Regular paper http://dx.doi.org/10.13067/JKIECS.2015.10.10.1115

하둡 프레임워크 기반 분산시스템 내의 작은 파일들을 효율적으로 처리하기 위한 방법의 설계

김승현* · 김영근* · 김원중**

The Design of Method for Efficient Processing of Small Files in the Distributed System based on Hadoop Framework

Seung-Hyun Kim* · Young-Geun Kim* · Won-Jung Kim**

요 약

하둡 프레임워크는 매우 큰 크기의 파일을 처리하기에 적합하도록 설계되었다. 반면 작은 크기의 파일을 처리할 경우, 분산 시스템의 자원 낭비와 분석 성능 저하가 발생하며 이는 작은 파일의 개수가 많을수록 현저하게 나타난다. 이 문제는 파일의 크기가 작기 때문에 발생하므로, 연관성 있는 작은 파일들의 병합을 통해 해결할 수 있다. 그러나 기존의 작은 파일 병합 방법들은 부차적인 한계점을 지니고 있다. 따라서 본 연구는 기존의 병합 방법의 문제점에 대하여 살펴보고, 작은 파일들의 효율적 처리를 위한 병합 방법을 설계하였다.

ABSTRACT

Hadoop framework was designed to be suitable for processing very large files. On the other hand, when processing the Small Files, it waste the resource of a distributed system, and occur performance degradation. It is shown noticeable the more the Small Files. This problem is caused by the Small Files, it can be solved through the merging of associated Small Files. But a way of merging of Small Files has some limited point. in this paper, examines existing limit of merging method, design merging method Small Files for effective process.

키워드

Small Files, HDFS, HDFS Block, NameNode, Merging
작은 파일, 하둡 분산 파일 시스템, 하둡 분산 파일 시스템 블록, 네임 노드, 병합

Ⅰ. 서 론

하둡은 다수의 컴퓨터를 네트워크로 연결한 시스템에서 단순한 프로그래밍 모델을 이용하여 크기가 큰 데이터를 분산 처리하는 프레임워크[1]이다. 오늘날 하둡은 클라우드 기반의 센서 데이터 센터 역할 뿐만 아니라 다양한 분야에서 널리 사용되고 있다[2-4].

하둡 프레임워크의 핵심 원리는 부하의 분산이며, 이러한 역할을 두 개의 컴포넌트; 하둡분산파일시스템(HDFS : Hadoop Distributed File System)과 맵리듀스(MapReduce)가 담당하고 있다.

HDFS는 분산 시스템을 위한 파일시스템이며, 일반

* 순천대학교 컴퓨터과학과(kimtmdgus22@gmail.com)
** 교신저자 : 순천대학교컴퓨터공학과
· 접 수 일 : 2015. 09. 11
· 수정완료일 : 2015. 10. 13
· 게재확정일 : 2015. 10. 23
· Received : Sep. 11, 2015, Revised : Oct. 13, 2015, Accepted : Oct. 23, 2015
· Corresponding Author : Won-Jung Kim
Dept. of Computer Engineering, Sunchon National University,
Email : kwj@sunchon.ac.kr

적인 UNIX/LINUX의 파일시스템과 유사하지만, 성능 향상을 위해 POSIX를 지원하지 않는다[5]. 이 파일시스템의 특징은 쪼개진 상태로 저장된 파일들을 사용자에게는 병합된 하나의 파일처럼 보이도록 추상화하며, 분산 환경에서 발생 가능한 여러 가지 실패에 대하여 안정된 성능을 제공한다.

MapReduce는 대규모 데이터의 병렬처리를 담당하며, Mapper와 Reducer를 통해 병렬 계산을 수행하고 결과를 병합한다. MapReduce는 하둡 프레임워크의 컴포넌트이면서, 분석을 위한 프로그래밍 틀(frame)을 제공하는 프레임워크이다[6].

공통적으로 HDFS와 MapReduce는 쪼개져 분산된 파일 조각들을 다루는 컴포넌트이다. 하둡의 분산 저장과 처리는 단일 디스크보다도 더 큰 데이터를 저장할 수 있도록 할뿐만 아니라, 대량의 데이터를 단시간에 분석할 수 있도록 한다. 그러나 파일시스템의 블록 크기보다 작은 파일들은 분산 저장과 처리의 이점을 얻지 못한다. 이들은 작은 크기로 인해 더 이상 쪼개지지 않는다. 소수의 작은 파일이 HDFS에 미치는 영향력은 거의 없지만, 다수의 작은 파일은 메타데이터가 저장되는 NameNode의 메모리 낭비와 MapReduce 분석 성능 저하의 원인이 된다.

