*The Korean Journal of Applied Statistics* (2014) DOI: http://dx.doi.org/10.5351/KJAS.2014.27.2.249
**27**(2), 249–261

# Measuring Hadoop Optimality by Lorenz Curve

Woo-Cheol Kim[a] · Changryong Baek[b,1]

[a]Mobigen Co.; [b]Department of Statistics, Sungkyunkwan University

(Received December 18, 2013; Revised February 10, 2014; Accepted March 11, 2014)

**Abstract**

Ever increasing "Big data" can only be effectively processed by parallel computing. Parallel computing refers to a high performance computational method that achieves effectiveness by dividing a big query into smaller subtasks and aggregating results from subtasks to provide an output. However, it is well-known that parallel computing does not achieve scalability which means that performance is improved linearly by adding more computers because it requires a very careful assignment of tasks to each node and collecting results in a timely manner. Hadoop is one of the most successful platforms to attain scalability. In this paper, we propose a measurement for Hadoop optimization by utilizing a Lorenz curve which is a proxy for the inequality of hardware resources. Our proposed index takes into account the intrinsic overhead of Hadoop systems such as CPU, disk I/O and network. Therefore, it also indicates that a given Hadoop can be improved explicitly and in what capacity. Our proposed method is illustrated with experimental data and substantiated by Monte Carlo simulations.

Keywords: Lorenz curve, Big data, Hadoop optimization, Gini coefficient.

## 1. 서론

웹(Web)이 개발된 이후 디지털 형태로 저장되는 데이터의 양은 빠르게 증가하였고 그 데이터들은 하나의 서버에 집중되기 시작했다. 예를 들어 최근 수년간 노트북과 디지털 카메라, 스마트폰 등 디지털 정보를 생산할 수 있는 다양한 기기들의 증가는 곧 디지털 형태로 저장되는 데이터의 양을 기하급수적으로 늘렸다. 또한 이러한 전통적인 정보과학(information Science) 분야뿐만 아니라 생물정보학(bioinformatics) 분야에서 생산해 내는 유전체(genome) 정보와 같이 다양한 분야에서 대용량의 데이터를 디지털로 기록하고 활용하고 있다.

물론 과거에도 다양한 디지털 데이터가 있었다. 하지만, 그 데이터는 크기가 수 메가바이트(mega byte)에서 수 기가바이트(giga byte) 정도로 한 대의 컴퓨터(또는 단일 서버)에서 처리 가능한 데이터의 크기였다. 그러나 최근 트위터나 페이스북에서 수집하고 저장하는 데이터는 수백 테라바이트(tera byte)를 넘어 수 페타바이트(peta byte)에 이르고 있다. 이러한 테라바이트를 넘어 페타바이트에 이르는 대용량 데이터를 빅데이터(big data)라고 하며 그 중요한 특징으로는 단일 컴퓨터를 사용하여 데이터를 저장하기도 처리하기도 힘들다는 것이다. 최근에 이슈가 되고 있는 "빅데이터"는 이러한 대용량 데이터를 자체를 의미하면서 동시에 그 대용량 데이터를 처리하는 방법에 대한 의미도 포함하고 있다.

[1]Corresponding author: Department of Statistics, Sungkyunkwan University, Sungkyunkwan-ro, Jongno-gu, Seoul 110-745, Korea. E-mail: crbaek@skku.edu

빅데이터에 대한 초창기 연구는 데이터의 수집 단계로 페타바이트의 데이터를 어떻게 저장을 하고 데이터베이스(database)화 하는지에 집중되었다. 이는 구글의 GFS (Ghemawat, 2003) 및 아파치 재단의 HDFS (Shafer 등, 2010)를 통해 효과적으로 해결되었으며 NoSQL을 통해 검색 가능한 형태로 데이터베이스를 구축할 수 있게 되었다. 이렇게 데이터를 효과적으로 저장한 이후에는, 대용량의 데이터를 어떻게 빠른 시간 안에 처리할 수 있는지에 대한 연구가 진행되었다. 즉, 데이터의 양은 기하급수적으로 증가한 반면 단일 컴퓨터의 하드웨어 성능은 일정한 수준 이상으로 향상되지 않았다. 이를 해결하기 위해서 다수의 컴퓨터를 활용해서 대용량의 데이터를 처리하는 분산처리 방법이 제안되었지만, 분산 처리와 관련된 가장 중요한 이슈는 사용된 컴퓨터의 개수만큼 처리 속도가 빨라지는 확장성(scalability)이 보장되지 못한다는 점이다. 예를 들어, 다섯 대의 컴퓨터를 병렬연결 하여 처리한다고 할지라도 그 처리 속도는 단일 컴퓨터를 사용했을 때와 비교해 5배 빨라지지 못하는 문제를 가지고 있었다.

이 문제는 전체 연산을 하위 컴퓨터 노드로 효율적으로 잘 분배하고 각 컴퓨터에서 계산된 결과를 효과적으로 합치는 구글의 맵리듀스(MapReduce, Dean과 Ghemawat, 2004) 알고리즘을 통해 해결되었다. 맵리듀스 알고리즘은 맵(map) 단계와 리듀스(Reduce) 단계로 이루어진 알고리즘으로 간략히 설명하면 다음과 같다. 먼저 연산을 행하고 싶은 잡(job)이 있다고 가정하자. 맵리듀스 알고리즘은 마스터 노드(master node)에 의해 통제되는데 먼저 맵(map) 단계에서 마스터 노드의 명령에 의해 수십 개의 하위 슬레이브 노드(slave nodes)로 분배해서 그 데이터를 처리한다. 모든 맵태스크가 성공적으로 수행되었으면 마스터 노드는 리듀스(reduce) 단계를 위해서 수십 개의 하위 작업 노드들을 할당하고 맵태스크의 출력 데이터를 전송받아서 최종적인 결과를 만들어 낸다. 이러한 맵리듀스 알고리즘은 확장성이 보장됨이 여러 연구를 통해 증명되었고 하둡(Hadoop)이란 맵리듀스 알고리즘을 아파치 재단에서 구현한 소프트웨어 플랫폼이다.

