JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 21, No. 1 : 17~28 Jan. 2017

아파치 스파크 기반 검색엔진의 설계 및 구현

박기성 · 최재현 · 김종배 · 박제원*

Design and Implementation of a Search Engine based on Apache Spark

Ki-Sung Park · Jae-Hyun Choi · Jong-Bae Kim · Jae-Won Park*

Graduate School of Software, Soongsil University, Seoul 06978, Korea

요 약

최근 데이터의 활용가치가 높아지면서 데이터에 관한 연구가 활발히 진행되고 있다. 데이터의 수집, 저장, 활용을 위한 대표적인 프로그램으로 웹 크롤러, 데이터베이스, 분산처리 등이 있으며, 최근에는 웹 크롤러가 다양한 분야에 활용할 수 있는 유용성으로 인해 크게 각광받고 있는 실정이다. 웹 크롤러란 자동화된 방법으로 웹서버를 순회하여 웹 페이지를 분석하고 URL을 수집하는 도구라고 정의할 수 있다. 인터넷 사용량의 증가로 매일 대량으로 생성되는 웹 페이지의 처리를 위해 하둡의 맵리듀스를 기반으로 하는 분산 웹 크롤러가 많이 사용되고 있다. 그러나 맵리듀스는 사용이 어렵고 성능에 제약이 있는 단점이 있다. 이러한 맵리듀스의 한계를 보완하여 제시된 인메모리 기반 연산 플랫폼인 아파치 스파크가 그 대안이 되고 있다. 웹 크롤러의 주요용도 중 하나인 검색엔진은 웹 크롤러로 수집한 정보 중 특정 검색어에 맞는 결과를 보여준다. 검색엔진을 기존 맵리듀스 기반의 웹 크롤러 대신 스파크 기반 웹 크롤러로 구현할 경우 더욱 빠른 데이터 수집이 가능할 것이다.

ABSTRACT

Recently, a study on data has been actively conducted because the value of the data has become more useful. Web crawler that is program of data collection recently spotlighted because it can take advantage of the various fields. Web crawler can be defined as a tool to analyze the web pages and collects the URL by traversing the web server in an automated manner. For the treatment of Big-data, distributed Web crawler is widely used which is based on the Hadoop MapReduce. But, it is difficult to use and has constraints on the performance. Apache spark that is the In-memory computing platform is an alternative to MapReduce. The search engine which is one of the main purposes of web crawler displays the information you search by keyword gathered by web crawler. If search engines implement a spark-based web crawler instead of traditional MapReduce-based web crawler, it would be a more rapid data collection.

키워드 : 검색엔진, 크롤러, 너치, 스파크, 솔라

Key word : Search Engine, Crawler, Nutch, Spark, Solr

Received 10 October 2016, **Revised** 17 October 2016, **Accepted** 28 October 2016

* **Corresponding Author** Jae-Won Park(E-mail:jwpark@ssu.ac.kr, Tel:+82-2-828-7014)
Graduate School of Software, Soongsil University, Seoul 06978, Korea

Open Access http://doi.org/10.6109/jkiice.2017.21.1.17 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

과거 활용이 불가능하다 여겨졌던 대량의 데이터들이 기술의 발전으로 활용이 가능해짐에 따라 데이터의 가치가 높아지고 있다. 그로인해 활용 가능한 데이터를 수집하는 방법에 대한 연구도 활발히 진행되고 있다.

데이터 수집에 관한 기술 중 대표적인 것이 웹 크롤러이다. 웹 크롤러란 조직적이고 자동화 된 방법으로 웹을 탐색하고 수집하는 컴퓨터 프로그램이다[1]. 기존 웹 크롤러로는 빅데이터를 수집하는 것에 한계가 있어 대량의 데이터를 빠르게 수집하기 위해 분산 웹 크롤러 기술이 등장하였다.

빅데이터를 다루는 가장 대표적인 도구는 아파치 하둡(Apache hadoop)의 맵리듀스(MapReduce)이다. 하둡은 클러스터 컴퓨팅 프레임워크로써, 저렴한 서버를 여러 대 연결하여 데이터 처리 성능을 높이는 기술이다. 맵리듀스는 하둡의 병렬 데이터 처리 프로그래밍 모델이다.

현재 많은 분산 처리 시스템이 맵리듀스를 기반으로 개발되어 사용되고 있다. 그러나 맵리듀스는 코드가 복잡하여 사용이 어렵고, 고정된 단일 데이터 흐름을 가지기 때문에 복잡한 연산이 어렵다. 또한 배치 방식으로 데이터를 처리하기 때문에 용도에 제약이 많으며, 이후 나온 기술들에 비해 성능이 낮고 속도가 느리다.

문제를 해결하기 위한 방법 중 하나는 맵리듀스를 다른 어플리케이션으로 대체하는 것이다. 맵리듀스의 대안으로 대두되고 있는 것이 아파치 스파크(Apache spark)이다. 스파크는 맵리듀스를 확장하고 일반화시킨 툴이다[2]. 스파크는 맵리듀스에 비해 사용하기 쉽고, 인메모리 처리를 기반으로 하여 속도가 빠르며, 범용 엔진이므로 여러 가지 용도로 활용이 가능하다. 또한 스칼라, 자바, 파이썬 등의 언어를 지원하여 접근이 용이하다. 스파크는 클러스터 매니저로 Standalone Scheduler, Hadoop YARN, Apache Mesos를 이용할 수 있다.

대표적인 오픈소스 웹 크롤러인 아파치 너치(Apache Nutch) 또한 맵리듀스를 기반으로 분산 작업을 할 수 있다. USC에서 아파치 너치를 스파크를 환경에서 동작할 수 있게 수정하여 스파클러(Sparkler)라는 프로그램을 개발하였다.

본 논문은 오픈소스 검색엔진인 아파치 솔라(Apache solr)를 기반으로 웹 크롤러인 아파치 너치와 스파클러와 함께 검색엔진을 구현하였다. 맵리듀스(YARN)를 기반으로 하는 너치 웹 크롤러와 스파크를 기반으로 하는 스파클러 웹 크롤러로 데이터를 수집하고, 스파크 기반의 크롤러가 맵리듀스 기반의 크롤러와 비교하였을 때 소요시간의 단축이 이루어지는지 실험하고 검증한다.