본 논문은 작은 파일 처리를 위한 기존의 방법을 살펴보고 효율적인 처리 방법을 설계하였다. 논문의 Ⅱ장에서는 작은 파일 처리의 문제점과 기존의 처리 방법, 작은 파일 처리의 한계에 대하여 기술하였다. Ⅲ장에서는 작은 파일들의 효율적 처리 방법 설계에 대하여 기술하였고, 마지막 Ⅳ장에서는 결론에 대하여 제시하였다.

Ⅱ. 관련연구

2.1 작은 파일의 문제점

초기 하둡 프레임워크 개발은 일괄 처리 방식을 중심으로 진행되었다[7]. 분산 시스템에서 일괄 처리 성능을 결정짓는 가장 중요한 요소는 디스크에서 데이터 블록을 최대한 빨리 탐색하여 같은 시간동안 더 많은 데이터를 전송하는 것이다. HDFS는 이를 충실히 반영하고 있다. 이것은 일반적인 파일시스템과 유사한 블록 개념을 갖고 있지만, 64MB 또는 128MB와 같이 훨씬 더 크다.

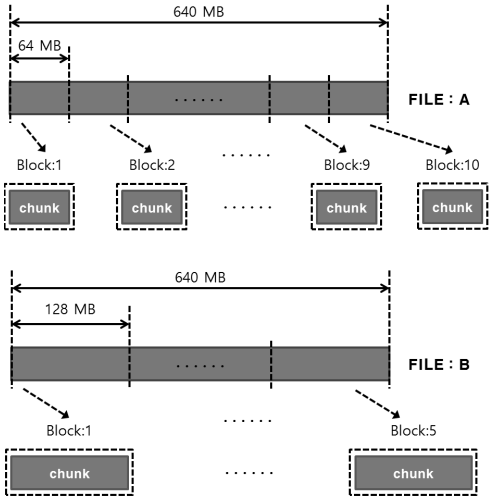


그림 1. 블록의 크기와 개수 사이의 상충관계

Fig. 1 Trade-off between the size and number of blocks

그림 1은 블록의 크기와 개수 사이의 상충관계를 나타낸다. 파일 A는 64MB 크기를 갖는 10개의 블록으로 분할되었고, 파일 B는 128MB 크기를 갖는 5개의 블록으로 분할되었다. 만약 디스크에서 해당 파일들을 탐색할 경우, 파일 A는 10개의 블록을 찾아야 하는 반면, 파일 B는 그의 절반인 5개의 블록을 찾기만 하면 된다.

이처럼 블록의 크기가 클수록 전체 블록의 개수가 줄어들기 때문에, 각 블록 시작점의 탐색시간에 소요되는 시간이 줄어들고, 그만큼 전송에 더 많은 시간을 투자할 수 있게 된다[8].

그러나 HDFS 블록보다 크기가 작은 파일들은 이와는 약간 다른 문제를 갖는다. 이것은 파일 자체의 크기가 블록 크기보다 작으므로 하나의 파일 저장을 위해 하나의 블록을 소비하게 된다.

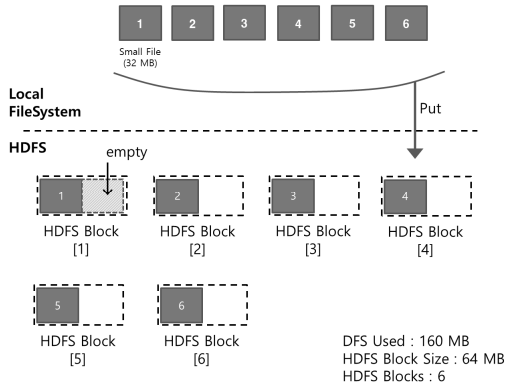
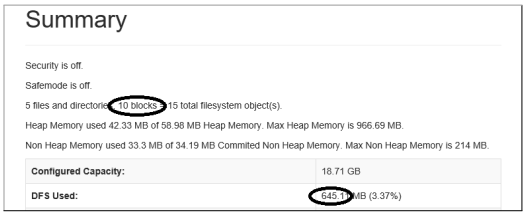


그림 2. HDFS에 저장된 작은 파일들

Fig. 2 Small files stored in HDFS

그림 2는 32MB 크기의 작은 파일 6개가 HDFS에 저장된 모습을 나타낸다. 각 파일은 HDFS 블록 크기보다 작기 때문에 나누어지지 않으며, 파일과 블록의 비율이 1:1로 같다. 즉, 총합 160MB의 작은 파일들의 저장을 위해 6개의 HDFS 블록을 사용한 것이다.