최근에는 하둡이라는 플랫폼 자체의 성능을 높일 수 있는 방법에 대한 연구가 활발히 진행되고 있다 (Herodotou와 Babu, 2011; Khoussainova 등, 2012; Jiang 등, 2010; Lee 등, 2012). 하지만, 어떠한 상태가 최적의 상태인지 또 어떠한 의미에서 성능 향상이 이루어졌다고 할지에 대해서는 확립된 정의가 없다. 따라서 본 연구는 하둡 성능 개선을 위한 사전단계로 꼭 필요한 "현재 하둡의 성능을 어떻게 측정할 것인가?"에 대한 문제를 심도 있게 다루고자 한다. 우리는 본 연구를 통해서 하둡 플랫폼의 최적화 상태란 필수 연산부분만 으로 운영되는 상태로 정의하고자 한다. 이를 바탕으로 제거 가능한 불필요한 연산이 포함된 부분을 찾아내고 이를 제거하였을 경우 얼마만큼의 성능 향상이 있을 수 있는지를 통계적인 방법을 사용하여 그 값을 지수화(index)하여 주어진 하둡 시스템의 최적화 정도를 측정하려고 한다.

## 2. 하둡의 최적화 상태 및 하둡 플랫폼의 성능 프로파일링

먼저 본 논문에서는 하둡의 최적화 상태를 주어진 잡을 처리하는데 사용되는 연산에 필요한 필수적인 시간만 이루어진 상태로 정의한다. 맵리듀스 알고리즘에서 살펴보듯이 분산처리 과정은 여러 대의 컴퓨터가 하나로 연결된 커다란 네트워크이다. 최적화된 상태란 이러한 커다란 하둡 시스템이 하드웨어/소프트웨어적인 병목현상(bottleneck)없이 모든 연산 과정이 지체(delay)없이 이루어질 때를 가리킨다. 바꾸어 말하면, 최적화 되지 않은 상태란 하드웨어/소프트웨어적인 통제를 통해서 병목현상 및 지체현상을 줄일 수 있는 상태이다.

따라서, 하둡의 최적화 정도를 측정하기 위해서는 병목현상이나 지체가 발생하는 부분을 프로파일링(profiling)해야 한다. 즉 만약 데이터의 기본 처리 단위인 레코드(record) 별로 읽기(read), 쓰기(write), 계산(processing) 등의 연산을 분리하여 시간을 측정할 수 있다면 어떠한 연산 과정때문에 지체가 발생하는지 알 수 있으므로 하둡의 최적화를 측정할 수 있을 것이다. 즉, 이론적으로는 각 연

산별 레코드별 처리 시간에 대한 분포(distribution)를 정확히 추정해 내고 이러한 분포를 바탕으로 평균 및 분산과 같은 다양한 통계량을 바탕으로 최적화 정도를 계산해 낼 수 있을 것이다. 또한 하둡의 성능에 영향을 주는 공변량(covariates)을 찾아내고 이러한 공변량을 제어함으로서 최적화를 이끌어 내는 보편적인 통계적 방법을 적용할 수 있을 것이다. 하지만, 이러한 세밀한 프로파일링을 하둡에 적용하기는 다음과 같은 이유로 무척 어렵다.

첫째, 하둡에서 성능 모니터링을 위해 자체적으로 제공하는 기초적인 정보는 맵태스크(map task)와 리듀스 태스크(reduce task)의 시작 시간과 종료 시간과 같은 매우 단순한 정보밖에 없어 분포 추정에 적합하지 않다. 스타피쉬(Starfish, Herodotou와 Babu, 2011)라는 좀 더 개선된 프로파일링을 제공해주는 프로그램이 있지만 이 역시 맵태스크와 리듀스 태스크를 구성하는 세부적인 부분 테스크(sub-task) 수준으로 정밀한 분석을 위한 데이터로는 한계가 있다.

둘째, 정확한 프로파일링을 위해서 수집하는 데이터의 세밀함을 무작정 높일 수 없다. 정확한 분포 추정을 위해서는 세밀한 단위의 정확한 관측치가 필요하다. 즉 데이터의 수준(granularity)이 세밀해질수록(fine-grained) 정확한 추정값을 준다. 하지만, 프로파일링에 사용되는 연산은 원래 프로그램에는 필요 없는 연산이므로 좀 더 세밀한 수준에서 진행하는 경우 프로파일링을 위해 더 많은 시간을 소모하게 되므로 하둡 시스템에 커다란 부하가 생긴다.

세번째 이유는 기술적으로 불가한 경우이다. 하둡의 경우에는 대량의 데이터를 작은 데이터 단위인 레코드(record)로 쪼개가면서 처리하는데 기본 연산 역시 읽기, 쓰기, 계산 등의 여러 가지 하부 처리 작업으로 구성되어 있어 이론적으로는 구분할 수 있다. 그러나 실제 실험에서 각 레코드별 그리고 레코드와 관련된 세부 연산별 수행 시간을 프로파일링 하는 것은 기술적으로 쉽지 않다. 또한 각 레코드에 연관된 읽기, 계산, 쓰기 연산을 위한 세부적인 실행 시간은 균일하지 않기 때문에 정확한 측정을 더 어렵게한다.