본 논문의 구성은 다음과 같다. 2장에서는 웹 크롤러 기술과 스파크 관련 연구에 대해 설명하고 각각의 장단점을 설명한다. 3장에서는 너치를 기반으로 하는 검색엔진과 스파클러를 기반으로 하는 검색엔진의 구성을 소개하고 각각의 역할을 설명한다.

4장에서는 구현된 시스템의 성능을 측정하고 기존 분산 웹 크롤러 대비 소요시간의 단축에 대해 평가한다. 5장에서는 4장의 결과를 토대로 결론과 향후 과제를 제시한다.

Ⅱ. 관련연구

2.1. 웹 크롤링

웹 크롤링(web crawling)은 자동화된 방법으로 웹서버를 순회하여 웹페이지를 분석하고 URL을 수집하는 것을 말한다. 웹 크롤러(web crawler)는 웹 크롤링 프로그램으로 마치 거미(spider)가 거미줄(hyperlink)을 기어 다니듯(crawling) 작동한다고 하여, 웹 크롤러, 웹 스파이더, 웹 로봇 등 다양한 용어로 사용된다.

보통 시드(seeds)라고 하는 URL리스트로부터 시작하여 페이지 내의 모든 하이퍼링크를 인식하여 URL 링크를 갱신하는 것을 재귀적으로 반복한다. 그림 1은 웹 크롤러의 기본 구조를 보여준다. 가장 대표적인 응용분야로는 수집된 URL을 인덱싱하여 서버에 보존한 뒤, 쿼리로 정보를 검색하는 검색엔진이 있다[1]. 대표적인 웹 크롤러로는 아파치 너치(Apache nutch)가 있고, 검색엔진으로는 아파치 솔라(Apache solr)와 엘라스틱 서치(Elastic Search)가 있다. 모두 아파치 루씬(Apache lucene)을 기반으로 하고 있다.

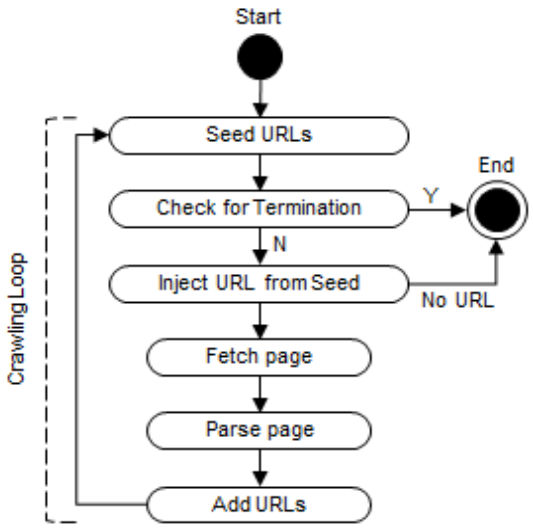

Fig. 1 Crawler structure [3]

[4]에는 웹 크롤러가 갖고 있는 3가지 한계점에 대해 언급되어 있다. 이는 Scale, Content selection tradeoff, Copyright and privacy problem으로 언급되어 있다. 매일 생성되는 방대한 양의 데이터를 수집하기 위해 웹 크롤러는 많은 처리량과 빠른 속도, 중복을 회피하는 방향으로 설계되어야 하며, 웹 크롤러가 무분별한 크롤링으로 저작권이나 프라이버시를 침해하지 않는 선에서 데이터를 수집하여야 한다는 것이다. 또한 최근 자바스크립트 등의 동적 URL은 HTML 파서만으로 데이터를 수집하기에는 한계가 있으므로 웹 스크래핑 등의 보조적인 수단이 필요하다.

대상 웹 페이지를 수집하기 위한 크롤러는 단일노드에서 동작하는 일반 웹 크롤러보다 멀티노드에서 동작하는 분산 웹 크롤러를 통해 탐색 시간을 줄이고 수집 효율을 높일 수 있다. 수집된 페이지에서 데이터를 추출함에 있어서도 분산 방법이 사용될 수 있다. 자바 등의 언어로 만든 웹 크롤러는 쓰레드(thread)를 사용하여 웹 크롤러의 처리 속도를 높일 수 있다. 그러나 [5]에서 언급한 바와 같이 파이썬(python)으로 만든 웹 크롤러에서는 GIL(Global Interpreter Lock) 때문에 멀티 쓰레딩을 할 경우 오히려 느려지는 문제가 생길 수 있다. Gevent는 libev 기반의 동시성 파이썬 라이브러리로써 쓰레드와 유사한 효과를 낼 수 있다.

[6]은 단일 시스템에서 수행되는 일반형 웹 크롤러의 성능향상과 부하분산을 위해 공유 메모리를 통한 동적 스케줄링 기법을 제안하고 있다. 일반형 웹 크롤러는 단일 시스템에서 웹 크롤링을 수행하여 중복수집의 문제는 없지만 시간이 오래 걸린다.

[7]은 서버-클라이언트 환경으로 크롤링 하는 분산 웹 크롤러 모델이다. RMI(Remote Method Invocation)와 NIO(New Input/Output) 기반의 분산 웹 크롤러를 제안하여 중복수집에 대한 문제를 해결하고자 하였다. 그러나 수집 속도의 한계가 있고 동적 웹 페이지에 대한 접근이 미흡하다.

2.2. 웹 스크래핑

전통적인 방식의 웹 크롤러들은 자바스크립트로 만들어진 동적 URL에서는 필요한 데이터를 가져오지 못한다. 이를 해결하기 위해서는 앞서 잠시 언급되었던 웹 스크래핑이 필요하다. 웹 스크래핑은 브라우저에 보이는 화면을 스크랩하여 필요한 데이터를 추출하는 프로그램으로, 기존 웹 크롤러가 처리하지 못했던 동적 웹 페이지에서도 데이터를 추출할 수 있다. 웹 스크래핑은 웹 브라우저를 기반으로 하는 방법과 프로토콜을 기반으로 하는 방법이 있다. 웹 브라우저를 기반으로 하는 대표적인 방법으로는 테스트 자동화 툴로 사용되고 있는 셀레늄 웹 드라이버(Selenium web dirver)를 이용한 방법이 있다.