만약 같은 크기의 160MB 단일 파일을 HDFS에 저장한다면, 그것은 블록 크기보다 크므로 분할될 것이다. 그러나 분할에 사용된 HDFS 블록의 개수는 전자의 작은 파일들이 사용한 블록의 개수보다 절반이나 적은 3개가 된다(160 ÷ 64 = 2.5 ≒ 3).



(a) 640MB File × 1

(b) 32MB File × 20

그림 3. HDFS 세부 상태 정보

Fig. 3 Detailed status information of HDFS

그림 3은 640MB 단일파일 1개(a)와 32MB 작은 파일 20개(b)를 HDFS에 실제로 저장했을 때, HDFS의 상태 정보를 나타낸 것이다. HDFS 블록의 크기가 64MB 일 때, 각각 (a)는 10개, (b)는 20개의 블록이 사용됐음을 알 수 있다.

HDFS는, 그림 4와 같이, 저장된 파일의 정보를 유지하기 위해 해당 파일이 몇 개의 HDFS 블록으로 저장되었는지 알고 있어야 하며, 이 정보는 파일의 이름이나 경로와 같은 정보들과 함께 메타데이터의 형태로 NameNode의 메모리에서 관리된다.

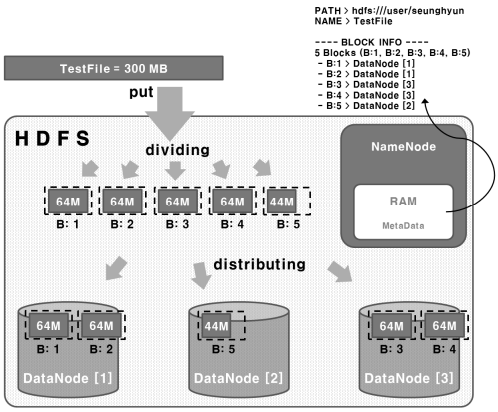


그림 4. 저장된 파일의 메타데이터 생성

Fig. 4 Generating metadata of the saved file

작은 파일들은 HDFS 블록을 과다하게 사용하며, 이는 곧 관리해야할 메타데이터량 증가와 함께 메모리 자원의 낭비를 초래하게 된다. 또한 수없이 많은 HDFS 블록은 데이터 분석 시점에서 탐색시간을 더욱 지연시키게 된다.

2.2 작은 파일 처리를 위한 기존의 방법

작은 파일의 대표적인 예는 로그 파일과 이미지 파일이며, 각각 다른 처리 방법이 존재한다.

텍스트 기반의 로그는 HDFS가 제공하는 sync() 메소드를 통해 HDFS 내부의 특정 파일에 지속적으로 내용을 추가할 수 있으며, 이미지 파일은 일종의 컨테이너를 이용하여 패키징 할 수 있다[9](현재 HDFS의 sync 메소드는 deprecated 되었으며, hflush로 대체되었다).

다음은 하둡에서 많이 사용되는 작은 파일 처리 방법이다.

- HAR(: Hadoop ARchive)

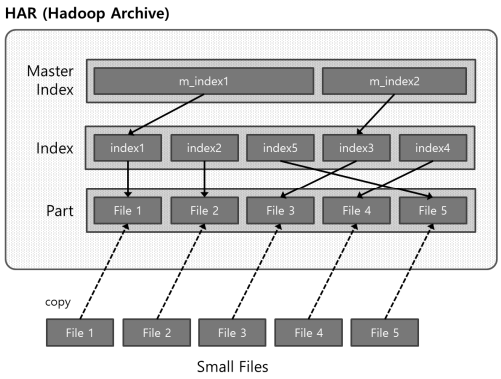


그림 5. 하둡 아카이브 레이아웃
Fig. 5 Layout of Hadoop archives

그림 5는 하둡이 제공하는 HAR[10]을 나타낸 것이다. 이 아카이브는 내부적으로 3단 계층의 구조를 가지며, 각각 해쉬(hash)와 오프셋(offset) 정보, 파일의 상태 정보, 실제 파일의 데이터를 담고 있다[11]. 이것은 작은 파일의 디렉토리 경로를 포함한 URL 스킴을 갖는다(har://).