그 결과 현재 하둡에서 제공하는 프로파일링은 데이터 처리 시간에 대한 분포를 정확하게 추정하기 어렵고 또 매우 세밀한 프로파일링은 기술적으로 불가능하거나, 가능하다 할지라도 시간이 많이 걸리는 연산을 하둡 시스템에 부여하는 단점을 가지고 있다. 본 연구는 앞서 설명한 세밀한 프로파일링의 장점을 이용하기 위해서 단점으로 지적된 문제점을 "각 레코드별 처리 시간은 동일하다"는 가정을 통한 통계적인 추정방법을 통해 극복하려고 한다. 이를 위하여 다음 장에서는 위 가정에 의하여 실제 하둡 시스템에서 얻어진 데이터를 토대로 각 레코드의 처리 시간이 동일한지 검증한다. 그리고 4장에서는 연산에 필수적인 시간과 연산에 필수적이지 않은 부가적인 시간을 로렌츠 커브를 이용하여 분리하고 어떻게 지니계수를 이용하여 계량화하는지를 설명한다.

## 3. 실험 데이터

맵태스크의 입력 데이터의 크기를 $N$이라 하자. 레코드의 크기를 $m$이라고 한다면 한 개의 맵태스크는 $T = N/m$개의 레코드를 처리한다. 앞서 우리는 각 레코드 단위별, 모든 연산 종류별로 시간을 측정하는 것은 비효율적이고 물리적으로도 매우 어렵다고 지적하였다. 따라서 우리는 빠른 시간 안에 측정할 수 있고 분포를 추정하기에 의미 있는 수준의 자료를 얻기 위해서 각 작업 노드에 대해 기본적인 연산 종류를 무시하고 개별 레코드의 처리 결과가 아닌 $T/1000$개의 레코드를, **1버킷(bucket)**이라 정의, 처리하는데 소요되는 시간을 측정하고자 한다.

예를 들어, 맵태스크는 64MB의 입력 데이터를 처리하게 되고 CPU에서 처리하는 레코드의 크기가 100Byte라면 각 태스크는 대략 64000개의 레코드를 처리하게 된다. 우리가 관측한 데이터는 $64000/1000 = 64$개의 레코드를 처리하는데 걸린 시간이다.

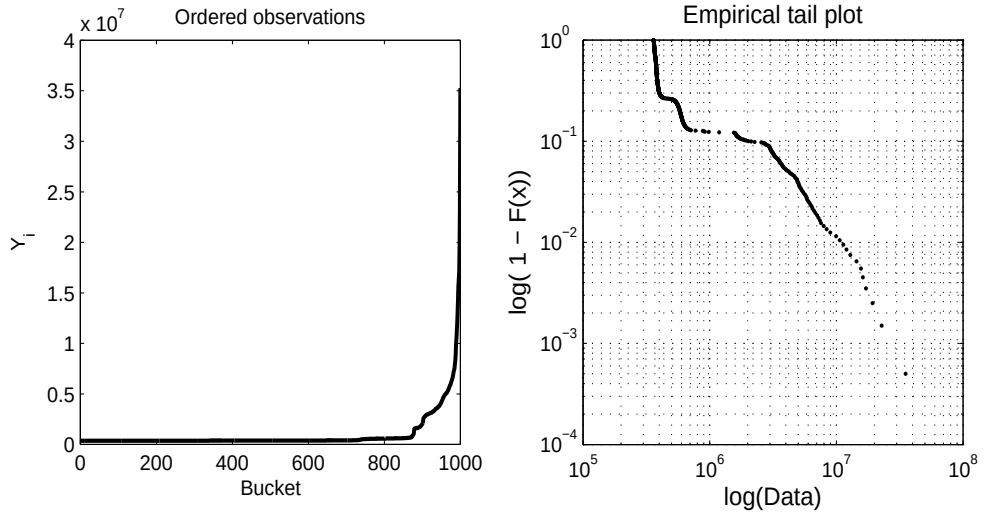


**Figure 3.1.** Bucket processing time (ordered) and corresponding empirical tail plot. It can be observed that bucket processing time is uniform in most of time, but also have few very long processing time. It means that record processing time can be considered as uniform in the optimized Hadoop system. However, in practice, such uniformity is disturbed by power-law tail due to delay/bottleneck in Hadoop system.

실제 본 논문을 위해서 수집된 데이터는 다음과 같은 환경에서 수집되었다. 하둡 클러스터는 총 5대의 서버로 구성되어 있다. 마스터 노드는 1대이고 슬레이브 노드는 4대이다. 각 서버 하드웨어는 4개의 코어로 구성된 CPU와 4GB의 메모리 그리고 1TB 7200RPM의 하드디스크를 가지고 있다. 그리고 5대의 서버는 모두 기가비트(gigabit)의 네트워크로 연결되어 있다. 기타 하둡 설정은 기본값을 사용하였다. 실험을 위해서 사용된 프로그램은 테라정렬(TeraSort)로 이 프로그램의 특징은 레코드의 데이터 크기가 동일하며 따라서 이론적으로 개별 레코드 처리 시간이 동일하다. 이러한 특성은 본 논문에서 전제하고 있는 "레코드를 처리하는 시간은 일정하다"에 가장 적절하다.

Figure 3.1은 위의 방법대로 정의한 태스크에서의 버킷 처리 시간(bucket processing time)에 대한 실험 관측 자료를 보여준다. 왼쪽 그림은 순서통계량을 나타낸다. 대부분의 버킷의 경우 그 처리 시간이 거의 일정함을 볼 수 있고, 몇몇 버킷의 처리 시간은 매우 오래 걸림을 알 수 있다. 즉 우리가 예상하였듯이 버킷 처리 시간은 대부분 일정하지만 몇몇 버킷의 경우 버킷 처리 시간은 매우 길어서 매우 두터운 꼬리를 가짐을 알 수 있다. 오른쪽 로그-로그 경험적 꼬리 분포그림에서 관측되는 직선관계를 통해 두터운 꼬리 분포가 멱함수꼴(파레토 분포형태)임을 알 수 있다.

