 결과를 응용으로 전송한다. 최근 RFID 기반의 미들웨어 제품 및 솔루션은 EPC 코드 등과 같이 간단한 형식의 데이터를 처리할 수 있으나 대량의 데이터를 실시간으로 필터링, 그룹핑 등의 처리를 통하여 보다 정확하고 유익한 자료를 제공하기 위한 고려가 미진한 상태이다[1].

한편, 최근에 제안된 데이터 센서, 웹 서버 로그나 전화 기록과 같은 다양한 트랜잭션 로그 분석등과 관련된 대량 데이터 스트림을 실시간으로 처리하는 것에 많은 관심이 집중되고 있다. 이는 RFID, 센서 데이터 처리, 인터넷 트래픽 분석, 웹 서버 로그와 같은 다양한 트랜잭션 로그 분석 등 다양한 분야에서 응용되고 있다[4].

또한 처리되어야 할 데이터의 양, 동시에 처리되는 필터링 조건의 수등을 고려하여 데이터 처리 방법을 채택하여야만 데이터의 손실 없이 실시간 처리가 가능하다. 최근에 RFID 기반 미들웨어 제품 및 솔루션들이 많이 개발되고 있으나, 주로 EPC 코드 등과 같은 간단한 형식의 데이터를 처리하며, 대량의 데이터 처리에 대한 고려가 미진한 상태이다. 기 개발된 해외 또는 국내업체의 RFID 미들웨어도 단순히 비즈니스 애플리케이션과 핵심 기반구조 사이의 통신만을 지원하는 형태이다[5].

따라서, 다양한 유비쿼터스 응용에 RFID 미들웨어를 활용하기 위해, 국제 표준 스펙을 기반으로 구현된 RFID 미들웨어를 이용하여 대량의 RFID 데이터를 처리할 수 있도록 분산 환경을 구축하고, 효과적으로 클라이언트의 RFID 데이터 수집 및 가공 처리 요청을 분산 할 수 있는 방법이 필요하다.

기존 RFID 미들웨어의 부하 분산 시스템에서는 CPU 사용율과 클라이언트의 RFID Tag 데이터 수집 및 가공 처리 유형이 명세된 ECSpec의 수, 로지컬 리더에서 수집되는 RFID Tag 데이터의 수, RFID 미들웨어서 처리하고 있는 데이터의 수만을 고려하여 부하 분산 처리를 수행하는데 필요한 자원 정보로는 부족하며, 보다 세부적이고 RFID 미들웨어의 특성을 고려한 부하 분산을 위해서는 추가적인 DB 접속수, 스레드 처리, 모듈 처리, ECSpec에 명세된 클라이언트의 RFID Tag 데이터 수집 및 가공 처리 주기에 해당하는 이벤트 사이클 처리와 같은 자원 정보가 필요하다.

그리고, RFID 미들웨어에서 처리하고자 하는 ECSpec과 이벤트 사이클 유형이 여러 가지일 경우에는 한 가지 유형에 비하여 상대적으로 복잡한 처리 알고리즘이 필요하며, 분산 환경에 구축된 RFID 미들웨어의 부하량 계산에 필요한 자원 정보가 많아지고 유형이 다양해질수록 계산에 필요한 식이 많아지고 계산 시간 역시 늘어나게 되어 부하량 계산을 보다 효율적으로 수행할 방법이 필요하다.

2. 연구 내용 및 방법

본 연구는 EPCglobal의 ALE 스펙을 기반으로 구현된 RFID 미들웨어를 이용하여 보다 대량의 RFID 데이터를 수집하여 클라이언트 응용에게 가공된 RFID 데이터를 효율적으로 제공하기 위한 방법을 제안하는 것이다.

대량의 RFID 데이터를 수집 및 처리하기 위하여 RFID 데이터 수집, 데이터 가공 처리, 클라이언트 요청 처리 등의 기능을 처리하도록 ALE 스펙을 기반으로 구현된 RFID 미들웨어들의 분산 환경을 구축하고, 클라이언트의 요청을 분산 할 수 있는 부하 분산 시스템을 설계 및 구현하고자 한다. 클라이언트의 RFID 데이터 수집 및 가공 처리 요청을 분산하기 위하여, 분산 환경에 구축된 여러 대의 RFID 미들웨어의 자원 정보를 수집하고, 각 RFID 미들웨어의 자원 사항에 맞게 클라이언트 요청을 처리할 수 있도록 해야 한다. 이를 위하여 기 구현된 ALE 스펙 기반 RFID 미들웨어의 자원 사항을 하드웨어 자원, 미들웨어 자원, ALE 자원으로 구분하여 개별 저장하며, 부하 분산 시스템에서 자원 정보를 요청했을 시에 개별 저장된 자원 정보를 SOAP(Simple Object Access Protocol) 통신을 이용하여 XML(eXtensible Markup Language) 형태로 제공한다.

한편, ALE 스펙을 기반으로 구현된 RFID 미들웨어는 클라이언트의 RFID Tag 데이터 수집 및 가공 처리 요청 유형을 ECSpec 명세서 형태로 관리하며, 보다 효율적으로 처리 요청을 분산하기 위해서는 ECSpec에 명시되어 있는 RFID 데이터의 수집방법 및 가공 처리 방법 등의 사항을 고려해야 한다.

이를 위하여 50가지 유형의 ECSpec 처리 유형을 분석하고, 분산 환경에 구축된 서로 사양이 다른 각 RFID 미들웨어에서 처리를 수행할 시에 자원 사용량을 바탕으로 각 ECSpec 처리 유형마다 최적으로 부하 균등이 잘 이뤄졌을 시에 상황을 찾는다.

본 논문에서는 각 RFID 미들웨어로부터 수집되는 하드웨어, 미들웨어, ALE 자원 정보로 관리되는 CPU 사용률, 메모리 사용량 등 30여개의 자원 정보 각각에 대해 부하 균등 상황을 찾고자 하였으며, 자원 정보 각각에 대해 자원 정규화 값의 평균을 구하고, 자원 정규화 값의 분포도 평균이 0과 비슷한 상황이 부하가

균등하게 된 상황을 의미한다.

이렇게 찾은 부하 균등 상황에서 자원 정규화 평균값을 실제 수집되는 각 자원 정보에 가중치 정보로 적용하면, 찾았던 부하 균등 상황이 이뤄 질수 있도록 유도하고, 30여개의 자원 정보의 특성상 이러한 부하 균등 상황을 이상적으로 찾을 수 없는 경우 최적으로 부하 균등이 이뤄진 상황을 찾기 위해 최적화 알고리즘인 유전자 알고리즘을 사용하였으며, 유전자 알고리즘을 사용하여 30여개의 자원 중요도 정보에 해당하는 가중치 정보를 생성하여 최적으로 클라이언트의 처리 사항을 분산 할 수 있는 방법을 제안하고자 한다.

또한, 분산 환경에 구축된 ALE 기반 RFID 미들웨어의 부하량을 판단하는 파라미터로는 서버에서 실행되고 있는 모든 프로세스 수, JAVA 기반으로 구현된 미들웨어에서 사용되고 있는 메모리 사용량, 미들웨어에서 사용되고 있는 CPU 사용률, 힙 메모리 사용량, 가상 메모리 사용량, 네트워크 속도 등의 하드웨어 자원 정보와 미들웨어에서 생성한 DB 접속 객체 수, 미들웨어에서 실행되고 있는 스레드 수, 미들웨어에서 실행된 스레드의 총 수, 미들웨어에서 생성되어 객체화된 클래스의 수, 미들웨어에서 객체로 생성되어 처리가 종료된 클래스의 수, 미들웨어에서 생성되어 객체화된 클래스의 총 수 등의 미들웨어 자원 및 미들웨어 등록된 논리 리더의 수, 미들웨어에서 실제 RFID Tag 데이터를 수집하고 있는 논리 리더의 수, 논리 리더에서 수집하고 있는 RFID Tag 수, 미들웨어에서 처리되고 있는 이벤트 사이클의 수, 이벤트 사이클에서 처리되고 있는 RFID Tag 데이터의 수, 가공 처리되어 처리를 요청한 클라이언트 응용에게 제공할 RFID Tag 데이터의 수 등의 ALE 자원 정보들이 있다.

이 정보들은 부하 균등 상황을 이루기 위한 가중치로 표현될 수 있고, 이들 파라미터들은 각각의 ECSpec 처리 사항에 따라 중요도가 달라질 수 있으며, 그 파라미터들은 서로에게 영향을 주는 Trade-off 관계에 있다.

따라서, 자원 정보들 간의 모호성과 분산 환경에 구축된 RFID 미들웨어의 수가 늘어남에 따른 계산의 복잡성, 부하 상황의 다양성을 충족시키고자 본 연구에서는 부하 분산 시에 필요한 가중치 정보를 찾기 위해 유전자 알고리즘을 사용하였다.

한편, 분산 환경에 구축된 RFID 미들웨어로 클라이언트의 요청을 분산하는 방법은 모든 RFID 미들웨어로 클라이언트의 요청사항을 일괄적으로 보내 해당 요청을 처리할 수 있는지에 대한 정보를 사용하여 분산하는 형식과 모든 RFID 미들웨어의 자원사항을 수집하여 부하량이 가장 적은 RFID 미들웨어로 클라이언트의 요청을 분산하는 로드밸런서를 사용한 부하분산 방법이 있다.

브로드캐스터 형식에서는 부하 균등 사항을 이루기 위한 별도의 구성이 없으며, 분산 환경에 구축된 RFID 미들웨어의 부하를 최대한 균등하게 하면 보다 대량의 데이터를 처리할 수 있기 때문에 본 논문에서는 로드밸런서를 사용하여 부하 균등 사항을 초기에 이루도록 유도하여 보다 대량의 RFID 데이터를 처리할 수 있는 방법을 제안하고자 한다.

3. 논문 구성

2장은 관련 연구로서 EPCglobal Network Architecture와 EPCglobal의 ALE 스펙 사항과 기존 RFID 미들웨어를 분석한다. 또한, 적절한 부하 방법 연구를 위하여 기존의 부하 분산 알고리즘과, 최적의 부하 분산 방법을 연구하기 위해 최적화 알고리즘인 유전자 알고리즘을 분석한다. 그리고, 기존의 일반 부하 분산 시스템과 RFID 미들웨어 부하 분산 시스템을 설명한다.

3장에서는 대량의 RFID 데이터를 처리할 수 있도록 본 논문에서 제안하는 부하 분산 시스템에 대해 설명하고, 최적의 부하 분산을 위한 가중치 정보와 부하 분산 시스템의 아키텍처에 대하여 기술한다.

4장에서는 구현한 부하 분산 시스템과 최적의 가중치 정보를 생성하기 위한 유전자 알고리즘 기반 가중치 생성 시스템을 설명하고, Round Robin, Random, 기존 RFID 미들웨어의 부하 분산 방법인 Weighted Round Robin, 본 논문에서 구현한 유전자 알고리즘을 이용하여 생성된 가중치 정보를 이용한 부하 분산 방법을 테스트하여 수행한 결과를 분석하며 5장에서는 본 논문의 결론과 향후 연구 과제에 대해 논한다.

II. 관련 연구

1. EPCglobal Network Architecture

EPCglobal은 RFID 표준의 개발과 보급을 위해 GS1이 설립한 비영리 국제표준기구이다. EPCglobal은 개별 물체의 유일 식별자인 EPC 기반의 “Internet of Physical Object"를 구성하기 위한 기술 집합을 EPC 네트워크라 정의하고 있다[6]. EPC 네트워크는 구현에 따른 기술요소 분야를 하드웨어, 소프트웨어, 비즈니스 분야로 구분하여 각 분야별 활동 그룹을 구성하고, EPCglobal에 가입한 업체를 중심으로 기술규격과 표준 제정을 추진하고 있다.

EPCglobal 네트워크는 RFID를 통해 객체(상품)를 자동식별하고, 식별된 객체 정보를 인터넷을 통해 공유함으로써 실시간으로 객체를 추적 조회할 수 있는 글로벌 네트워크 시스템이다.

1) 개요

그림 1은 EPCglobal 네트워크 아키텍처를 나타낸 것이다. EPCglobal의 주요 구성요소는 EPC, Tag, Reader, ALE, EPCIS(EPC Information Service), Discovery Service, Local ONS(Object Name Service)가 있다. EPC는 RFID를 이용한 객체 식별과 EPCglobal 네트워크를 통한 객체정보 접근 및 교환을 위한 key 역할을 하는 정보이다.

ALE는 리더로부터 읽혀진 RFID 태그 정보를 애플리케이션 계층으로 전달하는 역할을 하는 일종의 미들웨어이다. 즉, 데이터를 수집하고 필터링하여 비즈니스 로직의 해석을 시작하도록 하는 의미 있는 이벤트를 생성하는 것이다. EPCIS는 객체에 대한 정보접근과 교환을 위한 표준 인터페이스로서, 객체 이벤트를 저장하여 객체정보를 여러 애플리케이션에서 사용할 수 있도록 해주는 EPCglobal 네트워크의 가장 중요한 구성요소 중 하나이다.

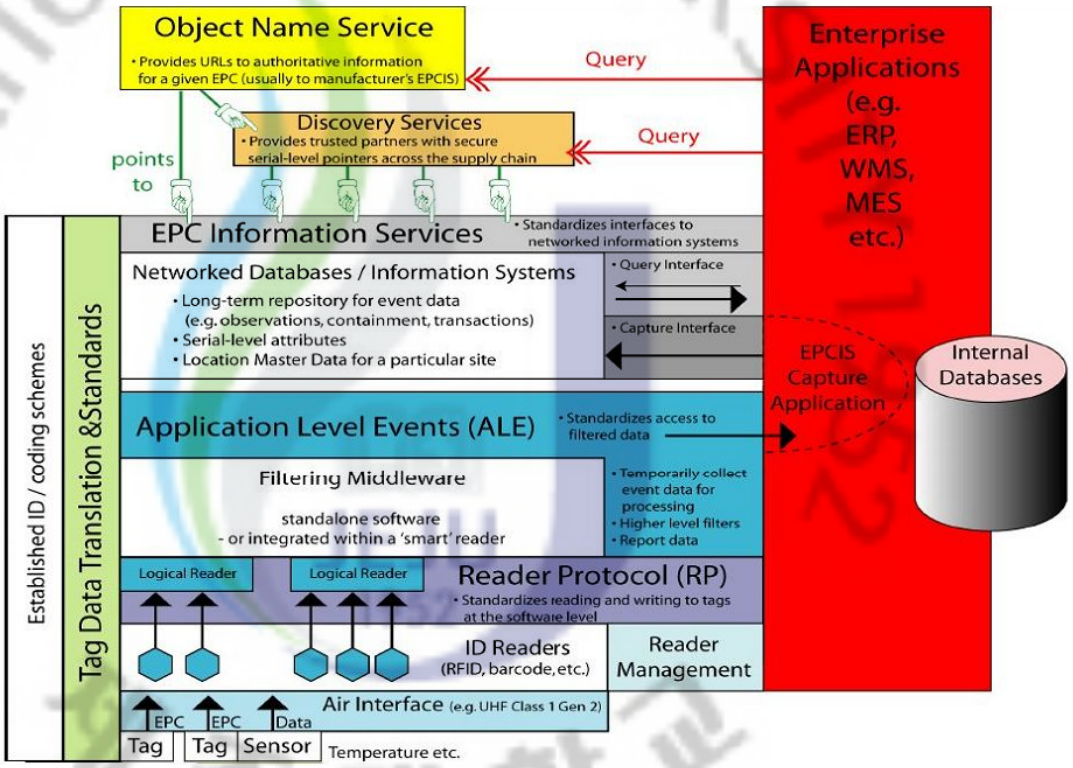

그림 1. EPCglobal Network Architecture

2) EPC Network 구성요소

(1) EPC(Electronic Product Code)

EPC는 MIT Auto-ID 센터에서 개발된 코드체계로 물리적 또는 가상적으로 존재하는 물품 또는 서비스에 고유한 일련번호를 부여하여 식별을 가능하게 해주는 코드이며, 이는 EPCglobal Tag Data Specification[7]에 의해 정의되어 있다. EPC 식별자는 메타 코드 체계로서 기존 체계와 새로운 체계간의 호환성을 보장하고 있다. EPC는 현재 사용되고 있는 바코드 체계뿐만 아니라 새롭게 개발된 코드 체계 모두를 수용하고 있다.

또한, 미래에 개발될 코드 체계까지를 고려한 확장성을 지원하며 사용목적에 따라 그림 2와 같이 다양한 코드체계(GID, GRAI, GIAI, SGLN, SSCC, SGTIN

등)가 존재하고 64비터, 96비트 등의 길이를 갖도록 정의되어 있다.

헤더 값 (2진수)	헤더 값 (16진수)	인코딩 체계	인코딩 길이 (비트)
0000 1000	08	SSCC-64	64
0000 1001	09	SGLN-64	64
0000 1010	0A	GRAI-64	64
0000 1011	0B	GIAI-64	64
0010 1111	2F	DoD-96	96
0011 0000	30	SGTIN-96	96
0011 0001	31	SSCC-96	96
0011 0010	32	SGLN- 96	96
0011 0011	33	GRAI-96	96
0011 0100	34	GIAI-96	96
0011 0101	35	GID-96	96
0011 0110	36	SGTIN-198	198
0011 0111	37	GRAI-170	170
0011 1000	38	GIAI-202	202
0011 1001	39	SGLN-195	195

그림 2. EPC 코드 체계

(2) 태그와 리더

RFID Tag는 기본적으로 상품코드인 EPC가 기록된 라벨 형태의 RFID IC칩과 전파를 송수신하는 안테나로 구성되어 있다. 태그는 메모리가 1bit인 태그부터 수 Kbit까지 태그 등 제조사에서 자유롭게 만들 수 있다.

EPCglobal에서는 64, 96, 128 bit 세 종류를 표준으로 정하였는데 그 중에서도 96bit가 가장 많이 활용될 것으로 예상하고 있다. 96bit를 사용하면, 약 2억 6800만 개의 회사(Domain manager), 그 회사의 1,600만종의 품종(Object class), 그 품종의 680억 개의 상품(일련번호)을 표현할 수 있다.

많은 정보를 태그에 기록하면 좋을 수도 있지만 태그 가격이 높아지기 때문에 태그에는 식별번호만 기록하고 그 회사나 기관의 정보시스템에 그 식별번호를 key로 하여 필요한 정보를 기록 유지하는 것을 권장하고 있다.

태그에 저장되는 데이터는 각 개별 상품에 대한 정보(EPC)이므로 그 구조가

통일되어야 하며, 이를 위해서는 표준이 매우 중요하다. 현재 표준화 된 태그 데이터의 구조는 다음과 같다.

Auto-ID 센터는 “Class 0″, ″Class 1 Generation 1″ 태그의 프로토콜을 개발했으며 ”Class 1 Generation 2″ 스펙은 EPCglobal에서 개발했다. “Class 0″은 ISO 18000-6A, ″Class 1 Gen 2″는 ISO 18000-6B에 대응된다. 하지만 ”Class 1 Gen 2″는 ISO 18000-6C의 표준으로 채택되었다. “Class 1 Gen 2″는 이전 두 클래스 규격의 장점을 취합하여 하나의 리더에서 ”Class 0″ 및 “Class 1″의 태그를 모두 읽을 수 있다.

Class 1 Gen 2의 메모리는 태그자체 정보 저장영역인 TID, EPC 저장용 영역인 EPC, 패스워드 저장용 영역인 RESERVED, 선택사항으로 End User가 임의로 사용할 수 있는 USER 영역으로 나누어진다.

EPCglobal Network에서 리더의 역할을 3가지로 정의하고 있는데, 장비로서의 리더, 그 리더와 다른 EPC Application 간의 데이터와 명령의 교환을 담당하는 리더 프로토콜(RP : Reader Protocol), 그리고 리더를 관리할 수 있는 기능 스펙인 리더 관리(RM : Reader Management)로 나눈다.

리더(Reader)[8]는 EPC 즉, 태그의 정보를 읽어 들이는 장치인 리더는 태그의 정보를 송, 수신 하고, 태그에서 수집된 정보를 관리 시스템으로 전송하며, 안테나, 제어장치, RF 장치, 통신장치로 구성된다.

리더 프로토콜은 EPC Application과 리더 사이에 존재하며 태그의 정보를 읽고, 쓰고, 비활성화 시키는 것과 같은 기능을 지원하는 역할을 수행한다. 리더 프로토콜은 Reader Protocol(RP) Standard, Version 1.1로 이미 EPCglobal에서 비준 되었다.

리더 관리는 리더 프로토콜을 이용하여 개별 RFID 리더의 설정, 준비, 모니터링과 경고 통보와 같은 표준화된 기능을 제공한다. 리더 관리는 Reader Management(RM) Version 1.0으로 EPCglobal에서 비준되었다.

이러한 리더는 일반 리더와 핸드헬드(hand held)형 리더, 그리고 복합기형 리더로 크게 나눈다. 핸드헬드형 리더는 안테나가 포함된 일체형이 일반적이며, 복합기형 리더는 단말기 또는 PDA(Personal Digital Assistants) cellular phone(핸드폰)에 리더와 안테나가 내장되거나 탈착 할 수 있는 리더를 최근 가장 많이 선

호한다.

리더는 응용 프로그램을 수행하는 관리 시스템(PC 혹은 기타 단말기)과 Serial(RS232-C, RS422 등) 혹은 랜 통신(TCP/IP)으로 연결되며, RFID 신호를 Encoding/Decoding하는 역할을 수행한다. 안테나는 RFID 수동형 또는 세미 능동형 태그에 전파를 송신하는 TX 모듈과 RFID 태그가 송신한 태그를 수신하는 RX 모듈로 구성되어 있으며 제어장치는 인코딩 및 디코딩, 데이터의 체크 및 저장, 태그 및 호스트와의 통신 등을 제어한다[9].

(3) ALE(Application Level Events)

ALE(Application Level Events)는 EPC 처리 시스템에서 RFID 태그를 인식한 리더가 Application 계층으로 데이터를 전달하는 역할을 하는 일종의 미들웨어이다.

ALE[10]는 클라이언트가 다양한 소스로부터 정제되고 통합된 EPC 데이터를 얻는 소프트웨어 인터페이스로 정의된다. ALE의 역할은 어플리케이션을 위한 인터페이스를 제공하고, 데이터 정제, 데이터 취합, 중복된 데이터 제거, 데이터 그룹화를 통해 태그 데이터를 처리하며 다양한 종류의 리더기를 지원하는 것이다.

ALE는 RFID 리더, 관리자, 클라이언트 애플리케이션을 포함하고 있는 환경에서 동작하게 되는데 동작원리를 살펴보면 다음과 같다. ALE 관리자는 RFID 리더의 환경을 설정하고 부분을 담당한다. 환경이 설정되면 EPC 데이터가 리더에서 ALE로 전달된다. 클라이언트는 수신하려는 데이터나 데이터를 갖고 있는 리포트 생성조건(ECSpec)을 ALE 인터페이스를 통해 지정하고 ALE로부터 ECReport의 형태로 정보를 제공 받는다.

EPCglobal Network에서 ALE 인터페이스의 중요한 역할은 데이터를 정제하고 카운팅하는 아키텍처 컴포넌트와 데이터를 사용하는 애플리케이션 간의 독립성을 보장하는 데 있다. 즉 클라이언트는 리더의 물리적 변화에 완전히 독립적으로 동일한 리포트 정보를 제공 받을 수 있다.

(4) EPCIS(EPC Information Service)

EPCglobal Network 구성원 간의 데이터 교환의 주요 수단이 EPCIS(EPC Information Service)[11]이다. EPCIS 데이터는 비즈니스(거래) 파트너가 직접 관장할 수 없는 위치에 있는 객체(상품, 박스, 팔레트 등)로부터 발생하는 객체 또는 트랜잭션에 대한 데이터를 얻기 위해 공유하는 정보이다. EPCIS는 정적 정보와 동적 정보를 가진다.

정적 정보는 물리 객체의 고유한 성격에 대한 데이터로서 변경되지 않는 클래스 레벨의 정보와 물리 객체마다 변경되는 인스턴스 레벨의 정보이다. 이를 통해 물리 객체의 동질성과 동시에 유일성을 제공한다.

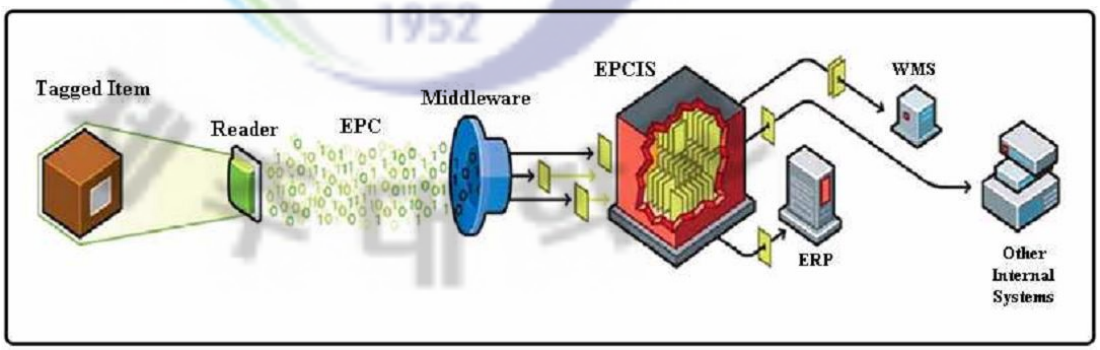

그림 3. EPCIS의 EPC 데이터 처리 구성도

(5) ONS(Object Naming Service)

EPCglobal Network 상에서 ONS[12]는 글로벌 검색 서비스는 제공하는 구성요소이다. 핵심적인 기능은 EPC에 대응되는 1개 또는 여러 개의 URI를 반환하는 것이고, 반환된 URI를 통해 EPCglobal Network 구성원은 객체에 대한 부가적인 정보를 획득할 수 있다.

EPCglobal Network 구조가 기존의 인터넷 표준과 인프라를 바탕으로 하므로 ONS에 질의하기 위해서는 DNS(Domain Name Service)를 사용한다. ONS는 정적(static) 서비스와 동적(dynamic) 서비스를 동시에 제공하는데, 정적 서비스는

객체의 생산자에 의해 관리되어지는 정보를 얻을 수 있는 URL을 제공하고, 동적 서비스는 객체가 공급사슬(Supply Chain) 상에서 이동함에 따라 발생하는 일련의 객체 관리 정보를 저장 및 관리한다.

최상단에 전체 ONS를 관장하는 Root ONS가 존재한다. 최상위 Root ONS의 상위에는 DNS가 존재해서 Root ONS까지의 경로 서비스를 제공한다. 최상의 Root ONS 하위에는 EPC에 따른 ONS들이 존재하고 개별 EPC 코드체계에 따른 EPCglobal Network 구성원의 Local ONS에 대한 정보를 관리한다. 각각의 Local ONS들은 해당 구성원 단위 정보를 Zone File을 관리하고 서비스하는 역할을 수행한다. 이러한 Zone File은 Root ONS에서도 사용되어 질수 있다. 또한, ONS는 Local Cache 기능을 제공하는데, 이 기능은 잦은 Query가 발생하는 Record에 대해 ONS 자체에서 이를 관리하여 동일한 Query가 발생할 경우 Cache에 있는 URI를 제공한다. 이 기능으로 ONS Network 상에서 Network 부하를 감소시켜 시스템의 자원 사용을 감소시킨다.

(6) EPC Discovery Service

EPC Discovery Service는 TBD(To Be Determine)상태이며, 현재 개념 단계에서 논의되고 있지만, 전 세계 대표 ONS 서버(root ONS)를 운영하고 있는 Verifying에서 서비스하고 있는 EPC Discovery Service와 EPCglobal에서 논의되어지고 있는 개념을 통해서 EPC Discovery Service의 형태를 짐작할 수 있고, ONS도 크게는 Discovery Service의 일부로 보고 있다.

Discovery Service는 Dynamic Discovery Service와 Static Discovery Service로 설명되는데, Dynamic Discovery Service는 EPC Discovery Service로서 변화하는 EPC의 이벤트 데이터의 위치를 검색하는 서비스이고, Static Discover Service는 ONS를 의미하며, 고정된 주소 값을 서비스하는 것이다.

3) ALE(Application Level Events)

EPCglobal은 RFID 미들웨어와 관련하여 Auto-ID 센터에서 개발한 “Savant" 라는 용어를 사용하여 ”The Savant Specification Version 0.1"과 “The Savant Specification Version 1.0"을 2002년과 2003년에 발표하였고, 2005년 11월에는 Savant 대신 ALE라는 용어를 사용하여 ”The Application Level Event(ALE) Specification Version 1.0"을 발표하였다[10].

ALE는 상위 레이더(layer)인 Application과 하위 레이어 인 Reader의 사이에 위치하며, 그 구성은 그림 4와 같다. ALE는 Applications에 의하여 전달되는 Request에 의하여 Reader에 읽힌 태그의 EPC를 Filtering과 Grouping을 적용하여 Reports를 생성 후 요구한 Applications에 전달한다.

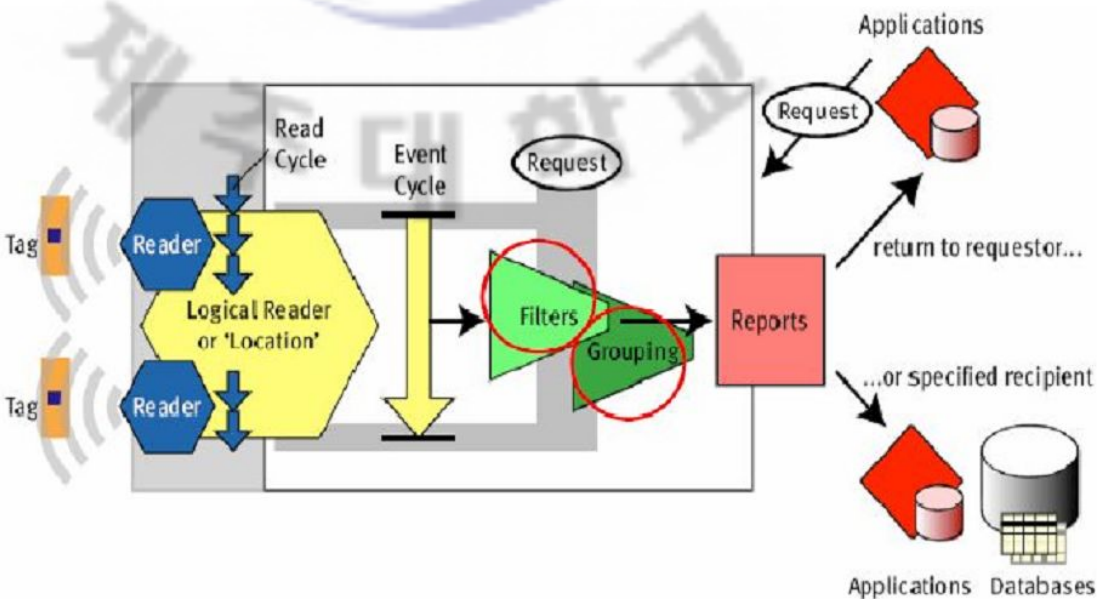

그림 4. ALE 개념도

ALE는 클라이언트가 다양한 소스로부터 정제되고 통합된 EPC 데이터를 얻는 소프트웨어 인터페이스로서, 애플리케이션을 위한 인터페이스를 제공하고 데이터 정제, 데이터 취합, 중복된 데이터 제거, 데이터 그룹화를 통해 태그 데이터를 처리하며 다양한 종류의 리더기를 지원한다.

(1) ALE 역할

ALE는 리더 등과 같은 한 개 이상의 데이터 소스로부터 EPC를 받고(receiving), 일정 시간 간격 동안 데이터를 축적(accumulating), 중복 EPC와 관계없는 태그를 필터링(filtering), 데이터를 분류하기 위한 그룹핑(grouping)과 그룹 태그 개수의 카운팅(counting), 다양한 형식의 리포팅(reporting) 등의 과정을 수행하며, 여기서 기술된 인터페이스와 인터페이스의 함축된 기능을 의미한다. 이런 역할은 미처리된 EPC 데이터를 얻는 인프라 컴포넌트, 수집한 데이터를 가공(filtering, grouping, counting)하는 아키텍처 컴포넌트, 데이터를 사용하는 애플리케이션 컴포넌트 간의 독립성을 보장한다.

EPCglobal ALE 표준에서는 내부적인 처리 방법에 대한 구현은 명시하지 않고, 상위 레벨인 클라이언트가 어떤 EPC 데이터에 관심이 있는지 선언적 방법으로 지정한다. EPC Network에서 ALE는 다른 레이어(Layer)와의 다음과 같은 차이점을 지닌다.

- ALE 인터페이스는 EPC 데이터에 대하여 실시간 프로세싱을 지향하며, 인터페이스를 통한 요청에 대하여 EPC 데이터를 처리할 영구 기억장치가 없다.

- ALE 인터페이스를 통해 전달된 이벤트는 비즈니스적인 의미(semantic) 없이 대상, 장소 및 시간에 대한 정보만 처리한다.

즉, ALE 계층은 데이터를 수집하여 이 데이터를 비즈니스 로직이 해석하는데 적절한 출발점이 되도록 의미 있는 이벤트로 필터링하는 메커니즘이다.

(2) ALE 동작

ALE에서는 리드 사이클(Read Cycle), 이벤트 사이클(Event Cycle), 리포트(Report)로 크게 3가지의 데이터 구조를 사용하여 실행된다[10].

● 리드 사이클

리더와 상호 작용하는 최소 단위(unit)이다. 가공되지 않은 태그 리드 계층(Raw Tag Read Layer)과 ALE 계층 사이의 인터페이스에서 리드 사이클의 출력은 ALE 계층의 입력이 된다. ALE 관점에서는 리드 사이클은 하나의 EPC 세트만을 보유한 1회 이벤트이다.

● 이벤트 사이클

1개 이상의 리더로부터 수집된 1개 이상의 리드 사이클을 포함하며, 응용 관점에서 한 단위로 취급된다. ALE 인터페이스와 응용 간 상호작용의 최소 단위이다.

● 리포트

ALE 계층에서 클라이언트로 전달되는 이벤트 사이클에 관한 데이터이다. 리포트는 ALE 계층의 출력으로 애플리케이션 비즈니스 로직 계층으로 전달된다.

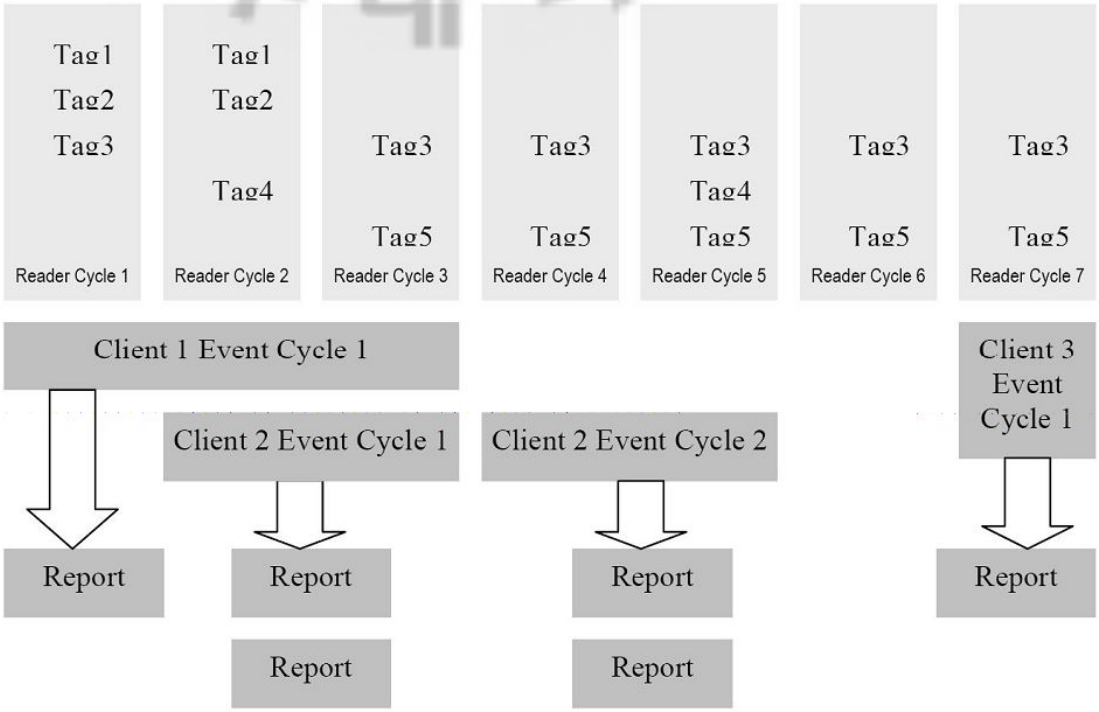

그림 5. Reading API - Event Cycles

그림 5에서는 Read Cycle, Event Cycle, Report의 관계를 설명하고 있다. 한 개 리더에서 발생하는 리더 사이클을 나타내지만 실제로는 해당 이벤트 사이클은 한 개 이 상 리더에서 발생한 리드 사이클을 수집한다[10].

그림 5에서 표시된 것처럼, 어느 시점에서는 동작(active)하는 이벤트 사이클이 한 개 이상이 있다. 다수의 이벤트 사이클은 다른 리드 사이클에 따라 시작, 종료되며 임의적으로 교차된다. 이벤트 사이클은 동일 리더, 다른 리더 또는 리더의 임의 교차 부분으로부터 데이터를 수집한다. 여러 동작 이벤트 사이클은 동시에 여러 요청을 하는 클라이언트나 독립된 클라이언트에서 발생한다. 그러나 모든 경우에 있어서, 동일 리드 사이클은 해당 리더로부터 데이터를 요청한 모든 동작 이벤트 사이클이 공유한다. 해당 리더에서 수집된 해당 리드 사이클의 태그 집합을 S로 표시하면 위 그림 5의 태그 집합은 아래와 같다.

S_1 = {Tag1, Tag2, Tag3}, S_2 = {Tag1, Tag2, Tag4}, S_3 = {Tag3, Tag5} …

클라이언트는 1개 이벤트 사이클을 하나의 단위로 취급한다. 클라이언트는 태그를 수집할 리더를 설정하게 되는데 이벤트 사이클은 해당 리더의 리드 사이클을 수집한다. 수집된 리드 사이클들에서 중복된 것은 제거하고 이벤트 사이클을 생성한다. 즉, 수집된 리드 사이클의 합집합을 이벤트 사이클로 간주하면 되는 것이다. 예를 들어, 그림 5에서 클라이언트 1의 이벤트 사이클 1은 아래와 같이 나타낼 수 있다.

$E_{C1E1} = S_1 \cup S_2 \cup S_3$ = {Tag1, Tag2, Tag3, Tag4, Tag5}

클라이언트는 리포트를 통해 이벤트 사이클 정보를 얻는다. 이에 대한 구현은 다양한 방법이 존재 할 수 있지만 클라이언트는 제공된 ALE API를 바탕으로 적합한 형태의 리포트를 요구하기만 하면 되고 내부의 구현을 알 필요가 없다.

(3) ALE API와 미들웨어 상태

ALE 표준은 클라이언트와 미들웨어가 서로 상호작용을 할 수 있게 인터페이스를 API로 제공한다. ALE API는 기본적으로 RFID 이벤트 사이클을 관장하는 ECSpec 관리에 주된 목적이다.

표 1. ALE APIs

API	Parameter	Descriptions
define	specName, ECSpec	ECSpec 명세를 미들웨어에 등록
undefine	specName	ECSpec 명세를 미들웨어에서 제거
subscribe	specName, notiURI	EventCycle 구동, 비동기적 동작
unsubscribe	specName, notiURI	ECSpec의 subscribe 서비스 제거
poll	specName	EventCycle 구동, 동기적 동작
immediate	ECSpec	ECSpec을 미들웨어에 정의 후 poll 동작을 수행 동작 완료된 ECSpec 명세를 삭제 (define -> poll -> undefine)
getECSpec	specName	지정한 ECSpec 명세를 요청
getECSpecNames		미들웨어에 정의되어 있는 ECSpec 이름 리스트
getSubscribes	specName	지정한 ECSpec의 subscribe 서비스 요청자 리스트
getStandardVersion		구현 기준 ALE 스펙 버전
getVendorVersion		구현 기준 벤더 버전

일반적으로 하나 이상의 클라이언트를 표 1과 같은 ALE 인터페이스에 있는 메소드를 호출한다. ALE 미들웨어는 그에 따른 적절한 동작을 수행하며 결과를 만들어 반환하는데, 여기에는 메소드 호출이 끝나야 결과를 반환하는 동기적인 특성을 갖는 poll, immediate 메소드가 정의되어 있다.