- SequenceFile

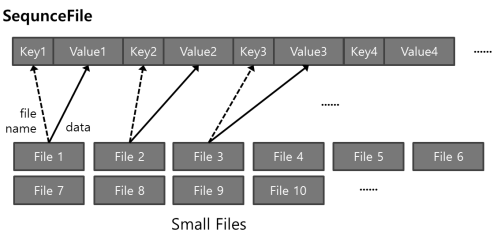


그림 6. 시퀀스파일 레이아웃
Fig. 6 Layout of sequencefile

이외에도 바이너리 타입의 효과적인 저장을 위한 컨테이너를 이용할 수 있다. SequenceFile은 그림 6과 같이 파일 이름과 파일 내용을 각각 키와 값으로 갖는 파일이며, 추가를 통해 다수의 작은 파일을 묶을 수 있다.

- CombineFileInputFormat 클래스

그림 7은 CombineFileInputFormat 클래스를 통해 어떻게 작은 파일들의 분석 작업을 분할하는지에 대한 과정을 나타내고 있다.

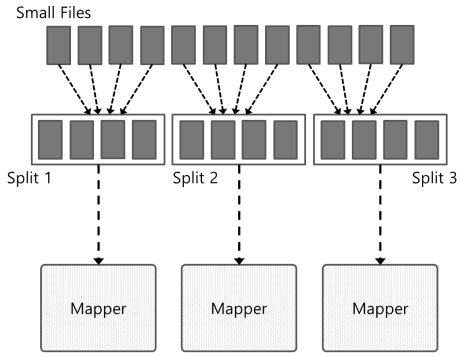


그림 7. CombineFileInputFormat의 스플릿 과정
Fig. 7 Split process of CombineFileInputFormat

스플릿은 맵리듀스의 매퍼(Mapper)가 처리할 수 있는 작업의 분량이며, 일반적으로 HDFS 블록의 크기와 동일하게 설정한다. 때문에 높은 성능을 위해 파일 입력을 스플릿 크기와 비슷하게 맞춰줄 필요가 있다. CombineFileInputFormat 클래스는 이러한 역할을 수행하며, 작은 파일들을 가상적으로 묶는다.

2.3 작은 파일 처리의 한계

HAR과 SequnceFile은 서로 다른 구조를 갖고 있지만, 작은 파일들을 하나의 파일로 병합하는 방식으로 문제를 해결한다는 점에서 공통점을 갖는다. 이 방식은 작은 파일들이 차지하는 메타데이터를 줄이고, 나아가 NameNode의 메모리 절약에 큰 도움이 된다.

그러나 파일 병합을 위해 HDFS에서 작은 파일들의 복사본을 만든다는 점에서 해당 파일들이 차지하고 있는 저장 공간만큼의 추가적인 여유 공간이 필요하다는 단점이 있으며, 이러한 작업 자체가 실제 분석 작업보다도 더 오래 걸릴 수 있다. 이는 애드혹(ad-hoc) 일괄 처리 분석에 비효율적이다.

HAR과 SequenceFile이 HDFS 저장 관점에서 살펴본 해결 방법이라면, CombineFileInputFormat 클래스는 분석적 관점에서 사용되는 해결 방법이다.

CombineFileInputFormat 클래스는 작은 파일들의 물리적 병합을 위한 사전 작업을 수행하지 않기 때문에 전체적인 처리시간은 그것들이 모두 병합된 하나의 파일을 처리하는 시간과 유사하다. 그러나 여전히 작은 파일이 HDFS 내에 존재한다는 점에서 저장 측면에서의 문제는 해결하지 못했다[12].