두터운 꼬리 함수를 가짐은 분산처리의 과정에서 한정된 하드웨어 자원을 효율적으로 분배하지 못하여 병목현상(overhead)이 생김으로써 하둡 시스템에 정체(delay)가 생기기 때문으로 이해할 수 있다. 컴퓨터 성능에 영향을 미치는 가장 중요한 자원은 크게 3가지다. 먼저 가장 중요한 자원은 CPU이다. CPU의 병목현상은 CPU가 가지고 있는 코어의 숫자보다 많은 수의 작업(thread)이 실행되는 경우에 발생한다. 예를 들어 CPU가 동시에 실행 가능한 작업의 수를 의미하는 코어를 2개만 가지고 있는데 동시에 실행되는 작업의 수가 4개라면 2개의 작업은 실행 가능하지만 나머지 2개의 작업은 실행되지 못하고 기다림(waiting)이 발생하거나 준비하는 과정인 문맥 전환(context-switching)이 발생한다. 더 중요한 병목현상은 하드디스크에서 발생한다. 하드디스크 역시 데이터의 읽기/쓰기 능력에 처리한계가 있다. 그 한계를 벗어나는 읽기/쓰기 요청이 들어오면 나중에 들어온 읽기/쓰기 기다려야하고 그 기다리는 시간

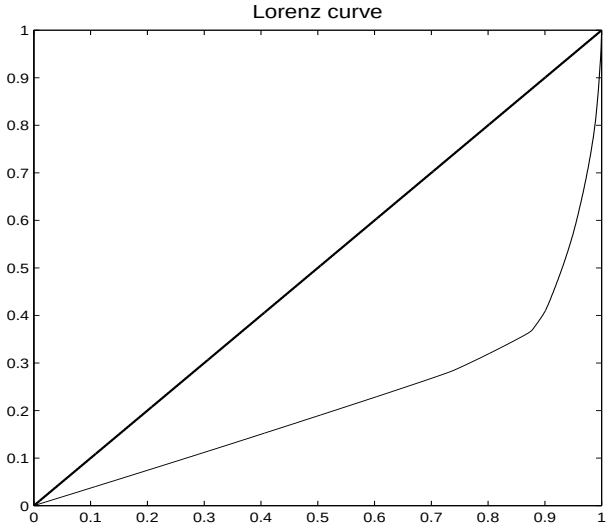


**Figure 4.1.** Lorenz curve for the experimental data in Chapter 3. The $y = x$ curve is a reference line meaning that underline distribution is uniformly distributed.

이 병목현상을 유발한다. 마지막 자원인 네트워크도 하드디스크와 비슷한 이유로 병목현상이 발생한다. 이러한 병목현상은 하둡에서 모두 발생 가능한 것이다. 그리고 하둡은 동시에 처리해야 하는 작업의 수가 많고 데이터의 읽기/쓰기가 많은 작업들을 처리하기 때문에 병목현상을 줄이는 것이 바로 하둡의 성능 최적화의 가장 중요한 부분이다.

요약하자면, 이 장에서는 실험데이터를 통해서 "레코드 처리 시간은 일정하다"는 우리의 가정이 합리적임을 살펴보았다. 또한 긴 처리 시간을 가지는 이유로는 CPU, 디스크 읽기/쓰기, 네트워크를 비롯한 여러 요인에서 발생되는 병목현상 때문임을 주장하였다. 다음 장에는 로렌츠 커브를 이용하여 레코드 처리 시간에 대한 분포를 추정하고 균일한 처리 시간과 멱함수 꼬리를 가지는 처리 시간을 변화점 추정을 통해 분리함으로서 하둡 시스템의 최적화 지수를 산출하는 방법을 소개한다.

## 4. 로렌츠 커브를 이용한 최적화 지수

확률변수 $X_1, \ldots, X_n$을 주어진 태스크에서 1버킷(bucket)의 레코드를 처리하는데 걸리는 시간이라고 하자. 그리고 이 확률변수의 순서통계량을 $Y_i = X_{(i)}$라 정의하자. 먼저, 긴 처리 시간을 가지는 것은 하드웨어 리소스를 효율적으로 사용하지 못한 결과 자원의 불균형 배분으로 인한 병목현상 때문이라고 이해하였다. 이러한 병목현상은 두터운 꼬리모양의 분포로 표현될 수 있고 두터운 꼬리 모형(heavy-tailed distributions) 중 sub-exponential 분포는 다음과 같이 정의 된다.

$$\lim_{x\to\infty} \frac{P(S_n > x)}{P\left(\max\limits_{1\le i\le n} X_i > x\right)} = 1, \quad S_n = X_1 + \cdots + X_n,\ n \ge 1. \tag{4.1}$$

즉, 최대값이 매우 커서 누적합과 비교해서도 거의 비슷한 그러한 분포로 이해할 수 있다 (Embrechts 등, 1997). 따라서, 수식 (4.1)에서 살펴보듯이 두터운 꼬리 모양은 누적합에 급격한 변화가 생김을 알 수 있다.

반면, 하둡 플랫폼이 최적화 되어 있다면 각 버킷을 처리하는 시간은 일정하다. 이 경우 누적합은 순증가하는 $y = x$ 꼴의 형태가 될 것이다. 이러한 누적합의 모양 차이는 하둡의 최적화를 요약해주는 좋은 지표로 자원의 불균형정도를 연구하는데 많이 쓰이는 로렌츠 커브(Lorenz curve)를 통해 요약할 수 있다. 먼저 레코드 처리 시간을 나타내는 순서 통계량 $Y_1, \ldots, Y_n$에 대해서 부분합을 정의하자

$$L_m = \frac{Y_1 + \cdots + Y_m}{Y_1 + \cdots + Y_n} = \frac{S_m}{S_n}, \quad S_m = Y_1 + \cdots + Y_m.$$

그러면 $(m/n, L_m)$의 그래프를 로렌츠 커브라 불리우며 $n \to \infty$일 때 $L(x)$ 로 표현한다 (Embrechts 등, 1997).