셀레늄은 자바스크립트 엔진을 탑재하고 있어 웹페이지 내에 있는 자바스크립트를 연산할 수 있다. 기본적으로 파이어폭스의 플러그인으로 제공되지만 IE, 크롬, 사파리, 오페라 등 다른 브라우저도 지원한다. 또한 파이썬 뿐만 아니라 다른 개발언어들도 지원한다. 셀레늄은 테스트 코드의 실행으로 브라우저에서의 액션을 테스트 할 수 있는 도구로써, 이를 이용하면 자바 스크립트 페이지를 HTML 페이지로 변환하는 것이 가능하다. 변환된 페이지를 다시 파싱하면 원하는 데이터를 출력 가능하다. 셀레늄을 일반 브라우저의 웹 드라이버와 사용하게 되면 페이지 변환 과정에서 브라우저가 실행되었다 닫히기 때문에 상당한 시간낭비가 발생한다. 이를 해소하기 위해 헤드리스 브라우저(Headless Browser)를 사용할 수 있다. 헤드리스 브라우저는 브라우저 화면 없이 동작하는 브라우저로써, 웹 크롤러는 GUI를 필요로 하지 않기 때문에 유용하게 사용될 수 있다.

대표적인 헤드리스 브라우저로는 PhantomJS가 있다. 너치는 너치-셀레늄 플러그인을 통해 셀레늄 모듈을 이용하여 동적 웹 컨텐츠를 탐색할 수 있다.

프로토콜 기반 방식은 브라우저 없이 HTTP 통신 모듈을 통해 직접 스크래핑하는 방법으로, 대표적으로 PHP의 cURL을 이용하는 방법이 있다. cURL은 명령줄(Command line)과 URL을 합성하여 만든 이름으로, 커맨드 라인 상에서 URL을 통해 프로토콜 통신을 지원한다.

[8]는 동적 웹페이지를 처리하기 위해 일반형 웹 크롤러에 스냅샷을 접목한 모델을 제시했다. 이는 변화하는 웹 기술에 효과적으로 대응하는 방법이지만 이미지 자료의 보관과 처리에는 한계가 있다.

[9]은 프로토콜 기반 웹 스크래핑을 적용한 병렬 웹 크롤러를 구현하였으며, 멀티 프로세스와 멀티 스레드 시스템까지 구현되었다. 특정 용도로 제작된 DBMS 기반 모델로 빅데이터 환경에 적합한 모델은 아니다.

2.3. 분산 처리

대량의 데이터를 처리하기 위해 사용되는 분산 처리 프레임워크 중 가장 널리 알려진 것이 자바 기반의 오픈소스 프레임워크인 하둡(Apache Hadoop)이다. 하둡은 빅데이터 처리를 위한 대안으로 상호보완적인 프로그램들과 함께 생태계를 구성하고 있다.

하둡은 분산처리 프로그래밍 모델인 맵리듀스를 기반으로 저렴한 서버를 병렬 연결하여, 그 수에 비례하여 데이터 처리 성능이 증가한다. 그러나 사용이 어렵고, 기능이 한정적이며, 속도가 느린 단점이 있다. 하둡 버전2부터 클러스터 리소스 매니저인 얀(YARN)을 도입하여 맵리듀스 뿐 아니라 다양한 데이터 처리 어플리케이션으로 변경 가능하게 되었다[10]. 그 중 가장 각광받고 있는 어플리케이션이 스파크이다.

아파치 스파크(Apache Spark)는 하둡과 같은 분산 컴퓨팅 프레임워크로써 클러스터 매니저로 단독 스케줄러(Standalone Scheduler), 하둡 얀(YARN), 아파치 메소스(Apache Mesos)를 이용할 수 있다. 스파크는 인메모리 컴퓨팅을 이용한 빠른 연산속도와 자바, 스칼라 등의 고급언어 지원, 범용엔진으로 다양한 분야에 활용 가능한 장점이 있다.

[11]에서는 분산 환경에서 맵리듀스와 스파크의 기계학습 성능을 비교하였다. 스파크의 MLlib가 맵리듀스 기반의 아파치 머하웃(Apache Mahout)보다 약 10배 정도 빠른 처리속도를 보인다. 이는 앞서 언급했듯이 스파크가 공유메모리를 캐시하여 처리하여 입출력 횟수가 줄어들기 때문이다. 또한 [12]에는 스파크SQL이 Hive보다 빠른 처리속도를 보인다는 연구 결과를 보여주고 있다.

Ⅲ. 검색엔진의 설계 및 구현

3.1. 검색엔진의 구성

3.1.1. 아파치 하둡

하둡(Hadoop)은 하둡 분산 파일 시스템(HDFS, Hadoop distributed file system)과 분산처리 프로그래밍 모델인 맵리듀스(MapReduce)로 구성된다.

맵리듀스는 구글에서 대용량 데이터 처리를 분산 병렬 컴퓨팅 환경에서 처리하기 위해 제작한 소프트웨어 프레임워크이다. 병렬 처리를 위해 하나의 잡(Job)을 여러 개의 태스크(task)로 나누어서 실행하고 각각의 태스크는 관련성 있는 데이터를 모으는 맵(Map) 과정과 여과(Filtering), 정렬(Sorting) 과정을 통해 중복된 데이터를 제거하고 필요한 데이터를 추출하는 리듀스(Reduce) 과정으로 구성되어 있다[13]. 맵리듀스는 일괄 배치작업, 단일 입력과 단일 출력을 갖는 알고리즘에 최적화 되어있다. 그러므로 복잡한 알고리즘 구성이 어렵고, 반복적인 데이터 처리에 있어 성능의 제약을 받는다.

하둡2(hadoop 2.x 버전 이상)에서는 맵리듀스의 단점을 보완하고자 얀(YARN)이라는 시스템을 도입하였다. 하둡1은 맵리듀스가 클러스터 리소스 관리 및 데이터 처리를 맡고 있었기 때문에 반드시 맵리스터API로 프로그램을 작성하여야 되었지만, 하둡2는 얀이 클러스터 리소스 관리를 맡고 데이터 처리를 위한 어플리케이션을 선택 가능하게 함으로써 맵리듀스 뿐 아니라 다양한 데이터 처리 어플리케이션을 수용할 수 있도록 하였다.