또한 비동기적으로 결과를 받을 수 있는 메소드로서 notificationURI를 파라미

터로 받는 subscribe가 있다. 이 메소드는 호출과 동시에 바로 반환되며, 이후 미들웨어는 해당 연산을 수행한 후 그 결과를 URI(Uniform Resource Identifier)에 명시되어 있는 클라이언트로 전달한다. 즉, 호출은 미리 끝나게 되고 미들웨어는 연산을 수행하는 대로 결과를 보내게 된다. 그림 6은 ALE 표준에 정의되어 있는 여러 메소드 및 메소드 호출 시 시스템의 상태가 어떻게 변하는지를 나타내는 상태 다이어그램이다.

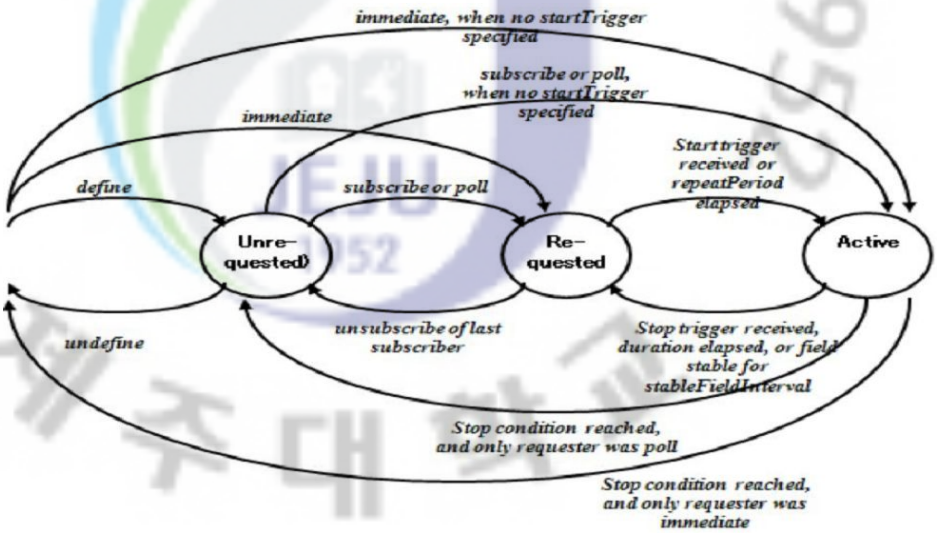

그림 6. ECSpec의 상태 다이어그램

ALE API 호출에 따라 ECSpec은 그림 6과 같은 ECSpec 명세가 정의되어 있는 ALE 미요청(Unrequested) 상태, 서비스 요청 후 이벤트 사이클이 동작하기 전까지의 대기(Requested) 상태, 명세에 의하여 Event Cycle이 시작된 동작(Active) 상태로 구분된다[10].

4) ALE 주요 구성요소

가장 핵심적 데이터 요소는 ECSpec과 ECReport 이다[2, 13]. ECSpec에는 클라이언트가 받고 싶어 하는 데이터를 지정할 수 있는 방법을 제공하는데, 데이터를 받고자 하는 리더 이름과 이벤트 사이클이 리더 사이클에 어떤 식으로 대응하는지도 지정할 수 있다. 이벤트 경계를 지정하는 ECBoundarySpec, 관측 결과를 필터링하는 ECFilterSpec, 다시 이 결과를 분류하는 방법을 제공하는 ECGroupSpec, 어떤 데이터를 어떻게 보고해야 할지 지정하는 ECReportSpec등이 있다[2, 13]. RCReprot는 ECSpec 기반으로 이벤트 사이클에 걸쳐 수집된 결과를 전달하는 보고서 양식이다.

(1) ECSpec

ECSpec[10]은 클라이언트가 ALE 미들웨어로부터 제공받고자 하는 정보의 상세 스펙으로써 다음 그림 7과 같이 수집대상(Logical Reader, 논리 리더)인 List 데이터형의 Reader, Event Cycle의 동작 조건인 ECBoundarySpec 데이터형의 boundaries, 전송받고자하는 데이터의 필터와 리포터 형식인 데이터형의 ReportSpec 등의 복잡한 데이터 형이다.

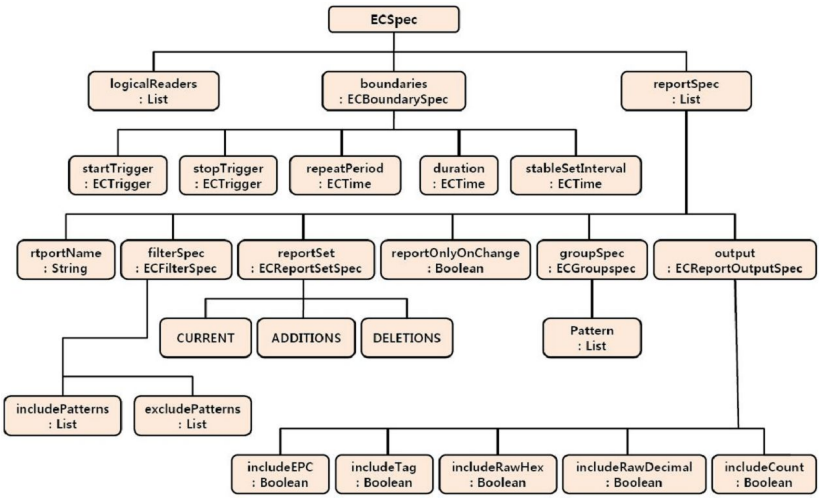

그림 7. ECSpec 명세

ECSpec은 하나의 이벤트 사이클과 대응되는데, ECSpec이 Active한 상태여서 현재 태그 정보를 수집하고 있는 상태일 때가 이벤트 사이클이 종료되는 시점에 보고된다.

표 2는 이벤트 사이클의 시작과 종료를 나타내고 있다. Event Cycle의 시작은 주기적으로 이루어질 수도 있으며, trigger 방식에 의해서 외부에서 주어지는 시점에 이루어질 수도 있다. 이벤트 사이클의 종료 역시 주기적으로 이루어지거나 trigger 방식에 의해서 외부에서 주어질 수도 있다. 이것은 ECBoundarySpec에 의해서 설정된다. 시작 조건은 ECBoundarySpec의 repeatPeriod와 startTrigger이고 종료 조건은 ECBoundarySpec의 duration과 stopTrigger과 그리고 stableSetInterval이다.

표 2. Event Cycle의 시작과 종료

동작 모드	시작	종료	관련 API
Trigger	Start Trigger 발생	stopTrigger duration stableSetInterval unrequested	Subscribe Poll Immediate
Period	RepeatPeriod 값에 의해 자동 실행	stableSetInterval duration unrequested	Subscribe Poll Immediate

표 3은 ECSpec의 자료구조로 이벤트 사이클의 시작과 끝을 결정하기 위한 규칙과 그 이벤트 사이클로부터 만들어야 하는 보고서 규격을 ECSpec을 통해 지정한다.

표 3. ECSpec 자료구조

속성	타입	설명
Readers	List	논리적 리더의 목록
Boundaries	ECBoundarySpec	이벤트의 시작과 끝을 결정하는 방법을 기술
reportSpecs	List	이벤트 사이클 결과를 반환하는 방법을 지정한 ECReportSpec 목록

표 4는 ECReportSpec의 자료구조로 이벤트 사이클 실행 결과로 반환되는 보고서를 기술하며, 보고해야 할 EPC 데이터에 대한 규칙을 제공한다.

표 4. ECReportSpec 자료구조

속성	타입	설명
reportName	String	리포트 명
reportSet	ECReportSetSpec	어떤 EPC 집합이 필터링용으로 입력되는지를 지정
Filter	ECFilterSpec	리더 관측 결과를 필터링하는 부분
Group	ECGroupSpec	필터링된 EPC를 분류하는 방법 지정
Output	ECReportOutputSpec	필터링 및 분류가 끝났을 때 EPC를 보고하는 방법을 지정

(2) ECReports

ECReports[10]는 ECSpec의 명세에 따라 동작한 이벤트 사이클에 대한 출력 결과로써, Event Cycle의 동작정보와 수집된 데이터에 대하여 다음 그림 8과 같이 클라이언트가 요청한 리포트 형식에 맞게 생성된다.

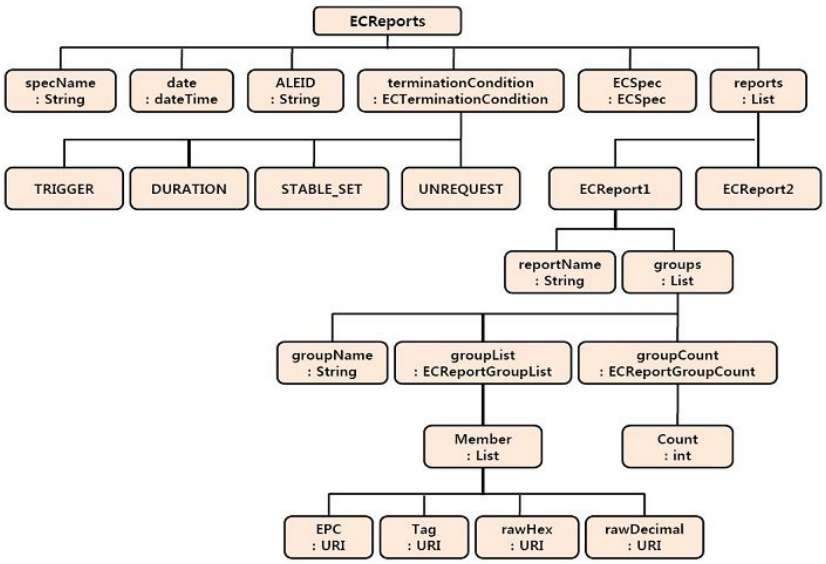

그림 8. ECReports 명세

서비스 명세인 ECSpec에 따라 EventCycle의 동작과 ECReports 형식이 결정된다. 표 5는 ECReprts의 형식을 결정하는 명세인 ECReportSpec[10]의 항목에 따른 ECReports의 처리 내용을 정리한 것이다.

표 5. ECReportsSpec과 ECReports 관계

ECReportSpec	ECReports 처리 내용
reportSet	CURRENT - 설정된 값에 따라 현태 읽고 있는 모든 태그 수집
	ADDITIONS - 전 이벤트 사이클 데이터를 기준으로 추가된 태그 수집
	DELETIONS - 전 이벤트 사이클 데이터를 기준으로 삭제된 태그 수집
group	설정된 패턴 개수 + 1만큼 그룹이 생성 단, 필드 패턴이 [x]이면, 상이한 값만큼 그룹 생성
output	그룹에 속하는 멤버 태그정보의 개수 표기 여부 지정한 EPC 표현방식에 따라 멤버의 태그 정보 표현이 결정
reportIfEmpty	수집 정보가 없고, true 설명이면 리포트 미생성
reportOnlyOnChage	true 이면 이전 이벤트 사이클의 태그 셋과 수집결과가 다르면 리포트 생성하지 않음 reportSet과 output 수행 전에 체크 됨

표 6은 ECReport의 자료구조로 하나의 이벤트 사이클로 만들어지는 단일 보고서로서, ECReport에 들어가는 데이터는 ECReportGorup 인스턴스로 분류된다.

표 6. ECReport 자료구조

속성	타입	설명
reprotName	String	리포트명
Group	List	정제, 분류된 데이터를 저장하는 ECReportGroup 목록

표 7은 ECReports의 자료구조로 이벤트 관리자에게서 출력되어 클라이언트로 전달되는 데이터 구조를 기술하며 개별 ECReport의 리스트를 포함한다.

표 7. ECReports 자료구조

속성	타입	설명
specName	String	ECSpec 명
Group	ECGroupSpec	데이터 분류 방법을 기술
totalMilliseconds	Long	이벤트 사이클 수행 시간
Reports	List	단일 보고서인 ECReport 목록

2. RFID 미들웨어

1) Sun's Java System RFID Software

SUN의 Java System RFID Software[14]는 EPCglobal의 EPC Network Architecture[6]의 한 구성 요소인 SAVANT 규격을 기반으로 구현된 RFID 기반 소프트웨어(RFID infrastructure software)이다. Java System RFID 소프트웨어는 두 종류의 소프트웨어 인프라 컴포넌트, 즉 Java System RFID Event Manager와 Java System RFID Information Server로 구성된다. 그림 9는 Sun's RFID Solution Architecture 이다.

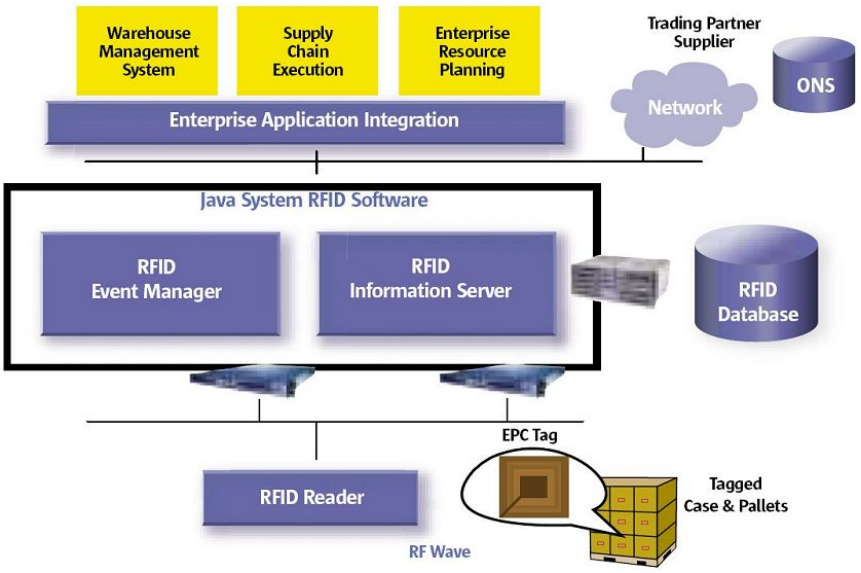

그림 9. Sun's RFID Solution Architecture

RFID Event Manager는 한 개 이상의 판독 장치에서 나온 태그 스트림이나 센서 데이터(이벤트 데이터)를 처리할 수 있도록 설계되었으며, 해당 애플리케이션으로 데이터를 전송하기 전에 데이터를 필터링하고 수집할 수 있는 기능을 갖추고 있다. RFID Event Manager의 주요 구성요소는 다음과 같다.

● 디바이스 어댑터

RFID Event Manager와 서로 다른 여러 회사에서 생산한 RFID 디바이스들과의 통신 및 상호작용을 가능하게 한다.

● 필터

태그 개체에 의해 일정하게 생성된 RFID 노이즈에서 나온 유용한 비즈니스 이벤트를 판독하는 역할을 한다. 미확인 이벤트 추정(event smoothing), 이벤트 배칭(batching), 이벤트 수정(태그인, 태그아웃) 그리고 이벤트 통과/차단을 위한 표준 필터가 제공된다.

● 로거(logger)

RFID 및 다른 종류의 이벤트 데이터 외부 시스템에 통지를 한다. 지원 가능한 인터페이스로는 Java Message Service(JMS) 큐, XML, HTTP, SOAP 메시지 그리고 표준 파일 시스템 등이 있다.

● 엔터프라이즈 게이트웨이

엔터프라이즈 애플리케이션(Enterprise Application) 이 RFID Event Manager로부터 데이터를 요청할 수 있도록 범용 인터페이스를 제공한다.

● 페일 오버

RFID Event Manager가 Java 및 JINI 프레임워크 아키텍처를 기반으로 하고 있으므로 처음부터 서비스 페일 오버 기능을 완벽하게 내장하고 있다.

RFID Event Manager는 관리의 용이성, 가용성 및 확장성(수평/수직 확장)에 영향을 주지 않고 유연하게 배치 기능을 수행할 수 있도록 설계되었다.

한편, RFID Information Server는 EPC 관련 데이터 획득 및 질의를 위한 인터페이스 역할을 하는 J2EE 애플리케이션이다. RFID Information Server는 RFID Event Manager에 의해 제공된 비즈니스 이벤트에 대한 외부 액세스 및 컨텍스트 룩업 서비스(context lookup server)를 제공한다. 그리고 RFID Event Manager에서 나온 데이터는 RFID Information Server로 전달되어 저장된 다음, 이 정보를 필요로 하는 모든 애플리케이션에 안정적으로 제공한다.

2) ConnecTerra/BEA

ConnecTerra의 RFTagAware[15]는 RFID 응용을 개발하고, 배치하고(deploying), 관리하기 위한 광범위한 소프트웨어 기반을 제공한다. 실시간 RFID 태그 데이터에 대해 필터링하기 위해 ALE API(Application Programming Interface)를 이용한다.

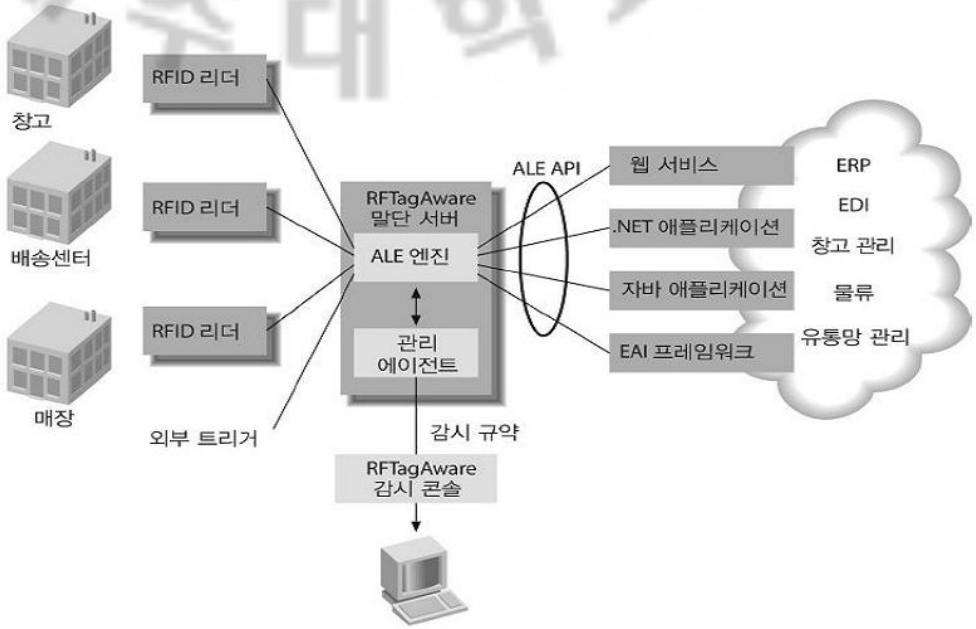

그림 10. ConnecTerra의 RFTagAware 미들웨어 플랫폼

RFTagAware 말단 서버(Edge Server)는 처리되지 않은 태그 정보를 가공한

다음 임의의 개수에 질의를 바탕으로 등록한 애플리케이션에 결과를 전송한다. 이 때 질의 및 등록 애플리케이션의 개수에는 제한이 없다. 다른 애플리케이션에서 사용하거나 제거하는 것이 가능하다. 말단 서버에서는 장치 사용을 질의 내용을 수집하는데 최적화시킴으로써 하드웨어 사용을 최적화 시키는 일까지 맡아서 해준다.

말단 서버의 주 구성 요소로는 필터링 및 수집 엔진과 장치 관리 에이전트를 들 수 있다. RFTagAware 말단 서버에서는 다양한 리더 및 프린터와 리더 제어용 trigger로 쓰이는 다양한 센서 입력 인터페이스로도 제공한다. 그리고 말단 서버 작동을 원격으로 직접 확인할 수 있는 관리 콘솔은 물론이고, 말단 서버 및 장치를 관리하고 감시하기 위한 API도 제공한다.

3) REMS(RFID Event Management System)

REMS[16]는 ETRI(Electronics and Telecommunications Research Institute)의 자동식별 미들웨어의 확장 버전으로 구조는 그림 11과 같다. REMS를 구성하는 구성요소로는 RIC(Reader Interface Component), LRIC(Logical Reader Interface Component), ALE(Application Level Event), EMC(Event Management Component), TDT(Tag Data Translation)로 구성된다.

RIC는 다수의 벤더에 의해 제공되는 RFID 리더들을 공통적인 인터페이스를 통해 미들웨어 소프트웨어와의 연계를 지원하는 컴포넌트이다.

LRIC는 물리적인 리더들을 논리적인 리더로 묶어서 실제적인 물리리더에 변경이 일어나더라도 상위계층에 영향을 미치지 않고, 상위계층에서 논리적인 리더로 접근할 수 있도록 해주는 컴포넌트 이며, ALE는 LRIC로부터 ECSpec(Event Cycle Specification) 이라 불리는 주기 동안 수집된 RFID 태그 ID를 필터링, 그룹핑 후 정해진 XML 형식으로 태그 ID 목록을 인코딩하여 다양한 프로토콜로 바꾸어 응용 프로그램에 전달하는 컴포넌트이다.

EMC는 사용자가 정의한 다양한 필터링 모듈 및 그들 간의 조합을 통한 일련의 정보 흐름을 정의하고 이에 기반하여 데이터 정제과정을 거치고 최종적으로

정보의 수용자인 RFID 응용 애플리케이션으로 전달하는 컴포넌트이며 RFID 태그에 저장된 바이너리 형식으로 전달되는 ID 정보를 URL 형태로 변환한다.

TDT는 ALE 규격에 적합한 필터링 및 그룹핑을 지원하기 위해 식별자 변환 과정을 독자적인 인터페이스와 엔진을 가지고 독립적인 프로세스로 처리하는 시스템이다.

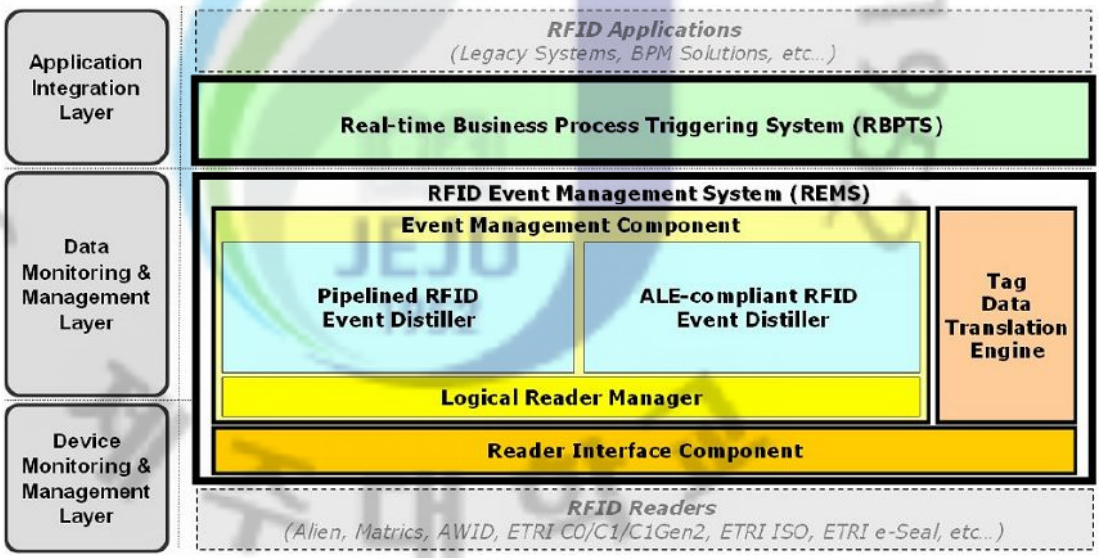

그림 11. REMS 구조

4) CARU(Context-Aware RFID Middleware System for Ubiquitous)

CARU[17]는 RFID 리더에서 생성하는 태그 데이터 및 센서 데이터를 수집, 필터링하여 의미 있는 정보를 요약해서 응용 시스템에 전달하는 미들웨어이다. CARU는 RFID 리더에서 데이터를 수집하는 Edge 미들웨어와 수집한 데이터를 사용자가 요구하는 의미있는 정보로 가공하는 ALE로 구분되며, EPCglobal에 제안하는 표준 스펙인 ALE 인터페이스를 제공한다.

또한 다양한 RFID/USN 환경에서 유비쿼터스 서비스 제공을 위해 이기종 RFID 리더에서 생성되는 태그 데이터 및 센서 데이터를 수집, 필터링하여 의미 있는 정보를 요약해서 응용 소프트웨어 및 사용자 요구사항에 필요한 정보를 제공하며, 여러 환경에 RFID 시스템 도입에 있어 소프트웨어 레벨에서 성능을 효

율적으로 지원하도록 표준에 기반하여 개발되었다. TCP/IP, RS232C 등 다양한 프로토콜을 지원하고 벤더별 인터페이스를 제공함으로써, RFID 기반 응용 개발 업무에 효과적인 활용이 가능하고, RFID 리더뿐만 아니라 일반적인 센서 프로토콜을 지원할 수 있는 인터페이스 기능을 갖고 있다.

CARU는 웹 기반에서 리더 관리 및 사용자 정의에 따른 요청을 처리할 수 있는 도구를 제공하며, 웹 브라우저를 통해 리더의 등록 및 삭제 등 리더를 관리할 수 있으며, 사용자에 요청에 따른 데이터 필터링, 데이터 전송 등 다양한 사용자 정의에 따른 요청을 효율적으로 처리할 수 있는 기능을 제공한다.

본 논문에서는 2009년 8월 3일 EPCglobal의 ALE 인증을 획득한 CARU 미들웨어를 미들웨어 분산 환경에 구축하여 대량 RFID 데이터 처리를 위한 부하 분산 방법을 실험하였다.

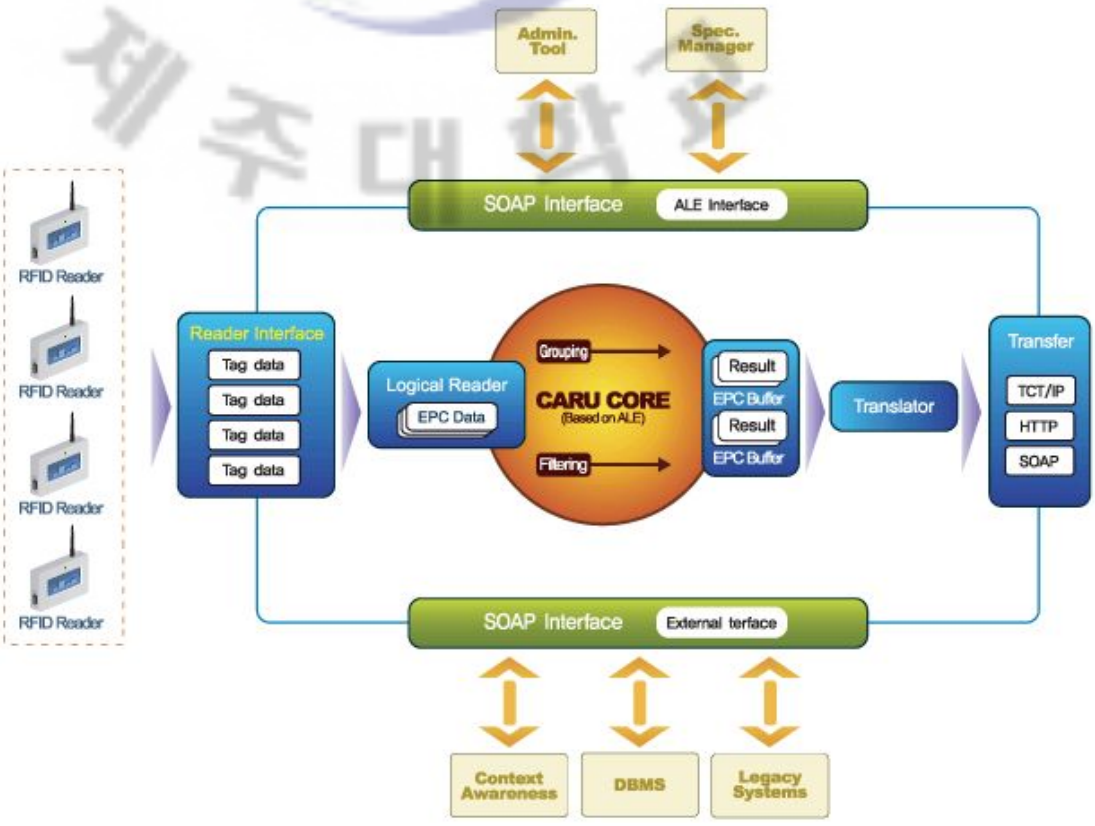

그림 12. CARU System 구조

3. 부하 분산 알고리즘

1) 라운드 로빈(RR)

라운드 로빈[18]은 그림 13과 같이 각 요구를 실제 서버(real server)에 풀에 순서대로 보낸다. 이 방식은 실제 서버들을 접속의 수나 응답시간, 네트워크 상황을 고려하지 않고 모두 동일한 것으로 취급한다.

라운드 로빈 DNS도 이와 같이 동작 하지만 약간 다르다. 라운드 로빈 DNS는 하나의 도메인(domain)을 다른 IP 주소들로 분산 시킨다. 작업 할당 기초(scheduling granularity)가 호스트 기반이라 DNS의 캐싱은 알고리즘의 효과를 방해한다. 이는 실제 서버들 사이에서 동적 부하 불균형의 발생할 수 있다. 가상 서버의 작업 할당 기초는 네트워크 접속기반이고, 좋은 작업 할당 기초 덕에 라운드 로빈 DNS 보다 우수하다.

실제로 서버의 사양과 네트워크 상태가 동일하다면 가장 단순하고 효율적이다.

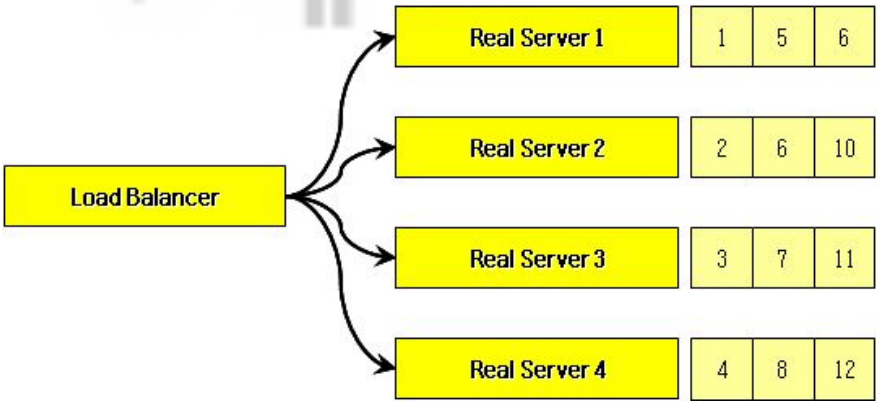

그림 13. 라운드 로빈

2) Weighted-라운드 로빈(WRR)

가중치 기반 라운드 로빈[19]은 성능이 다른 실제 서버들로 이루어진 클러스터에서 부하 균형에 유용하다. 각 서버들에 처리 용량을 나타내는 정수의 가중치가

할당 된다. 다른 서버보다 성능이나 기타 환경이 나아 더 많은 요청을 전달하고 싶을 때 해당 서버에 가중치를 주어 더 많은 요청을 처리할 수 있도록 하는 방식이다. 가중치 기반 라운드 로빈을 사용 사면서 실제 서버에서 네트워크 접속을 셀 필요가 없고 동적 알고리즘보다 과부하가 적으므로 더 많은 실제 서버를 운영할 수 있다. 그러나 요청에 대한 부하가 매우 많을 경우 실제 서버 사이에 동적인 부하 불균형 상태가 생길 수 있다.

실제 서버 A, B, C, D의 가중치가 각각 1, 2, 3, 4 라면, 그림 14와 같이 작업 할당 순서는 A->B->C->D->B->C->D->C->D->D 가 된다.

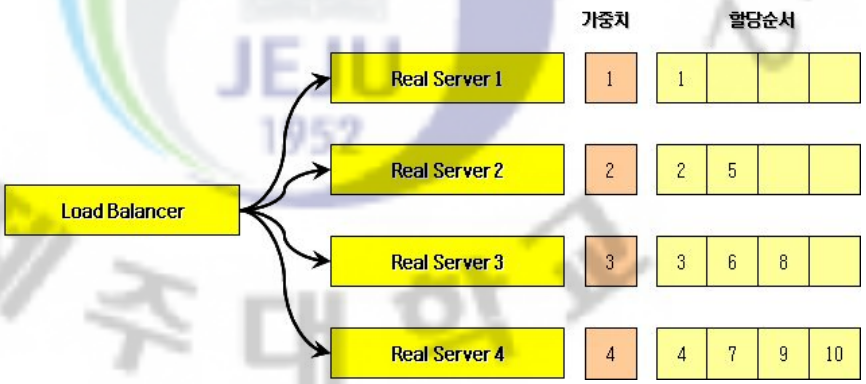

그림 14. 가중치 기반 라운드 로빈

3) Least-Connection(LC)

최소 접속[20]은 접속이 가장 적은 서버로 요청을 전달하는 방식을 말한다. 각 서버에 접속한 숫자를 세는 동적인 알고리즘 중 하나이다. 비슷한 성능의 서버로 구성된 가상 서버는 아주 큰 요구가 한 서버로만 집중되지 않기 때문에, 접속부하가 매우 큰 경우에도 아주 효과적으로 분산을 한다.

가장 빠른 서버에서 더 많은 네트워크 접속을 처리할 수 있다. 그러므로 다양한 처리 용량을 지닌 서버로 구성했을 경우에도 훌륭하게 작동한다는 것을 한눈에 알 수 있을 것이다. 그렇지만 실제로는 TCP의 TIME_WAIT 상태 때문에 아주 좋은 성능을 낼 수는 없다. TCP의 TIME_WAIT는 보통 2분이다. 그런데 접속자가 아주 많은 웹 사이트는 2분 동안에 몇 천 개의 접속을 처리해야 할 경우

가 있다. 서버 A는 서버 B보다 처리 용량이 두 배일 경우 서버 A는 수천 개의 요청을 처리하고 TCP의 TIME_WAIT 상황에 직면하게 된다. 그렇지만 서버 B는 몇 천 개의 요청이 처리되기만을 기다리게 된다. 그래서 최소 접속 스케줄링을 이용할 경우 다양한 처리 용량을 지닌 서버로 구성되었을 경우 부하분산이 효율적으로 되지 못할 수도 있다.

4) Weighted-Least-Connection(WLC)

가중치 기반 최소 접속[21]은 최소 접속 스케줄링의 한 부분으로서 각각의 실제 서버에 성능 가중치를 부여한다.

가중치가 있는 최소 접속은 다음과 같이 작동한다.

n개의 실제 서버가 있는 경우 각 서버 I는 가중치 Wi(i = 1, ... , n)를 가진다고 가정하자. 서버 I의 활동 접속(active connection)은 Ci(i = 1, ... , n)이고 모든 접속은 Ci(i = 1, ... , n)의 합이다. 서버 j로 가는 네트워크 접속은 아래와 같다.

(Cj/ALL_CONNECTIONS)/Wj = min{(Ci/ALL_CONNECTIONS)/Wi}(i=1, ... , n)

이 비교에서 ALL_CONNECTIONS는 상수이므로 Ci를 모든 접속으로 나눠줄 필요가 없다. 그러면 다음과 같이 최적화될 것이다.

Cj/Wj = min { { Ci/Wi } (i = 1, ... , n) }

5) 기타 (LBLC/LBLCR/DH/SH)

Locality-Based Least-Connection[22]은 Proxy 서버와 같은 보통 캐쉬서버 클러스터링에서 많이 사용하는데 목적지 IP 부하분산의 경우 서버에 과부하(active connection > weight)가 걸려있고, 부하가 적게 걸린 다른 서버가 있을 경우 부하가 적게 걸린 다른 서버들 간에 WLC 방식으로 부하를 분산한다.

Locality-Based Least-Connection with Replication[22]은 LBLC와 같지만 개별 서버가 아닌 서버셋(Server Set)이란 것을 설정한다. 만일 특정 서버 셋의 모든

실제서버가 과부하일 경우 다른 서버 셋에서 가장 연결수가 적은 실제서버로 서버 셋에 추가 시킨다.

Destination Hashing[22]은 목적지 IP 주소의 정적 해쉬에서 서버를 찾아 연결해 주며, Source Hashing[22]은 소스 IP 주소의 정적 해쉬에서 서버를 찾아 연결해준다.

4. 부하 분산을 위한 유전자 알고리즘

1950년대 후반 J. Holland[23]는 진화 생물학자인 Fisher의 저서 "The Genetic Theory of Natural Selection"를 접하면서 물리학 도구를 이용하는 시도를 하였다. 그 결과, 그는 학습과 같이 진화도 환경에 적응해 가는 한 형태이지만 단일 세대 기간에서만 동작하기보다는 세대를 통해 동작한다는 사실을 알게 되었다. s 그는 진화가 유기체에서 동작한다면 컴퓨터 프로그램 상에서 구현하는 것도 가능할 것이라는 가정 하에 유전자 도구를 적응 문제에 적용하는 연구를 수행하게 되었다.

이 시기에 J. Holland가 수행한 연구는 적응 시스템과 함께 환경을 다루게 되었다. 그의 목적은 임의 환경에 적응하는 무제한적인 능력을 가진 프로그램을 만드는데 필요한 이론과 절차를 개발하는 것이었다. 그는 최우량인자의 인위적인 적자생존과 같은 비자연적인 선택의 근본적인 역할을 인식하게 되었고, 탐색을 위해서는 단일구조 대 단일구조 접근법보다는 집단 접근법을 사용하게 되었다. 그리고 1965년에는 교차, 돌연변이, 재조합 등의 유전자 연산자들의 중요성에 대해 인식하게 되었다. 이러한 연구결과를 토대로 J. Holland는 유전자 알고리즘의 표준 모델을 제안하였다.

1) 기본 구조

유전자 알고리즘에서 해의 후보는 유전자형(genotype) 혹은 염기(gene)로서 염색체에 일차원적으로 표현된다. 그리고 각 세대는 염색체를 나타내는 개체

(individual)들의 집한 개체군(population)으로 표현된다. 유전자 알고리즘은 유전 연산자들을 개체군에 속해 있는 염색체들에 반복 적용함으로써 세대를 진행해 감에 따라 더 나은 해를 구하게 된다. 이 때 사용되는 유전자 연산자는 선택연산, 교차연산, 돌연변이 연산이 있다. 선택연산은 도태의 압력을 가하는 경우에 중요한 것으로 선택연산을 통해 개체군에서 교차연산의 대상이 될 염색체들을 선택한다. 교차연산은 선택된 두 부모의 염색체를 조합하여 유전자 정보를 교환함으로써 새로운 염색체를 만들고, 돌연변이 연산은 유전자를 일정한 확률로 변화시킴으로써 지역 해 에서의 탈출과 함께 더 넓은 해 공간을 탐색을 가능하게 한다.

일반적으로 유전자 알고리즘의 동작 순서는 그림 15와 같다.