로렌츠 커브는 $(0,0)$과 $(1,1)$을 잇는 비감소 함수로서 만약 $Y_i$가 일정한 값을 가진다면 $L_m = m/n$이 되므로 $y = x$의 45도 기울기를 갖는 그래프가 된다. 즉 하드웨어 리소스가 균일하게 배분된다면 로렌츠 커브는 45도 기울기의 직선의 형태를 가지게 된다. 극단적으로 만약 한 개의 최대값 $Y_n$의 값에 의해서 하드웨어 리소스가 점령되어버리게 되면, 로렌츠 커브는 $(0,0), (1,0)$ 그리고 $(1,1)$을 연결하는 계단함수가 된다. 따라서 로렌츠 커브를 통해서 $y = x$에 가까운 그래프인지 아니면 $x = 1$에 가까운 그래프인지를 통해서 자원의 배분이 얼마나 균등히 되고 있는지를 알 수 있다.

Figure 4.1는 3장에서 얻어진 관측 자료를 토대로 로렌츠 커브를 그린 것이다. 그림 중간의 $y = x$는 모든 리소스가 균일하게 배분되는 이상(oracle) 상태를 나타낸다. 관측 자료는 $y = x$에 오른쪽으로 벗어난 형태이므로 일단 자원의 불균형이 존재함을 알 수 있다. 하지만 .9를 경계로 매우 급격한 변화를 보임이 관측되었다. 또한 .9 이전은 곧은 직선관계를 보여 버킷(레코드) 처리 시간은 일정하나 자원의 불균형 때문에 병목현상이 생겨 하둡 플랫폼이 최적화되지 않았음을 알 수 있다.

이러한 불균등의 정도는 $y = x$와 주어진 로렌츠 커브 $L(x)$ 사이의 면적을 통해 계량화 할 수 있으며

$$G(L) = 2\left(.5 - \frac{1}{n}\sum_{m=1}^{n} L_m\right) \approx \frac{\left(\int_0^1 x dx - \int_0^1 L(x)dx\right)}{\int_0^1 x dx}, \tag{4.2}$$

이 지수가 바로 유명한 지니 계수(Gini coefficient)이다 (Yitzhaki와 Schechtman, 2012). 하지만, 지니 계수를 곧바로 하둡 최적화 지수로 사용할 수는 없다. 그 이유는 하드웨어 리소스의 기술적인 제약 때문에, 예를 들어 I/O 캐쉬(cache)에 의한 성능 차이 혹은 운영체제 수준에서 진행되는 스케줄링(scheduling)에 의해서 발생하는 성능차이, 현실에서는 결코 $y = x$와 같은 로렌츠 커브를 얻을 수 없기 때 문이다. 따라서 우리는 현실적으로 우리가 관측할 수 있는 최적화된(ideal) 하둡 플랫폼을 다음과 같이 추정하고자 한다.

먼저, 병목현상이 시작되는 시점에 대한 변화점을 최소제곱합(least squares estimator; LSE) 방식에 의하여 다음과 같이 추정한다 (Bai와 Perron, 1998).

$$\hat{m} = \underset{\omega \le m \le n-\omega}{\operatorname{argmin}} \left\{ \sum_{i=1}^{m} \left(L_i - f_1\left(\frac{i}{n}\right)\right)^2 + \sum_{i=m+1}^{n} \left(L_i - f_2\left(\frac{i}{n}\right)\right)^2 \right\}, \tag{4.3}$$

$f_j(x) = \beta_{j,0} + \beta_{j,1}x + \cdots + \beta_{j,p}x^p$, $j = 1, 2$이고 $\omega$는 회귀 적합을 위해 필요한 최소 데이터 개수를 보장해주는 작은 상수 값이다. 이렇게 추정된 $\hat{m}$에 대해서 로렌츠 커브 $L^*(x)$는 $(m/\hat{m}, S_m/S_{\hat{m}})$으로 주어진다. Figure 4.2는 3장에서 얻어진 실험 자료에 대해서 변화점 추정량 식 (4.3)를 이용하여 병목현상이 시작되는 시점에 대한 변화점을 직선 회귀를 통해 추정한 그림이다. 로렌츠 커브에서 관찰했던 .9근방의 변화점을 LSE 방법이 잘 추정함을 확인할 수 있다.

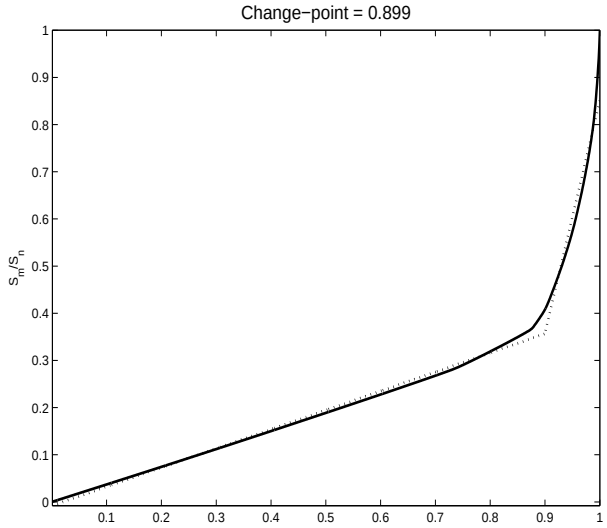


**Figure 4.2.** Estimated curve by LSE is represented in dotted line with Lorenz curve (solid line) for experimental data in Chapter 3. Lorentz curve looks linear for constant processing time. Once it is disturbed by very long processing time, Lorentz curve changes dramatically and LSE change-point estimator successfully detects such change-point.