3.1.2. 아파치 스파크

스파크(Spark)는 맵리듀스를 대체할 수 있는 범용적이면서도 고속의 클러스터용 연산 플랫폼이다[2]. 스파

크의 중요한 기능 중 하나는 연산을 메모리에서 수행하는 인메모리 처리 기능이지만, 메모리가 아닌 디스크에서 돌리더라도 맵리듀스보다 빠른 속도를 보여준다. 스파크의 구성은 간략하게 말하면 RDD (Resilient Distributed Datasets)와 인터페이스 구조이다[5]. 맵리듀스는 반복 작업을 할 때마다 HDFS를 통해 데이터를 공유하였는데, 스파크는 이를 램(RAM)에서 처리하는 개념으로 볼 수 있다. 램을 롬(ROM)처럼 Read-only로 사용함으로써 계보(lineage)를 통한 장애허용(fault-tolerant)이 가능하고 빠른 처리가 가능하다.

스파크는 스칼라, 파이썬, 자바 등 언어를 지원하여 맵리듀스에 비해 사용이 용이하다. 스파크는 배치 어플리케이션, 대화형 쿼리, 스트리밍, 머신러닝 등을 통합한 모델이다. 이는 단일 시스템으로 맵리듀스(배치처리), 스톰(스트리밍), 하이브(쿼리) 등을 대체할 수 있어 관리상의 이점을 가져온다. 또한 스파크는 얀, 메소스 등 다양한 분산처리 환경에서 운영될 수 있으며, HDFS 뿐 아니라 카산드라, HBase, S3, 몽고DB 등에서도 데이터를 가져올 수 있다.

3.1.3. 아파치 주키퍼

주키퍼(ZooKeeper)는 고가용성 분산 코디네이터이다. 주키퍼는 분산 환경에서 발생하는 정보공유, 클러스터 점검, 락킹 등의 문제를 처리할 수 있는 도구이다. 주키퍼는 Znode라는 데이터 객체와 Watcher라는 기능을 가진 디렉터리 구조의 저장소로 네임서비스를 통한 부하분산, 분산 락킹이나 동기화 문제 해결, 장애상황 판단 및 복구, 통합 환경설정 관리 기능 등을 가지고 있다. 주키퍼는 데이터의 정합성을 과반수의 룰로 판단하기 때문에 보통 3대 이상의 홀수 서버로 구성한다.

3.1.4. 아파치 카프카

카프카는 발행-구독을 기반으로 한 분산 커밋 로그 서비스이다. 링크드인(LinkedIn)에서 개발되어 2011년 오픈소스로 공개되었다. 대용량 실시간 로그처리가 가능하고 분산 시스템을 기본으로 하여 분산 및 복제가 변리하다. 가장 큰 특징은 파일 시스템을 이용한다는 것이고 그로인해 영속성이 보장된다. 대용량에 특화된 시스템이기 때문에 범용 시스템에 비해서 기능은 적다. 복제와 분산을 위해 주키퍼를 사용한다.

3.1.5. 아파치 너치

너치(Nutch)는 루씬을 기반으로 하는 오픈소스 웹 크롤러 소프트웨어이다. 웹 크롤러, HTML파서, 링크 그래프 데이터베이스 등 다양한 기능을 보유하고 있고, 플러그인을 통해 기능 확장이 용이하다. 너치는 너치1과 너치2로 나뉜다. 너치1은 하둡을 기반으로 하여 HDFS등에 자료를 저장하고, 너치2는 아파치 고라(Gora)를 데이터 추상화 계층으로 사용하여 너치1에서 사용하던 웹db뿐 아니라 Hbase, 카산드라와 같은 NoSQL 데이터베이스 사용이 가능하다.

너치는 단일노드에서 실행되지만 하둡 클러스터를 통해 분산처리가 가능하다. 파싱을 위해 아파치 티카(Apache tika)와 함께 사용할 수 있으며, 인덱싱을 위해 아파치 솔라(Apache solr)나 엘라스틱 서치(Elastic search) 등을 사용할 수 있다.

3.1.6. 아파치 솔라

솔라(Solr)는 루씬 기반의 오픈소스 기업용 검색엔진이다. 단독 어플리케이션 서버 또는 솔라 클라우드라는 분산 서버로 사용이 가능하다. 솔라 클라우드는 주키퍼를 이용하여 관리한다. 솔라는 REST 형식의 API를 사용하며, 색인을 위해 HTTP, XML, JSON, CSV, Binery 등을 추가하고 HTTP GET요청으로 Query하여, XML, JSON, CSV, 또는 Binery로 결과를 반환한다. 플러그인 아키텍처로 확장이 용이하고, 자체적인 관리용 웹 인터페이스를 제공하며, 4버전 이하는 아파치 톰캣(Apache tomcat)과 같은 서블릿 컨테이너와 함께 사용할 수 있다.

3.1.7. 스파클러

스파클러(Sparkler)는 USC IRDS에서 개발한 스파크 기반의 너치 크롤러이다[14]. 스파크, 카프카, 루씬/솔라, 티카, 펠릭스 등의 다양한 아파치 프로젝트들로 구성하며, RDD프로그램으로 인메모리 처리를 하여 성능이 뛰어나다. 너치 1.11을 기반으로 개발되어 사용법이 유사하다. 자바와 스칼라로 개발되었으며 현재 0.1버전으로 스파크 로컬모드 멀티코어에서 작동한다.

3.2. 검색엔진의 설계

클러스터는 인텔 기반 PC 물리노드 4대가 사용되었다. 너치 웹 크롤러 검색엔진 구현을 위해 [15]를 기반

으로 너치, 하둡, 주키퍼, 솔라를 설치하였고, 스파클러 웹 크롤러 검색엔진을 구현하기 위해 스파클러, 스파크, 카프카를 추가로 설치하였다. 제안하는 클러스터의 구조는 그림 2와 같다.

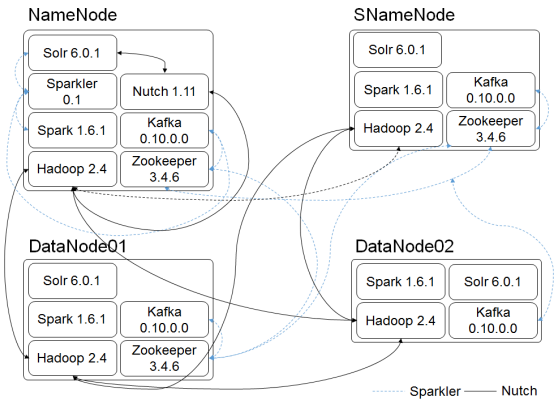

Fig. 2 Cluster structure

스파클러 기반 검색엔진의 연결은 파선으로, 너치 기반 검색엔진의 연결은 실선으로 표현하였다. 스파크와 솔라는 향후 연구의 확장을 위해 모든 노드에 설치하였다. 4대의 노드 중 1대는 네임노드, 1대는 세컨드네임노드 겸 데이터노드, 2대는 데이터노드로 구성하였다.