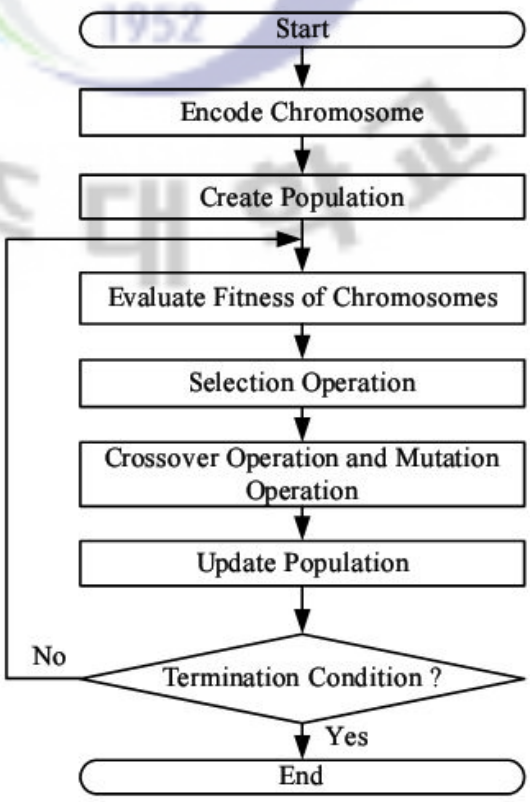

그림 15. 유전자 알고리즘의 동작

유전자 알고리즘의 동작은 먼저 탐색대상이 되는 해집합에서 해의 후보들을 선택하여 염색체를 인코딩하고, 이 염색체들을 이용하여 초기 모집단(initial

population)을 생성한다. 평가함수(evaluation function)를 통해 개체군 내에 있는 각 염색체들의 적합도를 평가하고 평가된 적합도를 선택연산에서 활용하여 교차 연산과 돌연변이 연산을 수행할 염색체를 선택한다. 이 때, 선택압력을 조절함으로써 도태확률을 변화시킬 수 있다. 이를 통해 최적해에 대한 수렴속도를 조절할 수 있다. 선택연산을 통해 선택된 부모염색체들에 교차 연산과 돌연변이 연산을 적용하여 자식 염색체를 생성한다. 부모 염색체와 자식 염색체 중 정합도가 높은 염색체로 다음 세대의 개체군을 재생성하고 종료조건에 이를 때까지 선택 연산, 교차 연산, 돌연변이 연산의 적용과 새로운 개체군의 재생성을 반복 수행한다. 알고리즘의 종료조건은 일반적으로 세대수로 결정된다[24].

2) 표현 기법

유전 알고리즘을 구축하는 첫 단계로 문제의 잠재해(candidate solution)를 유전적 표현, 즉 개체(individual)로 표현하여야 한다. 이 유전적 표현은 유전 알고리즘의 다른 절차(적응도 평가와 유전 연산자 적용 등)에 영향을 주므로 문제의 특성을 잘 반영할 수 있어야 한다. 대표적으로 이진 표현법(binary encoding)이 있으며 유전 연산자들은 0과 1의 이진 문자열을 기반으로 부호 공간상에서 연산되며, 개체 능력을 평가하는 목적함수에 적용 및 적합도 계산에서는 파라미터를 기반으로 해공간에서 이루어진다. 그 외에 순열 표현, 그룹번호 표현 등이 있다[25, 26].

3) 초기 개체집단의 생성

(1) 개체집단

개체집단(population)이란 임의 세대 k에서 n개의 염색체 집합을 의미하며, 많은 개체집단을 설정할 경우 수렴속도는 빨라지지만 연산속도가 느려지는 상충관계(trade-off)가 있다.

$$P(t) = \{ S_1(t), S_2(t), S_3(t), ..., S_i(t), ..., S_n(T) \}$$

여기서, S_i는 i 번째 염색체를 의미하고, 아래첨자 n은 개체집단의 크기(population size), t는 세대수를 나타낸다.

(2) 초기 개체집단 선정 방법

초기 개체집단의 형성방법은 임의 초기화(random initialization)와 의사 초기화(directed initialization) 크게 두 가지로 나눌 수 있다. 전자는 사전지식이나 경험 없이 단순히 난수 발생기에서 생성되는 개체집단의 크기와 스트링 백터의 차수의 곱에 해당하는 수의 비트로 초기화하며, 해에 대한 정보가 부족하거나 임의로 발생된 집단으로부터 전역해를 탐색해 가는 유전자 알고리즘의 성능평가를 위해 이용된다. 또한 후자는 사전지식과 경험을 바탕으로 간단한 알고리즘을 이용하여 얻게 되는 근사 결과치를 이용하거나 직관과 경험을 기초로 초기화하며, 해에 대한 사전지식과 정해는 초기조건이 경험적으로 추정 가능한 문제에 사용하는 방법이다.

(3) 세대

초기 개체군에 대한 적합도를 판별하고 재생산(reproduction), 교배(crossover), 돌연변이(mutation) 등을 거쳐 새로운 개체군이 만들어지는 과정을 세대(generation)라고 하며, 세대가 지날수록 각 세대들의 평균적합도 및 최대적합도가 상승하게 되므로 본 연구에서는 충분히 세대가 지난 후에는 최적치에 가까운 근사해를 얻을 수 있도록 하였다.

4) 적합도 평가

자연 도태설을 기반으로 하는 생물들은 생존경쟁을 통해 환경에 적응해 가며,

한 생물의 생존능력은 그 환경에 대한 적합도(fitness)를 통해 평가할 수 있다. 환경에 대한 생물의 적응능력이 적합도를 통해 평가받는 자연계와 마찬가로, 유전자 알고리즘에서는 개체간 환경 적응 능력의 여부는 적합도를 평가하는 과정에서 판단할 수 있다. 적합도 평가는 한 세대마다 유전자 연산자를 거쳐 새로운 집단이 완성되면 수행되며, 적합도가 크면 클수록 환경 적응능력이 큰 개체들을 의미하므로 적합도가 큰 개체가 더 많이 생존할 수 있도록 적합도 함수를 선정한다.

이러한 적합도 함수는 목적함수와 관련하여 최대화 문제의 형태로 정식화되고 양의 값을 갖는 조건이 만족되어야 하므로 목적함수의 적절한 사상(mapping)이 필요하다. 목적함수의 형태가 정의영역 내에서 항상 양의 값을 가지는 최대한 문제로 정식화되는 경우에는 목적함수 자체를 적합도 함수로 사용할 수 있으며, 정의영역 내에서 음의 값을 가질 경우에는 적정상수를 도입하여 양의 값을 갖는 적합도 함수로 재정식화 하여야 한다.

표 8은 함수 $f(x) = x^2$에 대해 염색체 스트링과 이에 대한 변수값 및 적합도를 나타내고 있으며, 각 염색체 스트링은 평가함수(evaluation function)를 통해 적합도를 산출하게 된다.

표 8. 염색체 문자열의 예

개체번호	변수값	적합도	적합도/총적합도
1	13	169	0.14
2	24	576	0.50
3	8	64	0.05
4	19	361	0.31

5) 유전자 연산

(1) 선택과 재생산 연산

선택(selection)은 적자생존의 자연법칙에 기초하여, 즉 환경에 대한 적응도에

의해 현 세대의 모집단으로부터 다음 세대에 생존할 개체를 선택하는 과정이다. 유전자 알고리즘에서 선택은 모집단의 다양성과 선택압력(select pressure)이 조화를 이룰 수 있어야 한다. 선택압력이란 우수한 해들과 열등한 해들 사이의 적합도 차이를 말한다. 강한 선택압력은 모집단의 개체들을 조기 수렴시키는 경향을 갖는다. 즉 모집단의 다양성을 약화시켜, 해공간의 다양한 탐색을 막는 경과를 가져온다. 한편, 약한 선택압력은 모집단의 다양성은 유지되나, 좋은 해를 효율적으로 탐색하지 못하여, 임의탐색(random search)과 비슷한 결과를 초래할 수 있다. 선택방법은 확률바퀴(roulette wheel), 순위선택(ranking selection), 토너먼트 선택(tournament select)등 여러 방법들이 있다. 이러한 방법들은 더 좋은 개체들에게 더 많은 특권을 부여한다는 점에 있어서는 공통적이다.

(가) 확률바퀴 방법

개체를 선택하기 위하여 적합도에 의해 슬롯의 크기가 정해지는 확률 바퀴 방법이 구현은 다음과 같은 초기 준비단계가 필요하다.

단계 1 : 각 개체 S_1(i = 1, ... , pop_size)에 대한 적합도 값 f(s_1) 을 계산한다. 단, 여기서 pop_size는 개체의 수를 의미한다.

단계 2 : 개체집단의 총 적합도를 계산한다.

$$F = \sum_{i=1}^{pop-size} f(s_i)$$

단계 3 : 각 개체 S_1(i = 1, ... , pop_size)에 대한 선택확률(selection probability) Pi 를 계산한다.

$$p_{i} = f(S_i)/F$$

단계 4 : 각 개체 S_i(i = 1, ... , pop_size)에 대한 누적확률(cumulative probability) qi 를 계산한다.

$$q_i = \sum_{j=0}^{i} P_i$$

그림 16과 같이 누적확률 백분율에 따라 각 개체를 차례로 확률바퀴의 슬롯 면적에 할당하고, 개체군 크기(pop_size)와 같은 횟수만큼 확률바퀴를 회전시켜 정지 시 개체를 선택하는 과정으로 이루어진다. 확률 바퀴를 한번 돌릴 때마다 새로운 개체군을 구성하기 위한 개체는 다음과 같이 선택된다.

단계 5 : 난수가 실수 r을 [0, 1]의 범위에서 발생시킨다.

단계 6 : 만약 $r < q_1$ 이면 첫 번째 개체 (S_1)를 선택하고, 그렇지 않으면 $q_{i-1} < r \leq q_i$ 인 i 번째 개체 S_1 $(2 \leq i \leq pop_size)$를 선택한다.

확률바퀴에 의한 새로운 개체 선택 시, 선택된 개체는 교배공간(mating pool)에 복제된다. 여러 개체들은 확률에 의해 선택되므로 가장 좋은 개체는 하나 이상을 재생산하며, 평균적인 값을 갖는 개체는 유지되고, 가장 나쁜 개체는 사라지게 될 가능성이 있다. 하지만 확률바퀴 선택방법은 확률적 속성으로 항상 최적의 선택을 보장하지 못하므로 이러한 단점을 보완해 줄 수 있는 방법이 요구된다.

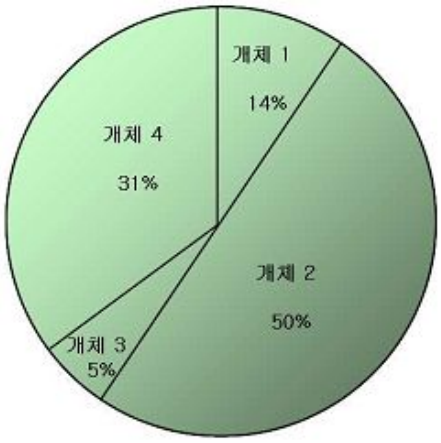

그림 16. 재생산 연산을 위한 확률바퀴

(나) 엘리티즘

엘리티즘(elitism)은 유전자 알고리즘의 진화과정에서 유전 연산자의 확률적 속성으로 인하여 한 세대의 최적 개체가 다음 세대에서 살아남지 못하고 소멸되어 탐색성능이 저하하는 원인을 제거하며, 국지적 최적해(local solution)로 수렴하는 문제를 개선한다. 특히, 앞에서 언급한 것처럼 확률바퀴 방법에 의한 재생산의 확률적 오차(stochastic error)를 감소시키는데 중요한 역할을 담당하며, 개체집단에서 가장 강한 개체가 다음 세대로 변경되지 않고 전달되는 것을 보장하여 부모세대보다 자손세대의 적합도가 크거나 같게 되는 결과를 낳게 된다. 하지만 적합도 값이 큰 상위 일정비율만을 그대로 재생상하기 때문에 전역탐색이 늦어지는 단점을 가지게 된다.

엘리티즘은 이전세대 동안에 알고리즘 수행 중 저장되어있는 최적개체 (s^{best})와 현 세대 t에서 적합도가 가장 큰 값을 갖는 개체 (s_t^{best})를 의미한다.

$$S^{best} = S_t^{best}, \text{if } s_{t-1}^{best} < s_t^{best}$$

$$S_t^{worst} = s^{best}, \text{if } s_{t-1}^{best} \geq s_t^{best}$$

여기서, t : 현재세대

t-1 : 이전세대

(2) 교배 연산

자연계의 생물들이 유성생식(sexual mating)을 통해 자손을 생산하고, 이 과정에서 유전의 정보교환이 이루어지듯이, 유전자 알고리즘 연산자 중 교배는 이러한 개체간의 정보교환을 가능하게 해주는 역할을 담당하게 된다. 교배는 탐색공간 상의 가능한 새로운 개체를 생산하기 위해 교배 공간에서 부모염색체 쌍을 임의로 선택하고 여러 가지 교배방법에 따라 스트링 비트를 교환함으로서 자손을 생성해나간다.

즉, 개체들 중 교배확률에 따라 교배할 개체를 선정하고, 새로운 개체군을 이

루는 각각의 개체들에는 재결합 작용자인 교배 작용자가 적용된다. 여기사, 교배 확률 p_c는 유전자 알고리즘에 사용되는 파라미터 중 하나이며, 교배 확률 p_c 와 개체 수에 의해 교배 적용 개체 수 (p_c × pop_size)가 결정된다. 다음은 교배과정을 각 단계별로 정리한 것이다.

단계 1 : 범위 [0, 1] 구간에서 난수 r_i 를 발생시킨다. (단, i = pop_size)

단계 2 : $r_i < P_c$ 이면 i 번째 개체를 선택한다.

단계 3 : 선택된 개체들 중에서 무작위로 쌍을 이룬다. (단, 선택된 개체수가 홀수일 경우, 임의로 개체를 추가 선택하거나 제거하여 쌍을 이루도록 한다.)

단계 4 : 무작위로 쌍을 이룬 개체들을 여러 가지 교배방법 중 하나를 이용하여 교배한다.

그림 17은 간단한 교배의 예를 종류별로 나타낸 것이다. 교배방법 중 일점교배(one point crossover)는 부모세대의 임의 한 곳에서 단 한번의 교배가 일어나며, 교배점은 염색체 길이(k) 사이의 정수(k-1) 중에서 임의 선택을 행한다. 그림 17(a)에서 보는 것처럼 교배점이 정해지면 교배점 이후 부모 염색체가 모두 교환되어 자손세대를 생산하게 된다. 자손세대의 새로운 염색체 구성으로 적합도 함수값의 변화는 있지만 부모세대의 형질을 전체적으로 변화하지 못하고 어느 정보의 형질은 유지되는 특징을 가진다. 하지만 일점교배는 부모염색체 내에 분산되어 있는 일정 유전자 정보는 결합하지 못하는 단점을 지니므로 이를 보완하기 위해 이점교배(two point crossover) 방법을 사용한다. 이점교배는 그림 17(b)에서 볼 수 있는 것처럼 교배점은 염색체 안에서 무작위로 선택된 2점이 되고, 교배되는 스트링은 2점내에 포함되어 있는 염색체에 해당한다. 또한 다점교배(multi-point crossover)는 첫 번째 교배점에서 일점교배를 수행하고 두 번째 교배점에서 다시 일점교배를 수행하는 방법으로 교배가 이루어지며 결국 짝수개의 교배점에 대하여는 이점교배가 이루어지고 나머지 교배점에서는 일점교배를 실행해서 교배가 완성된다. 다점교배는 일점 교배나 이점교배처럼 부모염색체의 일정 유전자 정보를 결합하지 못하는 단점은 없지만 쉽게 부모의 유전자 정보를

잃는 단점이 있다.

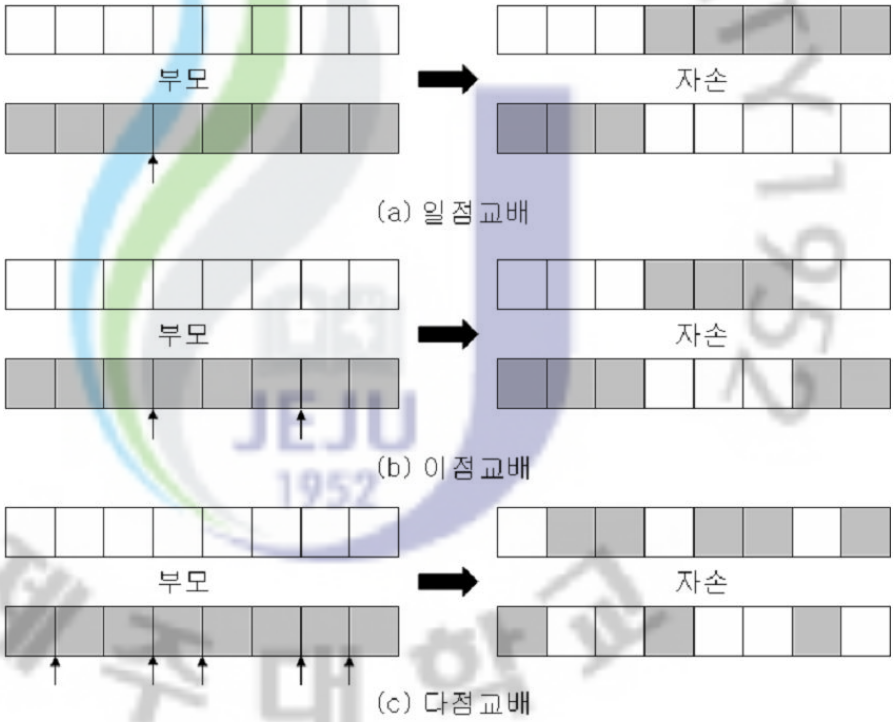

그림 17. 교배의 예

(3) 돌연변이 연산

재생산과 교배 연산자는 진화가 지속될수록 개체집단의 유전적 강도를 더욱 강하게 해주고, 염색체들은 서로 닮아가게 된다. 이러한 현상의 원인은 개체집단의 염색체 특정 비트가 모두 같게 되면 두 연산자를 통해 고정된 비트를 변경할 수 없게 되기 때문으로, 세대 말기에는 바람직하지만 세대 초기에 발생하게 되면 유전자의 다양성 결핍으로 지역해에 빠지게 되는 요인이 된다. 이러한 단점을 해결하기 위한 과정으로 돌연변이가 필요하다. 돌연변이는 자연계의 유성생식에서 자손의 형질이 부모의 형질과 전혀 다른 유전자 구조를 갖는 것을 말하며, 유전자 알고리즘에서의 돌연변이도 이러한 자연계의 현상을 모방한 형태로 구성된다. 이 연산자는 재생산과 교배 연산자에서 발생하는 세대초기 염색체 특정 비트가

고정되는 것을 방지해 줄뿐만 아니라, 준 최적해(suboptimal)로 수렴하는 요인을 제거하여 전역적인 해의 탐색을 가능하게 해준다.

이러한 돌연변이는 교배를 행한 후 개체의 스트링에서 각각의 유전자 돌연변이 확률만큼 랜덤하게 변화를 한다. 세대가 변하면서 환경에 대한 적응성과는 무관하면서, 랜덤하게 발생하므로 돌연변이 확률이 너무 크면 일반적으로 랜덤 탐색의 경향을 나타낸다. 유전자 알고리즘 파라미터인 돌연변이 확률 P_m, 개체의 스트링 길이 n, 그리고 개체의 수(pop_size)에 의해 돌연변이가 발생할 예상개수($P_m \times n \times pop \times pop\_size$)가 결정된다. 다음과 같은 방법에 의해 돌연변이를 일으킬 유전자를 선택한다.

단계 1 : 현재의 개체군을 구성하는 각 개체와 그 개체를 이루는 각 유전자에 대해서 난수인 실수 r을 [0, 1]의 범위 내에서 발생시킨다.

단계 2 : 만약 $r < P_m$ 이면 주어진 유전자를 선택한다.

단계 3 : 선택된 유전자에 대하여 여러 가지 돌연변이 연산자 중 하나를 이용하여 돌연변이를 일으킨다.

다음 그림 18은 간단한 돌연변이의 예를 보이고 있다.

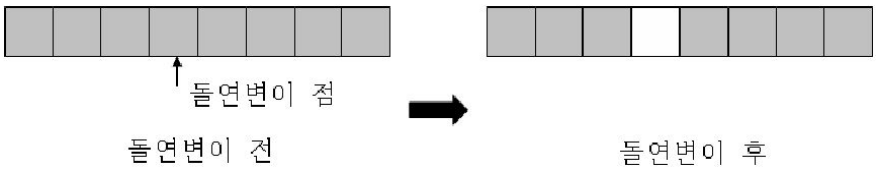

그림 18. 돌연변이의 예

6) 적정 매개변수 선택

유전자 알고리즘은 확률적인 탐색방법이므로 변수의 코딩방법, 유전연산자의 적정선택, 탐색 전략과 여러 가지 매개변수의 값의 선택에 따라 탐색성능에 영향을 주게 된다. 여기서, 매개변수들은 개체집단 크기, 교배확률, 돌연변이 확률 등

이 해당되며, 이들의 적절한 선정이 탐색효율이나 최적해 탐색에 큰 요인으로 작용된다.

첫 번째, 개체집단의 크기는 개체집단 내에 포함되어 있는 개체 수를 의미하며 그 수에 따라 탐색성능에 직접적인 영향을 끼치는 매개변수이다. 즉 개체 집단의 크기가 너무 작으면 유전적인 부동으로 부적합한 해로 빨리 적응해가며, 반대의 경우 탐색 성능은 향상되나 많은 연산시간으로 유전자 알고리즘의 동작이 느려진다.

두 번째, 교배확률은 교배가 일어나는 횟수를 조절하는 매개변수로 확률이 낮게 설정되면 다음 세대에서 새로운 개체 발생이 적게 되어 탐색이 침체되고, 반대로 높게 설정되면 탐색공간을 빨리 탐색하는 특징을 갖게 된다. 하지만, 교배의 전형적인 역할은 염색체간의 유전자 교환을 통해 전역탐색을 돕게 되므로 교배를 통해 제공되는 전역탐색의 양은 교배확률의 크기보다는 근본적으로 선택과정에 영향을 받게 된다.

세 번째, 돌연변이 확률은 유전자의 돌연변이 횟수를 조절하는 지표로 돌연변이 확률을 너무 낮게 설정하면 다른 탐색공간으로 이동하기 전에 지역해에 수렴할 수 있고, 반대로 너무 높게 설정하면 탐색공간을 불규칙적으로 이동하여 불안정한 탐색결과를 얻게 된다.

7) 유전자 알고리즘의 특징

유전자 알고리즘은 전역탐색과 지역탐색의 특성을 가지는 최적화 방법으로써 주어진 문제에 대한 해의 집단을 생성하는 과정에서 임의의 값들로 시작하여 생물학적 메커니즘에 바탕을 둔 재생산, 교배, 돌연변이 등의 연산자를 통해서 최적의 값을 찾아나가는 방법으로 기존의 방법들이 가지는 근본적인 제한들로부터 자유로워 다양한 분야에의 적용가능성이 크다. 이러한 유전자 알고리즘의 특징은 다음과 같다[27].

첫째, 파라미터 그 자체를 사용하는 것이 아니라 파라미터 집합을 코딩해서 처리하여 사용한다. 둘째, 하나의 지점부터 최적지점이 아닌 동시에 여러 지점에서

최적지점을 찾아 나간다. 셋째, 미분과 같은 수학적 연산이 아닌 결과의 적합도를 기준으로 수행한다. 넷째, 결정론적인 변화 규칙 대신 확률적인 변화규칙을 이용한다. 그림 19는 단순 유전자 알고리즘을 보이고 있다[28].

```
procedure GA
begin
    t = 0;
    initialize P(t)
    evaluate P(t)
    while not finished do
        begin
        t = t + 1;
        select P(t) from P(t-1);
        reproduce pairs in P(t);
    end
end
```

그림 19. 단순 유전자 알고리즘

일반적인 유전자 알고리즘 활용 연구에서는 우선 초기 모집단을 무작위로 발생시켰다. 이렇게 발생된 개체들은 목적함수에 의해 목적함수 값이 할당되어지고 할당된 목적함수 값에 의해 선택이 이루어진다. 또한 실시간 개체들의 선택은 비선형 순위에 의한 확률적 균등 표본선택을 이용한다[27].

다음 그림 20은 단순 유전자 학습 알고리즘을 보이고 있다[28].

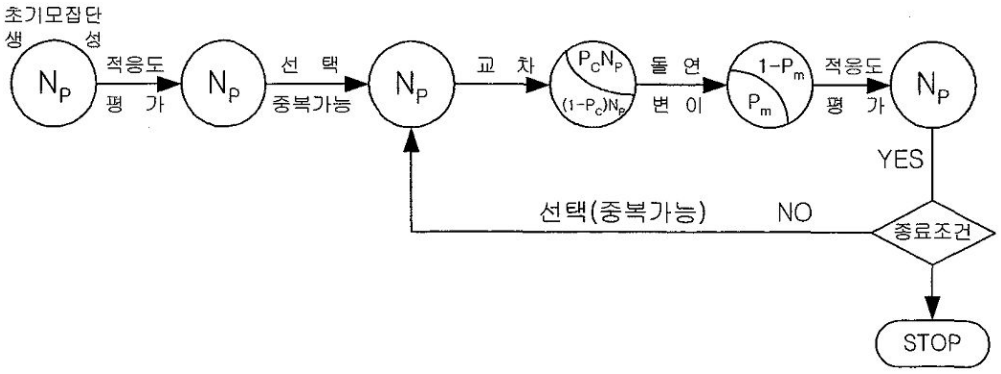

그림 20. 단순 유전자 학습 알고리즘

비선형 순위에 의한 알고리즘은 먼저 목적함수에 의해 각 개체들에 대한 목적함수 값을 평가한다. 최소화 문제의 최적화의 경우, 목적함수 값이 낮을수록 적합도가 좋다고 할 수 있다. 이렇게 평가된 목적함수 값을 내림차순으로 정렬한 후, 목적함수 값이 가장 낮은 개체는 첫 번째 위치에, 목적함수 값이 가장 높은 개체는 Nind 번째의 위치에 정렬된다. 그러면 개체집단에 정렬된 위치에 따라 각 개체에 적합도 값이 할당된다. 이것을 수식으로 표현하며 다음과 같다.

$$FitnV(Pos) = \frac{Nind \times X^{Pos-1}}{\sum_{i=1}^{Nind} X(i)}$$

여기서,

Fitn V(Pos) : 위치(Pos)에 따른 적합도값

Nind : 개체 집단 내 개체의 수

X는 다음 식의 다항식에 의해서 구할 수 있으며 SP는 선택압이다. 만약 선택압이 너무 클 경우, 우성개체가 열성개체보다 많이 선택되어 조기 수렴의 위험이 있다.

$$0 = (SP-1) \times X^{Nind-1} + SP \times X^{Nind-2} + \ldots + SP \times X + S$$

이렇게 각 개체에 적합도 값이 할당되면 확률적 균등 표본 선택법에 의해 적합도에 따라서 선택을 하게 된다. 확률적 균등 표본 선택은 바이어스가 0이고 최소한의 퍼짐 정도를 갖는 선택법이다. 개체들은 각자의 적합도 크기에 의해 선택확률을 갖게 되며 각 개체의 선택확률을 수식으로써 나타내면 다음 식과 같다. 이렇게 결정된 선택확률에 의해 적응도가 우수한 것부터 선택 확률값과 동일한 공간을 가지고 누적되어 직전상에 사상된다. 여기서, 선택확률에 전체 합은 1이 된다.

$$F(X_i) = \frac{f(x_i)}{\sum_{i=1}^{Nind} f(x_i)}$$

여기서,

$f(x_i)$: 개체 x_i의 적합도 값

$F(X_i)$: 개체가 선택될 확률

이렇게 직선상에 사상된 개체는 다음 세대를 위해 선택될 개체의 수를 결정하게 되고 초기 개체집단에서 다음 세대를 위해 선택될 무작위 수가 Npointer 라면 첫 번째 선택될 개체의 위치는 [1, 2/Npointer]의 구간에서 단지 하나의 무작위 수를 발생해서 결정된다. 그러면 첫 번째 개체로부터 다음 선택될 개체의 위치는 처음 선택된 개체로부터 거리가 2 × (1/Npointer)인 곳이 된다. 이와 같이 다음 선택될 개체는 이전 선택된 개체로부터 거리가 1/Npointer로 동일 간격을 유지하며 다음의 개체가 선택된다.

그림 21은 이 방법의 선택 형태를 보여주고 있으며 누적 선택확률은 F(Xn) = F(Xn-1) + F(Xi)이다.

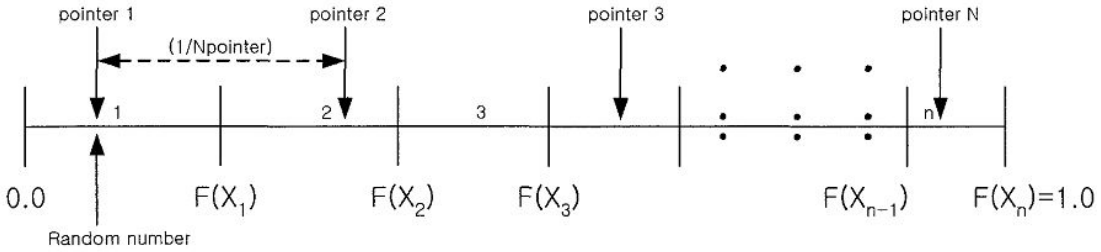

그림 21. 확률적 균등표본

이렇게 선택과정을 거친 개체들은 다시 교배 과정을 거친다. 일반적인 유전자 알고리즘 활용 연구에서는 다양한 교배방법 중 개체들 간에 변수의 값을 교환하는 불연속 재조합법을 이용한다[27].

그림 22에서 보는 바와 같이 각각의 부모는 마스크를 통하여 자손을 생성하게 된다. 교배과정을 거친 개체들은 돌연변이 과정을 수행한다. 돌연변이는 유전자

를 일정확률로 변화시키는 조작이다.

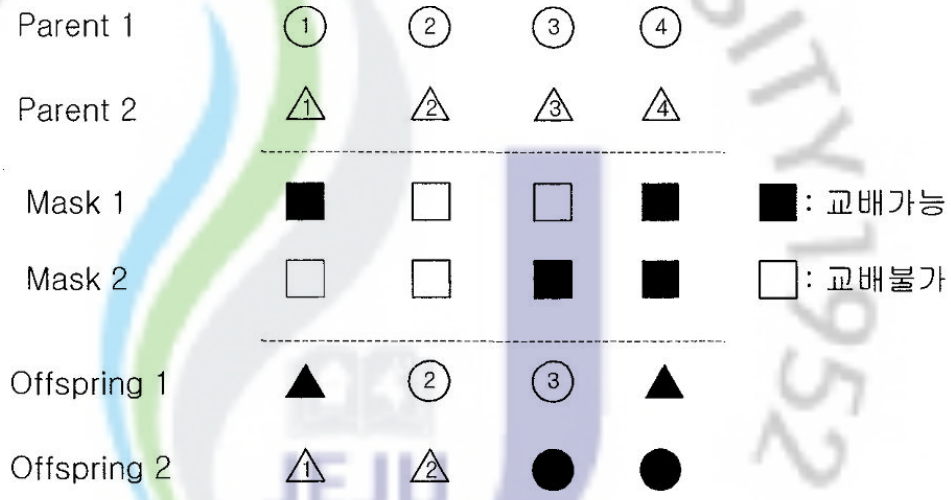

그림 22. 불연속 재조합법

돌연변이를 너무 큰 변이확률로 설정하면 원래 가지고 있는 염색체 고유의 특성을 상실하기 때문에 임의 탐색으로 변해 버리게 되지만 어느 정도의 변이는 필요하다. 돌연변이가 없는 경우에는 초기 유전자의 조합 이외의 공간을 탐색할 수 없으며 결국 찾고자 하는 해의 질에도 한계가 드러난다. 일반적으로 돌연변이는 고정된 확률로 각 유전자가 변이되도록 설정하지만 변이율을 동적으로 변화시키는 기법도 있다.

일반적인 유전자 알고리즘 활용 연구의 돌연변이 수행 과정의 알고리즘은 다음식과 같다[29].

$$\text{mutated variable} = \text{variable} \pm \text{range} \times \delta$$

여기서,

$$\text{range} = 0.5 \times \text{domain of variable(search interval)},$$

$\delta = \sum_{i=0}^{m-1} \alpha_i 2^{-i}$, 확률 1/m을 가지면 $\alpha_i = 1$,

그렇지 않으면 $\alpha_i = 0$,

만일 m=20이면, 돌연변이 연산자는 range×2^{-19}의 정밀도까지 최적화하고 이 세 가지 유전 연산과정을 수행한 후, 목적함수를 평가한다. 만일 종료조건에 목적함수 값이 도달했을 경우, 수행이 종료되지만 그렇지 않을 경우, 선택, 교배, 돌연변이의 유전 연산과정을 반복하게 된다.

8) 유전자 알고리즘의 응용

유전자 알고리즘은 근본적으로 탐색을 위한 알고리즘으로 탐색공간에 대한 제약, 예를 들어 공간의 연속성과 같은 제약 조건을 갖지 않기 때문에 여러 가지 문제에 적용이 가능하고 주어진 상황에 적응적으로 대처해서 탐색을 한다는 특징을 가지고 있다. 주의해야 할 점은 많은 응용에 있어 유전자 알고리즘은 그 응용에 적용되었던 구체적인 알고리즘에 비해 성능이 비슷하거나 떨어지는 경향이 있으나 경우의 수를 따는 조합의 문제에 있어서는 향상된 성능을 기대할 수 있다[30]. 실제적으로 유전자 알고리즘은 최적화 관련 문제에는 대부분 적용할 수 있으며 시스템의 특성이 멀티 모형 한 경우에 적용했을 때 어떤 다른 알고리즘보다 최적의 값을 찾아내는데 그 성능이 우수하다고 여러 연구에 의하여 확인되었다. 기존에 유전자 알고리즘은 신경만의 학습에서 가중치를 최적화하는 방법으로 이용되었다. 그러나 지금은 유전자 알고리즘의 유용한 특성으로 여러 산업 및 경제 분야에 적용되고 있다. 표 9는 유전자 알고리즘의 몇 가지 대표적인 응용의 예이다.

표 9. 유전자 알고리즘의 응용 분야

응용 분야	응용 사례
최적화	수치적 함수의 최적화, 가스 파이프라인의 최적화, 전력송전망의 최적화, 컴퓨터 자판의 최적 배정문제, 항공기 승무원 배정문제, 순회 외판원문제, 그래프 분할문제, 유전자 정보해석
설 계	VLSI 회로설계, 비행기 날개의 공기역학적 설계, 엔진 노즐의 설계, 컴퓨터 통신망의 최적 설계, 심장박동기의 설계, 디지털 필터 설계, 퍼지 제어기 설계

인공지능	LISP 프로그램의 자동 생성 문제해결 규칙의 자동 습득, 신경망 합성 및 학습, 패턴인식, 자연언어 처리, 멀티 에이전트 시스템
시스템 분석 및 예측	시스템 동정, 케이 오틱 시계열의 예측, 확률변화 예측, 단백질 구조의 분석, 재정 및 경제 분야에서의 예측 및 분석 문제
제어 및 로보틱스	이동 로봇의 경로 계획, 신경망 및 퍼지 로직과 유전자 알고리즘의 결합에 의한 제어

9) 유전자 알고리즘을 이용한 부하 분산

분산 환경에서 유전자 알고리즘을 기반으로 하여 사용자 요청을 분산하는 많은 연구들이 진행되어왔다. 유전자 알고리즘은 아주 큰 문제의 해 집단의 일부만을 이용하여 최적 또는 준 최적의 해를 찾을 수 있는 방법이다. 분산 환경에서 유전자 알고리즘을 이용하여 요청된 작업에 대한 최적의 해를 구하는 많은 연구들이 진행되어왔고, 효과적인 방법으로 인식되고 있다.

유전자 알고리즘은 정적 스케줄링 방식과 동적 스케줄링 방식으로 나눌 수 있다. 정적 스케줄링 방식은 실행시간 전에 스케줄이 생성된다. 이를 위해서는 모든 작업과 자원에 대한 상태를 미리 알고 있어야 하며 그 값들은 변경이 안 된다. 동적 스케줄링 방식은 실행시간에 스케줄이 정해지는 방식이며, 사용자 요청과 시스템의 상태에 대한 모든 정보를 알 필요는 없다. 따라서 동적 유전자 알고리즘이 실제 환경에 사용하기 적당하다는 것을 알 수 있다[31].

5. 부하 분산 시스템

분산 시스템 환경에 설정된 Connection 에서 데이터를 각 Connection 별로 분산 전송하기 위한 로드 밸런싱 기술로는 Round-Robin (RR), Least Connection (LC), Weighted Round-Robin (WRR), Weighted Least Connection (WLC) 방법들이 사용되고 있다. 일반적으로 Multi Connection 의 데이터 전송 방법에서는 구현이 가장 쉬운 Round Robin 방법이 주로 쓰이고 있다[32].

RFID 미들웨어를 위한 로드 밸런서 구현을 위해서는 먼저 현재 Connection 의 상태를 모니터링 할 수 있도록 관리해주는 Connection Manager의 구현이 요구된다. 이와 더불어, Multiple TCP Connection 상에서 QChained Clustering 기법을 사용하여 과중한 부하를 다른 채널로 분산시키는 로드 밸런서를 구현함으로써, 유/무선 네트워크의 효율적인 사용을 보장할 수 있도록 구현할 수 있다[33]. 네트워크 트래픽, 웹 로그 분석, 실시간 센서 데이터와 같이 시간에 따라 변하고 연속적으로 생성되는 대용량의 데이터를 데이터 스트림이라 한다. 새롭게 요구되고 있는 이 데이터 스트림을 처리하기 위해서 데이터 스트림 관리 시스템(DSMS)이 연구되었다[34]. 데이터 스트림 관리 시스템의 가용 작업량을 초과하는 데이터 스트림의 처리 요청 시, 작업에 일부를 포기하는 부하분산 기법(Load Shedding)을 사용 할 수 있다[35].

특점 시점에 서비스 요청 폭주로 인해 시스템 과부하가 발생할 때 효과적인 비용 측면에서 처리 속도는 약간 희생하더라도 시스템이 다운되거나 불능사태에 빠지는 것을 예방하는 것이 바로 Peak Load Control(PLC)이다. PLC는 현재 구성된 시스템이 최대로 수용할 수 있는 한도까지는 정상적으로 서비스 되지만, 그 이상을 넘어설 경우는 서비스를 제한하거나 처리하는 속도를 조절함으로 부하를 조절하는 기법이다[36].

대표적인 부하 분산 연구에는 퍼지 그룹핑 기반 부하 분산, 사용자 요청 패턴 히스토리와 유전자 알고리즘 기반 등의 일반 부하 분산 시스템과 RFID 미들웨어의 큐 사이즈를 고려한 부하 분산, 정책 기반의 RFID 미들웨어 부하 분산 시스템으로 나눠 설명하고자 한다.

1) 일반 부하 분산 시스템

퍼지 그룹핑 기반 부하 분산 방법[37]은 퍼지 그룹핑 기반 로드 밸런싱을 사용하여 분산 시스템의 성능을 향상시키고, 부하를 측정하기 위해 CPU와 메모리 사용량을 측정하여 서비스 우선순위를 정했다. 이 기법은 전체 서버 부하 상태를 실시간으로 모니터링 하여 서비스 요청이 있을 시 동적으로 최적 상태의 서버에게 부하를 분산 시킨다. 그러나, 국제 표준 ALE 스펙 기반 RFID 미들웨어 시스템의 특성을 고려하지 않아, RFID 데이터 수집 및 응용 클라이언트의 처리 요청을 효율적으로 처리하지 못한다.

사용자 요청 패턴 히스토리와 유전자 알고리즘 기반 부하 분산 방법[31]은 계층형 분산 VOD(Video on Demand) 시스템 모델의 서버들을 지역적으로 분한하고 제어 서버를 지역마다 설치하여 지역에 있는 VOD 서버들을 관리하도록 구성하였다. 지역적으로 서버를 분산함으로써 여러 지역에 산재되어 있는 사용자들의 요청이 한 지역에 집중되는 것을 막을 수 있다.

또한 사용자 요청을 지역 서버군 내에서 분산시키기 위해서 히스토리를 기반으로 한 유전자 알고리즘을 사용하였다. 사용자의 요청과 서비스에 대한 정보 데이터베이스에 히스토리로 구성하여 저장하였다가 참조한다. 이러한 히스토리 정보를 기반으로 유전자 알고리즘의 적합도 함수에 적요하여 VOD 시스템을 위한 유전자 알고리즘과 유전 연산을 제안하여, VOD 서비스 환경에서 사용자 요구에 대한 부하를 보다 정확하게 예측하여 부하를 분산할 수 있다. 그러나, 퍼지 그룹핑 방식과 같이 RFID 미들웨어 시스템에 적용하는데에는 부적합한 특징을 가진다.