이러한 변화점이 시작되기 이전의 자료만을 이용하여 새로운 로렌츠 커브 $L^*$를 계산할 수 있고 이렇게 얻어진 로렌츠 커브가 바로 최적화된(ideal) 하둡에 해당한다. 따라서, 주어진 하둡 시스템의 최적화 지수는 $L^*$와 $L$ 두 로렌츠 커브 사이의 면적을 0과 1 사이의 값으로 정규화시켜준 값으로

$$H(L) = \frac{\sum_{i=1}^{\hat{m}} \frac{L_i^*}{\hat{m}} - \sum_{i=1}^{n} \frac{L_i}{n}}{\sum_{i=1}^{n} \frac{L_i}{n}} \approx \frac{\int_0^1 L(x)dx - \int_0^1 L^*(x)dx}{\int_0^1 L^*(x)dx} \tag{4.4}$$

으로 주어진다.

Figure 4.3은 3장의 관측자료에 대해 $H$-index를 계산하는 과정을 보여준다. 맨 아래 실선은 전체 관측자료를 통해 계산한 로렌츠 커브이다. Figure 4.2에서는 이 로렌츠 커브에서 병목현상이 생기는 시점을 LSE 방법을 이용하여 추정하였고 .899로 주어짐을 알았다. 먼저 변화점 .899까지의 데이터만을 이용하여 로렌츠 커브를 구하면 이것이 바로 최적화된(ideal) 하둡에 해당하고 $L^*$(아래서 두 번째 점선)가 얻어진다. $H$-index는 위 두 로렌츠 커브 사이의 면적을 정규화 시킨 값으로 그 값은 .2176/.4289 = .493이다. 이에 대한 해석은 현재 하둡 플랫폼은 최적화된(ideal) 하둡 플랫폼과 비교해서 100(1 − .493) = 50.74%의 효율을 지님을 뜻하며, 병목현상을 없앰으로서 1/.5074 = 1.971배의 성능 개선을 할 수 있음을 의미한다.

또한 만약 다른 두 개의 하둡 플랫폼 $A$와 $B$에 대해서 최적화 정도를 절대적으로 비교하기 위해서는 상대 거리(relative difference)를 이용할 수 있으며

$$\mathrm{RD}\left(L^A, L^B\right) = \left|\frac{1}{n}\sum_{i=1}^{n} L_i^A - \frac{1}{n}\sum_{i=1}^{n} L_i^B\right| \approx \left|\int_0^1 L^A(x)dx - \int_0^1 L^B(x)dx\right|$$

으로 주어진다.

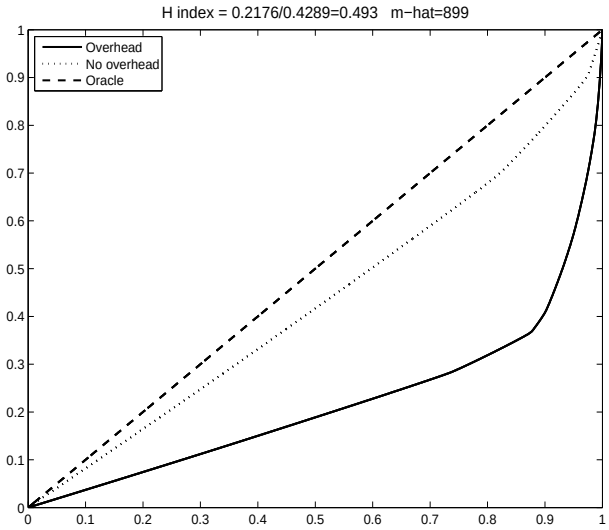


**Figure 4.3.** H-index for a task. Solid line represents Lorentz curve for a bucket processing time. Dotted line represents Lorentz curve without overhead, that is for ideal Hadoop system. Dashed line represents Lorentz curve for oracle Hadoop system, that is $y = x$ curve, and it means that all processing time is constant and observed without measurement errors. A (normalized) area between solid and dotted Lorentz curves, corresponding to current Hadoop and ideal Hadoop, respectively, is our proposed H-index.

## 5. 주어진 잡(job)의 최적화 지수

하둡 시스템에 입력된 잡(job)은 맵 단계에서 하위 태스크로 분배되어 연산을 수행한다. 그리고 하위 태스크에서의 최적화 정도를 로렌츠 커브에 기반한 $H$-index를 계산하여 계량화 할 수 있음을 앞 장에서 논의하였다. 이 장에서는 잡 전체의 최적화 정도를 어떻게 측정할 수 있는지 논의한다.

가장 먼저 생각할 수 있는 직관적인 방법은 각 태스크별 처리 시간을 통합(pooling)하여 $H$-index를 구하는 방법이다. 하지만 이렇게 할 경우, 데이터의 개수가 매우 커서 최적화 지수를 계산하는데 많은 추가적 시간이 걸리므로 실용적이지 못하다. 또한 각 태스크간 오차(between task error) 역시 고려해야 한다. 따라서 우리는 작은 샘플 개수를 가지는 태스크에서의 $H$-index를 이용하여 전체 잡(job)의 최적화 정도를 추정하고자 한다.

$H$-index의 핵심은 처리 시간이 긴 두터운 꼬리의 정보를 최대한 이용하는 것이다. 따라서 여기에서도 우리는 두터운 꼬리의 정보를 일어버리지 않는 가중 평균 방법을 사용하여 주어진 잡(job)에 대한 최적화 정도를 추정한다. 구체적으로

$$H_{job} = \sum_{t=1}^{N} w_t H_t, \quad w_t = \frac{P_t}{\sum_{t=1}^{N} P_t}, \tag{5.1}$$

$H_t$는 $t$번째 노드에서 처리된 태스크의 최적화 지수이며 $P_t$는 $t$번째 태스크의 처리 시간으로 하둡 기본 플래폼에서 제공된다. 따라서 잡에 대한 최적화 지수는 각각의 태스크를 처리하는데 걸린 시간에 비례한 가중평균을 통해 얻어진다.