네임노드(마스터 노드)는 메타데이터관리, 데이터노드 모니터링, 블록관리, 클라이언트 요청 접수 등의 기능을 수행한다. 클라이언트는 네임노드를 통해 클러스터에 접근하므로 너치와 스파클러는 네임노드에 설치되고 실행된다. 네임노드의 주요 구성은 그림 3과 같다.

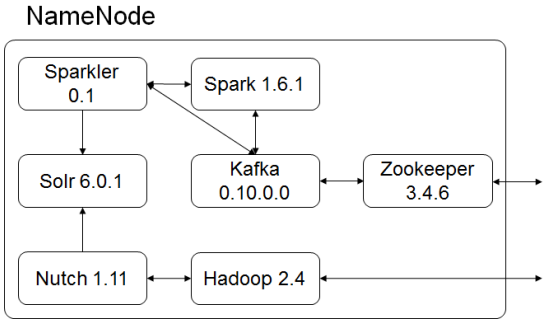

Fig. 3 NameNode structure

세컨드네임노드는 네임노드가 효과적으로 운영될 수 있도록 하며 데이터 노드의 분산 처리와 저장을 담당한다. 데이터노드(슬레이브 노드)는 블록의 백업 저장소로 로우데이터와 메타데이터를 보관한다.

너치 웹 크롤러는 하둡 맵리듀스(YARN)와 주키퍼 환경에서 분산으로 처리되며, 솔라를 통해 수집 결과를 확인한다. 스파클러 웹 크롤러는 스파크 환경에서 멀티코어로 동작하고 주키퍼 클러스터를 통해 고가용성을 유지하며 카프카로 메시지를 전달한다. 수집 결과는 솔라를 통해 확인할 수 있다. 솔라는 향후 분산 인덱싱 및 검색이 가능한 클러스터 구성인 솔라클라우드(SolrCloud)를 구성하기 위해 모든 노드에 설치하였다.

[16]에 의하면 스파크는 하둡보다 연산이 빠르고, [11], [12] 등 다양한 분야에서 그 성능이 검증되었다. 그렇다면 스파크 기반의 스파클러 웹 크롤러를 맵리듀스 기반의 너치 웹 크롤러와 비교하였을 때, 수집속도의 개선을 확인할 수 있을 것이다. 수집의 결과를 확인하기 위해 솔라와 함께 검색엔진을 구성한다. 작업 순서도는 그림 4와 같다.

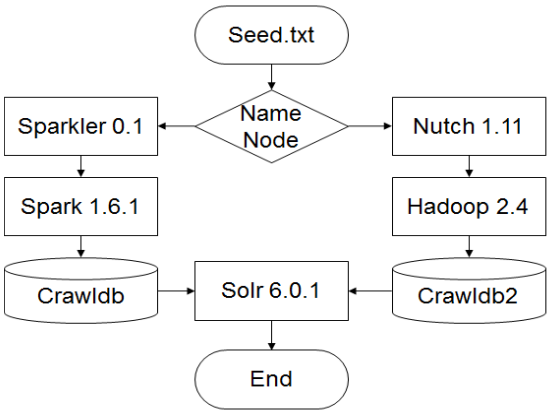

Fig. 4 Search engine work flow

3.3. 검색엔진의 구현

하둡 클러스터에 사용된 운영체제는 CentOS 7 이며, 네임노드, 세컨드 네임노드, 그리고 데이터 노드에 사용된 하드웨어 사양은 표 1과 같다.

Table. 1 Hardware option

NameNode	
CPU	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
RAM	6GB
HDD	500GB
SNameNode	
CPU	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
RAM	4GB
HDD	1500GB
DataNode01	
CPU	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
RAM	4GB
HDD	500GB
DataNode02	
CPU	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
RAM	4GB
HDD	500GB

분산처리의 원활한 통신을 위해 각 노드에 Hadoop이라는 계정을 만들고 SSH 기반으로 공개키를 통합하여 다른 노드들과 비밀번호 없이 통신이 가능하게 하였다.

노드 내의 소프트웨어 구성과 버전은 스파클러와의 호환성을 토대로 선정하였으며, 스파클러는 너치 1.11에 기반을 두고 있어 비교의 편의를 위해 너치 버전 또한 1.11버전을 설치하였다. 현재 너치의 최신버전은 2.3.1버전이나, 고라를 도입해서 확장성이 좋아진 반면 1버전에 비해 속도가 느리다고 알려져 있다[17]. 사용된 대부분의 프로그램이 자바 기반으로 동작하여 JVM환경이 필요하고, 솔라 6버전부터는 자바8 이상에서만 동작하여 Oracle JDK 8을 설치하였다. 또한 하둡 Native library와 스파클러의 빌드를 위해 아파치 메이븐(Apache Maven)과 스칼라(Scala)를 설치하였다. 설치된 소프트웨어의 버전은 표 2와 같다.

스파크는 자바8에서 동작할 경우 자바7의 경우와는 달리 메모리 설정에 민감하게 반응하여 메모리가 충분하지 않을 경우 기본 설정만으로는 메모리 오류와 함께 동작하지 않을 수 있다. 하둡의 yarn-site.xml 파일에서 yarn.scheduler.minimum-allocation-mb를 100mb정도로 컨테이너의 최소메모리 값을 적게 주고 yarn.nodemanager.resource.memory-mb 파일은 물리메모리보다 값을 크게 준다. 그리고 스파크의 spark-defaults.conf 파일에서 spark.yarn.am.memory의 값을 4g이상으로 주면 적은 메모리를 가진 클러스터에서도 실행된다.

Table. 2 Software Version

Software	Version
CentOS	7.2.1511
Oracle JDK	1.8.0u92
Scala	2.11.8
Maven	3.3.9
Hadoop	2.4.0
Spark	1.6.1
ZooKeeper	3.4.6
Kafka	0.10.0.0
Solr	6.0.1
Nutch	1.11
Sparkler	0.1

하둡은 32bit용으로 제작되었기 때문에 64bit 컴퓨터에서 사용 시 네이티브 라이브러리를 실행하지 못하는 문제가 발생한다. 이를 해결하기 위해서 하둡 소스파일을 받아 메이븐으로 빌드하여 빌드된 네이티브 라이브러리를 사용하는 하둡 폴더로 옮긴다.