Abed A.K.등은 각 클러스터(Cluster) 내의 컴퓨팅 자원을 부하분산(Load-balancing)하는 알고리즘을 제안했다. 제안된 알고리즘은 서로 협력하며 실제적으로 컴퓨팅을 수행하는 워크스테이션 (Workstation)들이 모여 클러스터를 구성하고 각 클러스터마다 부하(Workload)가 고루 분배되도록 하는 역할의 워크스테이션을 Cluster Controller(CC)로써 하나를 설정한다. 각각의 워크스테이션에 에이전트(Agent)를 두어 부하를 관리하고 그 정보를 교환할 수 있도록 한다. CC

의 에이전트는 자신이 속한 클러스터의 남은 컴퓨팅 자원량을 종합하고 다른 클러스터의 CC의 에이전트로부터 타 클러스터 내의 컴퓨팅 자원 정보를 받아 모자라는 컴퓨팅 자원을 공유할 수 있도록 한다[38].

Di Wu등은 DHT(Distributed Hash Table)를 사용하는 P2P(Peer-to-Peer) 시스템의 마이그레이션 기반의 부하분산 기법인 RDS (Rendezvous Directory Strategy)와 ISS (Independent Searching Strategy)를 비교 분석하고 두 가지 방법의 장점을 사용한 GBS(Gossip Based Strategy)를 제안했다. 비교 분석한 내용을 보면 RDS의 경우 각 피어(Peer)는 랑데부 디렉토리 내의 다른 피어들에게 주기적으로 부하정보를 보낸다. ISS는 오직 부하정보에 대한 요청이 있을 때만 요청한 피어에게 부하정보를 제공한다. RDS는 대부분의 경우 ISS보다 효율적이다. 하지만 ISS의 경우 확장성과 강인성에서 RDS보다 더 나은 효율을 보인다. GBS는 전체 시스템을 그룹으로 묶고 그 그룹 내에서는 가쉽 프로토콜(Gossip Protocol)을 사용하여 부하정보를 공유하여 그룹 내에 부하분산 상태를 이룬다. 시스템이 여러 그룹으로 구성되는 경우 각 그룹 내에서 부하분산 상태를 만든 후 다른 그룹의 부하정보를 받아 부하분산 상태를 만든다[39].

부하 분산 메커니즘은 정보 수집 정책, 초기화 정책, 작업 전송 정책, 선택 정책, 위치 선정 정책과 같은 정책들을 포함하고 있다[40].

정보 수집 정책(Information Gathering Policy)은 클러스터 내의 노드들로부터 작업 부하에 대한 정보를 수집하고 관리하는 것을 뜻한다. 정보 수집 정책을 위하여 고려해야 할 것은 부하 관리를 위한 정보 수집을 수행하는 빈도와 정보 수집 방법으로 나누어 볼 수 있다. 정확한 최신 부하 정보를 수집하는 것과 비용을 최소화 시키는 것은 서로 상충된다. 정보 수집 정책은 작업 부하의 추정치와 명세에 대한 정보도 역시 포함하게 된다. 이러한 정보들은 시스템 정보인 CPU 사용률(CPU Utilization), 프로세스 큐 길이(CPU run queue length)와 어플리케이션 정보인 초당 요청 수(Request per Second), 처리량(Throughput), 요청에 대한 응답 시간(Response Time) 등으로 나눌 수 있다[40, 41].

시작 정책(Initiation Policy)은 누가 부하 분배 프로세스의 수행을 트리거 할 것인지에 관련된 것이다. Initiator는 근원 노드(source node)또는 목적 노드(destination node)가 될 수 있으며, 두 가지 모두가 될 수도 있다(symmetric

initiations).

중앙 집중식(Centralized Initiation) 시작 정책은 전체 시스템 내에 부하 분산을 책임지는 하나의 Initiator가 존재한다. 각 노드는 부하 정보를 수집하여 노드에게 제공하고, 노드는 부하 분산 스케줄링을 통하여 부하를 목적 노드에 분산시킨다[42].

분산 방식(Distributed Initiation) 시작 정책은 어떤 노드도 Initiator가 될 수 있다. 각 노드는 자신의 부하에 변화가 있으면 다른 노드들에게 이를 알리게 된다. Initiator는 이렇게 수집된 부하 정보와 부하 정보의 시간 경과를 고려하여 부하를 분산할 목적 노드를 선택하게 된다. 분산 방식에는 송신자-시작(Sender-Initiation) 정책과 수신자-시작(Receiver-Initiation)정책이 있다[43, 44].

작업 전송 정책(Job Transfer Policy)은 Initiator가 다른 노드들에 작업 부하를 재할당하는 것을 고려해야 하는 시점을 결정한다. 즉, 부하 분산이 필요한 시점이 언제인가를 고려하는 것이다. 이러한 시점에 대한 결정은 노드의 상태만을 기초로 만들어 지기도 하며 서로 교환된 부하 정보를 취합한 노드의 정보를 기초로 하기도 한다.

선택 정책(Selection Policy)은 어떤 작업을 재할당할 것인지 결정하는 것이다. 비 선점 정책에서는 한번 할당된 작업은 작업이 완료 될 때까지 이동되지 않는다. 하지만 선점 정책에서는 노드에서 수행중인 작업이라도 부하 분배의 대상이 될 수 있다. 위치 선정 정책(Location Policy)은 재할당하기 위해 선정된 작업을 어느 노드에 할당할 것이지를 결정한다. 가장 간단한 위치 선정 정책은 노드를 랜덤 또는 순차적으로 선택하도록 하는 것이다. 보다 복잡한 정책을 사용하게 될 때에는 클러스터내의 각 노드에 대한 다양한 정보들을 고려하여 부하를 분배 받게 될 대상을 결정한다[40].

2) RFID 미들웨어 부하 분산 시스템

RFID 미들웨어의 큐 사이즈를 고려한 부하 분산 방법[45]은 RFID 시스템이 설치되는 현장의 다양한 요구 사항과 부하 수준에 맞춰 성능을 최적화하고 어플

리케이션 레벨의 비즈니스 요구 사항을 수렴하기 위해 미들웨어 구성요소를 각각 독립된 서비스로 식별하고 각 요소들이 메시지 기반의 약 결합을 통해 쉽게 분산하여, 유연하게 필요한 서비스를 추가할 수 있는 방식의 RFID 미들웨어를 제안하였다. 다양한 RFID 애플리케이션들과 표준 기반으로 인터페이스를 사용할 수 있도록 ALE 스펙을 준수하고 RFID 데이터의 효율적 처리와 안정적인 시스템 운영을 위한 부하 부산 및 Fail Over 기능이 가능한 미들웨어 시스템을 구현하였다.

미들웨어 각 요소별 부하분산이 적절히 이루어질 수 있도록 구성 요소들이 메시징(Messaging) 시스템을 기반으로 약결합하여 여러 노드에 쉽게 분산될 수 있는 미들웨어 프레임워크 아키텍처와 수집, 정제된 RFID 태그 데이터에 대한 비즈니스 프로세스를 효과적으로 적용할 수 있도록 비즈니스 룰 엔진도 설계 및 구현하였다. 그러나 로지컬 리더에서 수집되는 RFID Tag 데이터의 수와 미들웨어에서 처리되는 ECSpec의 수만을 고려하여 완벽한 부하 분산을 처리하는데 부족하다.

정책 기반의 RFID 미들웨어 부하 분산 방법[46]은 표준 RFID 미들웨어의 확장성 향상을 위한 적응적 부하 분산 기술 도입 방법과 적용 사례를 분석하여, 첫째, 기존의 부하 분산 기본 기법과 정책들을 RFID 미들웨어에 적용하기 위한 방안과 RFID 미들웨어에 특화된 정책을 제시하였다. 둘째, EPCglobal ALE 표준 인터페이스를 만족하는 RFID 미들웨어의 적응적 부하 분산을 위한 시스템을 제시하였다. 셋째, RFID 미들웨어의 부하에 많은 영향을 주는 부하 요소와 성능 및 부하 상태 판단을 위한 지표를 제시하고 작업 부하 모델을 결정하였다. 넷째, 전체 시스템 부하량의 변화에 따른 적응적 부하 분산 기법을 적용하기 위한 시뮬레이션 결과를 제시하였다. 그러나 ALE Spec 기반의 RFID 미들웨어의 CPU 사용율과 ECSpec의 수, 리더들로부터 수집되는 태그 수와, RFID 미들웨어서 처리하고 있는 데이터의 수만을 고려하여 RFID 미들웨어의 큐 사이즈를 고려한 부하 분산 방법과 같이 완벽한 부하 분산을 처리하는데 부족하다.

6. 기존의 부하 분산 시스템의 분석

기존 부하분산 시스템을 살펴보면 첫 번째 부하 분산의 리얼 서버인 미들웨어의 일반적인 자원 정보만을 수집하고 있다. 일반적인 자원 정보로 CPU 정보를 공통적으로 수집하고 있으며, CPU 정보는 미들웨어의 실행 환경에서 가장 중요한 정보이고, Memory 자원 정보 등을 추가로 수집하여 부하 분산 시에 활용한다. 이는 다양한 처리를 수행하는 미들웨어 분산 환경에서 부하 분산을 하기 위해 대표적인 정보이기는 하나, 보다 세부적이고 미들웨어의 특성을 고려한 부하 분산을 위해서는 추가적인 자원 정보가 필요하다.

두 번째 기존의 부하 분산 시스템은 특정 기능을 수행하는 미들웨어 형태로 구현되었다. 그러나 미들웨어의 DB 접속수, 스레드 처리, 모듈 처리와 같은 자원 정보를 고려하고 있지 않으며, 미들웨어에서 처리되는 상황 정보에 해당하는 이러한 정보를 부하 분산 시에 활용할 필요가 있다.

세 번째 기존의 ALE 기반 RFID 미들웨어의 부하 분산을 위해 사용자의 처리 사항이 명시된 ECSpec과 RFID 리더들로부터 수집된 RFID Tag 수 등만을 고려하여 부하 분산을 하고 있다. 그러나 ALE 스펙을 기반으로 처리되는 RFID 미들웨어에서는 실제 처리되는 Event Cycle에 대한 자세한 정보가 필요하며, 다양한 유형의 ECSpec을 분석하여 적절한 RFID 미들웨어로 부하 분산 처리를 할 필요가 있다.

끝으로 이렇게 수집된 다양한 정보를 부하 분산 시스템에 제공하기 위한 효율적인 방법이 필요하며, 이렇게 수집된 다양한 자원 정보를 바탕으로 최적의 미들웨어 서버를 선정하기 위한 방법에 대한 연구가 필요하다.

7. RFID 부하 분산 시스템의 고려사항

위와 같이 기존의 부하 분산 시스템의 문제점들에 따라 본 논문에서 고려해야 할 핵심적인 요구사항을 도출하였다.

첫 번째로 다양한 RFID 미들웨어의 자원 정보를 수집하여 효과적으로 관리하

고, 공용 인터페이스를 바탕으로 부하 분산 시스템에 제공할 수 있어야 한다.

두 번째로 다양한 자원 정보를 바탕으로 다양한 클라이언트의 처리 요청을 처리할 최적의 미들웨어 서버를 선정할 수 있어야 한다.

세 번째로 일반적인 ALE 스펙을 기반으로한 RFID 미들웨어의 부하 분산 시스템에는 몇 가지 문제가 있으며 이러한 문제점을 해결하기 위한 구조로 설계되어야 한다.

이러한 문제점에는 첫째, 그림 23과 같이 Tag 데이터 수집 측면의 문제이다. 로지컬 리더가 특정 미들웨어에만 셋팅될 경우, 로지컬 리더간 수집에 제약을 갖는다.

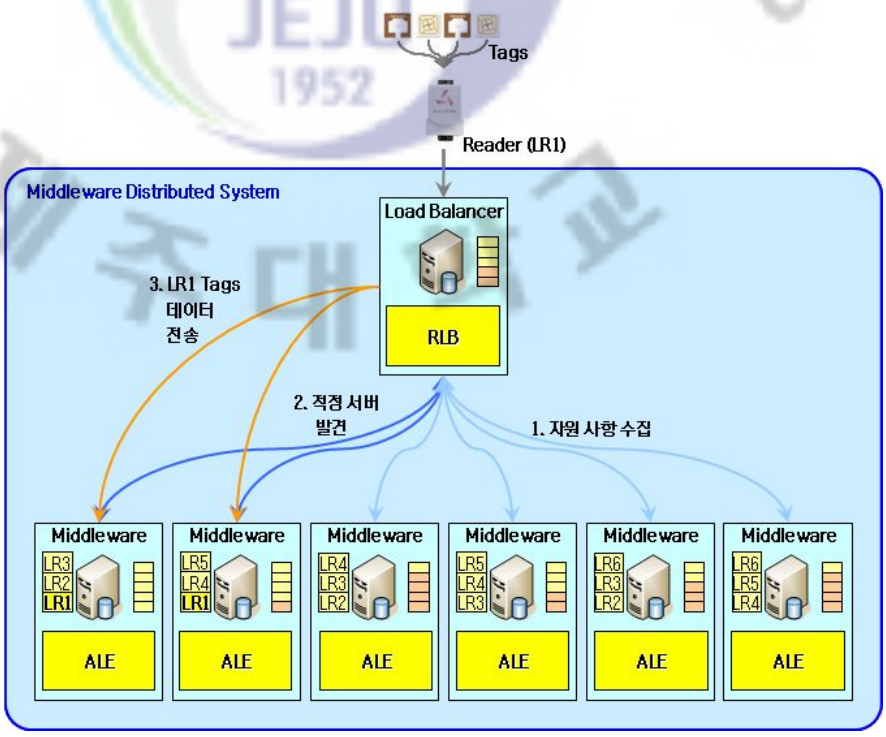

그림 23. Tag 데이터 수집

ECSpec에 로지컬 리더 사항이 LR1, LR2, LR5, LR6 4개의 로지컬 리더가 정의될 경우 분산 시스템에서 처리하는데 부가적인 미들웨어간 Tag 데이터 병합 처리와 위치 이동 처리가 필요하며, 정책 기반의 RFID 미들웨어 부하 분산 방법에서 이러한 문제점이 있다.

둘째, 그림 24와 같이 ECSpec 처리 측면의 문제이다. 로지컬 리터 스펙이 먼저 등록되고, 특정 미들웨어만 셋팅될 경우 ECSpec에 정의되는 로지컬 리더에 대한 제약이 생기며, 특정 로지컬 리더는 특정 ALE 미들웨어서만 수집이 가능하며, 부하 분산에 대한 효율이 떨어진다.

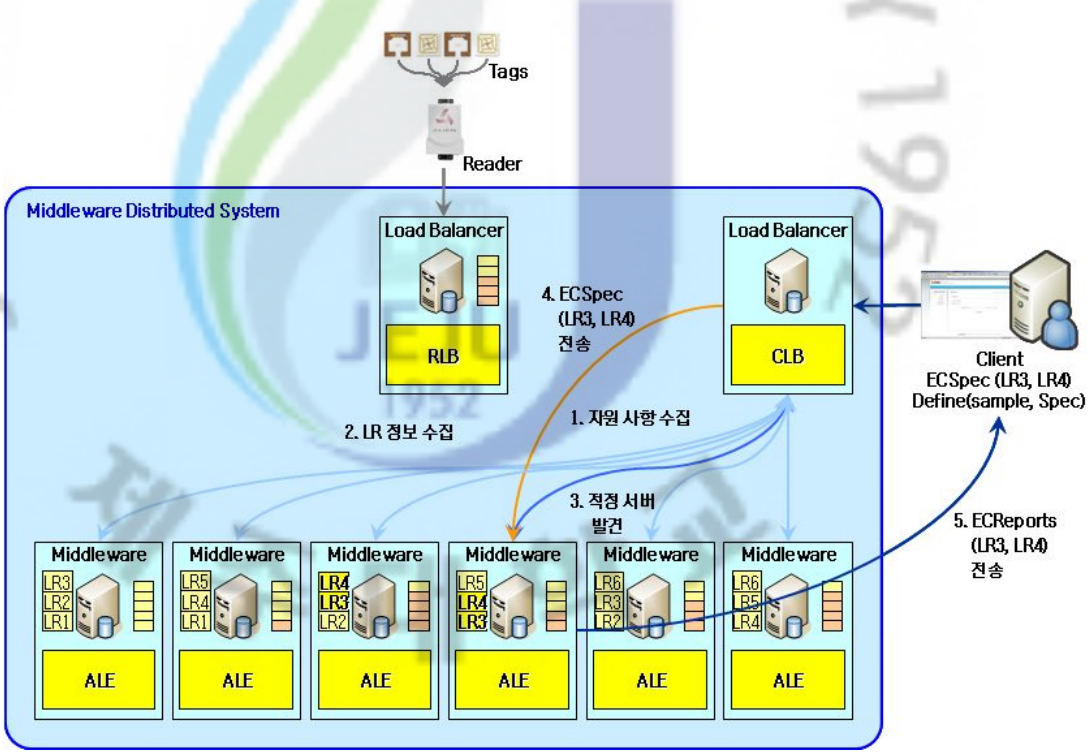

그림 24. ECSpec 처리

셋째, 그림 25와 같이 ECSpec 저장 처리 측면의 문제이다. 미들웨어 분산 시스템 내에서 동일한 ECSpec 정보를 모든 DB 저장하여 저장 공간을 낭비할 수 있으며, 공동 DB 서버 1대를 구축하여 공유하면, DB 서버에 장애 발생 시 저장된 ECSpec 등의 자료가 손실 되는 문제가 있으며, ECSpec 등의 공통 정보를 저장하여 공유하기 위한 방법이 필요하다.

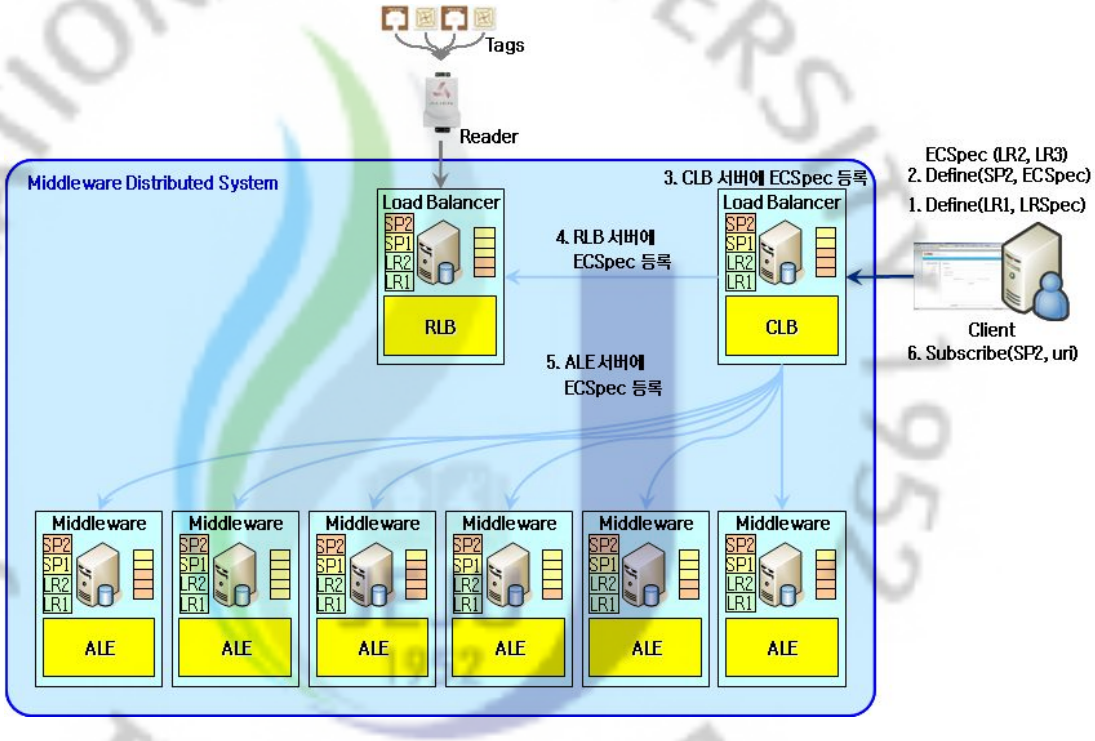

그림 25. ECSpec 저장 처리

따라서 본 논문은 대량의 RFID 데이터를 수집 및 처리를 하기 위하여 RFID 데이터 수집, 데이터 가공 처리, 클라이언트 요청 처리 등의 기능을 처리하도록 ALE 스펙을 기반으로 구현된 RFID 미들웨어들의 분산 환경을 구축하고, 클라이언트의 요청을 분산 할 수 있는 부하 분산 시스템을 설계 및 구현하고자 한다.

본 논문에서는 클라이언트의 RFID 데이터 수집 및 가공 처리 요청을 분산하기 위하여, 분산 환경에 구축된 여러 개의 RFID 미들웨어의 자원 정보를 수집하고, 각 RFID 미들웨어의 자원 사항에 맞게 클라이언트 요청을 처리할 수 있도록 해야 한다. 이를 위하여 기 구현된 ALE 스펙 기반 RFID 미들웨어의 자원 사항을 하드웨어 자원, 미들웨어 자원, ALE 자원으로 구분하여 개별 저장하며, 부하 분산 시스템에서 자원 정보를 요청했을 시에 개별 저장된 자원 정보를 SOAP 통신을 이용하여 XML 형태로 제공하고자 한다. 또한, 클라이언트의 요청 처리를 수행할 최적의 ALE 기반 RFID 미들웨어를 선정하기 위해 유전자 알고리즘을 이용하여 가중치를 생성하여 부하 분산 처리에 활용하고자 한다.

분산 시스템에 부하 분산 기능을 제공하기 위해서는 위와 같은 정책들을 필요

로 하게 된다. 그리고 이러한 메커니즘은 시스템 설계에 반영되어야 하며 구현되어야 한다[46].

RFID 미들웨어의 과부하를 분산시키기 위해 RFID 미들웨어의 몇 가지 요소를 모니터링하고 부하분산을 위해 결정해야 한다. 먼저 현재 RFID 미들웨어에 대해 얼마나 많은 부하가 가해지고 있는지 CPU 사용량, 메모리 사용량 등을 지속적으로 모니터링 해야 한다. 그리고 RFID 미들웨어의 한계 수준으로 과부하가 가해지는 경우 어떤 RFID 미들웨어로 부하를 옮겨 RFID 미들웨어 전반적으로 부하가 균형을 이루게 할 것인지 결정한 후 부하를 옮기는 작업을 수행하여 부하분산을 완성한다. 이를 위해 RFID 미들웨어의 부하상태를 모니터링하고 RFID 미들웨어의 부하정보를 공유할 수 있는 방법이 필요하다.

또한, 부하 분배는 분산되어 있는 자원을 효율적으로 사용하고 높은 성능을 얻기 위해서 자원의 상태와 각 노드의 부하 상태에 따라 자원을 적절히 분배하여 부하를 균등하게 분배 하는 것이다[47]. RFID 미들웨어에 부하가 집중되는 현상이 지속 된다면 처리의 효율성이 낮아짐은 물론 처리 지연으로 인한 비즈니스 플로우의 지연 및 장애가 발생하게 된다. 이와 같이 부하가 특정 노드에 집중되는 현상을 방지하기 위해서는 부하 분산기법의 도입이 필요하다[48].

과부하 문제를 해결하기 위한 방법은 단일 노드의 성능을 향상시키는 것과 클러스터링 기법을 이용하는 것으로 나누어질 수 있다. 단일 노드의 성능을 개선하여 하는 방법을 선택할 경우 성능 개선동안 서비스가 중단되어야함은 물론 요청이 증가되면 머지않아 다시 과부하가 되어 더 높은 성능의 노드로 다시 개선해 주어야한다. 이와는 반대로 클러스터링 기법을 이용하면, 확장성과 비용의 효율성 및 신뢰성의 측면에서 큰 이점을 얻을 수 있다[48].

III. 제안하는 부하 분산 시스템

1. 개요

그림 26은 본 논문에서 제안하는 대량 RFID 데이터 처리를 위한 부하 분산 방법의 개요이다. 다양한 유형의 대량 RFID Tag 데이터를 로지컬 리더 데이터 브로드캐스터가 수집하여 미들웨어 분산 시스템에 구축되어 있는 여러 대의 ALE 기반 RFID 미들웨어로 전송한다. 이후 이렇게 수집되는 RFID 데이터를 제공받고자 하는 클라이언트 응용으로부터 수집 및 가공 처리 요청을 로드밸런서가 받아 미들웨어 분산 시스템 환경에 구축되어 있는 각 ALE RFID 미들웨어의 CPU, Memory, Process 수, DB 커넥션 등 다양한 자원 정보를 수집한다.

그리고, Weighted Round Robin 방식의 알고리즘을 사용하여 부하 분산을 하며, 다양한 factor의 서버 한계 상황 반영 비율을 계산하기 위하여 유전자 알고리즘을 이용하여 최적의 반영 비율을 찾는다.

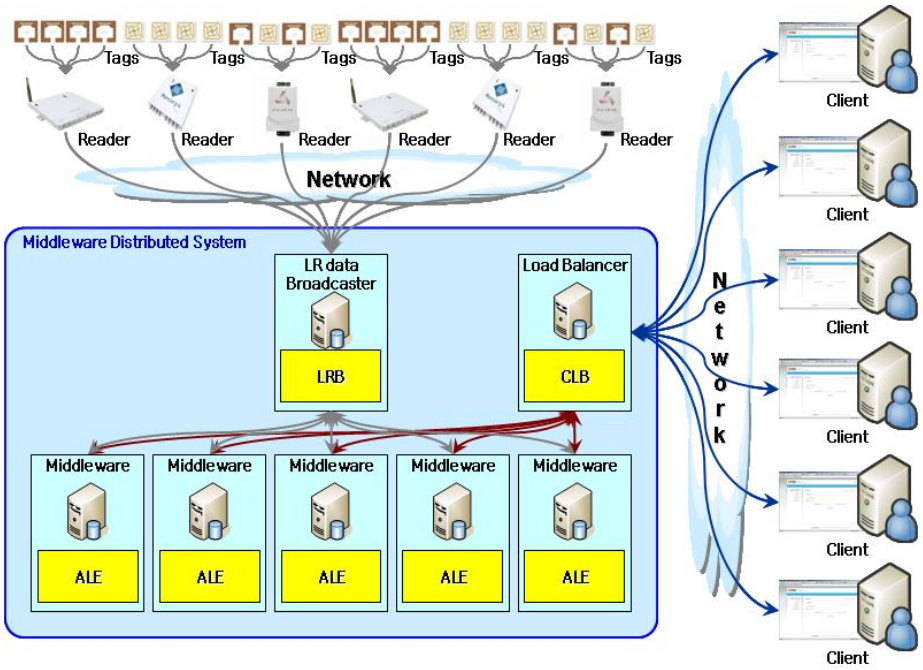

그림 26. 제안 시스템의 개요

2. RFID 미들웨어 분산 시스템

그림 27은 본 논문에서 제안하는 ALE 처리 측면의 구조도이다. 로지컬 리더 데이터 브로드 캐스터는 RFID Tag 데이터들을 수집하여 미들웨어 분산 시스템의 ALE RFID 미들웨어로 전송하며, 로드밸런서는 클라이언트의 RFID 데이터 수집 및 가공 처리 요청을 받아 처리할 적절한 미들웨어를 선정하여 ECSpec 정보와 클라이언트 처리 요청에 따른 ALE API를 호출하고, 처리 요청을 받은 ALE RFID 미들웨어는 해당 ECSpec에 따라 RFID 데이터를 처리하여 ECReports 형태로 정리하여 클라이언트에게 전송한다.

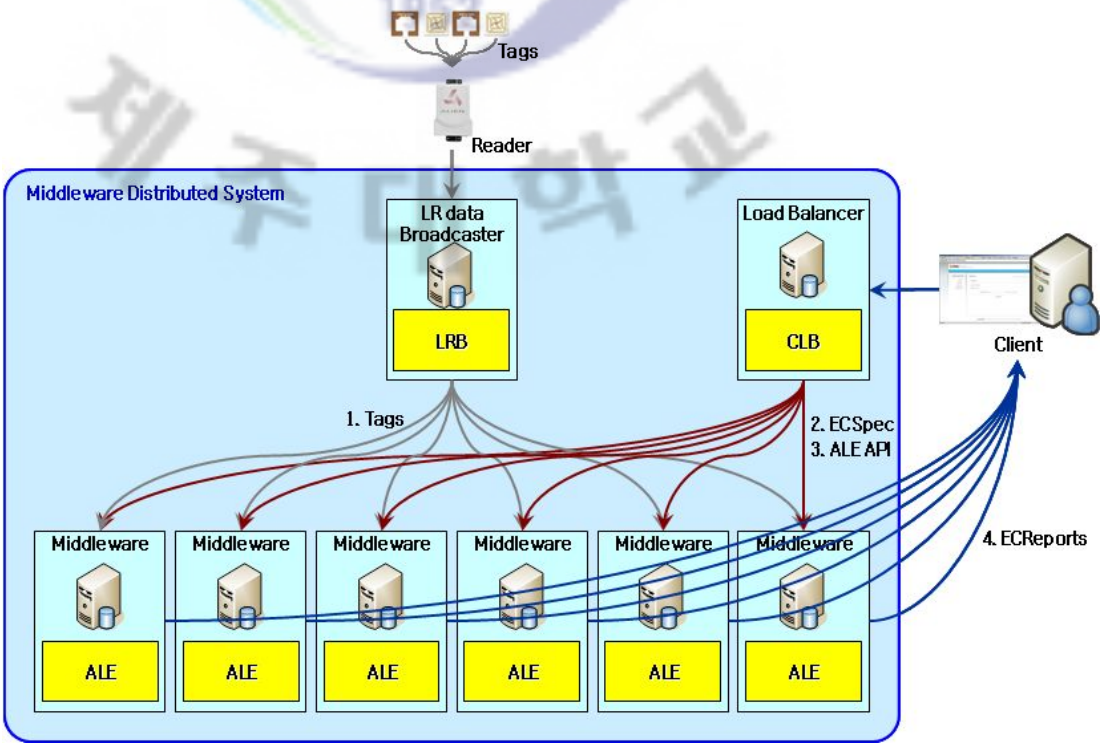

그림 27. ALE 처리 구조

그림 28은 본 논문에서 제안하는 부하 분산 처리 구조도이다. 로지컬 리더 데이터 브로드 캐스터는 모든 미들웨어로 로지컬 리더 1, 2, 3 으로부터 수집된

RFID Tag 데이터들을 전송한다. 클라이언트는 로지컬 리더 2와 3에서 수집되는 RFID 데이터의 수집 및 가공 처리를 하기 위하여 처리 사항에 따른 명세서인 sample 이라는 ECSpec을 Define ALE API를 호출하여 미들웨어 분산 시스템의 공용 DB에 저장한다. 이후 클라이언트 응용의 RFID 데이터 수집 및 가공 처리 요청을 Subscribe ALE API를 호출하여 처리한다.

클라이언트의 처리 요청을 전달 받은 로드밸런서는 미들웨어 분산 시스템의 각 ALE RFID 미들웨어의 자원 사항을 수집하고, 공용 DB에서 sample 이라는 이름을 갖는 ECSpec 정보를 제공 받아 처리 부하 사항 정보에 따라 유전자 알고리즘을 이용하여 계산되어진 가중치 정보를 로드밸런서 DB에서 불러온다. 이후 로드밸런서는 각 ALE RFID 미들웨어서 수집된 자원 정보와 가중치 정보를 계산하여 적정 ALE RFID 미들웨어를 선정한다. 끝으로 로드밸런서는 클라이언트 응용의 수집 및 가공 처리 사항을 선정된 ALE RFID 미들웨어로 Subscribe(Sample, url) 라는 ALE API를 호출하여 전송한다. 이러한 처리 사항을 전달 받은 ALE RFID 미들웨어는 sample 이라는 ECSpec에 명시된 처리사항에 맞게 처리하여 Subscribe API 에 명시되어 있는 클라이언트 응용의 url 주소로 처리된 결과인 ECReports 정보를 제공한다.

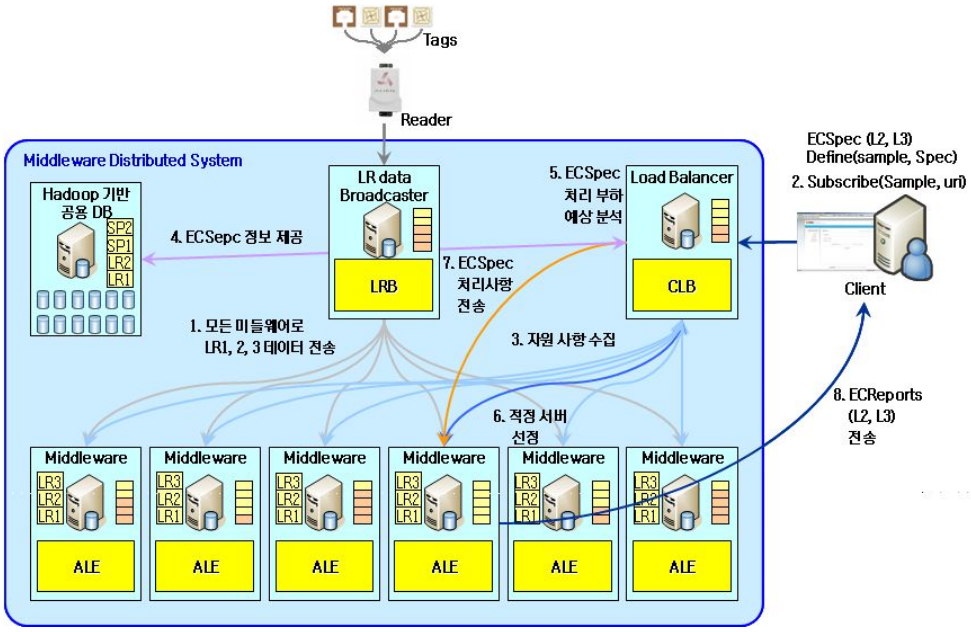

그림 28. 부하 분산 처리 구조

3. RFID 미들웨어 자원 정보 수집

본 논문에서는 클라이언트의 RFID 데이터 수집 및 가공 처리 요청을 분산하기 위하여, 분산 환경에 구축된 여러 대의 RFID 미들웨어의 자원 정보를 수집하고, 각 RFID 미들웨어의 자원 사항에 맞게 클라이언트 요청을 처리할 수 있도록 해야 한다. 이를 위하여 기 구현된 ALE 스펙 기반 RFID 미들웨어의 자원 사항을 하드웨어 자원, 미들웨어 자원, ALE 자원으로 구분하여 개별 저장하며, 부하 분산 시스템에서 자원 정보를 요청했을 시에 개별 저장된 자원 정보를 SOAP 통신을 이용하여 XML 형태로 로드밸런서에 제공한다.

1) 하드웨어 자원 정보

표 10은 미들웨어 분산 시스템에 구축된 ALE RFID 미들웨어에서 로드밸런서가 수집하는 하드웨어 자원 정보이다.

표 10. 하드웨어 자원 정보

자원 정보명	설 명
ProcessCount	서버에서 실행되고 있는 모든 프로세스 수
TotalMemory	서버에서 사용되고 있는 메모리 자원
JavaUseMomory	JAVA 기반으로 구현된 미들웨어에서 사용되고 있는 메모리 자원
TCPUUse	서버에서 사용되고 있는 CPU 자원
CPUUse	미들웨어에서 사용되고 있는 CPU 자원
HeapMemoryUsage	미들웨어 실행 시 사용되는 힙 메모리 자원
NonHeapMemoryUsage	미들웨어 실행 시 아용되는 논힙 메모리 자원
AvailableProcessors	서버의 CPU 수 (예 : Dual Core = 2)
TotalPhysicalMemorySize	서버의 RAM 총 크기
FreePhysicalMemorySize	서버에서 사용할 수 있는 RAM 크기
TotalSwapSpaceSize	서버의 Swap 메모리 총 크기
FreeSwapSpaceSize	서버에서 사용할 수 있는 Swap 메모리 크기
CommittedVirtualMemorySize	서버에서 활용되고 있는 가상 메모리 크기
ConnectionSpeed	서버와 로드밸런서간의 네트워크 초단위 속도

2) 미들웨어 자원 정보

표 11은 미들웨어 분산 시스템에 구축된 ALE RFID 미들웨어에서 로드밸런서가 수집하는 일반적인 미들웨어 자원 정보이다.

표 11. 미들웨어 자원 정보

자원 정보명	설 명
DBConnectionCount	미들웨어에서 생성한 DB 접속 객체 수
ThreadCount	미들웨어에서 실행되고 있는 스레드 수
PeakThreadCount	미들웨어에서 절정으로 실행되고 있는 스레드 수
DaemonThreadCount	Back Ground 형태로 실행되는 스레드 수
TotalStartedThreadCount	미들웨어에서 실행된 스레드의 총 수
LoadedClassCount	미들웨어에서 생성되어 객체화된 클래스의 수
UnloadedClassCount	미들웨어에서 객체로 생성되어 처리가 종료된 클래스의 수
TotalLoadedClassCount	미들웨어에서 생성되어 객체화된 클래스의 총 수

3) ALE 자원 정보

표 12는 미들웨어 분산 시스템에 구축된 ALE RFID 미들웨어에서 로드밸런서가 수집하는 ALE 스펙을 기반으로 개발된 미들웨어에서만 수집 가능한 ALE 자원 정보이다.

표 12. ALE 자원 정보

자원 정보명	설 명
LogicalReaderCount	미들웨어 등록된 논리 리더의 수
ActivedLogicalReaderCount	미들웨어에서 실제 RFID Tag 데이터를 수집하고 있는 논리 리더의 수
ECSpecCount	미들웨어 등록된 ECSpec의 수
ActivedEventCycleCount	미들웨어에서 처리되고 있는 EventCycle의 수
EventCycleDataCount	전 EventCycle에서 클라이언트로 보내진

	RFID Tag 데이터의 수
LogicalQueueCount	미들웨어에서 논리 리더로부터 수집되는 RFID Tag 데이터를 저장하기 위해 생성된 논리 큐 수
EventQueueCount	미들웨어에서 처리되고 있는 EventCycle에서 클라이언트 응용에게 전송하기 위해 논리 큐에 저장되어진 RFID Tag 데이터를 ECSpec 조건에 따라 가공 처리된 데이터를 저장하기 위해 생성된 이벤트 큐 수
LogicalQueueSize	논리 큐에 저장된 RFID Tag 데이터의 수
EventQueueSize	이벤트 큐에 저장된 클라이언트 응용으로 보낼 RFID Tag 데이터의 수

4. RFID 미들웨어 부하 부산 방법

본 절에서는 대량 RFID 데이터 처리를 위한 부하 분산을 위한 방법을 기술한다. 본 논문에서는 대량 RFID 데이터 처리를 위한 부하 분산을 위하여 미들웨어 분산 환경에 구축된 각 ALE RFID 미들웨어의 하드웨어, 미들웨어, ALE 자원 정보를 수집하고, 유전자 알고리즘을 이용하여 구한 가중치 정보를 자원 정보와 연계하여 ALE RFID 미들웨어의 자원 정보를 계산한다. 이렇게 계산된 자원 정보를 바탕으로 가장 낮은 자원 정보를 갖고 있는 ALE RFID 미들웨어로 클라이언트의 RFID Tag 데이터 수집 및 가공 처리를 전달한다.

1) RFID 미들웨어 부하 정보 계산

본 논문에서는 대량 RFID 데이터 처리를 위한 부하 분산을 위하여 미들웨어 분산 환경에 구축된 ALE RFID 미들웨어 중 요청된 클라이언트의 수집 및 가공 처리 사항을 처리할 최적의 ALE RFID 미들웨어를 선정하기 위해 미들웨어 분산 환경에 구축된 각 ALE RFID 미들웨어의 하드웨어, 미들웨어, ALE 자원 정보를 수집하고, 유전자 알고리즘을 이용하여 구한 가중치 정보를 자원 정보와 연계하여 ALE RFID 미들웨어의 부하 정보를 계산한다.