Figure 5.1은 주어진 job에 대한 실험 자료를 통해 얻어진 60개의 태스크별 $H$-index 및 가중치를 보여

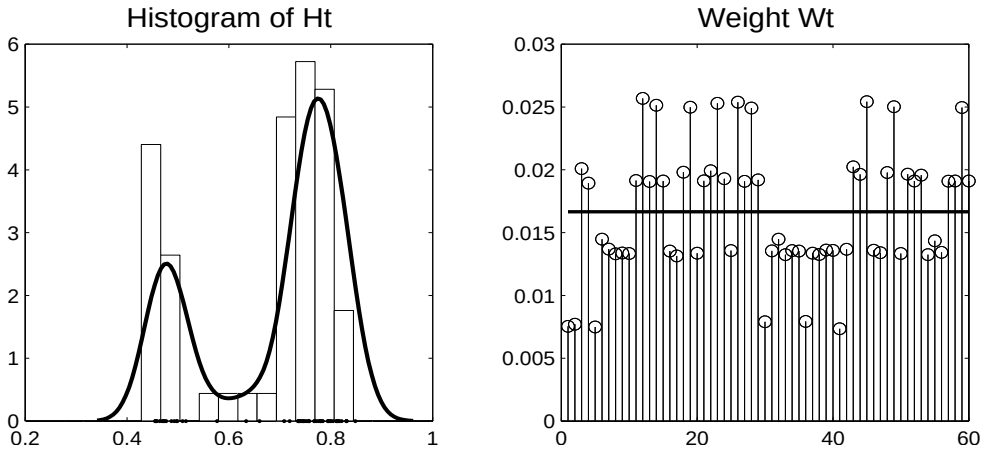


**Figure 5.1.** Left panel shows $H$-indexes calculated for 60 tasks in a job. It can be observed that there are some variations between tasks. Right panel shows its corresponding weights to calculate $H_{job}$ based on the task processing time. A horizontal solid line is 1/60, a uniform weight for each task.

준다. 왼쪽 패널은 히스토그램 및 커널 밀도 함수(실선)을 보여주고 있으며 이봉분포(bimodal distribution)를 보임을 통해서 $H$-index의 태스크간 변동이 존재함을 알 수 있다. 오른쪽은 각 태스크를 처리하는데 소요된 시간에 따른 가중치를 보여주며 중간의 수평 직선은 1/60 즉 균등 가중치를 나타낸다. 가중 평균을 통해 얻어진 $H_{job}$은 .7186이다. 단순 평균은 .69 그리고 중간값은 .7479로 우리가 제안한 가중 평균은 각 태스크에 기반한 $H$-index의 33% 분위수에 해당한다.

## 6. 모의 실험 결과

이 논문에서 사용된 로렌츠 커브를 이용한 $H$-index의 유용성을 알아보기 위해서 다음과 같은 몬테카를로(Monte Carlo) 모의실험을 실행하였다. 먼저 1 버킷의 레코드를 처리하는데 걸린 시간은 3장에서 관측한 실험 자료에 기반 하여 다음과 같은 파레토 혼합분포로 생성하였다.

$$X_i \sim .9\,\text{Pareto}(4, .1) + .1\,\text{Pareto}(1.7, 1), \tag{6.1}$$

여기에서 Pareto($\alpha$, $c$)는 파레토 분포로서 누적분포함수는

$$F(x) = 1 - \left(\frac{x}{c}\right)^{-\alpha}, \quad x \geq c \tag{6.2}$$

로 주어진다. 전자인 파레토$(4, .1)$ 분포는 병목현상이 없는 경우 버킷 처리 시간을 나타낸다. 각 버킷은 처리 시간이 일정하다는 가정에 제어할 수 없는 관측 오차를 더하여 파레토 분포를 생각하였으며 그 꼬리 지수가 4로 (4차 적률이 유한함) 두터운 꼬리를 가지지는 않는다. 반면 두 번째 파레토$(1.7, 1)$ 분포는 그 꼬리 지수가 1.7로 평균은 존재하지만 분산이 존재하지 않는다. 이는 곧 병목현상으로 인해 생기는 두터운 꼬리 분포를 반영하기 위함이며 1.7이라는 꼬리 지수는 3장의 관측 자료를 토대로 힐 추정량(Hill estimator)을 통해 얻어졌다.

따라서 변화점 추정의 경우 혼합 분포의 비율이 9 : 1이고 두 파레토 분포가 서로 멀게 떨어져 있음을 고려하면 변화점은 900에 가깝게 주어짐을 알 수 있으며, $H$-index의 경우 병목현상이 없는 Pareto$(4, .1)$

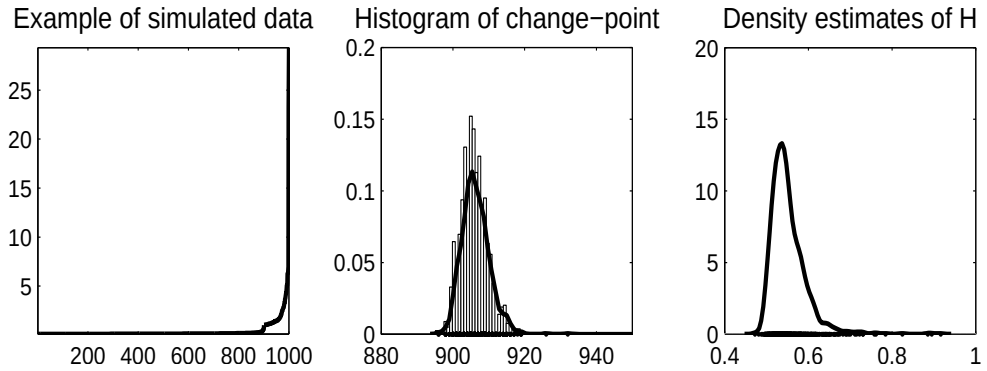


**Figure 6.1.** Monte Carlo simulation results for Pareto mixture in (6.1) to calculate $H$-index. Parameters are chosen to mimic the experimental data in Chapter 3. Left panel shows one realization and it can be observed that it resembles Figure 3.1 up to some scale factor. Middle panel shows the histogram of change-point estimator with the true value around 900 and right panel is the density of $H$-index where the true value is .5778.