스파클러는 카프카와 함께 사용하는 것을 권장하고 있다. 각 노드에서 카프카를 실행하기 전에 주키퍼를 먼저 실행하여야 하며 실행 후 명령창으로 돌아오지 않을 경우 '&'명령으로 백그라운드에서 실행한다. 앞서 언급하였듯 주키퍼는 데이터 불일치의 판단과 복구를 과반수로 결정하기 때문에 보통 홀수로 구성되며, 구성한 클러스터에서도 3개의 노드에만 설치하여 구동하였다.

클러스터를 제대로 구성했는지는 jps명령을 통해 확인할 수 있다. 그림 5는 각 노드의 jps명령으로 확인되는 프로세스의 목록이다.

본 논문은 복잡성을 줄이기 위해 네임노드에서만 솔라를 동작시켰으나 단일모드에서 동작시키는 것과 마찬가지로 주키퍼를 기반으로 각 노드에서 솔라를 실행하면 솔라클라우드 모드를 이용할 수 있다.

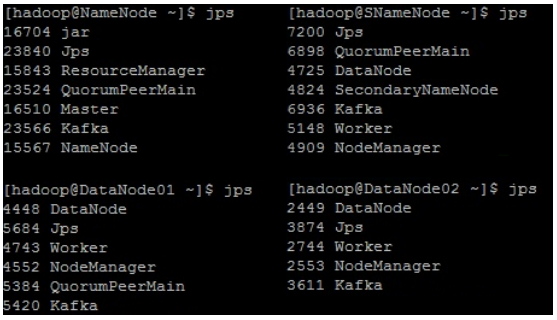

Fig. 5 Process list by nodes

너치와 솔라를 연동시키기 위해 core를 생성하여 그 안에 있는 managed-schema 파일을 수정하는 단일모드와 같이, 솔라클라우드에서도 collection을 생성하여 주키퍼 내에 위치한 managed-schema 파일을 Schema API 또는 주키퍼 명령어로 너치의 schema.xml 파일에 맞게 조정할 수 있다. Solr Admin UI에 접속하여 클러스터가 잘 구성되었는지 확인할 수 있다.

Ⅳ. 실험 및 검증

4.1. 스파클러 크롤링

스파클러는 메이븐을 통해 빌드하며 $SPARKLER_HOME의 pom.xml 파일을 수정하여 연동할 아파치 프로젝트의 버전을 설정할 수 있다. 빌드를 하여 target 디렉토리에 생성된 sparkler-app-XX.jar 파일로 실행한다. 우선 빌드 후 생성된 $SPARKLER_HOME/sparkler-app/target 디렉토리에 seed.txt 파일을 생성하여 수집을 시작할 URL을 입력한다. 본 논문에서는 http://nutch.apache.org/를 Seed URL로 한다. 스파클러는 regex- urlfilter.txt 파일에서 수집조건의 디폴드 값이 +^https?:// 으로 주어지어 모든 URL을 수집하게 설정되어 있고, $SPARKLER_HOME/conf 디렉토리의 regex-urlfilter.txt 파일을 수정하여 설정을 변경할 수 있다.

java -jar sparkler-app-0.1-SNAPSHOT.jar inject –sf seed.txt 명령을 통해서 seed URL을 inject하여 jobId를 생성할 수 있으며 생성된 JobId로 Crawl 명령을 java -jar sparkler-app-0.1-SNAPSHOT.jar crawl –id [JobId] -m local[*] -i 2 와 같이 입력 후 실행한다. 정상적으로 실행된 스파클러 크롤링 화면은 그림 6과 같다.

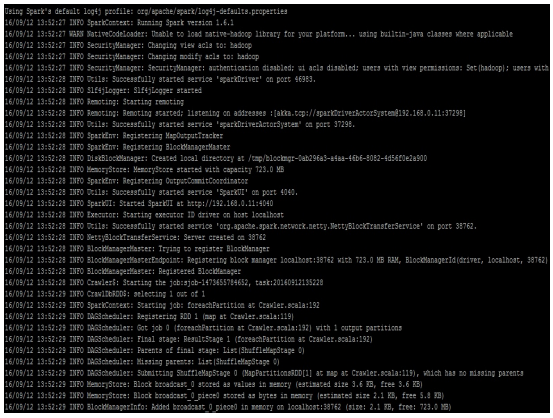

Fig. 6 Sparkler execution

4.2. 너치 크롤링

너치는 $NUTCH_HOME/bin 디렉토리 안에 nutch와 crawl 2개의 실행파일이 존재한다. nutch 실행파일은 크롤링을 inject, generate, fetch, parse, updatadb, invertlinks, index, dedup, sclean 단계별로 순차적으로 실행할 수 있게 하는 실행파일이며, crawl 실행파일은 그러한 모든 과정을 한 번에 처리할 수 있게 해주는 실행파일이다. $NUTCH_HOME에 urls와 crawl폴더를 생성하고 urls폴더 내에 seed.txt 파일을 생성하여 수집을 시작하고자 하는 URL을 적는다.

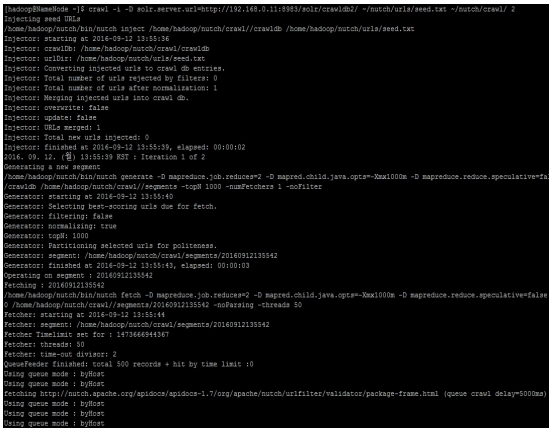

Fig. 7 Nutch execution

또한 $NUTCH_ HOME/conf 디렉토리의 regex-urlfilter.txt 파일에서 수집조건을 정규식을 포함한 수집하고자 하는 URL로 +^http://([a-z0-9]*\.)*nutch.apache.org/ 와 같이 수정한다. crawl 실행파일을 통해 crawl -i -D <Solr WebDB> <seed Dir> <Crawl Dir> <Num Rounds> 와 같이 사용할 경우 Inject, Crawl, Index까지 한 번에 처리한다. 정상적으로 실행된 너치 크롤링 화면은 그림 7과 같다.