그림 29는 ALE RFID 미들웨어의 하드웨어 부하 정보인 $L_{iHARDWARE}$ 값을 수집된 ProcessCount, TotalMemory 등과 같은 하드웨어 자원 정보와 유전자 알고리즘을 이용하여 구한 W_{pc}, W_{tm} 등과 같은 가중치 값과 계산하는 식이다.

$$
\begin{aligned}
L_{iHARDWARE} = {} & W_{pc} \times L_{iProcessCount} \\
& + W_{tm} \times L_{iTotalMomory} \\
& + W_{jum} \times L_{iJavaUseMomory} \\
& + W_{tcu} \times L_{iTCPUUse} \\
& + W_{cu} \times L_{iCPUUse} \\
& + W_{hmu} \times L_{iHeapMemoryUsage} \\
& + W_{nhmu} \times L_{iNonHeapMemoryUsage} \\
& + W_{ap} \times L_{iAvailableProcessors} \\
& + W_{tpms} \times L_{iTotalPhysicalMemorySize} \\
& + W_{fpms} \times L_{iFreePhysicalMemorySize} \\
& + W_{tsss} \times L_{iTotalSwapSpaceSize} \\
& + W_{fwss} \times L_{iFreeSwapSpaceSize} \\
& + W_{cvms} \times L_{iCommittedVirtualMemorySize} \\
& + W_{cs} \times L_{iConnectionSpeed}
\end{aligned}
$$

그림 29. 하드웨어 부하 정보 계산

그림 30은 ALE RFID 미들웨어의 미들웨어 부하 정보인 $L_{iMIDDLEWARE}$ 값을 수집된 DBConectionCount, ThreadCount 등과 같은 미들웨어 자원 정보와 유전자 알고리즘을 이용하여 구한 W_{dc}, W_{tc} 등과 같은 가중치 값과 계산하는 식이다.

$$
\begin{aligned}
L_{iMIDDLEWARE} = {} & W_{dc} \times L_{iDBConnectionCount} \\
& + W_{tc} \times L_{iThreadCount} \\
& + W_{ptc} \times L_{iPeakThreadCount} \\
& + W_{dtc} \times L_{iDaemonThreadCount} \\
& + W_{tstc} \times L_{iTotalStartedThreadCount} \\
& + W_{lcc} \times L_{iLoadedClassCount} \\
& + W_{ucc} \times L_{iUnloadedClassCount} \\
& + W_{tlcc} \times L_{iTotalLoadedClassCount}
\end{aligned}
$$

그림 30. 미들웨어 부하 정보 계산

그림 31은 ALE RFID 미들웨어의 미들웨어 부하 정보인 L_{iALE} 값을 수집된 LogicalReaderCount, ActivedLogicalReaderCount 등과 같은 ALE 자원 정보와 유전자 알고리즘을 이용하여 구한 W_{lrc}, W_{alrc} 등과 같은 가중치 값과 계산하는 식이다.

$$
\begin{aligned}
L_{iALE} = {} & W_{lrc} \times L_{iLogicalReaderCount} \\
& + W_{alrc} \times L_{iActivedLogicalReaderCount} \\
& + W_{ecsc} \times L_{iECSpecCount} \\
& + W_{aecc} \times L_{iActivedEventCycleCount} \\
& + W_{ecdc} \times L_{iEventCycleDataCount} \\
& + W_{lqc} \times L_{iLogicalQueueCount} \\
& + W_{eqc} \times L_{iEventQueueCount} \\
& + W_{lqs} \times L_{iLogicalQueueSize} \\
& + W_{eqs} \times L_{iEventQueueSize}
\end{aligned}
$$

그림 31. ALE 부하 정보 계산

그림 32는 ALE RFID 미들웨어의 총 부하 정보인 T_{iLoad} 값을 위와 같이 계산된 $L_{iHARDWARE}$, $L_{iMIDDLEWARE}$, L_{iALE} 와 같은 ALE RFID 미들웨어 부하 정보를 사용하여 계산하는 식이다.

$$
\begin{aligned}
T_{iLoad} = {} & L_{iHARDWARE} \\
& + L_{iMIDDLEWARE} \\
& + L_{iALE}
\end{aligned}
$$

그림 32. 총 부하 정보 계산

2) 부하 분산을 위한 가중치 생성

본 논문에서는 분산 환경에 구축된 ALE RFID 미들웨어 중 요청된 클라이언트의 수집 및 가공 처리 사항을 처리할 최적의 ALE RFID 미들웨어를 선정하기 위해 하드웨어, 미들웨어, ALE 자원 정보와 유전자 알고리즘을 이용하여 구한

가중치 정보를 연계하여 ALE RFID 미들웨어의 부하 정보를 계산하며, 부하 정보가 가장 적은 ALE RFID 미들웨어로 처리사항을 전달한다.

분산 환경에 구축된 서로 다른 사양의 ALE RFID 미들웨어에서 특정 ECSpec이 처리될 시의 자원 정보를 수집하여, 분산 환경에서 이 ECSpec이 처리 될 시에 부하가 균등하게 될 수 있도록 분산하기 위한 가중치 정보를 유전자 알고리즘의 연산 방법을 이용하여 구한다.

(1) 표현 방법

유전자 알고리즘을 이용하여 클라이언트 응용의 RFID Tag 데이터 수집 및 가공처리 요청을 ALE RFID 미들웨어에 할당하는 최적의 가중치 수치를 찾기 위해서는 우선 유전자 알고리즘에서 요구하는 형식의 표현 방법이 필요하다. 일반적으로 모의 해집단을 표현하기 위해서는 바이너리 스트링(binary string) 표현을 많이 사용하지만 다양한 형태의 ECSpec 처리 요청을 분산하기 위해서는 적합하지 않다.

유전자 알고리즘의 염색체 표현방법에는 이진표현 방법과 부동점 표현방법이 있다. 이진표현 방법은 염색체를 유한 길이의 2진 스트링으로 부호화하고 스트링에 대한 복호화 절차를 필요로 하며, 부동점 표현 방법은 각 염색체 벡터를 해 벡터와 같은 길이의 부동점 수의 벡터로 코딩하기 때문에 부호화나 복호화의 절차가 필요치 않다.

전자의 코딩기법은 전통적으로 사용되어온 코딩기법으로 다차원 고정밀의 수치해석 문제에 적용시 정밀도와 정의영역의 한계 내에서 넓은 탐색공간을 가지며 정밀도를 확장하기 위해 더 많은 염색체 비트를 도입할 수 있으나 알고리즘의 속도가 느려지는 단점을 가진다.

후자의 코딩기법은 각 염색체 벡터를 해 벡터와 같은 길이의 부동점수로 코딩하기 때문에 전자보다는 표준편차나 속도 및 염색체 총 길이에 효율적인 성능향상을 나타낸다. 특히, 대규모 정의역역 한계 내에서 특별히 고안된 유전자 연산자에 의해 높은 정밀도를 가지며, 표현하는 기법 면에서 주어진 문제의 변수에 가깝게 코딩하므로 문제와 관련된 특성을 포함하는 연산자의 설계가 용이하고

구속조건을 다루는데 편리한 장법을 가진다[49].

본 논문에서는 전체 개체집단의 코딩기법에 따라 비전역 최적값으로 조기 수렴하고 미세한 국소조정을 하지 못하거나 구속조건이 존재할 때 동작하지 않는 단점을 보안하기 위해 한 개의 유전자 대 한 개의 변수가 일치하고 돌연변이 연산의 비트 오류나 인공적인 작업이 불필요한 실변수 코딩기법을 사용한다[25].

이러한 코딩기법을 사용한 실변수 유전자 알고리즘(Real-Variable Genetic Algorithm : RVGA)의 기본요소는 탐색할 각각의 변수를 실제 값으로 구성하며, 스트링은 모든 변수를 포함하는 벡터로서 표현할 수 있다. 예를 들어 함수 $f(x_1, x_2, ..., x_n)$를 최소화하는 문제를 고려하면 변수들은 다음의 스트링 $s = (x_1, x_2, ..., x_n) \in R^N$ 로 표현할 수 있으며, R^N 은 N 차원 탐색공간이다. 스트링 s을 하나의 개체(individual)라고 하며, 개체들의 집합을 개체집단(population)이라고 한다.

본 논문에서는 유전자에 그림 33과 같은 ECSpec을 서로 사양이 다른 4대의 ALE RFID 미들웨어에서 처리할 시에 메모리 사용량 등의 자원 사항을 정규화하여 표현하였다.

```
<?xml version="1.0" encoding="UTF-8"?>
<ale:ECSpec xmlns:ale="urn:epcglobal:ale:xsd:1" xmlns:epcglobal="urn:epcglobal:ale:xsd:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" includeSpecInReports="true" schemaVersion="1.0" xsi:schemaLocation="urn:epcglobal:ale:xsd:1 Ale.xsd">
<logicalReaders>
        <logicalReader>FRONT_DOOR</logicalReader>
        <logicalReader>BACK_DOOR</logicalReader>
</logicalReaders>

<boundarySpec>
        <repeatPeriod unit="MS">10000</repeatPeriod>
        <duration unit="MS">5000</duration>
</boundarySpec>

<reportSpecs>
        <reportSpec reportIfEmpty="true" reportName="report1" reportOnlyOnChange="true">
        <reportSet set="CURRENT"/>
                <filterSpec>
                        <includePatterns>
                                <includePattern>urn:epc:tag:sgtin-64:1,1,*,*</includePattern>
                        </includePatterns>
                        <excludePatterns>
                                <excludePattern>urn:epc:tag:sgtin-64:1,1,1,*</excludePattern>
                                <excludePattern>urn:epc:tag:sgtin-64:1,1,2*</excludePattern>
                        </excludePatterns>
                </filterSpec>
                <groupSpec>
                        <pattern>urn:epc:tag:sgtin-64:1,1,X,*</pattern>
                </groupSpec>
                <output includeCount="true" includeEPC="true" includeRawDecimal="true" includeRawHex="true" includeTag="true"/>
        </reportSpec>
</reportSpecs>
</ale:ECSpec>
```

수집할 논리 리더 정보 정의

논리 리더로 부터 수집할 주기 : 5초간 수집, 10초 마다 보고

필터링 처리 조건 정의

그룹핑 처리 조건 정의

그림 33. ecspec_lb40 ECSpec 내역

그림 33의 ECSpec은 미들웨어 분산 환경의 ALE RFID 미들웨어 ecspec_lb40이라는 이름으로 등록되어 있으며, FRONT_DOOR, BACK_DOOR라는 2개의 논리 리더로부터 RFID Tag 데이터를 5초간 수집하여, 명세된 필터링, 그룹핑 등의 가공 처리 사항들을 처리하여 10초마다 이 스펙의 처리를 요청한 클라이언트 응용에게 ECReports 형태로 전송한다.

미들웨어 분산 환경에 구축된 서로 사양이 다른 4대의 ALE RFID 미들웨어에서 각각 ecspec_lb40 스펙을 처리할 시에 각 미들웨어에서 사용되는 하드웨어, 미들웨어, ALE 자원 정보는 모두 다르며, 이렇게 다른 자원 정보에 따라 가중치 정보를 구하여 자원 사항을 정리할 필요가 있다.

그림 33의 ECSpec를 서로 다른 사양의 A, B, C, D 4대의 미들웨어에서 처리했을 시에 실제 메모리 사용량은 A 미들웨어 86,576KB, B 미들웨어 145,056KB, C 미들웨어 149,052KB, D 미들웨어 148,260KB이며, 이 메모리 사용량을 정규화하면, 각각 A 미들웨어 0.163677062, B 미들웨어 0.27423697, C 미들웨어 0.281791645, D 미들웨어 0.280294322이다.

이렇게 정규화 된 수치는 ecspec_lb40 스펙이 4대의 서버에 분산 되었을 경우 비례하게 증가하며, 최대로 부하 분산이 이뤄지면 이 수치의 총 합이 4와 가까워진다. 4개 수치의 총합이 4에 가까워지면 졌을 시에 정규화된 4개의 평균값을 구해 본 논문에서 제안하는 부하 분산 방법의 가중치 정보로 활용한다.

그림 34는 이렇게 구한 4개의 정규화된 자원 정보를 유전자 염색체에 표현한 방법이다.

그림 34. 유전자 알고리즘을 위한 문제의 표현

(2) 모의 해집단의 구성

모의 해 집단은 그림 35와 같이 m 개의 클라이언트 응용의 RFID Tag 데이터 수집 및 가공 처리 요청을 n개의 서버에 할당하는 경우 중 랜덤하게 모의 해 집단을 구성한다.

본 논문의 n 값은 미들웨어 분산 환경에 4대의 ALE RFID 미들웨어 서버를 구축하였기 때문에 4가 된다.

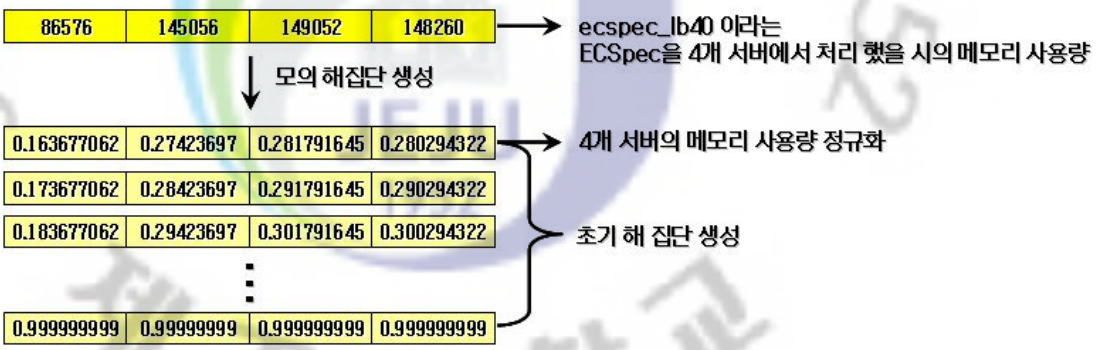

그림 35. 초기 해 집단 생성

이러한 방법을 사용하여 ecspec_lb40과 유사하나 처리 조건이 서로 다른 50개의 ecspec_lb1 ~ ecspec_lb50 의 ECSpec들을 분산 미들웨어 환경에 구축된 서로 사양이 다른 4개의 ALE RFID 미들웨어 서버에서 실행됐을 시의 표10, 11, 12의 자원 각각을 정규화하여 그림 35와 같은 초기 해 집단을 생성한다.

이렇게 생성한 해 집단을 본 논문에서 제안하는 적합도 함수를 바탕으로 최적으로 부하 균등이 이뤄질 수 있는 상황 정보를 찾아, 이후 서로 사양이 다른 4개의 ALE RFID 미들웨어 서버에서 실행됐을 시에 최대한 부하가 균등하게 될 수 있는 상황 정보로 활용된다.

(3) 적합도 함수

각 모의해의 질을 평가하기 위해 적합도 함수를 작성한다. 50개의 ecspec_lb1

~ ecspec_lb50 의 ECSpec을 중 ecspec_lb40이라는 이름의 ECSpec을 4대의 서로 사양이 다른 ALE 미들웨어 서버로 클라이언트의 수집 및 가공 처리사항을 분산하기 위하여 ecspec_lb40을 4대의 각 서버에서 실행하여 최적의 부하 균등 상황이 됐을 시의 값을 유전자가 연산을 통해 얻으며, 적합도 함수는 다음과 같다.

n = 부하 분산 대상 ALE RFID 미들웨어의 수

D_i^{avg} = ECSpec을 처리할 시 자원 정규화 값의 분포도 평균 값

$$F(x) = \sum_{i=1}^{n} D_i^{avg}$$

$$\text{적합도 값} = \frac{1}{F(x)}$$

적합도 함수가 0에 가까울수록 부하 분산이 잘 되었음을 의미하고, 1에 가까울수록 부하 분산이 잘되지 않았음을 의미함으로 0에 가까운 값을 갖는 해집합이 선택될 확률이 높다. 자원 정규화 분포도 평균값인 D_i^{avg}값을 구하기 위하여 특정 ECSpec이 미들웨어 분산 환경의 각 ALE RFID 미들웨어에서 실행됐을 시의 자원 정보의 정규화 값과 분포도 값이 필요하다.

특정 ECSpec이 처리될 시의 ALE RFID 미들웨어의 CPU 부하 정보의 정규화 값과 정규화 값의 분포도 값을 구하는 식은 다음과 같다.

LN_i^{CPU} = i 번째 ALE RFID 미들웨어의 CPU 정규화 값

D_i^{CPU} = I 번째 ALE RFID 미들웨어의 CPU 분포도 값

L_S^{CPU} = i 번째 ALE RFID 미들웨어의 CPU 부하 값

$\sum_{k=1}^{i} L_k^{CPU}$ = 미들웨어 분산 환경에 구축된 ALE RFID 미들웨어의 CPU 부하 값 총 합계

$\sum_{k=1}^{i} LN_k^{CPU}$ = 미들웨어 분산 환경에 구축된 ALE RFID 미들웨어의 CPU 정규화 값 총 합계

$$LN_i^{CPU} = \frac{L_S^{CPU}}{\sum_{k=1}^{i} L_k^{CPU}} (S = 1, 2, ..., i)$$

$$D_i^{CPU} = \frac{LN_s^{CPU}}{\sum_{k=1}^{i} LN_k^{CPU}} (S = 1, 2, ..., i)$$

다음은 ALE RFID 미들웨어의 CPU 부하 정규화 값의 평균과 분포도 값의 평균을 구하는 식이다. D^{CPUarg}은 적합도 함수에서 D_i^{avg}와 같은 값이며, 유전자 연산을 통해 부하 분산이 최적으로 잘 되었을 경우의 값을 찾으면 D^{CPUarg}은 0에 가까워진다. 본 논문에서는 D^{CPUarg}값을 클라이언트의 RFID Tag 데이터 수집 및 가공 처리 사항을 처리할 최적의 ALE RFID 미들웨어 선정을 위한 자원 가중치 정보로 활용한다.

이렇게 찾은 부하 균등 상황에서 자원 정규화 평균값을 실제 수집되는 각 자원 정보에 가중치 정보로 적용하면, 찾았던 부하 균등 상황이 이뤄 질수 있도록 유도하고, 30여개의 자원 정보의 특성상 이러한 상황을 이상적으로 찾을 수 없는 경우 최적으로 부하 균등이 이뤄진 상황을 찾기 위해 최적화 알고리즘인 유전자 알고리즘을 사용하였으며, 유전자 알고리즘을 사용하여 30여개의 자원 중요도 정보에 해당하는 가중치 정보를 생성하여 최적으로 클라이언트의 처리 사항을 분산 할 수 있도록 하였다.

$$LN^{CPUarg} = \frac{\sum_{k=1}^{i} LN_k^{CPU}}{n}$$

$$D^{CPUarg} = \frac{\sum_{k=1}^{i} D_k^{CPU}}{n}$$

(4) 부하 분산을 위한 유전자 알고리즘의 흐름

유전자 알고리즘은 m 개의 클라이언트 응용의 RFID Tag 데이터 수집 및 가공처리 요청을 n개의 서버에 할당하기 위한 최적의 가중치 정보를 제시하는 알고리즘이다. 유전자 알고리즘은 그림 36과 같은 흐름을 갖는다.

P(t)는 세대 t의 모의 집단이며, Ps는 선택 확률, Pc는 교배 확률, Pm은 돌연변이 확률이다.

```
procedure GA()
begin
        t = 0;
        initialize P(t);
        evaluate P(t);
        while not finished do
        begin
                t = t + 1;
                selection P(t), Ps;
                Crossover P(t), Pc;
                Mutation P(t), Pm;
                evaluate P(t);
                reproduce P(t);
        end
end
```

그림 36. 유전자 알고리즘

유전자 알고리즘은 다음과 같은 절차에 따라 진화한다.

단계 1 : (초기모집단), t ← 으로 초기화 하고 P(t)를 생성한다. 즉, 초기 모집단을 생성한다.

단계 2 : (적합도 평가), P(t)에 있는 모든 개체의 적합도를 평가한다.

단계 3 : (선택), t ← t + 1 로 증가시키고 P(t-1)로부터 P(t)를 선별한다.

단계 4 : (교차), 1) [0, 1] 사이의 난수를 r을 발생시켜, r < Pc 이면 그 개체를 교차하는 개체로 둔다. 모든 개체의 대해 이를 수행한다.

2) 교차 대상 개체 중에서 임의로 쌍(두 부모)을 선택한다.

3) 각 쌍(두 부모)의 개체를 교차하여 두 자손을 생산한다. P(t)에서 교차된 부모 개체를 제거하고 생산된 자손을 P(t)에 삽입한다.

단계 5 : (돌연변이), P(t)를 돌연변이 시킨다. 개체의 각 원소에 대해 [0, 1] 사이의 난수 r을 발생시켜, r < Pm 이면 그 원소를 돌연변이 시킨다.

단계 6 : (적합도 평가), P(t)의 각 개체에 대해 적합도를 평가한다.

5. 부하 분산 시스템의 아키텍처

그림 37은 본 논문에서 제안하는 대량 RFID 데이터 처리를 위한 부하 분산 시스템의 아키텍처이다. 부하 분산 시스템은 크게 ALE 기반 RFID 미들웨어의 자원 정보 수집, 저장, 제공 모듈과 로드밸런서의 ALE RFID 미들웨어 자원 정보 수집, GA(유전자 알고리즘) 기반 가중치 관리, 부하 분산 관리등으로 나뉜다.

ALE 기반 RFID 미들웨어에는 부하 분산을 위해 각 ALE RFID 미들웨어의 하드웨어, 미들웨어, ALE 자원 정보를 5초 간격으로 수집하여 DB에 저장하며, 로드밸런서가 부하 분산을 위해 자원 정보를 요청할 시에 DB에 저장된 부하 상황에 해당하는 자원 정보를 XML 형태로 생성하고, SOAP 통신을 이용하여 제공한다. 이후 로드밸런서는 부하 상황에 해당하는 자원 정보에 유전자 알고리즘(GA)을 이용하여 구한 가중치 정보를 적용하여 자원 정보가 가장 적은 ALE RFID 미들웨어로 클라이언트의 RFID Tag 데이터 수집 및 처리 요청을 분산한다.

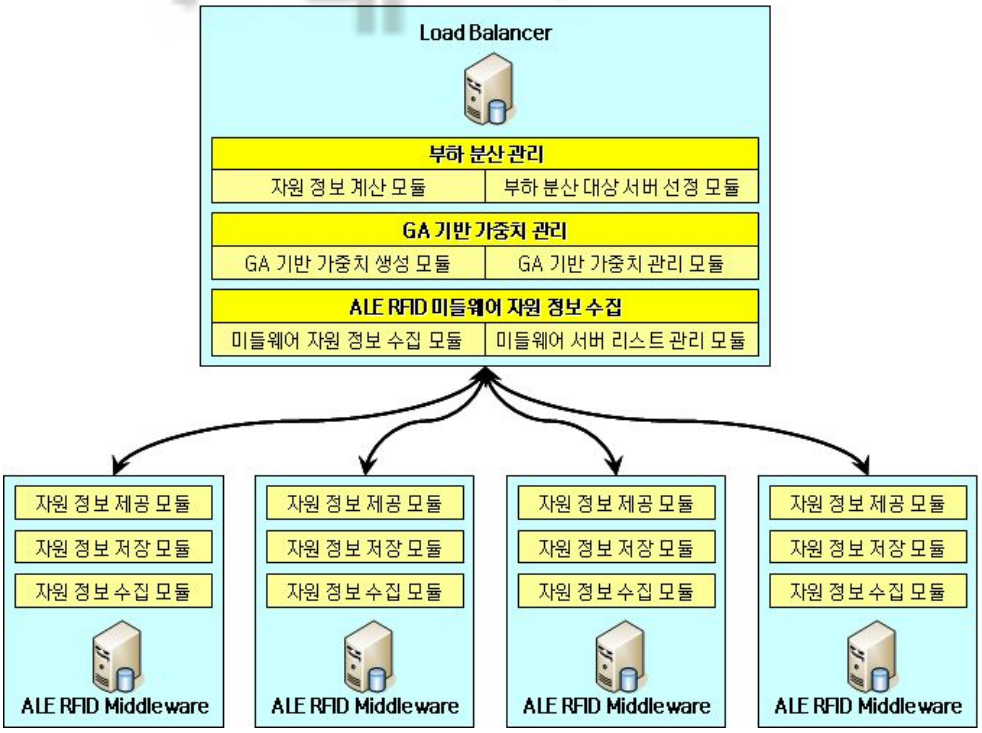

그림 37. 부하 분산시스템의 아키텍처

6. 부하 분산 시스템의 구성요소

1) ALE 기반 RFID 미들웨어

본 논문의 부하 분산 시스템에서 ALE RFID 미들웨어에는 자원 정보 수집, 자원 정보 저장, 자원 정보 제공 모듈이 ALE 스펙을 기반으로 구성된 미들웨어 기능과 함께 구성된다.

자원 정보 수집 모듈은 ALE 기반 RFID 미들웨어 클라이언트의 RFID Tag 데이터 수집 및 가공 처리 사항을 수행 시에 서버에서 실행되고 있는 모든 프로세스 수, 서버에서 사용되고 있는 메모리 자원 등의 하드웨어 부하 상황에 해당하는 자원 정보와 미들웨어에서 생성한 DB 접속 객체 수, 미들웨어에서 실행되고 있는 스레드 수 등의 미들웨어 부하 상황에 해당하는 자원 정보와 미들웨어 등록된 논리 리더의 수, 미들웨어에서 실제 RFID Tag 데이터를 수집하고 있는 논리 리더의 수 등의 ALE 부하 상황에 해당하는 자원 정보를 java 및 sun의 management API 등을 사용하여 수집하고, 하드웨어 자원, 미들웨어 자원, ALE 자원 공용 자료 구조 형태로 관리한다.

자원 정보 저장 모듈은 자원 정보 수집 모듈에서 관리되고 있는 하드웨어 자원, 미들웨어 자원, ALE 자원 공용 자료 구조의 정보를 데이터베이스에 저장한다. 자원 정보 제공 모듈은 자원 정보 저장 모듈이 DB에 저장한 ALE RFID 미들웨어의 부하 상황에 해당하는 자원 정보를 부하 분산 시스템의 로드밸런서가 요청할 시에 XML 구조 형태로 생성하여 SOAP 통신을 이용하여 제공한다.

2) 부하 분산 시스템

본 논문의 부하 분산 시스템에서 로드밸런서는 미들웨어 서버 리스트 관리, 미들웨어 자원 정보 수집, GA 기반 가중치 생성, GA 기반 가중치 관리, 자원 정보 계산, 부하 분산 대상 서버 선정 모듈로 구성된다.

로드밸런서는 클라이언트의 RFID Tag 데이터 수집 및 가공 처리 요청을 받게

되면 미들웨어 서버 리스트 모듈은 분산 미들웨어 환경에 구축된 각 ALE RFID 미들웨어의 서버 정보를 데이터베이스에서 불러오며, 미들웨어 자원 정보 수집 모듈은 미들웨어 서버 리스트 관리 모듈에서 불러온 서버 정보를 바탕으로 각 ALE RFID 미들웨어의 부하 상황에 해당에는 자원 정보를 SOAP 통신을 이용하여 요청하고, 결과로 반환된 XML 구조 형태의 하드웨어, 미들웨어, ALE 자원 정보를 로드밸런서의 데이터베이스에 저장한다.

GA 기반 가중치 생성 모듈은 본 논문에서 제안한 유전자 연산 과정을 통하여 특정 ECSpec이 최적으로 부하 분산이 이뤄졌을 때의 분포도의 평균값인 가중치 정보를 생성하여 로드밸런서의 데이터베이스에 저장한다. GA 기반 가중치 관리 모듈은 클라이언트의 RFID Tag 데이터 수집 및 가공 처리 요청을 분산할 ALE RFID 미들웨어 선정을 위한 자원 정보 계산 시에 가중치 정보를 적용할 수 있도록 가중치 정보를 관리한다.

자원 정보 계산 모듈은 미들웨어 자원 정보 수집 모듈이 저장한 ALE RFID 미들웨어의 부하 상황에 해당하는 각 자원 정보와 GA 기반 가중치 생성 모듈에서 구한 각 자원 가중치 정보를 그림 29, 30, 31의 계산식에 의해 자원 정보를 계산한다. 부하 분산 대상 서버 선정 모듈은 자원 정보 계산 모듈에서 계산한 자원 정보를 바탕으로 순위화하여 클라이언트의 RFID Tag 데이터 수집 및 가공 처리 사항을 수행할 ALE RFID 미들웨어를 선정한다.

7. 가중치 기반 부하 분선 처리 방법

1) 부하 분산 대상 서버 선정 방법

본 논문의 부하 분산 시스템에서 부하 분산 대상 서버를 선정하는 방법은 클라이언트의 RFID Tag 데이터 수집 및 가공 처리 사항이 부하 분산 시스템으로 요청이 되면 분산 미들웨어 환경에 구축된 각 ALE RFID 미들웨어의 하드웨어 부하 상황에 해당하는 자원 정보, 미들웨어 부하 상황에 해당하는 자원 정보, ALE 부하 상황에 해당하는 자원 정보를 수집한다. 이후 클라이언트가 요청한 특

정 ECSpec의 처리사항을 바탕으로 유전자 알고리즘을 이용하여 구한 각 자원 정보의 가중치 정보를 수집한 하드웨어, 미들웨어, ALE 자원 사항을 그림 29, 30, 31의 계산식을 바탕으로 자원 정보를 계산한다.

이렇게 계산된 자원 정보를 바탕으로 종합 분석 및 순위화 작업을 통하여 가장 자원 정보가 적은 서버를 클라이언트 요청 처리 서버로 선정하며, 1순위 서버가 2개 이상일 경우 Round Robin 방식을 적용하여 최종 부하 분산 대상 서버를 선정한다. 표 13은 하드웨어, 표 14는 미들웨어, 표 15는 ALE 자원 정보 및 유전자 알고리즘을 이용하여 구한 ecspec_lb1의 가중치 정보의 예이다.

표 13. 하드웨어 자원 정보 및 ecspec_lb1의 가중치 예

자원 정보명	자원 정보	가중치 정보
ProcessCount	46	0.576131687
TotalMemory	353136	0.623681913
JavaUseMomory	85632	0.5663124
TCPUUse	99	0.580249457
CPUUse	2.542306	1.263723303
HeapMemoryUsage	10163	0.54119347
NonHeapMemoryUsage	19513	0.503746813
AvailableProcessors	4	0.727272727
TotalPhysicalMemorySize	2620984	0.588481918
FreePhysicalMemorySize	1620512	0.699681562
TotalSwapSpaceSize	4194303	0.500287877
FreeSwapSpaceSize	3770800	0.566284844
CommittedVirtualMemorySize	43344	0.512480294
ConnectionSpeed	1	0.571428571

표 14. 미들웨어 자원 정보 및 ecspec_lb1의 가중치 예

자원 정보명	자원 정보	가중치 정보
DBConnectionCount	99	0.534005038
ThreadCount	30	0.512820512
PeakThreadCount	31	0.504065041
DaemonThreadCount	13	0.530612245
TotalStartedThreadCount	42	0.51497006
LoadedClassCount	3329	0.501353383
UnloadedClassCount	4	0.5
TotalLoadedClassCount	3333	0.501351757

표 15. ALE 자원 정보 및 ecspec_lb1의 가중치 예

자원 정보명	자원 정보	가중치 정보
LogicalReaderCount	4	0.5
ActivedLogicalReaderCount	1	0.5
ECSpecCount	62	0.5
ActivedEventCycleCount	1	0.5
EventCycleDataCount	282	0.50222618
LogicalQueueCount	4	0.5
EventQueueCount	1	0.5
LogicalQueueSize	38	0.583850932
EventQueueSize	94	0.637974683

가중치 정보는 유전자 알고리즘 연산을 이용하여 ecspec_lb1 이라는 이름을 갖는 ECSpec을 4대의 ALE RFID 미들웨어서 실험하여, CPU 사용률 등의 자원 정보 4개를 구하고 이를 정규화하여 유전자 알고리즘의 초기 유전자로 활용한다. 이 정규화 값을 이용하여 유전자 알고리즘은 부하가 가장 최적으로 이뤄졌을 때의 자원 정규화 평균값과 분포도 평균값의 유사도 값을 적합도 함수를 이용하여 찾아내며, 이 값이 적합도 값이다. 이 값을 ecspec_lb1 ECSpec이 처리 될 시 CPU 자원 정보에 가중치 정보로 활용하며, 표 15의 LogicalReaderCount의 가중치 값이 최소값 0.5 인 이유는 ecspec_lb1 ECSpec을 4대의 ALE RFID 미들웨어에서 실행 했을 때 자원 정보가 모두 4로 동일하기 때문이다.

가중치 정보가 0.5인 자원 정보는 4대의 미들웨어에서 ECSpec를 처리 했을 때 자원 정보가 서로 같다는 것을 의미하며, 이 가중치 정보가 표 13의 CPUUse 처럼 수치가 클수록 자원 정보가 서로 차이가 많이 남을 의미한다.

즉, 서로 사양이 다른 4대의 ALE RFID 미들웨어에서 ECSpec를 처리하여 자원 정보의 차이가 클수록 부하 분산 처리 시 중요한 자원 정보로 활용해야 하기 때문에 가중치 정보를 크게 둔다.

2) 부하 분산 처리 흐름도

그림 38은 본 논문에서 제안하는 부하 분산 시스템의 처리 흐름도이다. 시스템이 시작되면 미들웨어 분산 미들웨어 환경에 구축된 각 ALE RFID 미들웨어와 부하 분산 시스템의 로드밸런서의 ALE Interface를 실행하여 클라이언트 응용이 호출할 수 있도록 대기상태가 된다.

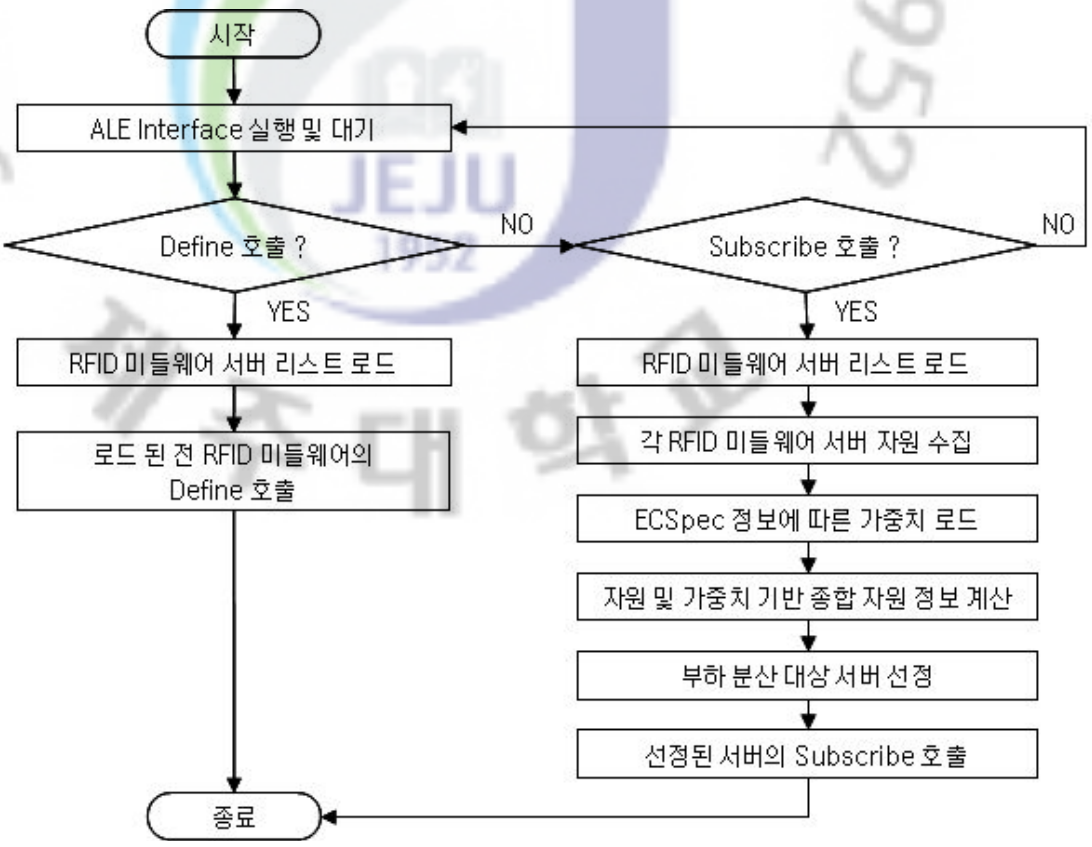

그림 38. 부하 분산 처리 흐름도

이후 클라이언트 응용은 로드밸런서의 define API[ex : define(LR1, LRSpec)]을 호출하여 LRSpec을 로드밸런서에 등록시키고, 미들웨어 분산 환경에 구축된 모든 ALE RFID 미들웨어의 define API를 호출하여 LRSpec에 명시된 논리 리더 정보를 등록한다.

다음으로 클라이언트 응용은 로드밸런서의 define API[ex : define(ecspec_lb1,

ECSpec)]을 호출하여 ECSpec을 로드밸런서에 등록시키고, 모든 ALE RFID 미들웨어의 define API를 호출하여 클라이언트의 수집 및 가공 처리 사항이 명시된 ECSpec을 등록한다.

클라이언트의 RFID Tag 데이터 수집 및 가공 처리 요청을 로드밸랜서의 subscribe API[ex : subscribe(ecspec_lb1, url)]을 호출하면 로드밸런서는 미들웨어 분산 환경에 구축된 모든 ALE RFID 미들웨어의 부하 상황에 해당하는 자원 정보를 수집한다.

로드밸런서는 유전자 알고리즘을 기반으로 구한 ecspec_lb1 이름을 갖는 ECSpec의 처리사항에 대한 가중치 정보를 불러오며, 모든 ALE RFID 미들웨어서 수집한 자원 정보와 ecspec_lb1의 각 자원 가중치 정보를 바탕으로 그림 29, 30, 31의 계산식을 이용하여 종합 자원 정보를 계산한다. 이렇게 계산된 자원 정보를 바탕으로 순위화 작업을 통하여 가장 자원 정보가 적은 서버를 클라이언트 요청 처리 서버로 선정하고, 선정된 ALE RFID 미들웨어 서버의 subscribe API[ex : subscribe(ecspec_lb1, url)]를 호출한다.

선정된 ALE RFID 미들웨어 서버는 subscribe API[ex : subscribe(ecspec_lb1, url)] 사항을 처리하여 클라이언트가 명시한 url 주소로 ecspec_lb1이라는 이름의 ECSpec에 명시된 논리 리더로 부터 RFID Tag 데이터 수집하고, ECSpec에 명시된 필터링 및 그룹핑등의 가공처리를 하여 ECReports 형태로 보낸다.

IV. 시스템 구현 및 성능평가

1. 구현 및 실험 환경

본 논문에서 제안한 대량 RFID 데이터 처리를 위한 부하 분산 시스템은 다음 환경에서 구현 및 실험 하였다. 표 16과 같이 한 대의 로드밸런서와 4대의 ALE RFID 미들웨어를 바탕으로 미들웨어 분산 환경을 구축하였다.

표 16. 시스템 구현 및 실험 환경

구 분	하드웨어	소프트웨어	ID
Load Balancer System	AMD Athlon(tm) 32 X1 Single Core 2.00GHz RAM 2048MB	Microsoft Windows XP Professional Sun Java SDK 1.5.0_07 Eclipse SDK 3.1	LB
ALE RFID Middleware System	Intel Core(tm) 64 X2 Dual Core 2.00GHz RAM 2048MB	Microsoft Windows 7 Ultimate Sun Java SDK 1.5.0_07 Eclipse SDK 3.1	KGY
ALE RFID Middleware System	AMD Phenom(tm) 64 X4 Quard Core 3.00GHz RAM 2048MB	Microsoft Windows 7 Professional Sun Java SDK 1.5.0_07 Eclipse SDK Helios Service Release 1	KMJ
ALE RFID Middleware System	AMD Phenom(tm) 64 X4 Quard Core 3.00GHz RAM 4096MB	Microsoft Windows 7 Professional Sun Java SDK 1.5.0_07 Eclipse SDK Helios Service Release 1	YJG
ALE RFID Middleware System	AMD Phenom(tm) 32 X4 Quard Core 3.00GHz RAM 3072MB	Microsoft Windows XP Professional Sun Java SDK 1.5.0_07 Eclipse SDK 3.1	NYS

본 연구에서 제안하고자 하는 시스템 개발의 중점은 미들웨어 분산 환경에 구축된 ALE RFID 미들웨어들의 자원 정보를 효율적으로 수집하고, 유전자 알고리즘을 이용하여 구한 가중치 정보를 적용하여 부하 상황에 해당하는 자원 정보가 가장 적은 서버로 적절하게 클라이언트의 처리 요청을 분산하는 것이다.