III. 작은 파일들의 효율적 처리 방법 설계

3.1 전송 시점에서의 병합

작은 파일의 병합 작업이 특정 파일시스템 내부에서 수행될 경우, 기존의 작은 파일들의 내용을 복사하여 하나의 큰 파일(복제본)을 만들게 된다. 이러한 과정은 비록 병합 후 기존의 작은 파일들을 삭제한다고 하더라도, 복사 시점에서는 더 많은 디스크 공간을 요구하게 된다.

따라서 로컬 파일시스템에서 HDFS로 작은 파일들을 전송할 때, 개개의 작은 파일들이 연속적인 하나의 큰 파일로 자연스럽게 병합될 수 있도록 만들 필요가 있다.

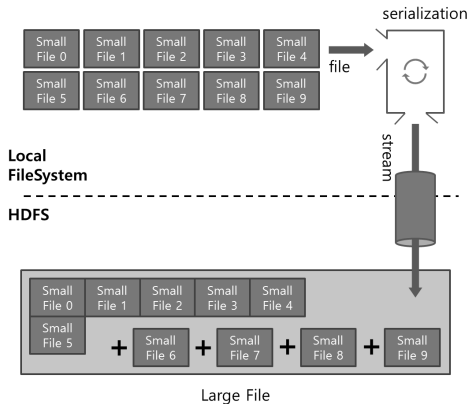


그림 8. 전송 시점에서의 작은 파일 병합

Fig. 8 Merging small files on the transmission time

그림 8는 이러한 과정을 나타낸다. 작은 파일들의 직렬화된 모든 바이트스트림을 하나의 스트림으로 반복적으로 병합하며, 맨 마지막에 close() 를 호출한다.

전송 과정에서는 작은 파일들의 스트림을 별다른 처리 없이 보내는 것 같지만, HDFS에서는 마치 처음부터 큰 파일을 전송받는 것과 같은 효과를 얻기 때문에 추후 병합작업을 별도로 수행할 필요가 없어진다. 이러한 작업은 하둡이 제공하는 입출력 API를 통해 단순하게 구현될 수 있다.

3.2 동적인 파일추가와 인덱싱

작은 파일은 언제든지 새롭게 생성될 수 있다. 반면 새로 생성된 작은 파일이 기존의 파일들과 연관되어 있다면, 이전에 병합한 파일에 새로운 내용을 추가해야 할 것이다.

그러나 내용 갱신을 위해 처음부터 다시 병합 작업을 수행하는 것은 매우 비효율적이다. 따라서 병합되어 만들어진 기존의 파일의 끝에 새로운 파일을 추가하는 방식으로 처리해야 한다.

또한 사용자는 저장된 파일에 대하여 여전히 작은 파일 형태로 접근하기를 원할 수 있다. 단순히 내용 병합을 통해 만들어진 큰 파일은 NameNode의 메모리 문제는 해결하였지만, 이전의 작은 파일들에 대한 구분자를 저장하고 있지 않다. 이를 위해 병합시점에서 원본 파일들의 시작과 끝을 구분하기 위해 인덱스를 남겨야할 필요가 있다.

그림 5는 병합된 파일 내부에 인덱스 정보를 포함한 모습을 나타낸 것이다. 인덱스는 파일의 끝에 새로운 파일을 추가할 수 있도록 포인트를 제공할 수 있으며, 특정 작은 파일들의 내용에 접근할 수 있도록 한다.