4.3. Solr Admin UI

수집된 정보들은 솔라 웹DB에 인덱싱 된다. 인덱싱된 정보들은 Solr Admin UI에서 확인할 수 있다. 그 밖에도 수집 실패 시 확인할 수 있는 로그 정보나 코어관리 등의 기능을 제공한다. 그림 8은 Solr Admin UI에서 데이터를 확인한 화면이다.

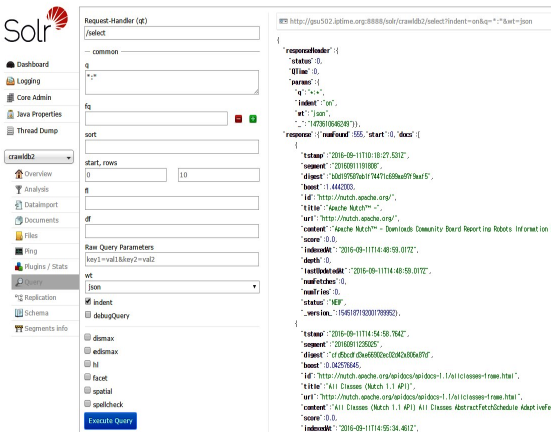

Fig. 8 Solr Admin UI

4.4. 스파클러와 너치 비교

너치 웹 크롤러와 스파클러 웹 크롤러를 가지고 실험을 수행하여, 그 결과로 제안하는 검색엔진 모델의 타당성을 검증한다.

평균적인 수집속도 확인을 위해 스파클러 웹 크롤러와 너치 웹 크롤러로 5회의 테스트를 하여 비교하였다. Seed URL은 http://nutch.apache.org/로 테스트를 하였고, 깊이(-depth)는 4로 내용(-topN)은 1000으로 크롤링을 진행하였다. 너치 웹 크롤러와 스파클러 웹 크롤러의 데이터 수집 결과와 시간은 표 3과 같다.

Table. 3 Crawling Test for 5 times

Crawler	Test	Num of Doc	Time (s)
Nutch	1	80	513
	2	80	511
	3	80	510
	4	80	516
	5	80	516
	Avg.	80	513
Sparkler	1	80	111
	2	80	113
	3	80	108
	4	80	99
	5	80	100
	Avg.	80	106

상기 결과와 같이 너치 웹 크롤러와 스파클러 웹 크롤러 모두 크롤링 깊이의 값을 4로 설정하였을 때 Seed URL을 기점으로 총 80개의 웹 문서를 동일하게 수집하였다. 두 웹 크롤러의 수집한 문서의 수는 같았으나 속도는 스파클러 웹 크롤러가 약 5배 정도 빠름을 알 수 있다. 이로 인해 인메모리 처리를 하는 스파크 머신 1대가 유사한 스펙의 머신 4대로 구성된 하둡 클러스터보다 뛰어난 성능을 보임을 알 수 있다. 그림 9는 평균 속도를 다이어그램으로 나타낸 것이다.

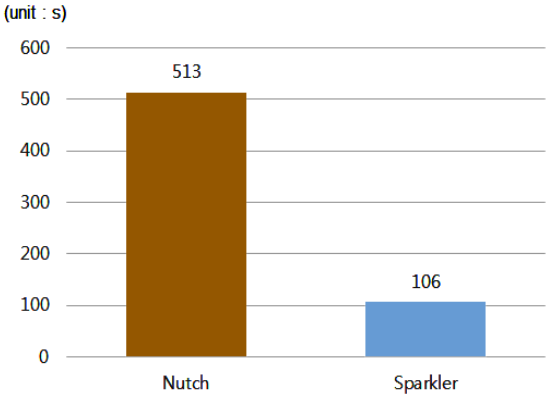

Fig. 9 Comparison of the average collection time

수집하는 문서의 수가 늘어났을 때 스파클러 웹 크롤러와 너치 웹 크롤러의 성능차이를 확인하기 위해서 깊이를 2, 3, 4, 5, 6으로 다르게 하여 추가로 5회의 테스트를 진행하였다. 결과는 표 4와 같다.

Table. 4 Crawling Test by different Depth

Crawler	Test	Num of Doc	Time (s)
Nutch	2	7	82
	3	26	200
	4	80	516
	5	579	3285
	6	1078	6047
Sparkler	2	21	4
	3	60	48
	4	80	99
	5	1446	135
	6	41656	1665

동일 깊이를 비교하였을 때 스파클러 웹 크롤러가 너치 웹 크롤러보다 전 구간에서 빠른 수집속도를 나타내고 있다. 결과를 꺾은선 그래프로 나타낸 그림 10을 보면 깊이가 깊어질수록 그 차이가 더욱 벌어지고 있다. 너치 웹 크롤러가 일부 특수문자가 포함된 URL을 수집하지 못해 스파클러 웹 크롤러가 인덱싱에 더 많은 시간을 들인 것을 감안하면 그 차이가 명확하다.

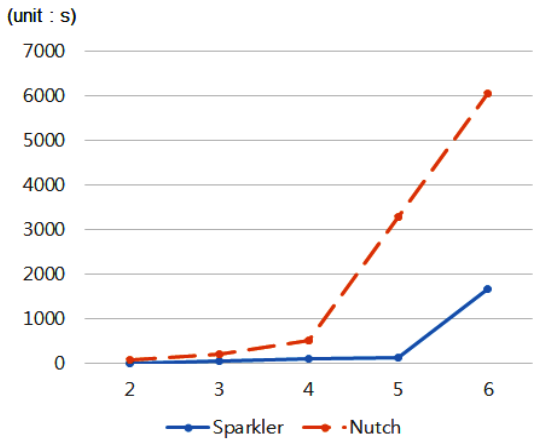

Fig. 10 Comparison of the collection time by depth

V. 결 론

하드웨어 가격의 하락과 하둡, NoSQL과 같은 새로운 패러다임의 등장으로 빅데이터 활용의 시대가 열리고 있다. 그로인해 데이터의 가치가 높아지면서 데이터 수집에 관한 연구도 활발히 진행되고 있다. 넘쳐나는 정보 속에서 필요한 정보를 탐색하여 데이터를 수집하는 것은 데이터를 활용함에 있어 필수적인 기능이다.