2. 시스템 모듈 구성

1) 시스템 패키지 다이어그램

본 논문에서는 대량 RFID 데이터 처리를 위한 부하 분산 방법을 제공하기 위하여 다음과 같은 시스템을 구현하였다. ALE RFID 미들웨어 분산 환경에서 Java를 이용하여 다양한 미들웨어의 부하 상황에 맞게 최적으로 클라이언트의 처리사항을 분산 할 수 있도록 시스템을 설계 및 구현하였으며, 클라이언트 요청의 부하 분산을 위한 공용 인터페이스와 부하 분산 시스템의 데이터베이스 관리, SOAP 통신을 이용한 데이터 송/수신 관리, ALE RFID 미들웨어의 하드웨어, 미들웨어, ALE 자원 정보관리와 ECSpec 조건에 따른 가중치 정보 관리등의 기능을 그림 37의 부하 분산시스템 아키텍처를 기반으로 구현하였다

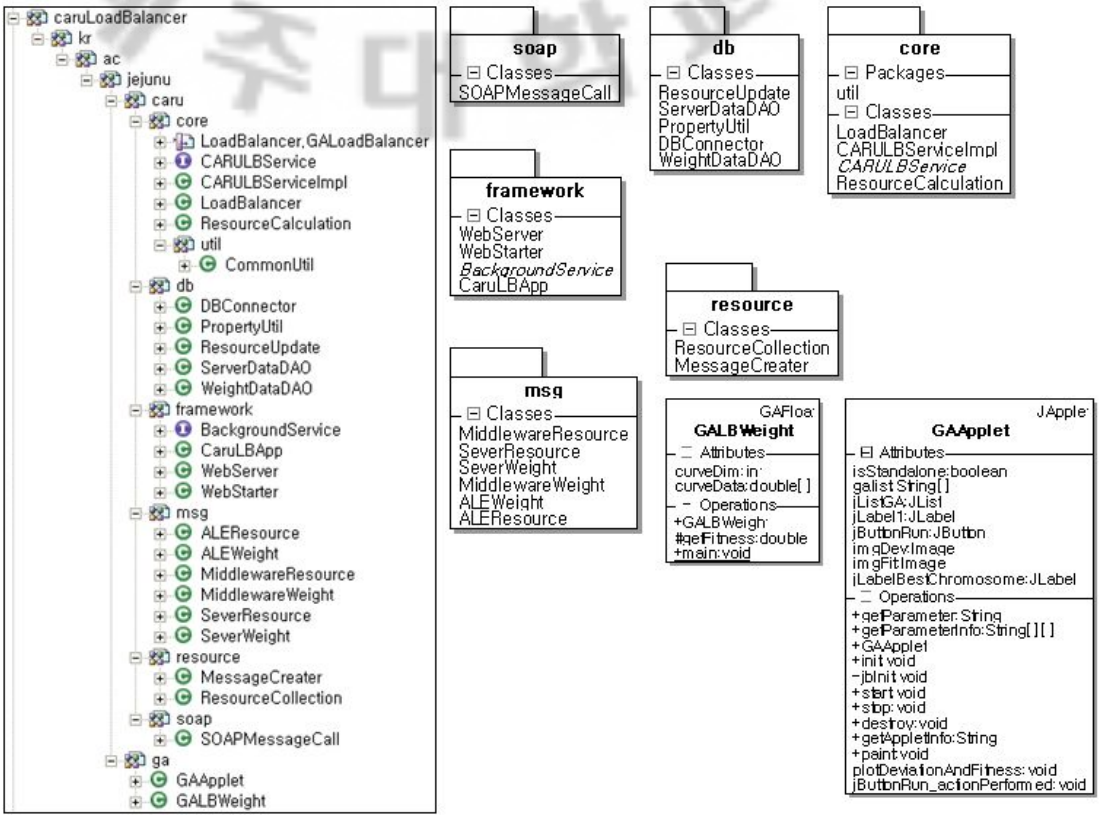

그림 39. 부하 분산 시스템 및 패키지 다이어그램

(1) ga 패키지

ga 패키지는 그림 40과 같이 GALB Weight 클래스, GAApplet 클래스로 구성된다. GALB Weight 클래스는 실수형 유전자 인코딩 방법을 이용하여 그림 35와 같은 초기 유전자를 정의하여 모의 집단을 생성하고, 스레드 형태로 실행되어 파라미터로 정의된 세대 수만큼 반복처리 되면서 각 세대마다 랜덤으로 생성된 부하 상황의 적합도 값을 계산하여, 가장 최적으로 부하 분산이 이루어 졌을 때의 유전자를 찾는다. 적합도 평가 처리는 세대마다 랜덤으로 실수형 값을 생성하고, 이렇게 생성된 유전자 값을 유전자 값의 총합 값을 이용하여 정규화 한다.

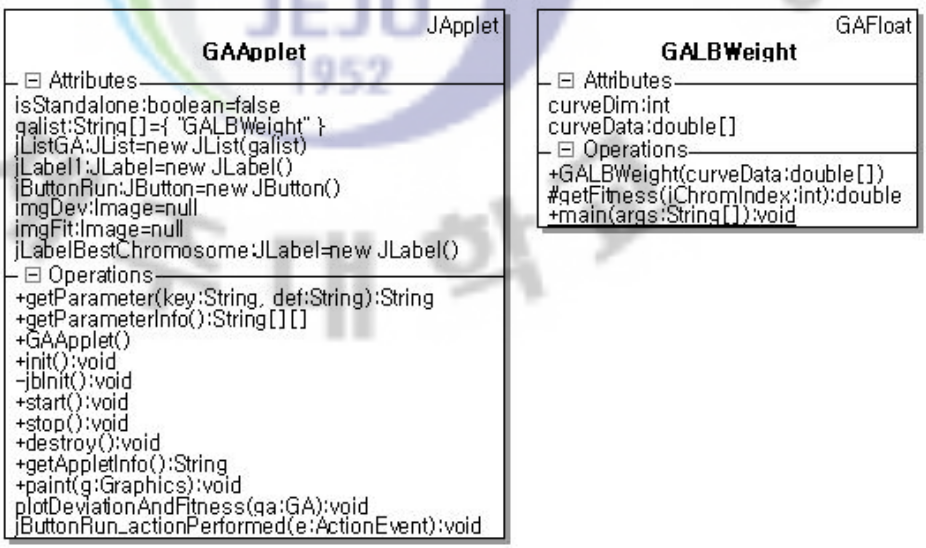

그림 40. ga 패키지의 클래스

이렇게 정규화된 값과 ECSpec을 처리하여 정규화된 자원 정보를 더하고, 더한 유전자 값의 분포도 값을 구하며, 이렇게 구한 분포도 값의 총합을 바탕으로 분포도의 평균값을 구해 최적으로 부하 상황이 잘 된 사항의 유전자 값을 찾아 적합도 값을 구한다. 이렇게 구한 적합도 값은 클라이언트의 요청 사항을 분산하기 위한 가중치 정보로 활용된다. GAApplet 클래스는 GALB Weight 클래스를 바탕으로 유전자 알고리즘의 처리 과정에서 최우성 유전자를 찾아가는 과정을 JApplet을 이용하여 평균 유전자 값과 평균 적합도 값을 각 세대마다 그래프 형

태로 표현한다.

(2) core 패키지

core 패키지는 그림 41과 같이 CARULBService 인터페이스 클래스, CARULBServiceImpl 인터페이스 구현 클래스, LoadBalancer 클래스, ResourceCalculation 클래스로 구성된다. CARULBService 인터페이스 클래스는 클라이언트의 RFID Tag 데이터 수집 및 가공 처리 요청을 수행하기 위한 subscribe 인터페이스를 관리하며, ARULBServiceImpl 인터페이스 구현 클래스는 이러한 subscribe 인터페이스가 클라이언트에 의해 호출될 시에 처리되는 부하 분산 과정과 선정된 ALE RFID 미들웨어로 클라이언트의 처리 사항을 전달하는 역할을 수행한다.

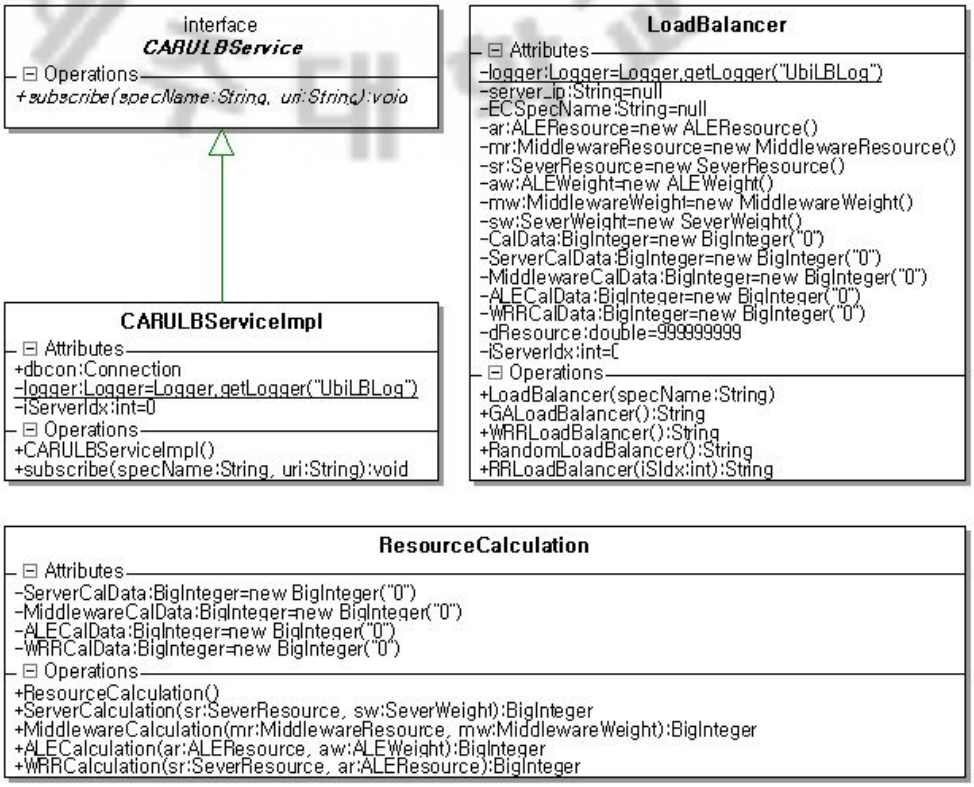

그림 41. core 패키지의 클래스

LoadBalancer 클래스는 부하 분산 시스템에서 클라이언트의 처리 사항을 분산하는 주된 처리를 수행하며, 그림 38의 부하 분산 처리에서 subscribe가 호출될 시의 부하 분산 과정을 처리한다. ResourceCalculation 클래스는 LoadBalancer 클래스에서 수집한 미들웨어 분산 환경에 구축된 ALE RFID 미들웨어의 자원정보와 ECSpec을 바탕으로 유전자 알고리즘을 이용하여 구한 가중치 정보를 그림 29, 30, 31, 32의 계산식을 바탕으로 미들웨어의 종합 자원을 계산한다.

(3) db 패키지

db 패키지는 그림 42와 같이 DBConnector 클래스, PropertyUtil 클래스, ResourceUpdate 클래스, ServerDataDAO 클래스, WeightDataDAO 클래스로 구성된다. DBConnector 클래스는 부하 분산 시스템의 데이터베이스에 저장된 자원정보 및 가중치 정보등을 불러오기 위해 데이터베이스에 접속하기 위한 커넥션 객체를 관리하고, PropertyUtil 클래스는 데이터베이스에 접속하기 위한 환경 설정 파일을 시스템에서 불러오기 위한 처리를 수행한다.

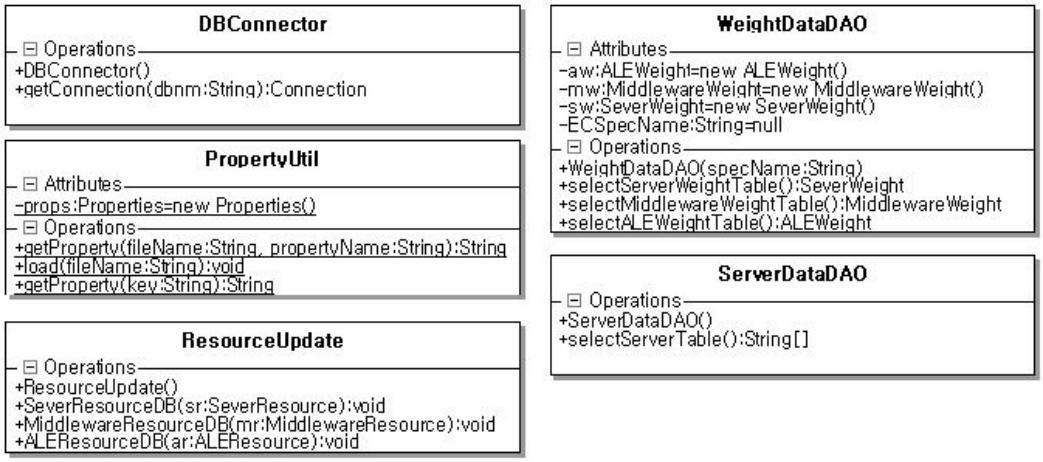

그림 42. db 패키지의 클래스

ResourceUpdate 클래스는 데이터베이스 커넥션 객체를 기반으로 분산 미들웨어 환경에 구축된 각 ALE RFID 미들웨어의 자원 정보를 부하 분산 시스템의

하드웨어, 미들웨어, ALE 자원 테이블에 저장하며, ServerDataDAO 클래스는 데이터베이스 커넥션 객체를 기반으로 분산 미들웨어 환경에 구축된 ALE RFID 미들웨어의 서버 정보를 불러와 관리하고, WeightDataDAO 클래스는 데이터베이스에 저장된 ecspec_lb1부터 ecspec_lb50까지 50개 유형의 ECSpec 가중치 정보를 불러와 관리한다.

(4) framework 패키지

framework 패키지는 그림 43과 같이 CaruLBApp 메인 클래스, WebStarter 클래스, WebServer 클래스, BackgroundService 클래스로 구성된다. CaruLBApp 메인 클래스는 부하 분산 시스템의 메일 클래스로 SOAP 통신을 이용하여 클라이언트의 처리 요청을 수행하기 위해 Jetty WebServer 실행시키고, CARULBService 인터페이스를 실행하여 클라이언트의 요청 사항을 처리하기 위해 대기한다.

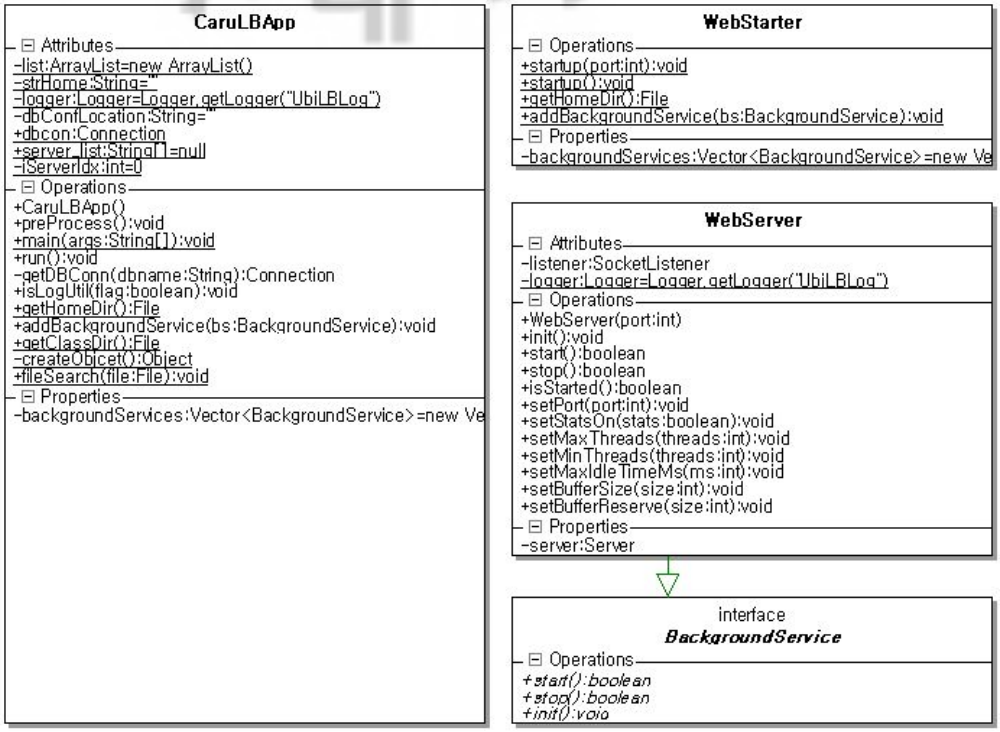

그림 43. framework 패키지의 클래스

WebStarter 클래스는 Jetty 웹서버를 실행하는 역할을 수행하며, WebServer 클래스는 Jetty 웹서버의 각 설정사항과 서버와 리스너 객체 사항을 관리하고, BackgroundService 클래스는 WebServer 클래스의 Jetty 웹서버를 백그라운드 서비스로 실행하기 위한 처리를 수행한다.

(5) msg 패키지

msg 패키지는 그림 44와 같이 ALEResoure 클래스, ALEWeight 클래스, MiddlewareResource 클래스, MiddlewareWeight 클래스, ServerResource 클래스, ServerWeight 클래스로 구성된다. ALEResoure 클래스는 표 12의 ALE 자원 정보들을 자료구조 형태로 관리하며, ALEWeight 클래스는 50개 유형의 ECSpec 중 클라이언트가 요청한 ecspec_lb1 이름을 갖는 ECSpec의 ALE 가중치 정보들을 자료구조 형태로 관리한다.

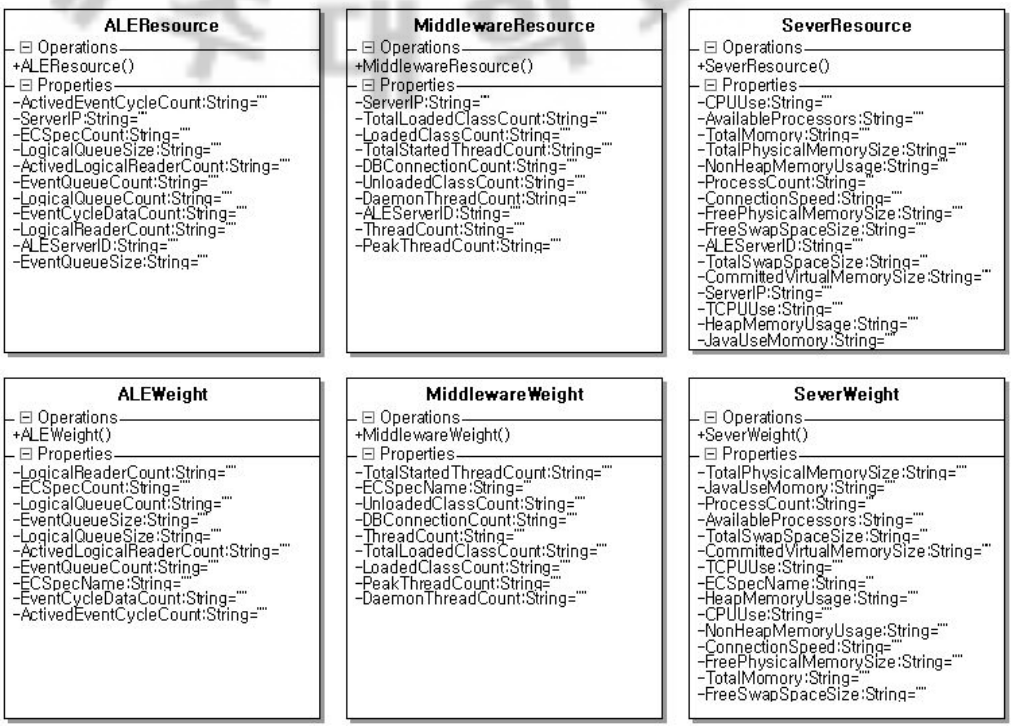

그림 44. msg 패키지의 클래스

MiddlewareResource 클래스는 표 11의 미들웨어 자원 정보들을 자료구조 형태로 관리하며, MiddlewareWeight 클래스는 50개 유형의 ECSpec 중 클라이언트가 요청한 ecspec_lb1 이름을 갖는 ECSpec의 Middleware 가중치 정보들을 자료구조 형태로 관리한다. ServerResource 클래스는 표 10의 하드웨어 자원 정보들을 자료구조 형태로 관리하며, ServerWeight 클래스는 50개 유형의 ECSpec 중 클라이언트가 요청한 ecspec_lb1 이름을 갖는 ECSpec의 Hardware 가중치 정보들을 자료구조 형태로 관리한다.

(6) resource 패키지

resource 패키지는 그림 45와 같이 ResourceCollection 클래스, MessageCreater 클래스로 구성된다. ResourceCollection 클래스는 분산 미들웨어 환경에 구축된 ALE RFID 미들웨어의 getLoadInfo()라는 SOAP 인터페이스를 호출하여 각 ALE RFID 미들웨어에 저장된 하드웨어, 미들웨어, ALE 자원 정보를 XML 형태로 제공 받으며, 제공 받은 하드웨어, 미들웨어, ALE 자원 정보를 각각 나눠 관리한다.

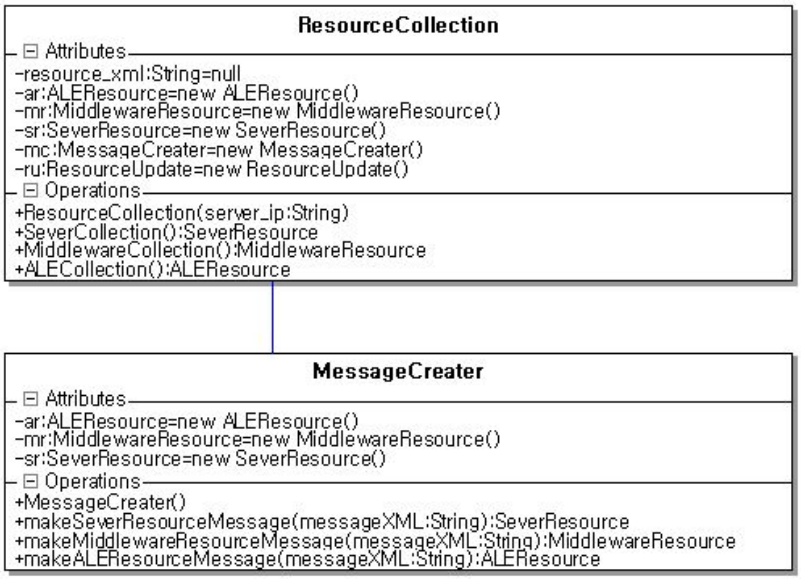

그림 45. resource 패키지의 클래스

MessageCreater 클래스는 ResourceCollection 클래스에서 수집한 XML 정보의 프로토콜을 기반으로 XML Tag 형태를 분석하여 실제 XML 형태로 제공된 하드웨어, 미들웨어, ALE 자원 정보를 msg 패키지의 Hardware, Middleware, ALE Resource 클래스의 자료구조에 저장한다.

(7) soap 패키지

soap 패키지는 그림 46의 SOAPMessageCall 클래스로 구성되며, SOAP 통신을 이용한 인터페이스를 호출 처리를 수행한다. ResourceCollection 클래스에서 호출하는 ALE RFID 미들웨어의 getLoadInfo()와 CARULBServiceImpl 클래스에서 호출하는 subscribe(String specName, String uri) 메소드를 SOAP 프로토콜을 이용하여 호출한다. SOAP 프로토콜을 사용하기 위해 제공되는 WSDL(Web Services Description Language)의 메소드 정보와 파라미터, URL, URI 정보 등을 기술하여 분산 미들웨어 환경에 구축된 ALE RFID 미들웨어의 SOAP 인터페이스를 호출한다.

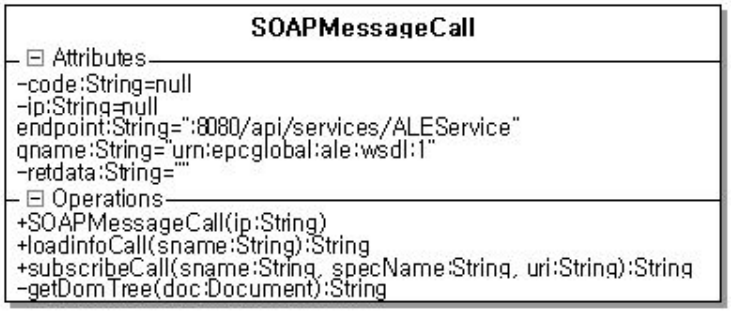

그림 46. soap 패키지의 클래스

2) 시퀀스 다이어그램

그림 47은 본 논문에서는 대량 RFID 데이터 처리를 위한 부하 분산 방법을 제공하기 위해 구현한 시스템의 시퀀스 다이어그램이다. 그림 38의 부하 분산 처리 흐름도를 기반으로 구현한 부하 분산 시스템의 각 모듈의 실행 흐름을 볼 수 있다.

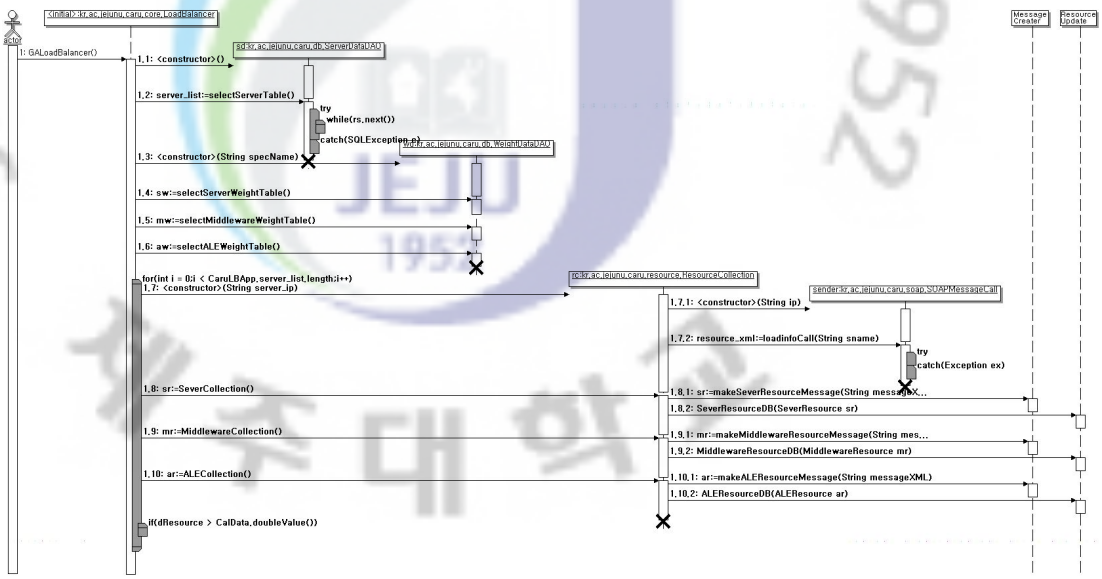

그림 47. 부하 분산 시스템의 시퀀스 다이어그램

클라이언트의 처리 요청을 받은 CARULBServiceImpl의 subscribe 메소드는 LoadBalancer 클래스의 GALoadBalancer() 메소드를 호출하여 부하 분산 처리를 수행한다. LoadBalancer는 ServerDataDAO 클래스의 selectServerTable() 메소드를 호출하여 분산 미들웨어 환경에 구축된 ALE RFID 미들웨어의 서버 리스트 정보를 반환받는다. 이후 WeightDataDAO 클래스의 selectServerWeightTable(), selectMiddlwareWeightTable(), selectALEWeightTable() 메소드를 호출하여 부하 분산 시스템의 데이터베이스에 저장된 가중치 정보를 로드하여 msg 패키지의 자료구조에 저장한다.

그리고, LoadBalancer는 ServerDataDAO 클래스의 selectServerTable() 메소드를 호출하여 반환된 서버 리스트의 수만큼 ResourceCollection 클래스에서 SOAPMessageCall 클래스의 loadinfoCall(String sname)을 호출하여 분산 미들웨어 환경에 구축한 각 ALE RFID 미들웨어의 자원 정보를 요청하여 그 결과로 XML 형태의 하드웨어, 미들웨어, ALE 자원 정보를 수집한다.

또한, ResourceCollection 클래스의 ServerCollection(), MiddlewareCollection(), ALECollection() 메소드를 호출하여 각각 MessageCreater 클래스의 makeServerResourceManager, makeMiddlewareResource, makeALEResource 메소드를 호출하고, ALE 미들웨어로부터 수집한 XML 형태의 하드웨어, 미들웨어, ALE 자원 정보를 분석하여 msg 패키지의 자료구조에 저장한다.

이렇게 저장된 자원 자료구조 정보는 각각 ResourceUpdate 클래스의 ServerResourceDB, MiddlewareResourceDB, ALEResourceDB 메소드를 호출하여 부하 분산 시스템의 데이터베이스에 저장한다.

이렇게 저장된 자원 정보와 msg 패키지의 자료구조에 저장된 가중치 정보를 종합 연산하여 분산 미들웨어 환경에 구축된 ALE RFID 미들웨어들 중 최적의 처리 서버를 선정한다.

3. GA 가중치 생성 시스템

1) GA 파라미터 요소

본 논문에서는 제안한 부하 분산 처리를 수행할 때 필요한 가중치 정보를 유전자 알고리즘을 이용하여 구하기 위해서 표 17과 같이 유전자 알고리즘의 파라미터들을 설정하였다. 그림 35의 유전자 알고리즘을 위한 문제의 표현에서 정의한 4개의 염색체를 갖는 유전자를 정의하며, 그림 36의 초기 해의 집단인 염색체를 갖고 있는 유전자의 수는 100개로 설정하였다.

표 17. GA 파라미터 요소

GA 파라미터 명	GA 파라미터 값
염색체 수	4
염색체를 갖고 있는 유전자의 수	100
교배 확률	0.7
무작위 선택 확률	90
최대 진화 세대 수	200
예비 실행 수	10
최대 예비 세대 수	20
돌연변이 확률	0.5
교배 방법	Two Point
정밀도	2
실수형만 사용 여부	True
통계 계산 여부	True

분산 환경에 표 16과 같이 4대의 ALE RFID 미들웨어를 구축하였기 때문에 염색체의 수는 4개를 정의하였으며, 초기 해 집단인 염색체를 갖고 있는 유전자의 수는 100개로 정의하였다.

교배확률은 교배가 일어나는 횟수를 조절하는 매개변수로 확률이 낮게 설정되면 다음 세대에서 새로운 개체 발생이 적게 되어 탐색이 침체되고, 반대로 높게

설정되면 탐색 공간을 빨리 탐색하는 특징을 갖게 된다.

일점교배는 부모염색체 내에 분산되어 있는 일정 유전자 정보는 결합하지 못하는 단점을 지니므로 이를 보완하기 위해 이점교배(two point) 방법을 사용하며, 다점교배는 일점 교배나 이점교배처럼 부모염색체의 일정 유전자 정보를 결합하지 못하는 단점은 없지만 쉽게 부모의 유전자 정보를 잃는 단점이 있다. 이러한 이유로, 교배 방법은 그림 17(b)의 이점 교배 방법을 사용하였으며, 교배 확률은 다양한 부하 사항을 표현하기 위해 0.7 과 같이 교배 연산을 많이 수행하였다.

돌연변이 연산은 최적해(suboptimal)로 수렴하는 요인을 제거하여 전역적인 해의 탐색을 가능하게 해준다. 돌연변이 확률은 유전자의 돌연변이 횟수를 조절하는 지표로 돌연변이 확률을 너무 낮게 설정하면 다른 탐색공간으로 이동하기 전에 지역해에 수렴할 수 있고, 반대로 너무 높게 설정하면 탐색공간을 불규칙적으로 이동하여 불안정한 탐색결과를 얻게 되어, 돌연변이 확률은 절반 수준인 0.5로 정의하였다.

무작위 선택 확률 또한 다양한 부하 사항을 표현하기 위해 90으로 수치를 높게 정의하였다. 그림 34의 유전자 알고리즘을 위한 문제의 표현에서 기술한 실수형태만을 사용하고, 최종적인 적합도 값의 통계 계산을 수행한다.

2) GA 기반 가중치 생성 실험

그림 48은 본 논문에서 구현한 GAApplet 클래스를 기반으로 유전자 알고리즘을 이용하여 표 16의 ALE RFID 미들웨어에서 특정 ECSpec 처리하여 최적으로 부하 상황이 이뤄졌을 때의 평가도 값을 구한 화면이다. 4개 서버의 자원 정규화 정보인 초기 유전자 값으로, 표 16의 시스템 실험환경과 같이 4대의 ALE RFID 미들웨어 ID를 갖는 서버 NYS(0.10811), YJG(0.27027), KMJ(0.21622), KGY(0.40541)로 정의하였다.

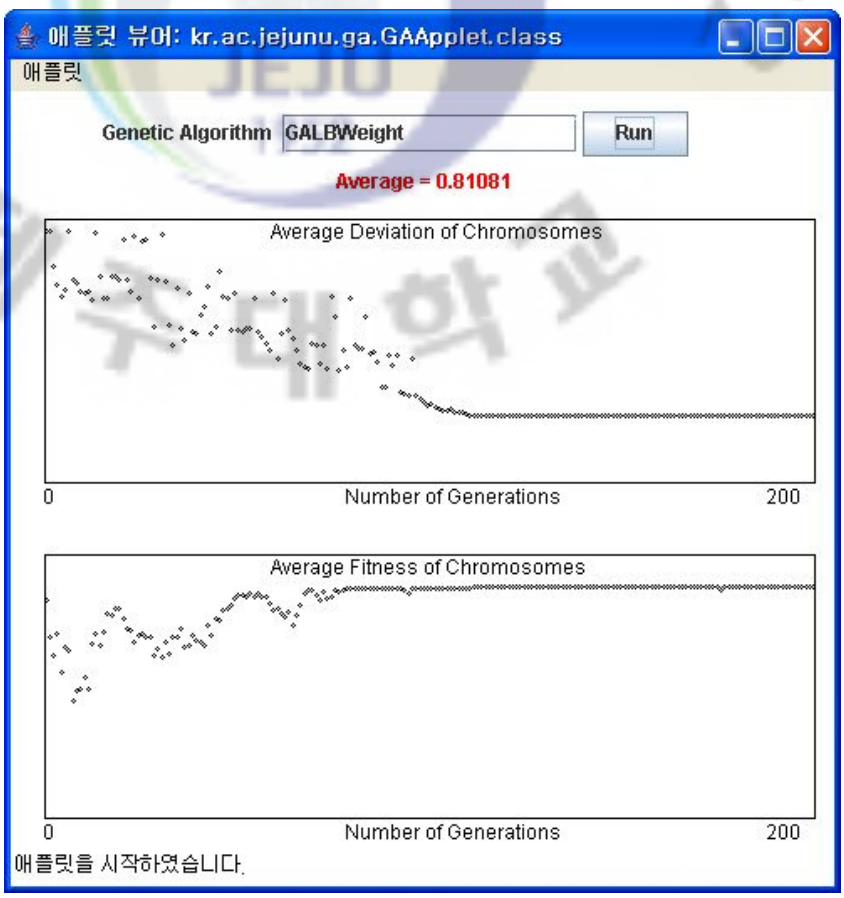

그림 48. 유전자 알고리즘 실험

초기 유전자 값의 자원 정보는 NYS(4), YJG(10), KMJ(8), KGY(15)이며, 그림 49는 본 논문에서 제안한 적합도 함수의 수식을 이용하여 초기 자원 정보 값과

자원 정보를 정규화 한 값과 정규화 한 자원 정보의 분포도 값이다.

	NYS	YJG	KMJ	KGY		
	4	10	8	15		37
자원 정규화	0.10811	0.27027	0.21622	0.40541	1	0.25
	1	1	1	1	4	
분포도, 디스턴스	0.14189	0.02027	0.03378	0.15541		0.08784

그림 49. 초기 자원 정보

초기 유전자 값의 자원 평균은 0.25이며 분포도 값의 평균은 0.08784이다. 본 논문에서 기술한 적합도 함수에서 설명했듯이, 자원 정규화 값의 평균이 1과 비슷해지고, 유전자 평가 값인 분포도 값의 평균이 0에 가까울수록 부하 분산이 잘 되었음을 의미한다.

그림 50은 가중치 기반 라운드 로빈(Weighted Round Robin) 방법을 이용하여 최대로 부하를 분산했을 시의 자원 정규화 값과 분포도 값이다. 자원 정규화 값의 평균은 0.86486이고, 유전자 평가 값인 분포도 값의 평균은 0.05405 이다. 그림 48과 같이 본 논문에서 제안한 유전자 알고리즘을 이용하면 평균값은 0.81081이며, 가중치 기반 라운드 로빈을 이용하면 그림 50과 같이 0.86486이다.

자원 정규화	0.972973	0.810811	0.864865	0.810811	3.45946	0.86486
	2	0	0	0	2	
분포도, 디스턴스	0.108108	0.054054	0	0.054054		0.05405

그림 50. 가중치 기반 라운드 로빈의 부하 분산

그림 51은 가중치 기반 라운드 로빈(Weighted Round Robin) 방법과 유전자 알고리즘을 바탕으로 구한 최적으로 부하를 분산했을 시의 자원 정규화의 평균값과 이 자원 정규화 값의 분포도 값을 비교한 수치를 보여준다.

0.1081	0.05405	0.21621	0.05405	0.86486
0.75676	0.81081	0.64865	0.81081	0.75676
분포도	평가 기준		평가 진행	
88%	93%		진행중	
0.875	-> 이 값이 제일 컷을 때의 가중치 선정			

0.05405	1.9E-10	0.16216	1.9E-10	0.81081
0.75676	0.81081	0.64865	0.81081	0.75676
분포도	평가 기준		평가 진행	
93%	93%		종료	
0.93333	-> 이 값이 제일 컷을 때의 가중치 선정			

그림 51. WRR과 GA 비교

그림 51의 0.86486은 가중치 기반 라운드 로빈 방법을 이용하여 구한 자원 정규화 값의 평균값이고, 0.81081은 유전자 알고리즘을 이용하여 구한 자원 정규화 값의 평균값이다. 그림 51과 같이 분포도 평균값과 초기 유전자 값인 NYS = 0.10811, YJG = 0.27027, KMJ = 0.21622, KGY = 0.40541와 나눈 나머지 값 0.1081, 0.05405, 0.21621, 0.05405 수치에 대한 분포 값을 구해 평균을 구하면 0.75676이 된다.

이 0.75676 수치와 구한 분포도 평균값을 나눠 유사도 값을 구하며, 유사도 값이 1이 되면 가장 완벽하게 부하 분산이 이뤄졌음을 의미하고, 수치가 1과 가장 가까웠을 최적으로 부하 분산이 이뤄졌음을 의미한다.

가중치 기반 라운드 로빈 방법의 유사도 값은 그림 51과 같이 0.875이고, 유전자 알고리즘을 이용한 방법의 유사도 값은 0.93333 이며 유전자 알고리즘을 이용한 방법이 최적의 부하 분산 상황 정보를 찾아낼 있었다. 위와 같이 구한 0.81081과 같은 자원 정규화의 평균값을 본 논문에서 분산 미들웨어 환경에 구축된 ALE RFID 미들웨어 중에 클라이언트의 요청을 처리할 최적의 미들웨어 선정에 필요한 가중치 정보로 활용한다.

4. 부하 분산 시스템

1) 시스템 구현

그림 52는 본 논문에서는 대량 RFID 데이터 처리를 위해 구현한 부하 분산 시스템을 실행한 화면이다. SOAP 인터페이스를 기반으로 클라이언트의 요청을 처리하기 위해 Jetty 기반의 WebServer를 실행한 CARU Load Balancer WEB SERVER - START! 로그와 CARULBService 인터페이스 클래스, CARULBServiceImpl 인터페이스 구현 클래스가 실행되어 클라이언트의 요청을 처리하기 위해 미들웨어의 실행 준비가 됐음을 의미하는 CARU Load Balancer Middleware Ver 1.0 ready 로그가 Log4j 기반의 Logger 클래스에서 표시되고 있는 미들웨어의 처리 내역을 보여준다.