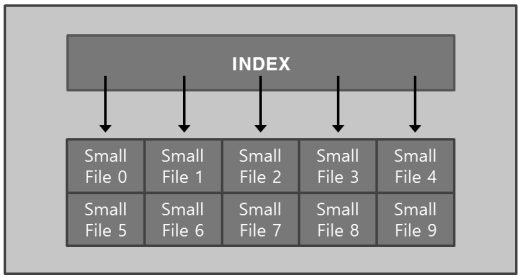


그림 9. 작은 파일들의 인덱싱

Fig. 9 Indexing of small files

Ⅳ. 결 론

본 논문은 작은 파일들을 처리하는 기존의 파일 병합 방법에 대하여 논의하였고, 효율적인 처리를 위해 부가적으로 필요한 기능들에 대하여 나열하였다. 새롭게 생성되는 작은 파일들을 전송시점에서 병합하는 것은 복제본 생성 시 필요한 추가적인 저장 공간 발생 문제를 해결할 수 있다. 그러나 새롭게 저장되는 작은 파일이 아닌, 이미 HDFS에 저장되어 있는 작은 파일들에 대해서는 효율적인 병합 방법을 제시하지 못하였다. 이들은 전송이 완료된 상태의 파일들이므로 앞서 제시한 전송시점에서의 병합이 적용될 수 없기 때문이다. 또한 하둡 프레임워크의 구조적 특수성으로 인하여 인덱싱된 개별의 작은 파일에 대해 접근하려고 할 때, 작은 파일 문제는 근본적으로 해결되지 않는다. 이러한 모든 문제를 가장 효과적으로 해결할 수 있는 방안은 하둡 생태의 HBase를 함께 사용하는 것이다. 하지만 HBase를 사용하지 않고 작은 파일들을 처리해야 한다면, 하둡의 구조적 문제와 함께 작은 파일 문제를 해결할 수 있는 연구가 진행되어야 할 것이다.

References

[1] Apache Hadoop, "What Is Apache Hadoop?," *The Apache Software Foundation*, https://hadoop.apache.org, Sept. 2015.

[2] K. Park, K. Kim, K. Ban, and E. Kim, "Design and Implementation of Cloud-based Sensor Data Management System," *J. of the Korea Institute of Electronic Communication Sciences*, vol. 5, no. 6, 2010, pp. 672-677.

[3] Y. Kim, S. Kim, M. Jo, and W. Kim, "The Bigdata Processing Environment Building for the Learning System," *J. of the Korea Institute of Electronic Communication Sciences*, vol. 9, no. 7, 2014, pp. 791-797.

[4] S. Jung and C. Sim, "A Study on a Working Pattern Analysis Prototype using Correlation Analysis and Linear Regression Analysis in Welding BigData Environment," *J. of the Korea Institute of Electronic Communication Sciences*, vol. 9, no. 10, Oct. 2014, pp. 1071-1078.

[5] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," In *Proc. IEEE Int. Symp. on Mass Storage Systems and Technologies(MSST)*, Incline Village NV, May. 2010, pp. 1-10.

[6] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, 2008, pp. 107-113.

[7] M. Asay, "Beyond Hadoop: The streaming future of big data", *InfoWorld*, http://www.infoworld.com/article/2900504/big-data/beyond-hadoop-streaming-future-of-big-data.html, Mar. 2015.

[8] T. White, *Hadoop: The Definitive Guide Fourth Edition*. Sebastopol, USA: O'Reilly, Apr. 2015.

[9] T. White, "*The Small Files Problem*," Cloudera, http://blog.cloudera.com/blog/2009/02/the-small-files-problem, Feb. 2009.

[10] Apache Hadoop, "Hadoop Archives Guide," The Apache Software Foundation, http://hadoop.apache.org/docs/r1.2.1/hadoop_archives, Aug. 2013.

[11] C. Vorapongkitipun and N. Nupairoj, "Improving performance of small-file accessing in Hadoop," In *Computer Science and Software Engineering (JCSSE), 2014 11th Int. Joint Conf. on IEEE*, Chon Buri, Thailand, May. 2014. pp. 200-205.

[12] C. Kim and J. Chung, "Processing Method of Mass Small File Using Hadoop Platform," *J. of Advanced Navigation Technology(JANT)*, vol. 18, no. 4, Aug. 2014, pp. 401-408.

저자 소개

김승현(Seung-Hyun Kim)

2014년 순천대학교 컴퓨터공학과 졸업(공학사)

2014년~현재 순천대학교 대학원 컴퓨터과학과 석사과정

※ 관심분야 : 분산처리, 하둡프레임워크

김영근(Young-Geun Kim)

2001년 한려대학교 전자계산학과 졸업(공학사)
2012년 순천대학교 대학원 컴퓨터과학과 졸업(이학석사)
2014년 순천대학교 대학원 컴퓨터과학과 박사과정 수료
※ 관심분야 : 빅데이터, 병렬분산처리시스템

김원중(Won-Jung Kim)

1987년 전남대학교 계산통계학과 졸업(이학사)
1989년 전남대학교 대학원 전산통계학과 졸업(이학석사)
1991년 전남대학교 대학원 전산통계학과 졸업(이학박사)
1992년 ~ 현재 순천대학교 컴퓨터공학과 교수
※ 관심분야 : RFID/USN, 빅데이터, Context Awareness, 인터넷 서비스