데이터를 수집하는 대표적인 프로그램이 웹 크롤러로써 웹의 정보를 수집하는 도구로 사용되고 있다. 루씬의 등장한 후로 그에서 파생된 하둡, 너치, 솔라 등으로 인해 웹 크롤러 및 검색엔진의 구현을 개인이 제작할 수 있게 되었고, 분산 컴퓨팅과 인메모리 기법으로 저렴하고 빠른 웹 크롤러의 구현이 가능하게 되었다.

본 논문에서는 스파크를 기반으로 한 인메모리 처리를 기존 하둡 기반의 분산처리와 대비하여 성능차이를 확인하기 위해 하둡을 기반으로 하는 너치 웹 크롤러와 스파크를 기반으로 하는 스파클러 웹 크롤러로 검색엔진을 구현하여 비교하였다. 그 결과 스파크 기반 검색엔진이 기존 방식보다 나은 성능을 보임을 확인하였다. 본 논문에서는 저가의 컴퓨터로 클러스터 환경을 구성하고 하둡, 스파크, 너치와 솔라 등의 오픈소스 소프트웨어를 이용하여 누구나 간단하게 구현할 수 있는 검색엔진 모델을 제시하였고, 스파크의 성능을 명확하게 보여줌으로써 다양한 연구로 확장할 수 있는 기반을 마련하였다.

향후 연구과제로는 스파크 기반의 크롤러가 단일노드가 아닌 하둡의 얀을 기반으로 멀티노드에서 동작할 수 있도록 개선하는 것과 너치2와 스파크를 결합하여 확장성과 빠른 검색 속도를 만족하는 크롤러를 구현하는 것, 그리고 구현한 크롤러와 솔라클라우드를 결합하여 분산 인덱싱이 가능한 검색엔진을 구현하는 것에 대한 연구가 필요하다.

REFERENCES

[1] Wikipedia. Web Crawler [Internet] Available: https://en.wikipedia.org/wiki/Web_crawler.

[2] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning Spark*, 1st ed. Sebastopol, CA: O'Reilly Media, pp.1-9, 2015.
[3] F. Pant, P. Srinivasn, F. Menczer, “Crawling the Web” in *Web Dynamics*, 1st ed. Berlin, Germany: Springer-Verlag, pp.153-177, 2003.
[4] M. S. Ahuja , J. Singh, and B. Varnica. “Web Crawler: Extracting the Web Data,” *International Journal of Computer Trends and Technology(IJCTT)*, vol. 13, no. 3, pp.132-137, Jul. 2014.
[5] Pycon. Web Scraper in 30 Minutes [Online]. Available: https://www.pycon.kr/2014/program/15.
[6] H. C. Kim and S. H. Chae. “Design and Implementation of a High Performance Web Crawler,” *Journal of Digital Contents Society*, vol. 4, no. 2, pp.127-137, Dec. 2003.
[7] D. M. Seo and H. M. Jung. “Intelligent Web Crawler for Supporting Big Data Analysis Services,” *The Journal of the Korea Contents Association*, vol. 13, no. 12, pp.575-584, Dec. 2013.
[8] K. Y. Kim, W. G. Lee, H. M. Yoon, S. H. Shin, and M. H. Lee. “Development of Web Crawler for Archiving Web Resources,” *The Journal of the Korea Contents Association*, vol. 11, no. 9, pp.9-16, Sep. 2011.
[9] S. H. Hong, “An Implementation of Smart Price Tracker System Using Web Crawling,” M.S. Thesis, Seoul National University of Science and Technology, Seoul, Korea, 2015.
[10] V. K. Vavilapalli, et al., “Apache hadoop yarn: Yet another resource negotiator,” in *ACM Proceedings of the 4th annual Symposium on Cloud Computing*, Santa Clara: CA, 2013.
[11] Y. K. Lee, “The Comparison Between Hadoop MapReduce and Spark Device's Machine Learning Performance,” M.S. Thesis, Soongsil University, Seoul, Korea, 2015.
[12] B. S. Kim, “Performance Evaluation of HDFS Based SQL-On-Hadoop,” M.S. Thesis, Chungbuk National University, Cheongju, Korea, 2015.
[13] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp.107-113, Jan. 2008.
[14] USCDataScience. Sparkler [Internet]. Available: https://github.com/USCDataScience/sparkler.
[15] H. O. Song, A. Y. Kim, and H. K. Jung. “Implement on Search Machine using Open Source Framework,” *Journal of the Korea Institute of Information and Communication Engineering*, vol. 19, no. 3, pp.552-557, Mar. 2015.
[16] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. “Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing,” in *Proceedings of the 9th USENIX conference on networked systems design and implementation*, San Jose: CA, pp.2-2, 2012.
[17] C. Klaussne, J. Nioch. (2013, September). Nutch fight! 1.7 vs 2.2.1 [Internet]. Available: http://digitalpebble.blogspot.co.uk/2013/09/nutch-fight-17-vs-221.html.

박기성(Ki-Sung Park)

2015년~현재 : 숭실대학교 SW특성화대학원 석사과정
관심분야 : 분산처리, 임베디드, 계측제어

최재현(Jae-Hyun Choi)

2004년 2월 : 숭실대학교 컴퓨터학부(석사)
2011년 8월 : 숭실대학교 컴퓨터학부(박사)
2013년 4월 : 숭실대학교 SW특성화대학원 교수
관심분야 : SW프레임워크, 서비스 엔지니어링, 클라우드, 데이터 마이닝

김종배(Jong-Bae Kim)

2002년 8월 숭실대학교 정보과학대학원 석사
2006년 8월 숭실대학교 대학원 컴퓨터학과 박사
2001년~2012년 (주)이엔터프라이즈 대표이사
2012년~현재 숭실대학교 SW특성화대학원 교수
※관심분야 : 소프트웨어공학, 정보보호, 오픈소스소프트웨어

박제원(Jea-Won Park)

2004년 2월 : 숭실대학교 컴퓨터학부(석사)
2011년 2월 : 숭실대학교 컴퓨터학부(박사)
2013년 4월 : 숭실대학교 SW특성화대학원 교수
관심분야 : 소프트웨어테스팅, 클라우드컴퓨팅, 웹서비스, SOA/ESB, 프로젝트관리