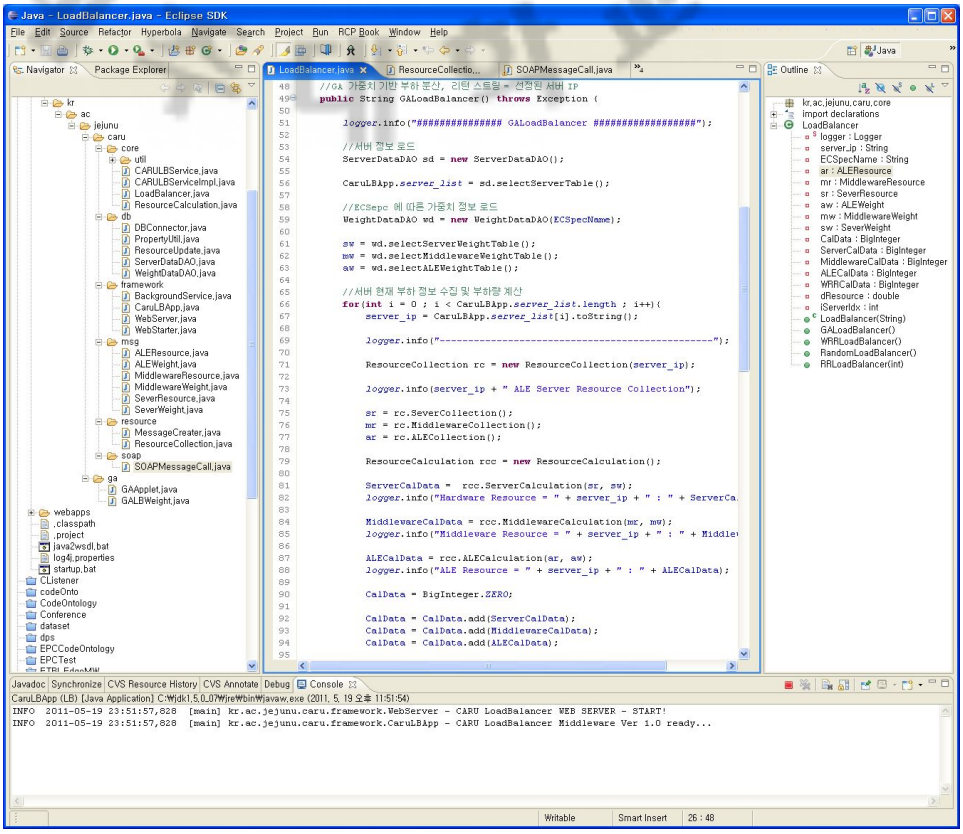

그림 52. 부하 분산 시스템의 실행

2) 부하 분산 시스템 실행

그림 53은 대량 RFID 데이터 처리를 위해 구현한 부하 분산 시스템으로 클라이언트의 RFID Tag 데이터 수집 및 가공 처리를 요청할 시에 본 논문에서 테스트한 화면이다. 테스트는 클라이언트의 요청 처리를 SOAP 통신을 이용하여 요청하기 때문에 SOAP 통신을 테스트하기에 적합한 soapUI 1.7.1 툴을 사용하였다. 그림 53은 부하 분산 시스템의 RFID Tag 데이터 수집 및 가공 처리 요청 메소드인 subscribe(specName, notiURI) 메소드를 soapUI 툴을 기반으로 호출하는 화면으로 subscribe 메소드의 파라미터 값으로 specName = ecspec_lb1, notiURI = 117.17.102.162:5500을 입력하여 클라이언트의 처리 사항을 부하 분산 시스템으로 요청한다. 그림 53은 ecspec_lb1 이라는 이름의 ECSpec을 부하 분산 시스템이 선정한 ALE RFID 미들웨어서 처리하여 117.17.102.162:5500로 ecspec_lb1 ECSpec을 처리한 결과인 그림 54와 같은 ECReprots를 반환한다.

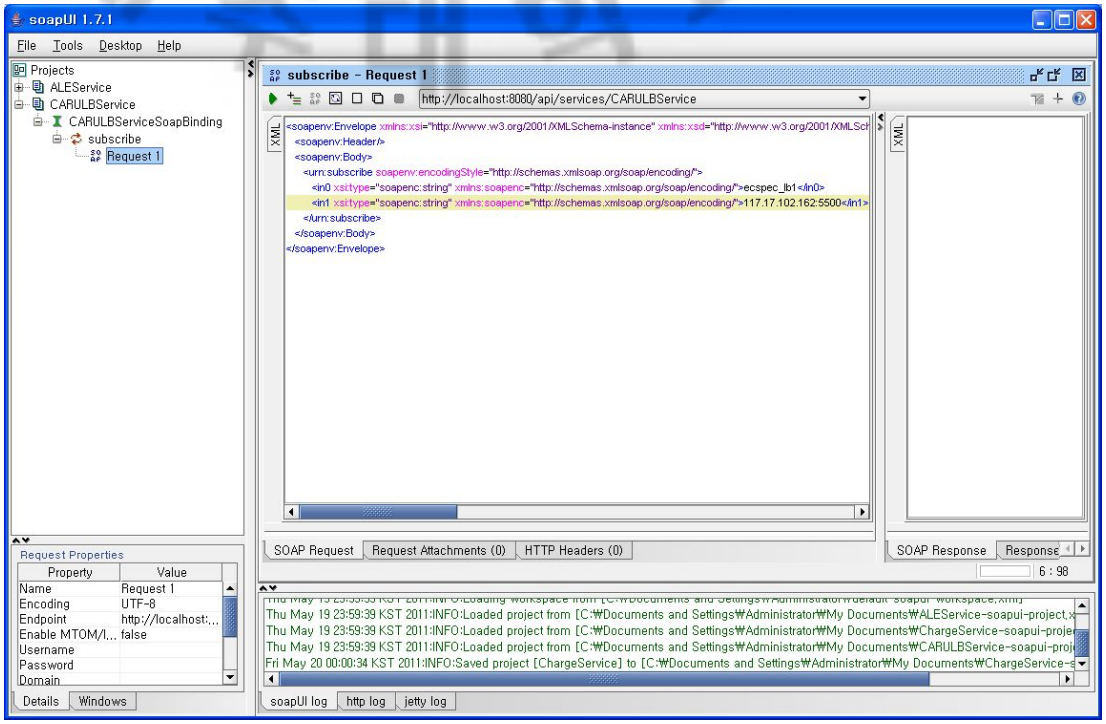

그림 53. 클라이언트 요청 처리

그림 54. 클라이언트 요청 처리 결과

3) 부하 분산 실험

(1) 라운드 로빈(RR) 방식 실험

그림 55는 본 논문에서 구현한 유전자 알고리즘을 이용하여 구한 가중치 기반 부하 분산 시스템과 비교하기 위해 구현한 라운드 로빈(Round Robin) 부하 분산 시스템을 구현하여 실행한 화면이다.

LoadBalancer의 RRLoadBalancer 메소드를 호출하여 부하 분산 처리를 수행하는 화면으로 미들웨어 분산 환경에 구축된 KGY(117.17.102.166), KMJ(117.17.102.212), NYS(117.17.102.104), YJG(117.17.102.89) 4대의 ALE RFID 미들웨어 리스트 정보를 순차적으로 선택하여 클라이언트의 요청을 분산한다.

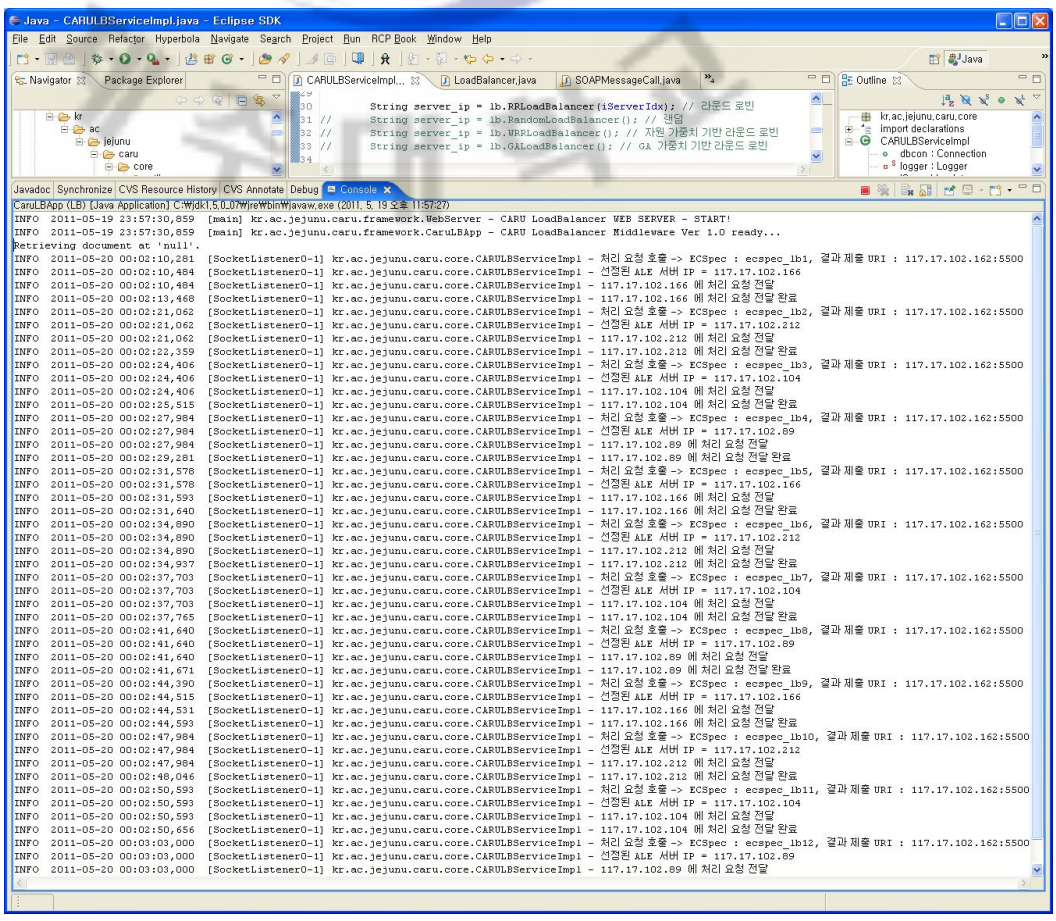

그림 55. RRLoadBalancer 실행

(2) 랜덤(Random) 방식 실험

그림 56은 본 논문에서 구현한 유전자 알고리즘을 이용하여 구한 가중치 기반 부하 분산 시스템과 비교하기 위해 구현한 무작위(Random) 부하 분산 시스템을 구현하여 실행한 화면이다.

LoadBalancer의 RandomLoadBalancer 메소드를 호출하여 부하 분산 처리를 수행하는 화면으로 미들웨어 분산 환경에 구축된 KGY(117.17.102.166), KMJ(117.17.102.212), NYS(117.17.102.104), YJG(117.17.102.89) 4대의 ALE RFID 미들웨어 서버 리스트 정보를 무작위로 선택하여 클라이언트의 요청을 분산한다.

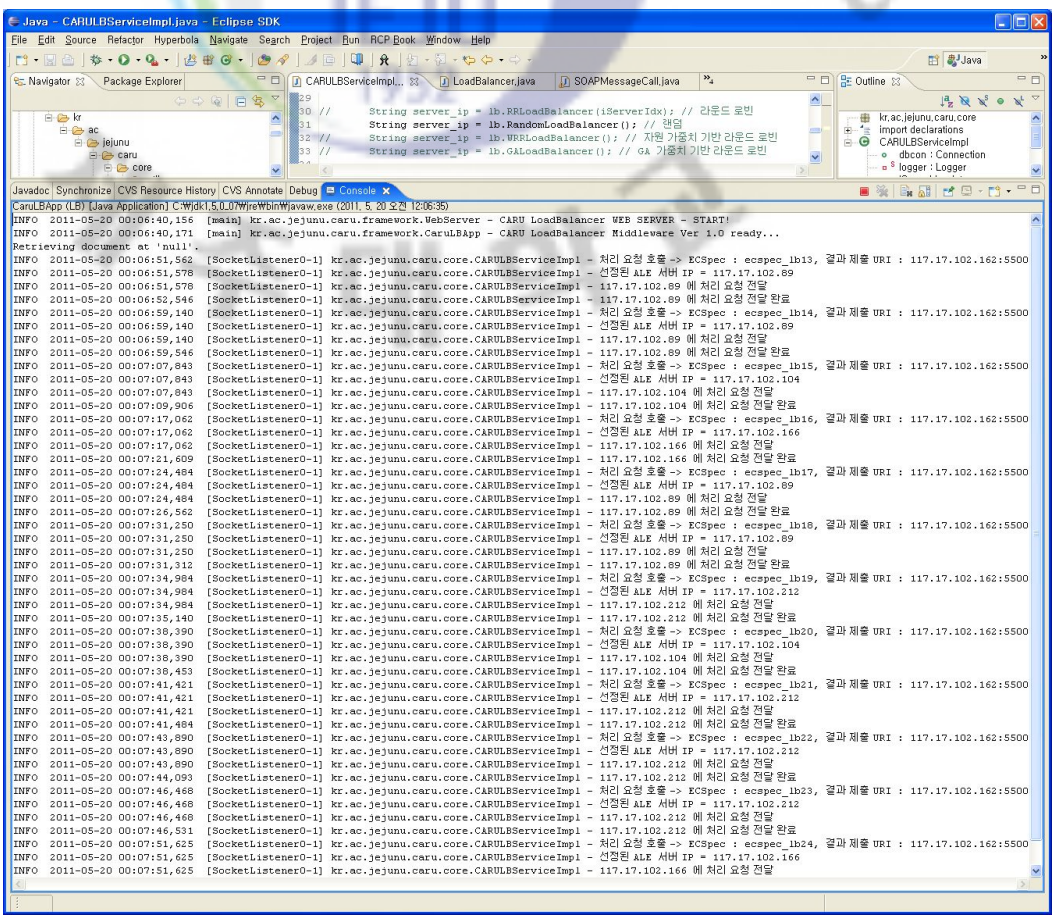

그림 56. RandomLoadBalancer 실행

(3) 가중치 기반 라운드 로빈(WRR) 방식 실험

그림 57은 본 논문에서 구현한 유전자 알고리즘을 이용하여 구한 가중치 기반 부하 분산 시스템과 비교하기 위해 구현한 일반 자원 정보 가중치 기반 라운드 로빈(Weighted Round Robin) 부하 분산 시스템을 구현하여 실행한 화면이다. LoadBalancer의 WRRLoadBalancer 메소드를 호출하여 부하 분산 처리를 수행하는 화면으로 미들웨어 분산 환경에 구축된 KGY(117.17.102.166), KMJ(117.17.102.212), NYS(117.17.102.104), YJG(117.17.102.89) 4대의 ALE RFID 미들웨어 CPU 사용률, ECSpec 수, RFID 리더 수, 수집된 RFID Tag 수등을 고려하여 부하 정보를 계산하고, 부하 정보가 적은 ALE RFID 미들웨어를 클라이언트의 처리 요청을 수행할 서버로 선정하여 클라이언트의 요청을 분산한다.

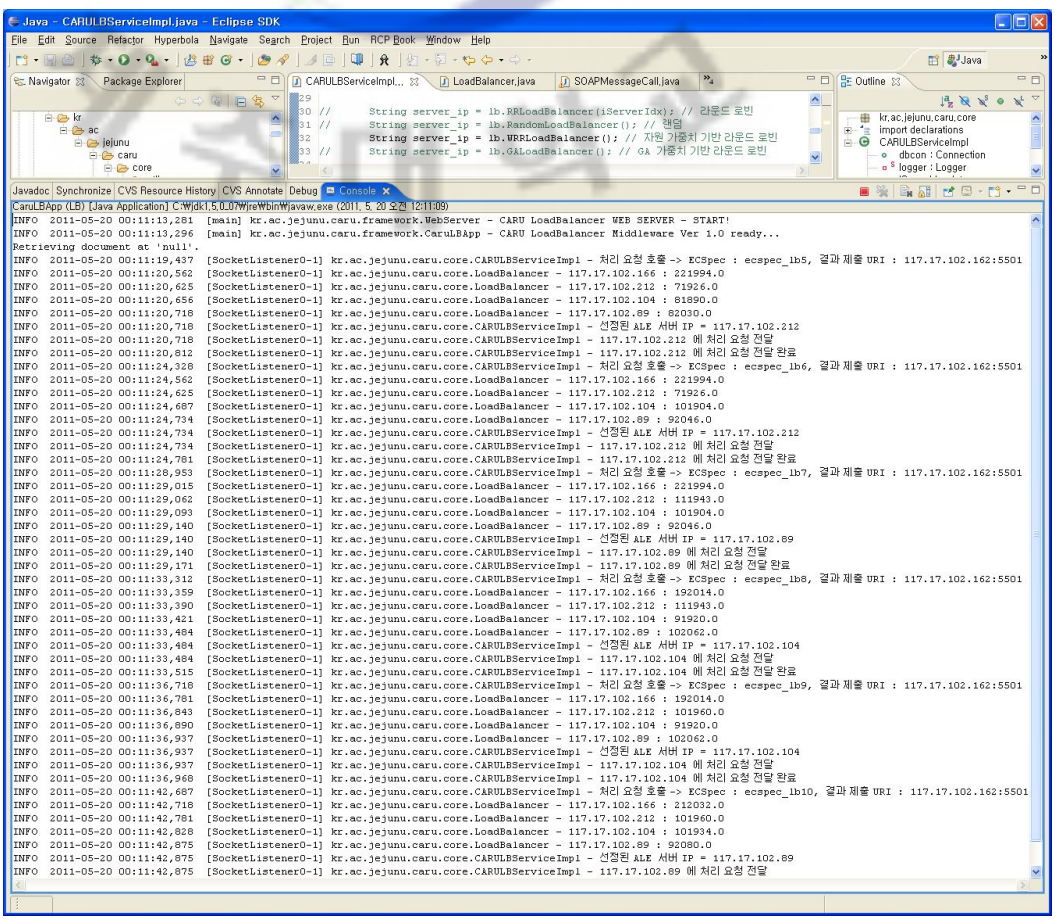

그림 57. WRRLoadBalancer 실행

(4) GA 가중치 기반 방식 실험

그림 58은 본 논문에서 구현한 유전자 알고리즘을 이용하여 구한 가중치 기반 부하 분산 시스템을 구현하여 실행한 화면이다. LoadBalancer의 GALoadBalancer 메소드를 호출하여 부하 분산 처리를 수행하는 화면으로 미들웨어 분산 환경에 구축된 KGY(117.17.102.166), KMJ(117.17.102.212), NYS(117.17.102.104), YJG(117.17.102.89) 4대의 ALE RFID 미들웨어 하드웨어, 미들웨어, ALE 자원 정보와 ECSpec에 따른 가중치 정보를 이용하여 부하 정보를 계산하고, 부하 정보가 가장 적은 ALE RFID 미들웨어를 클라이언트의 처리 요청을 수행할 서버로 선정하여 클라이언트의 요청을 분산한다.

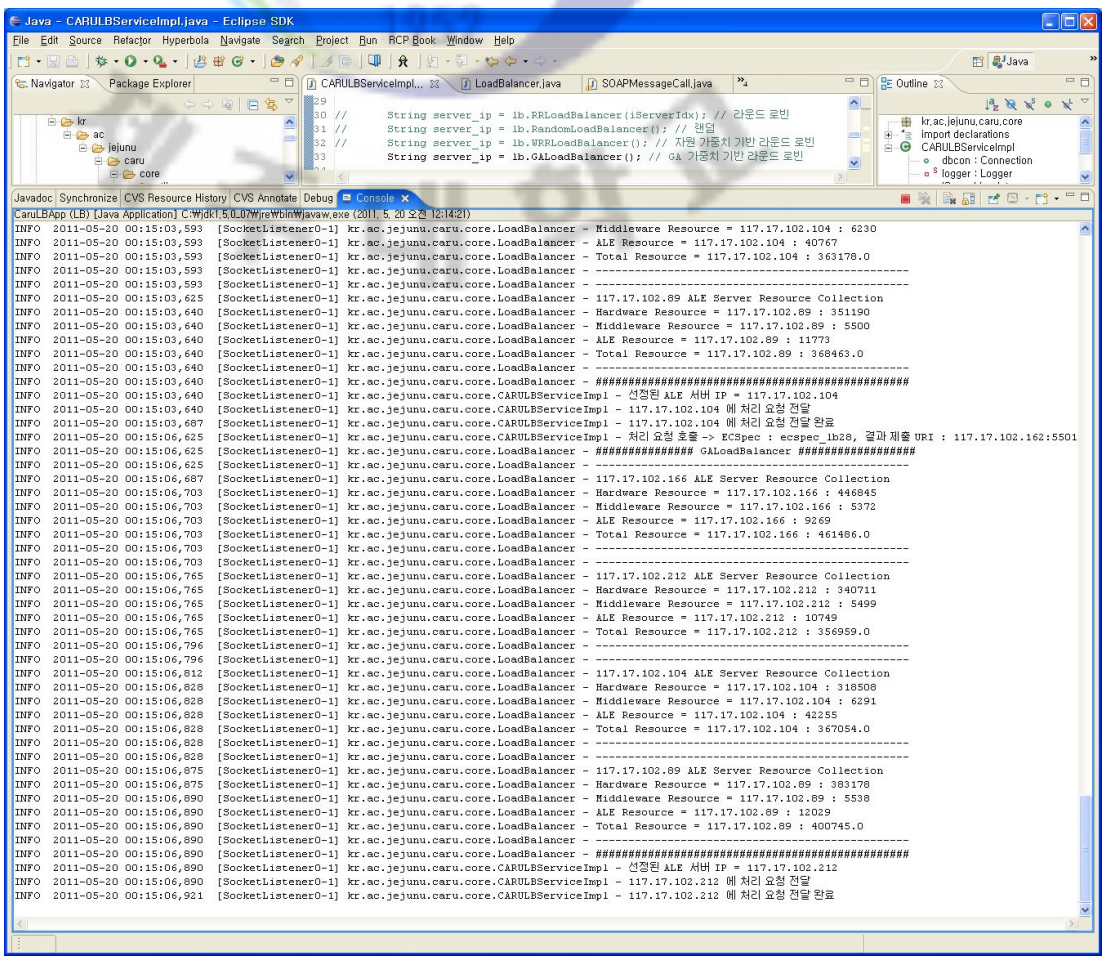

그림 58. GALoadBalancer 실행

그림 59는 유전자 알고리즘을 이용하여 구한 가중치 기반 부하 분산 시스템에서 분산 환경에 구축된 ALE RFID 미들웨어 가운데 NYS(117.17.102.104) 서버에서 SOAP 통신을 이용하여 수집한 XML 기반의 하드웨어, 미들웨어, ALE 자원 정보이다.

```
- <xml>
    <ALEServerID>NYS_ALE1</ALEServerID>
    <ServerIP>117.17.102.104</ServerIP>
  - <HardwareResource>
      <ProcessCount>45</ProcessCount>
      <TotalMomory>322212</TotalMomory>
      <JavaUseMomory>71120</JavaUseMomory>
      <TotalCPUUse>99.0</TotalCPUUse>
      <CPUUse>0.75234437</CPUUse>
      <HeapMemoryUsage>9576</HeapMemoryUsage>
      <NonHeapMemoryUsage>23891</NonHeapMemoryUsage>
      <AvailableProcessors>4</AvailableProcessors>
      <TotalPhysicalMemorySize>2097151</TotalPhysicalMemorySize>
      <FreePhysicalMemorySize>2097151</FreePhysicalMemorySize>
      <TotalSwapSpaceSize>4194303</TotalSwapSpaceSize>
      <FreeSwapSpaceSize>4194303</FreeSwapSpaceSize>
      <CommittedVirtualMemorySize>50540</CommittedVirtualMemorySize>
      <ConnectionSpeed>3</ConnectionSpeed>
    </HardwareResource>
  - <MiddlewareResource>
      <DBConnectionCount>790</DBConnectionCount>
      <ThreadCount>28</ThreadCount>
      <PeakThreadCount>30</PeakThreadCount>
      <DaemonThreadCount>12</DaemonThreadCount>
      <TotalStartedThreadCount>33</TotalStartedThreadCount>
      <LoadedClassCount>4259</LoadedClassCount>
      <UnloadedClassCount>9</UnloadedClassCount>
      <TotalLoadedClassCount>4268</TotalLoadedClassCount>
    </MiddlewareResource>
  - <ALEResource>
      <LogicalReaderCount>4</LogicalReaderCount>
      <ActivedLogicalReaderCount>0</ActivedLogicalReaderCount>
      <ECSpecCount>62</ECSpecCount>
      <ActivedEventCycleCount>0</ActivedEventCycleCount>
      <EventCycleDataCount>842</EventCycleDataCount>
      <LogicalQueueCount>4</LogicalQueueCount>
      <EventQueueCount>0</EventQueueCount>
      <LogicalQueueSize>2079</LogicalQueueSize>
      <EventQueueSize>0</EventQueueSize>
    </ALEResource>
  </xml>
```

그림 59. ALE RFID 미들웨어 자원 XML

5. 성능 평가

본 논문에서 제안하는 대량 RFID 데이터 처리를 위해 구현한 부하 분산 시스템의 성능 분석은 기존의 일반 부하 분산 방법인 라운드 로빈(Round Robin), 랜덤(Random), 기존 ALE RFID 미들웨어에서 제안한 CPU 사용률, ECSpec 수, RFID 리더 수, 수집된 RFID Tag 수 등을 고려한 가중치 기반 라운드 로빈 방식과 본 논문에서 제안한 ecspec_lb1 ~ ecspec_lb50의 50개 ECSpec 각 처리 유형별로 표 16의 환경에서 실험하여 얻은 초기 자원 정보의 정규화 값을 유전자 알고리즘을 사용하여 최적으로 부하 분산이 이뤄진 상황의 유전자 염색체 정보를 찾았을 때의 적합도 값인 자원 정규화 분포 값의 평균값을 가중치로 활용한 방법과, ecspec_lb1 ~ ecspec_lb50의 각 50개 가중치 값의 평균값을 일괄적으로 가중치 값으로 활용한 방법을 실험하여 성능을 평가한다.

실험 방법은 Round Robin, Random, Weighted Round Robin, GA Weighted, GA Average Weighted 5가지 부하 분산 방법을 사용하여 ecspec_lb1 ~ ecspec_lb50 ECSpec을 그림 53의 soapUI 1.7.1 툴을 사용하여 50회 단위로 500회 까지 클라이언트 요청 처리인 Event Cycle를 표 16의 분산 미들웨어 환경에 구축된 4대의 ALE RFID 미들웨어로 분산 처리를 수행할 시의 특정 기능을 수행하는 미들웨어의 가장 대표적인 부하량 산정에 대한 척도가 되는 CPU 사용률, Memory 사용량을 측정하였으며, 하드웨어 자원 사용량, 미들웨어 자원 사용량, ALE 자원 사용량도 수집하여 분석하였다.

1) 하드웨어 자원 사용량 평가

그림 60은 기존의 부하 분산 방법인 Round Robin, Random, Weighted Round Robin 방법과 본 논문에서 제안하는 GA Weighted, GA Average Weighted 부하 분산 방법을 실험하여 본 논문에서 정의한 표 10의 하드웨어 자원 사용량의 평균값을 측정한 것이다.

ecspec_lb1부터 ecspec_lb50까지 50개 처리 유형의 ECSpec를 50회에서 500회까지 실험을 하였으며 50회 단위로 5가지 각 부하 방법을 사용하여 하드웨어 자원 사용량의 평균값을 측정하였다. 그림 60의 평균 하드웨어 자원 사용량을 살펴보면 본 논문에서 제안하는 GA Weighted(GAW), GA Average Weighted(GAAW) 방식이 Round Robin, Random 방식이 보다 평균 하드웨어 자원 사용량이 적고, Weighted Round Robin과 비슷하나 수치상 보다 평균 하드웨어 자원 사용량이 적음을 알 수 있다. 이는 본 논문에서 제안한 GAW, GAAW 방식이 부하 분산 처리를 적절하게 잘 수행했음을 의미하며, GAW보다 GAAW 방식이 분산 처리를 잘 수행했음을 알 수 있다.

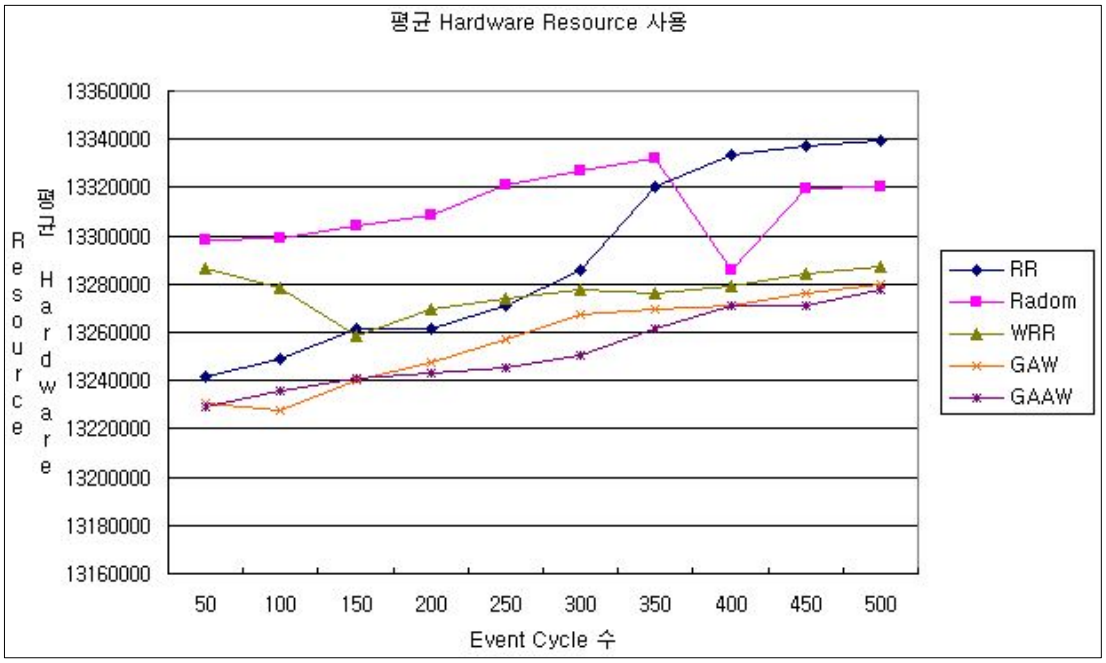

그림 60. 평균 Hardware 자원 사용량 비교 실험 결과

(1) CPU 사용률 평가

그림 61과 62는 표 16의 실험 환경에서 Round Robin 방법과 Random 방법을 이용하여 4대의 ALE RFID 미들웨어로 부하 분산을 했을 때 CPU 사용률을 측정한 것이다. 그림 61과 62의 CPU 사용률을 살펴보면 Round Robin 방법과 Random 방법의 경우 KGY ALE RFID 미들웨어로 처리 부하가 높은 ECSpec 처리 사항이 몰려있어 서버의 CPU 자원이 불균형 됨을 알 수 있다.

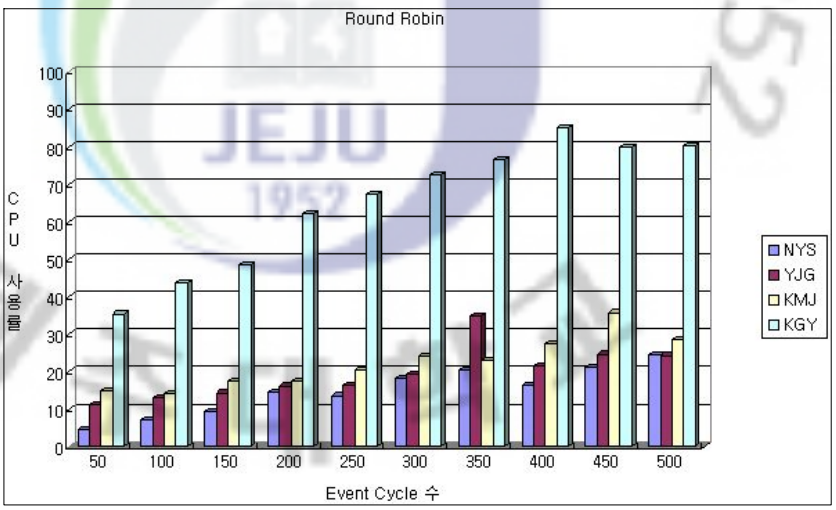

그림 61. Round Robin CPU 사용률 실험 결과

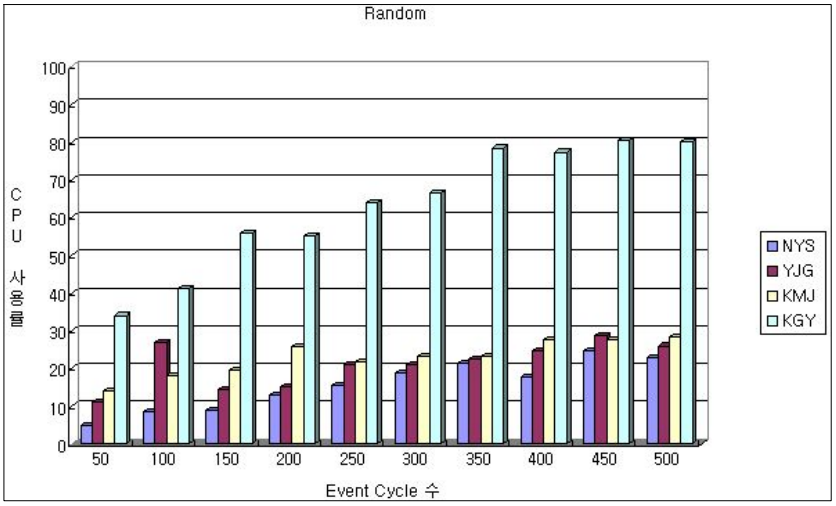

그림 62. Random CPU 사용률 실험 결과

그림 63은 표 16의 실험 환경에서 Weighted Round Robin 방법을 이용하여 4대의 ALE RFID 미들웨어로 부하 분산을 했을 때 CPU 사용률을 측정한 것이다. 그림 63의 CPU 사용률을 살펴보면 Weighted Round Robin 방법의 경우 Round Robin 방법과 Random 방법 보다는 양호하나 역시 KGY ALE RFID 미들웨어로 처리 부하가 높은 ECSpec 처리 사항이 몰려있어 서버의 CPU 자원이 불균형 됨을 알 수 있다.

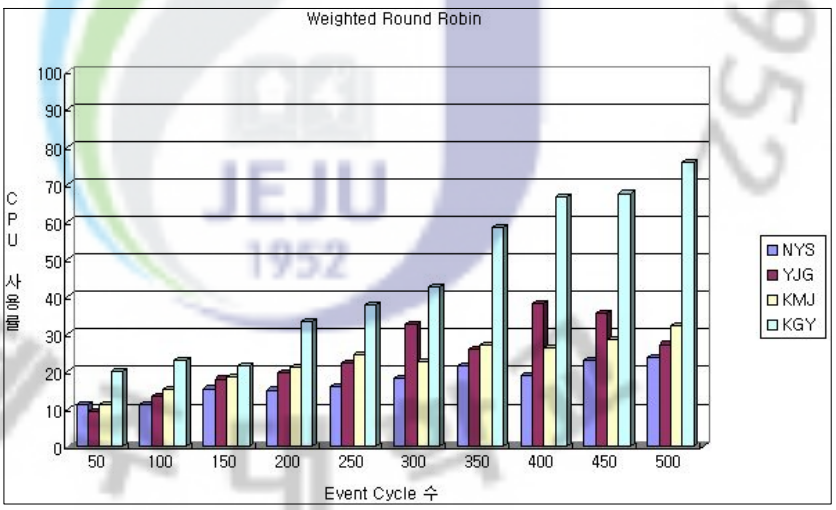

그림 63. Weighted Round Robin CPU 사용률 실험 결과

그림 63의 Weighted Round Robin 방법을 이용하면 이벤트 사이클이 50회에서 ~ 150회까지 4개의 ALE RFID 미들웨어 서버에서 처리될 시 CPU 사용률이 균등하게 될 수 있도록 분산되나, 200회를 초과 하면서 CPU 사용률이 차이가 남을 알 수 있었다.

그림 64는 본 논문에서 제안하는 유전자 알고리즘 50개의 ECSpec 개별 가중치 기반 부하 분산 방법을 이용하여 4대의 ALE RFID 미들웨어로 부하 분산을 했을 때 CPU 사용률을 측정한 것이고, 그림 65는 50개의 ECSpec 개별 가중치의 평균값을 50개 유형이 ECSpec이 처리될 시 일괄적으로 사용하여 부하 분산을 했을 때 CPU 사용률을 측정한 것이다.

Round Robin, Random, Weighted Round Robin 방법과 비교하면 이벤트 사이클이 50회에서 500회까지 4개의 ALE RFID 미들웨어 서버에서 처리될 시 CPU

사용률이 보다 균등하게 될 수 있도록 분산됨을 알 수 있었다.

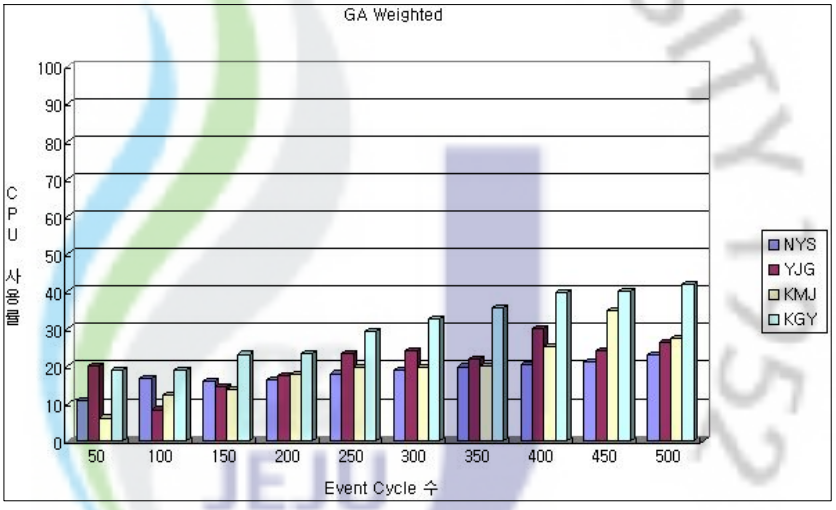

그림 64. GA Weighted CPU 사용률 실험 결과

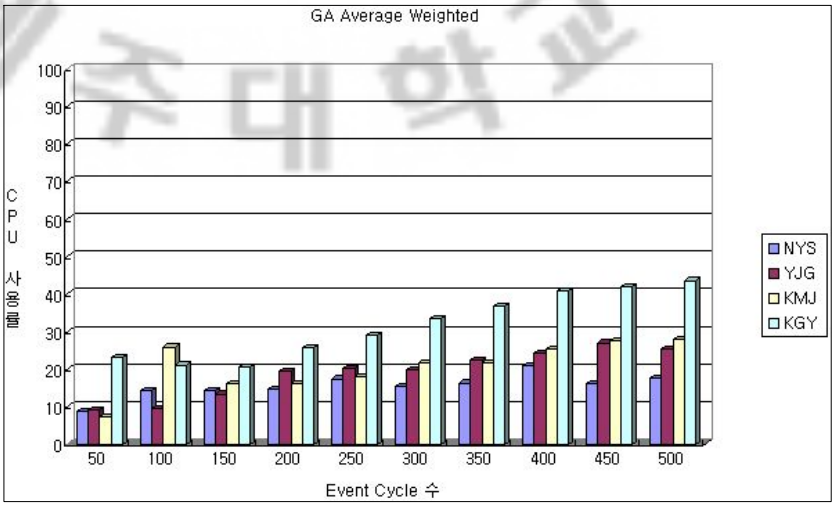

그림 65. GA Average Weighted CPU 사용률 실험 결과

그림 66은 기존의 부하 분산 방법인 Round Robin, Random, Weighted Round Robin 방법과 본 논문에서 제안하는 GA Weighted, GA Average Weighted 부하 분산 방법을 실험하여 CPU 사용률의 평균값을 측정한 것이다.

ecspec_lb1부터 ecspec_lb50까지 50개 처리 유형의 ECSpec를 50회에서 500회까지 실험을 하였으며 50회 단위로 5가지 각 부하 방법을 사용하여 CPU 사용률

의 평균값을 측정하였다. 그림 66의 평균 CPU 사용률을 살펴보면 본 논문에서 제안하는 GA Weighted(GAW), GA Average Weighted(GAAW) 방식이 Round Robin, Random, Weighted Round Robin 방식 보다 평균 CPU 사용률이 낮음을 알 수 있으며, 이는 본 논문에서 제안한 GAW, GAAW 방식이 50회에서 500회의 이벤트 사이클 처리사항을 4개의 ALE RFID 미들웨어로 부하 분산 처리를 적절하게 잘 수행했음을 의미한다.

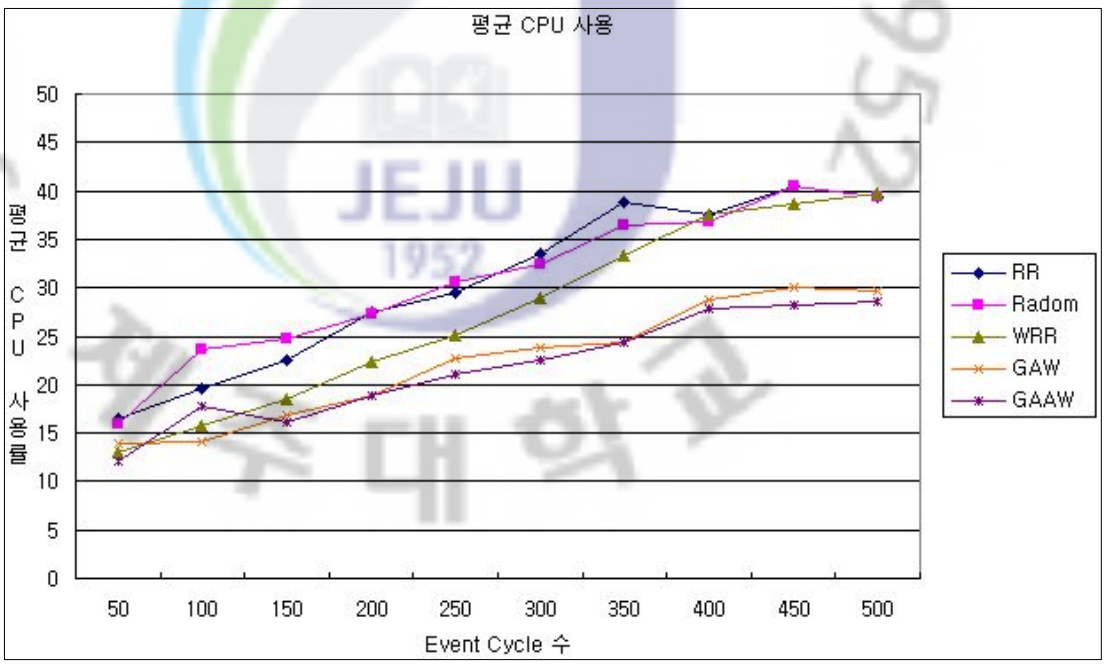

그림 66. 평균 CPU 사용률 비교 실험 결과

(2) Memory 사용량 평가

그림 67과 68은 표 16의 실험 환경에서 Round Robin 방법과 Random 방법을 이용하여 4대의 ALE RFID 미들웨어로 부하 분산을 했을 때 Memory 사용량을 측정한 것이다. 그림 67과 68의 Memory 사용량을 살펴보면 Round Robin 방법과 Random 방법의 경우 YJG, KMJ ALE RFID 미들웨어로 처리 부하가 높은 ECSpec 처리 사항이 몰려있어 서버의 Memory 자원이 불균형 됨을 알 수 있다.

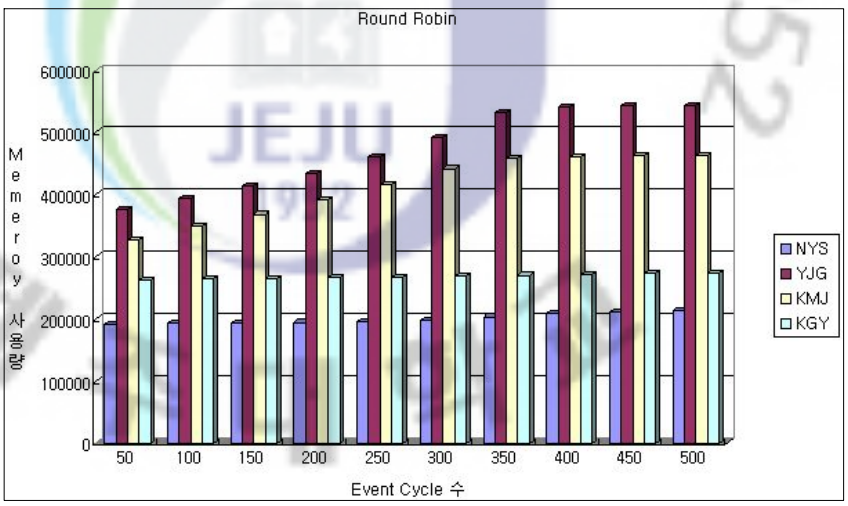

그림 67. Round Robin Memory 사용량 실험 결과

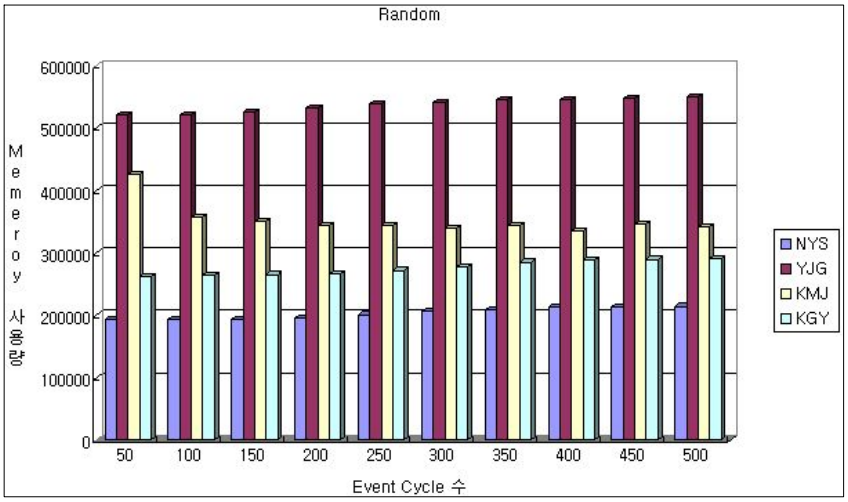

그림 68. Random Memory 사용량 실험 결과

그림 69는 표 16의 실험 환경에서 Weighted Round Robin 방법을 이용하여 4대의 ALE RFID 미들웨어로 부하 분산을 했을 때 Memory 사용량을 측정한 것이다. 그림 69의 Momoery 사용률을 살펴보면 Weighted Round Robin 방법의 경우 Round Robin 방법과 Random 방법 보다는 양호하나 역시 YJG, KMJ ALE RFID 미들웨어로 처리 부하가 높은 ECSpec 처리 사항이 몰려있어 서버의 Memory 자원이 불균형 됨을 알 수 있다.

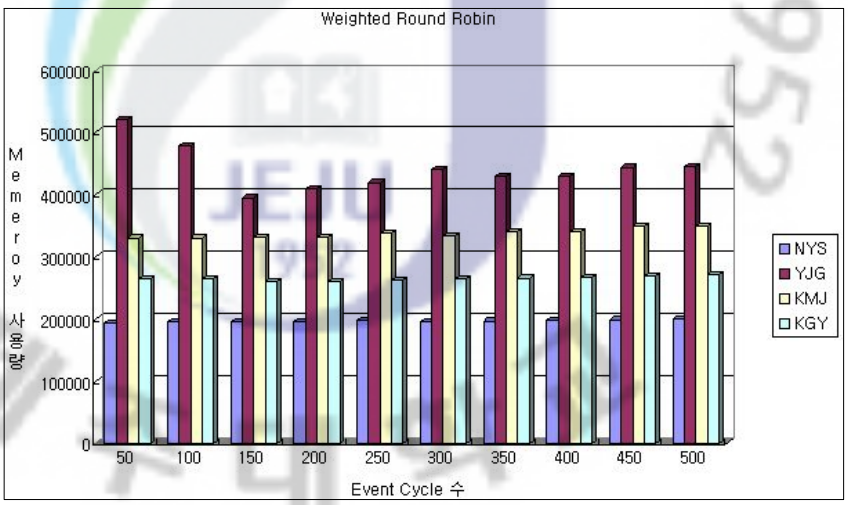

그림 69. Weighted Round Robin Memory 사용량 실험 결과

그림 70은 본 논문에서 제안하는 유전자 알고리즘 50개의 ECSpec 개별 가중치 기반 부하 분산 방법을 이용하여 4대의 ALE RFID 미들웨어로 부하 분산을 했을 때 Memory 사용량을 측정한 것이고, 그림 71은 50개의 ECSpec 개별 가중치의 평균값을 50개 유형이 ECSpec이 처리될 시 일괄적으로 사용하여 부하 분산을 했을 때 Memory 사용량을 측정한 것이다. 기존 방법과 비교하면 이벤트 사이클이 50회에서 500회까지 4개의 ALE RFID 미들웨어 서버에서 처리될 시 Memory 사용률이 보다 균등하게 될 수 있도록 분산됨을 알 수 있었다.

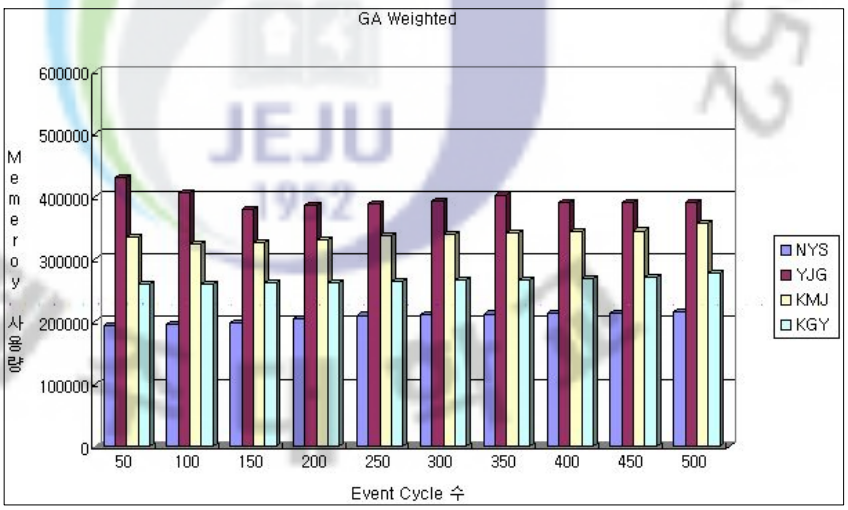

그림 70. GA Weighted Memory 사용량 실험 결과

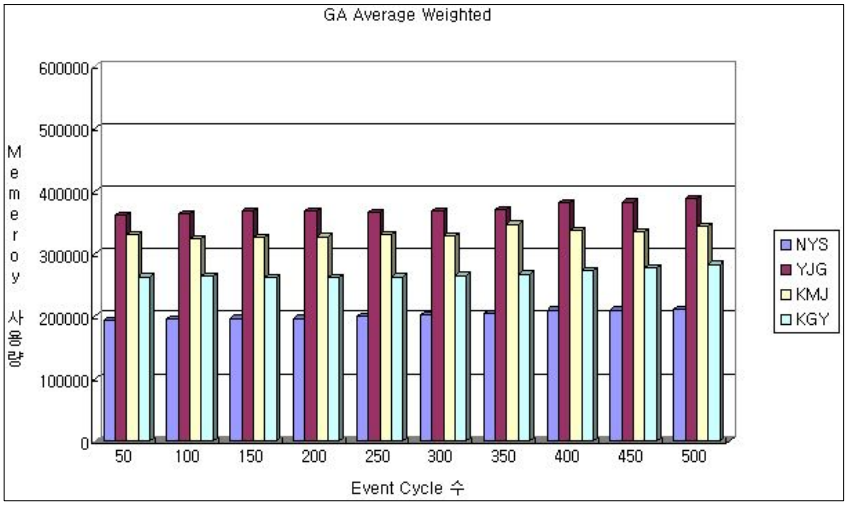

그림 71. GA Average Weighted Memory 사용량 실험 결과

그림 72는 기존의 부하 분산 방법인 Round Robin, Random, Weighted Round Robin 방법과 본 논문에서 제안하는 GA Weighted, GA Average Weighted 부하 분산 방법을 실험하여 Memory 사용량의 평균값을 측정한 것이다.

ecspec_lb1부터 ecspec_lb50까지 50개 처리 유형의 ECSpec를 50회에서 500회까지 실험을 하였으며 50회 단위로 5가지 각 부하 방법을 사용하여 Memory 사용량의 평균값을 측정하였다. 그림 72의 평균 Memory 사용량을 살펴보면 본 논문에서 제안하는 GA Weighted(GAW), GA Average Weighted(GAAW) 방식이 Round Robin, Random 방식이 보다 평균 Memory 사용량이 적고, Weighted Round Robin과 비슷함을 알 수 있다. 이는 본 논문에서 제안한 GAW, GAAW 방식이 50회에서 500회의 이벤트 사이클 처리사항을 4개의 ALE RFID 미들웨어로 부하 분산 처리를 적절하게 잘 수행했음을 의미한다.

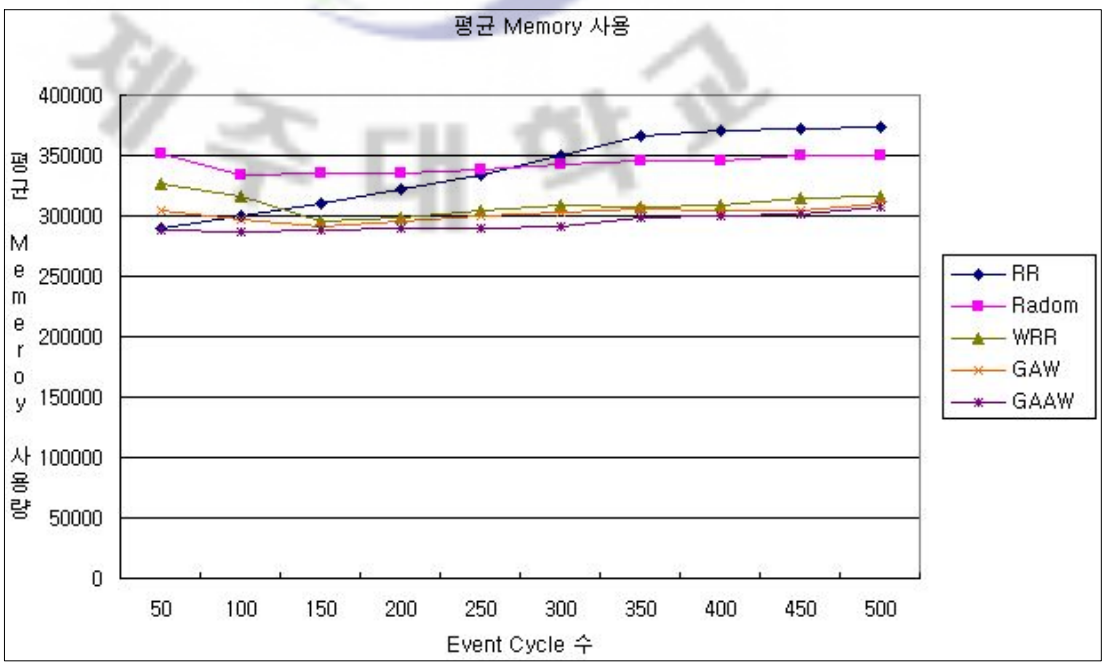

그림 72. 평균 Memory 사용량 비교 실험 결과

2) 미들웨어 자원 사용량 평가

그림 73은 기존의 부하 분산 방법과 본 논문에서 제안하는 GA Weighted, GA Average Weighted 부하 분산 방법을 실험하여 본 논문에서 정의한 표 11의 미들웨어 자원 사용량의 평균값을 측정한 것이다. 하드웨어 자원 사항 같은 방법으로 미들웨어 자원 사용량의 평균값을 측정하였으며, 그림 73의 평균 미들웨어 자원 사용량을 살펴보면 본 논문에서 제안하는 GA Average Weighted(GAW) 방식과 Round Robin, Random, Weighted Round Robin 방식은 유사하게 증가함을 알 수 있으며, 350개의 이벤트 사이클을 처리할 때부터 GA Weighted 방식만 평균 미들웨어 자원 사용량이 증가함을 알 수 있다. 이는 GA Weighted 방식을 제외한 4개 방식은 4개의 ALE RFID 미들웨어로 일정하게 Event Cycle 처리를 분산하며, GA Weighted 방식은 ECSpec의 개별 가중치 정보를 이용하여, 상대적으로 고성능의 서버 사양을 갖는 미들웨어로 Event Cycle 처리 사항을 집중시켜 부하를 균등하게 하기 때문에 Event Cycle 처리 사항이 증가함에 따라 비례적으로 증가하는 미들웨어 자원 사용량 역시 평균값이 증가하게 된다.

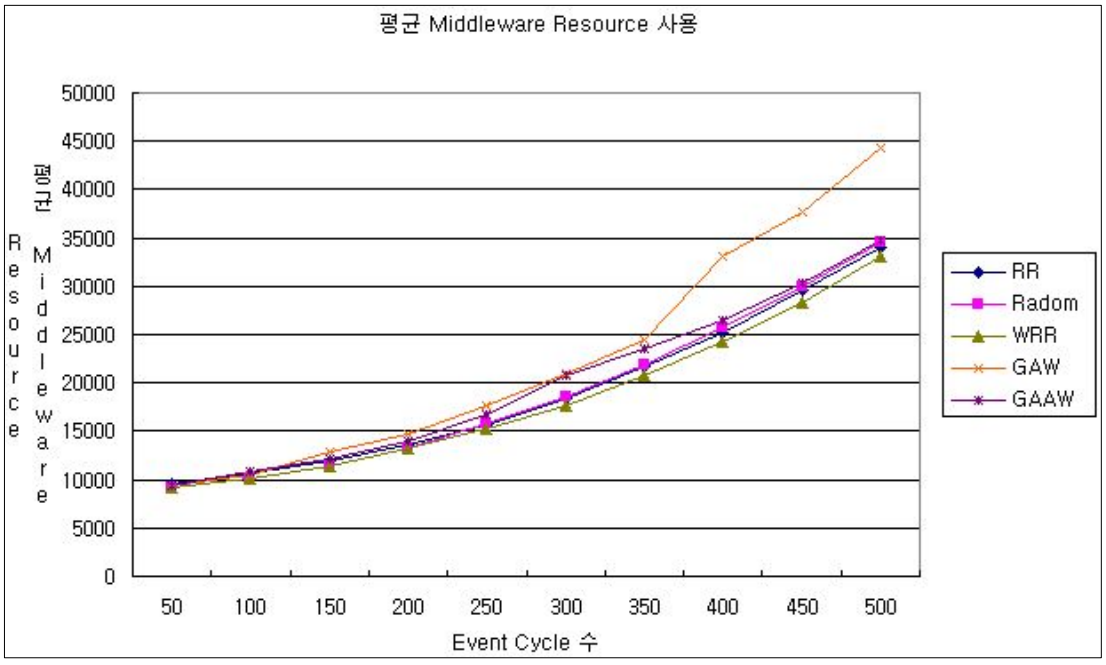

그림 73. 평균 Middleware 자원 사용량 비교 실험 결과

3) ALE 자원 사용량 평가

그림 74는 기존의 부하 분산 방법과 본 논문에서 제안하는 GA Weighted, GA Average Weighted 부하 분산 방법을 실험하여 본 논문에서 정의한 표 12의 ALE 자원 사용량의 평균값을 측정한 것이다. 하드웨어 자원 사항 같은 방법으로 ALE 자원 사용량의 평균값을 측정하였으며, 그림 73의 평균 ALE 자원 사용량을 살펴보면 Round Robin, Random, Weighted Round Robin 방식은 유사하게 증가함을 알 수 있으며, 본 논문에서 제안하는 GA Average Weighted(GAW) 방식은 기존 3개의 방식과 차이는 있으나 유사하게 증가하며, GA Weighted 방식은 미들웨어 자원 사항 평가 시와 같은 이유로 350개의 이벤트 사이클을 처리할 때부터 평균 ALE 자원 사용량이 증가함을 알 수 있다. 이는 Event Cycle 처리 사항이 증가함에 따라 미들웨어 자원 정보와 ALE 자원 정보가 유사하게 증가함을 알 수 있다.

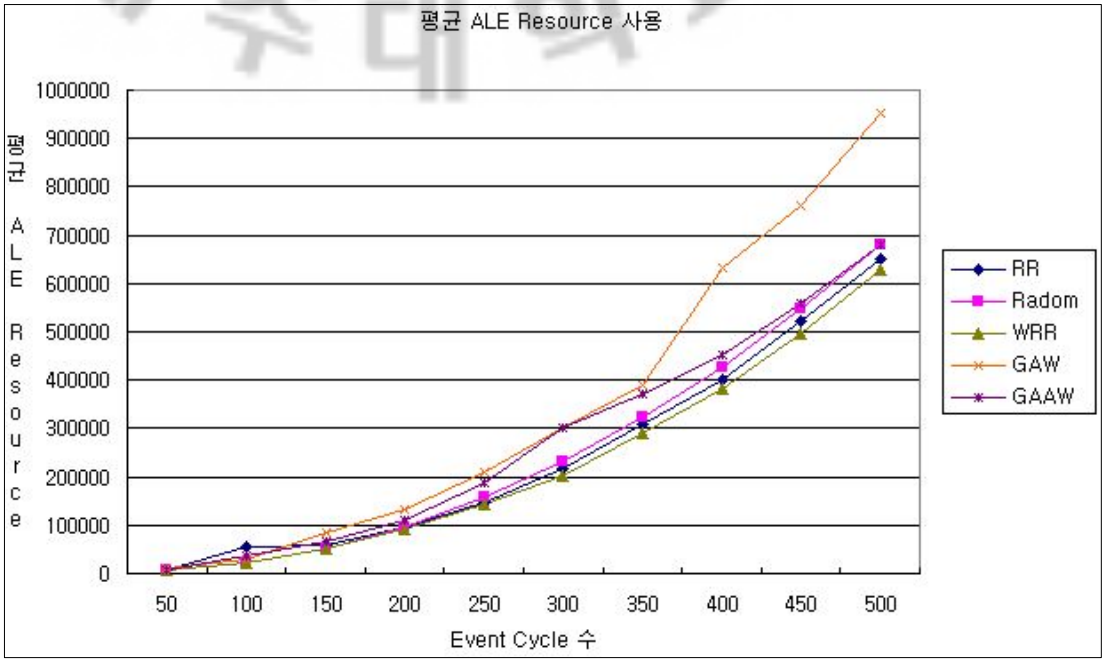

그림 74. 평균 ALE 자원 사용량 비교 실험 결과

6. 성능 평가 결과 분석

본 논문에서 제안하는 대량 RFID 데이터 처리를 위한 부하 분산 방법인 GA Weighted, GA Average Weighted의 성능을 분석하고 평가하기 위해, 기존의 부하 분산 방법인 Round Robin, Random, Weighted Round Robin 방법과 비교 실험 하였다.

ALE RFID 미들웨어의 부하 상황 정보에 해당하는 그림 60, 73, 74와 같은 평균 하드웨어, 미들웨어, ALE 자원 사용량을 비교하여 기존의 부하 분산 방법보다 클라이언트의 RFID Tag 데이터 수집 및 가공 처리 요청을 잘 분산할 수 있음을 확인하였고, 분산 환경에 구축된 4대의 ALE RFID 미들웨어의 자원 사항을 최대한 활용할 수 있어 보다 많은 이벤트 사이클을 처리하여 대량의 RFID 데이터를 처리할 수 있음을 확인하였다.

분산 미들웨어 환경에 구축된 ALE RFID 미들웨어의 가장 중요한 부하 상황 정보인 그림 66의 평균 CPU 사용률을 살펴보면 본 논문에서 제안하는 GA Weighted(GAW), GA Average Weighted(GAAW) 방식이 Round Robin, Random, Weighted Round Robin 방식이 보다 평균 CPU 사용률이 낮음을 알 수 있었으며, 이는 본 논문에서 제안한 GAW, GAAW 방식이 50회에서 500회의 이벤트 사이클 처리사항을 4개의 ALE RFID 미들웨어로 부하 분산 처리를 적절하게 잘 수행했음을 의미한다.

특히, 50개 유형의 ECSpec 개별 가중치 보다, 50개 유형의 ECSpec 평균 가중치 값을 기반으로한 부하 분산 방법이 보다 부하 분산 처리를 잘 수행했음을 확인하였고, 50개 유형 이외의 ECSpec 처리 사항에도 평균 가중치 값을 사용할 수 있을 것으로 기대 된다.

V. 결론 및 향후 연구

1. 결론

본 논문에서는 분산 미들웨어 환경에 EPCglobal의 ALE 스펙을 기반으로 구현된 RFID 미들웨어를 구축하였고, 이를 이용하여 보다 대량의 RFID 데이터를 수집하여 클라이언트 응용에게 가공된 RFID 데이터를 효율적으로 제공하기 위한 방법을 제안하였다.

제안하는 대량 RFID 데이터 처리를 위한 부하 분산 방법은 미들웨어 분산 환경에 구축된 ALE RFID 미들웨어들의 클라이언트 RFID Tag 데이터 수집 및 가공 처리 성능을 향상시키기 위해 ALE RFID 미들웨어 처리되는 50개 처리 유형의 ECSpec이 소모되는 자원에 정보를 분석하고, 최적화 알고리즘인 유전자 알고리즘 연산을 이용하여 ECSpec 각각 마다 미들웨어 분산 환경에서 최적으로 부하 균등을 이뤘을 경우의 유전자 정보를 적합도 함수를 바탕으로 찾아 가중치 정보를 생성한다. 이렇게 생성한 가중치 정보를 이용한 클라이언트의 요청을 분산하는 방법을 제시하였고, 실험을 통해 ALE RFID 미들웨어 서버들 간의 자원이 편중되지 않고 효율적으로 분배된다는 것을 알 수가 있었다.

기존 부하분산 시스템은 부하 분산의 리얼 서버인 미들웨어의 일반적인 자원 정보만을 수집하고 있다. 일반적인 자원 정보로 CPU, Memory 자원 정보 등을 수집하여 부하 분산 시 활용하며 이는 부하 분산을 하기 위한 중요한 정보이기는 하나, 보다 세부적이고 미들웨어의 특성을 고려한 부하 분산을 위해서는 추가적인 자원 정보가 필요하다.

또한, 기존의 부하 부산 시스템은 특정 기능을 수행하는 미들웨어 형태로 구현되었으나 미들웨어의 DB 접속수, 스레드 처리, 모듈 처리와 같은 자원 정보를 고려하고 있지 않으며, 미들웨어에서 처리되는 부하 상황 정보에 해당하는 이러한 정보를 부하 분산 시 활용할 필요가 있다.

끝으로 기존의 ALE 기반 RFID 미들웨어의 부하 분산 방법은 CPU 사용률,

ECSpec 수, RFID 리더 수, 수집된 RFID Tag 수 등만을 고려하였다. ALE 스펙을 기반으로 처리되는 RFID 미들웨어에서는 실제 처리되는 Event Cycle에 대한 자세한 정보가 필요하며, 다양한 유형의 ECSpec의 처리사항을 분석하여 적절한 RFID 미들웨어로 부하 부산 처리를 할 필요가 있다.

본 논문에서는 이러한 문제를 해결하기 위하여 기 구현된 ALE 스펙 기반 RFID 미들웨어의 부하 상황 정보에 해당하는 자원 사항을 표 10의 하드웨어 자원, 표 11의 미들웨어 자원, 표 12의 ALE 자원 정보 등 다양한 부하 상황 자원 정보를 수집하여 효율적인 부하 분산과 미들웨어 분산 환경에 구축된 ALE 기반 RFID 미들웨어들이 일정하게 부하 균형을 이룰 수 있도록 활용하였다.

그리고, 미들웨어 분산 환경에 구축된 서로 사양이 다른 4대의 ALE RFID 미들웨어에서 각각 ecspec_lb40이라는 이름을 갖는 ECSpec을 처리할 시에 각 미들웨어에서 사용되는 하드웨어, 미들웨어, ALE 자원 정보는 모두 다르며, 이렇게 다른 자원 정보에 따라 가중치 정보를 구하여 클라이언트의 RFID Tag 데이터 수집 및 가공 처리가 요청될 시 각 미들웨어로부터 수집된 하드웨어, 미들웨어, ALE 자원 각각에 가중치 정보를 적용하여 종합 부하 정보를 계산하였다. 이렇게 계산된 정보를 바탕으로 부하 정보가 적은 ALE RFID 미들웨어를 선정하여 클라이언트의 요청을 분산 처리하였다.

본 논문에서 제안하는 부하 분산 방법의 기대 효과 및 활용 방안은 다음과 같다. 기존의 연구 보다 대량의 RFID Tag 데이터를 처리할 수 있어 증가되는 RFID 사용 수요에 대한 유비쿼터스 서비스를 효과적으로 제공할 수 있으며, 처리 속도보다는 대량 및 지속적인 처리와 처리량에 중점을 두어 방대한 RFID 데이터를 처리할 수 있다.

국제 표준 방법을 기반으로한 RFID 미들웨어에 적용하여 다양한 응용 환경에 적용 가능하며, RFID 뿐만 아니라 USN 등의 환경에서도 활용 될 수 있으며, 다양한 유형의 ECSpec 처리사항을 분석하고, 최적화 알고리즘인 유전자 알고리즘을 이용하여 다양한 서버 상황에 대한 가중치 정보를 구하고, 최적의 ALE RFID 미들웨어 서버 선정에 사용될 수 있다.

끝으로 50개 유형의 ECSpec 개별 가중치 보다, 50개 유형의 ECSpec 평균 가중치 값을 기반으로한 부하 분산 방법이 보다 부하 분산 처리를 잘 수행했음을

확인할 수 있었고, 본 논문에서 제시한 50개 유형 이외의 ECSpec 처리 사항에도 평균 가중치 값을 사용할 수 있을 것으로 기대 된다.

2. 향후 연구

본 논문에서는 대량 RFID 데이터 처리를 위한 부하 분산 방법을 제안하였다. 제안하는 방법은 서로 다른 50개 처리 유형의 ECSpec를 분산 미들웨어 환경에 구축된 사양이 다른 4대의 ALE RFID 미들웨어에서 실험하여 서로 다른 자원 정보에 따라 가중치를 구하고, 부하 분산 시 수집된 미들웨어의 하드웨어, 미들웨어, ALE 자원 정보 각각에 적용하였다.

또한, CPU 사용률, OS 메모리 사용량, 힙 메모리 사용량 등의 하드웨어 자원 정보와 DB 접속 수, 스레드 실행 수, 객체화된 클래스 수 등의 미들웨어 자원 정보, 그리고, 등록된 논리 리더 수, 실제 Tag 데이터를 수집하고 있는 논리 리더 수, 논리 리더에서 수집하고 있는 RFID Tag 수, 처리 되고 있는 Event Cycle 수 등의 ALE 자원 정보를 수집하여 다양한 부하 상황을 처리하고자 하였다.

그러나, 본 연구에서는 ecspec_lb1에서 ecspec_lb50까지 50개의 처리 유형을 갖는 ECSpec을 고려하여, 각각 ECSpec 마다 서로 다르게 소모하고 있는 자원 정보에 대한 가중치를 구하여 정의한 50개 유형이외의 ECSpec에 대해서는 적절한 부하 분산 처리를 수행하지 못할 가능성이 있다.

본 연구에서 50개 유형의 평균 가중치 값을 이용한 부하 분산 방법이 개별 가중치 보다 나은 부하 분산 처리를 수행할 수 있는 가능성을 보였기 때문에 향후 RFID Tag 데이터를 필요로 하는 다양한 유비쿼터스 응용을 모두 수용할 수 있도록 다양한 ECSpec 조건을 처리할 수 있는 사실상의 국제 표준인 EPCglobal의 ALE 스펙을 기반으로한 RFID 미들웨어 분산 시스템의 부하 분산 방법에 대해 연구할 필요가 있다.

참고문헌

[1] 원종호, 이미영, 김명준, "유비쿼터스 컴퓨팅 환경을 위한 RFID 기반 센서 데이터 처리 미들웨어 기술 동향", 전자통신동향분석, 제 19권, 제 5호, pp.21-30, 2004.

[2] EPCglobal Inc, "The EPCglobal network: overview of design, benefits, and security", Available: http://www.epcglobalinc.org, September, 2004.

[3] EPCglobal, "The Application Level Event(ALE) Specification, Version 1.0", September, 2005.

[4] The STREAM groups STREAM: The Stanford Stream Data Manager (short overview paper) IEEE Data Engineering Bulletin, March, 2003.

[5] 박병섭, "대용량 데이터처리를 위한 XML기반의 RFID 미들웨어시스템" 한국콘텐츠학회논문지, 제7권, 제7호, pp.31-38, 2007.

[6] EPCglobal, "The EPCglobal Architecture Framework", Final Version of 1.0 July, 2005.

[7] EPCglobal, "EPCglobal Tag Data Standards Version 1.3, EPCglobal Standard Specification", March 2008.

[8] EPCglobal, "RM Standard Version 1.0.1", May, 2007.

[9] 박미선, "실시간 RFID 미들웨어에서의 태그 데이터 고속 필터링 방법", 한양대학교, 석사학위논문, 2010.

[10] EPCglobal, "The Application Level Events(ALE) Specification, Version 1.1", EPCglobal Standard Specification, September, 2008.

[11] EPCglobal, "EPC Information Service(EPCIS) Version 1.0.1 Specification", September, 2007.

[12] EPCglobal, "Object Name Service(ONS) Version 1.0.1 Specification", May, 2008.

[13] Bill Glover, Himanshu Bhatt 저, 서환수 번역, "실무자를 위한 RFID 이해와 활용", 한빛미디어, 2007.

[14] Sun Java System RFID Software, http://www.sun.com/software/products/rfid/index.xml, July, 2006.

[15] ConnecTerra, "RFTagAware", http://www.connecterra.com, March, 2004.

[16] Taesu Cheong, Youngil Kim, and Yongjoon Lee, "REMS and RBPTS : ALE-compliant RFID Middleware Software Platform", ICACT 2006, February, 2006.

[17] 홍연미, "ALE 기반 RFID미들웨어 설계 및 구현", 제주대학교, 석사학위논문, 2008.

[18] Chan, W., and Nieh, J. Group ratio round-robin: An O(1) proportional share scheduler. Tech. Rep. CUCS-012-03, Department of Computer Science, Columbia University, April 2003.

[19] A. Raha, N. Malcolm, Wei Zhao, "Hard real-time communications with weighted round robin service in ATM local area networks," iceccs, First IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'95), pp. 96-103, August, 1995.

[20] Yu Shengsheng, Yang Lihui, Lu Song, Zhou Jingli, "least-connection algorithm based on variable weight for multimedia transmission", Proceedings of the WSEAS International Conferences, pp. 1441-1445, September, 2002.

[21] Li-Hui Yang, Sheng-Sheng Yu, "A variable weighted least-connection algorithm for multimedia transmission", Journal of Shanghai University, Volume 7, Number 3, 256-260, July, 2003.

[22] 남궁영, "저사양 기반의 하드웨어를 이용한 Linux 라우터 서버 구현", 단국대학교, 석사학위논문, 2007.

[23] J. H. Holland, "Adaptation in Natural and Artificial Systems", University of Michigan Press, 1975.

[24] 성기택, "유전자 알고리즘을 이용한 센서 네트워크의 수명연장에 관한 연구", 부경대학교, 박사학위논문, 2007.

[25] A. J. Page, T. J. Naughton, "Dynamic Task Scheduling using Genetic

Algorithms for Heterogeneous Distributed Computing", The 19th IEEE International Parallel and Distributed Processing Symposium, pp. 179-189, New York, April, 2005.

[26] S. H. Lee, C. S. Hwang, "A Dynamic Load Balancing Approach using Genetic Algorithm in Distributed Systems", IEEE International Conference, pp. 639-644, 1998.

[27] 김중효, "유전자 알고리즘과 신경망 이론의 결합에 의한 신호교차로 위험도 예측모형에 관한 연구", 전남대학교, 박사학위논문, 2009.

[28] 박성수, 박해영, "C++로 구현한 유전자 알고리즘", 한출판사, 2001.

[29] 박문용, "뉴로컴퓨터", 박영사, 1991.

[30] Hiroaki Kitano, "Empirical Studies on the Speed of Convergence of Neural Network Training using Genetic Algorithms", Machine Learning, pp. 789-795, 1990.

[31] 문종배, "분산 VOD 서버 환경에서 히스토리 기반의 동적 부하분산 스케줄러에 관한 연구", 숭실대학교, 박사학위논문, 2007.

[32] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel, "Scalable Content-Aware Request Distribution in Cluster-Based Network Servers", Proceedings of 2000 USENIX Annual Technical Conference, San Diego, June, 2000.

[33] Joo-Ho Kim, Bo-Seok Moon, and Myong-Soon Park, "MiSC : A New Availability Remote Storage System for Mobile Appliance", International Conference on Networking(ICN'05), 2005.

[34] Olston, C., Rosenstein, J. and Varma. R., "Query Processing, Resource Management, and Approximation in a Data Stream Management System", CIDR, 2003.

[35] 백성하, 이동욱, 어상훈, 정원일, 김경배, 오영환, 배해영, "블록 단위 트랜잭션을 이용한 대용량 데이터의 실시간 저장관리기", 한국공간정보시스템학회 논문지 제10권 제2호, pp.1-12, 2008.

[36] Robert Steinke, Micah Clark, Elihu Mcmahon, "A new pattern for flexible

worker threads with in-place consumption message queues", Volume 39, Issue 2, pp. 71-73, April, 2005.

[37] 안효철, "분산 객체 컴퓨팅 시스템에서 퍼지 그룹핑 기반 부하분산 기법", 성균관대학교, 석사학위논문, 2006.

[38] Abed A.K., et al., "Competition-Based Load Balancing for Distributed Systems", Proceedings of 7th International Symposium on Computer Networks, pp. 230-235, 2006.

[39] Di Wu, et al., "On the Effectiveness of Migration based Load Balancing Strategies in DHT Systems", Proceedings of 15th International Conference on Computer Communications and Networks, pp. 406-410, 2006.

[40] Jaiganesh Balasubramanian, Douglas C.Schmidt, Lawrence Dowdy, and Ossama Othman, "Evaluating the performance of middleware load balancing strategies", Proc. 8th IEEE Conf. on Enterprise Distributed Object Computing, 2004.

[41] N. G. Shivaratri, P. Krueger, and M. Singhal, "Load Distributing for Locally Distributed Systems", IEEE Int. Conf. Dist. Computer Systems, pp. 502-509, 1990.

[42] M. M. Theimer and K. A. Lantz "Two adaptive Location Policies for Global Scheduling Algorithms.", IEEE Int. Conf. Dist. Computer Systems, pp. 502-509, 1990.

[43] S. Zhou, X. Zheng, J. Wang and P. Delisle, "Utopia: a Load Sharing Facility for Large , Heterogeneous Distributed Computer Systems.", Software-Practice and Experience, vol. 23, no. 12, pp. 1305-1336, 1993.

[44] D.L. Eager, E. D. Lazowska, and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing", Performance Evaluation, Vol.6, pp.53-68, 1986.

[45] 곽노욱, "부하 분산을 고려한 ALE 기반의 RFID 미들웨어 시스템 설계 및 구현", 성균관대학교, 석사학위논문, 2009.

[46] 박재걸, 임종호, 이광민, 황경수, 채홍석, "RFID 미들웨어에 적합한 부하 분

산 기법 연구", 한국정보과학회 학술발표논문집(C), pp.145-147, 2006.

[47] Richard Korry, "A load sharing algorithm for a workstation environment", Department Computer Science, FR 35, University Washington Seattle, Washington 98195, Tech. Report 86-10-03, October, 1986.

[48] Wensong Zhang, Shiyao Jin and Quanyuan Wu, "Creating Linux virtual Servers", LinuxExpo Conf. 1999.

[49] P. Francis, S. Jamin, C. Jun, Y. Jin, D. Raz, Y. Shavitt, L. Zhang, "IDMaps : A Global Internet Host Distance Estimation Service", IEEE/ACM Transactions on Networking, Volume 9, Issue 5, pp. 525-540, October, 2001.