분포와 혼합분포 식 (6.1)에서 계산한 로렌츠 커브 사이의 면적을 수치 적분을 통해 계산한 결과 .5778로 주어졌다. 이는 곧 90%에 가까운 버킷의 경우 처리 시간이 거의 일정하고 10%에 가까운 버킷은 병목현상으로 인해서 처리 시간이 매우 긴 분포를 따른다고 해석할 수 있다. 이론적인 $H$-index를 통해서 우리는 버킷 처리 분포가 식 (6.1)를 따르는 하둡 플랫폼의 최적화 지수는 .5778로 최적화된 상태가 아니며 최적화(병목현상을 없앰)을 통해서 $1/(1-.5778) = 2.368$배의 성능을 개선(자원의 불균형 해소) 할 수 있음을 알 수 있다.

Figure 6.1은 우리가 제안한 방법을 통해 $H$-index값을 추정하였을 때의 결과로 1000번의 반복을 통하여 얻어진 결과이다. 왼쪽 그림을 통해서 파레토 혼합 분포 식 (6.1)이 3장에서 관측한 실험 데이터와 비슷한 형태를 가짐을 확인할 수 있다 (Figure 3.1과 비교). 중간 그림은 변화점 추정으로 그 평균은 906으로 참값 900에 매우 가까움을 확인할 수 있으며, $H$-index의 경우도 평균 .5525, 중간값 .5433으로 이론적인 계산 값인 .5778과 상당히 가까움을 확인할 수 있었다. 이는 우리가 제안한 방법이 모의실험을 통해서 참값에 잘 추정됨을 알려준다.

두 번째 시뮬레이션은 우리가 전제한 모든 버킷(레코드) 처리 시간이 동일하다는 전제에서 많이 벗어난 경우를 고려하여 다음의 혼합 분포를 사용하였다.

$$X_i \sim .9\mathcal{N}(.5, .4) + .1\,\mathrm{Pareto}(1.7, 1). \tag{6.3}$$

즉, 레코드 처리 시간은 정규분포를 통해 재현하였고 이러한 경우라 할지라도 병목현상에 의한 두터운 꼬리는 최적화 되지 않은 하둡에서는 불가피한 것으로 판단하여 파레토 분포를 사용하였다. 그 결과는 Figure 6.2과 같다. 먼저, 왼쪽 패널에서 보듯이 버킷 처리 시간이 일정하지 못함을 알 수 있고 두터운 꼬리 분포의 영향으로 매우 큰 값들 역시 관측됨을 알 수 있다. 변화점 추정의 경우 참값인 900보다 약간 큰 값을 추정하는 경향을 보였다. 하지만, $H$-index의 경우 평균이 .144로 참값인 .1409에 가깝게 잘 추정됨을 볼 수 있다. $H$-index가 앞선 파레토 혼합분포의 경우와 비교하여 작게 나온 것은 레코드 처리 시간이 균일하지 않기 때문에 최적화를 통해 개선되는 부분 역시 앞선 경우보다 크지 않음을 반영한 결과로 볼 수 있다. 이 시뮬레이션 결과는 레코드 처리 시간이 균일하다는 전제에서 조금 벗어나더라도 본 논문에서 제안한 $H$-index가 유용함을 보여준다.

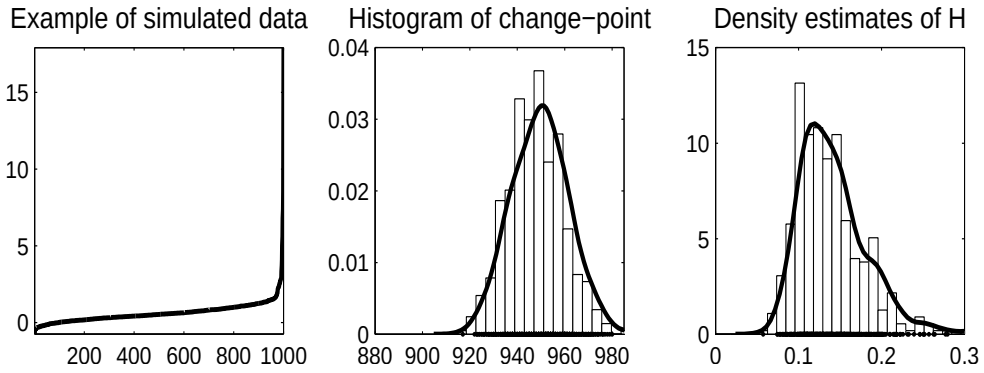


**Figure 6.2.** Monte Carlo simulation results for Normal-Pareto mixture in (6.3) to calculate $H$-index. Left panel shows one realization and it can be observed that processing time is away from constant, but still power-law tailed. Middle panel shows the histogram of change-point estimator where the true value is around 900 and right panel is the density of $H$-index where the true value is .143.

## 7. 결론

이 논문에서는 최근 뜨거운 관심을 받는 "빅데이터"의 처리에 핵심인 분산처리를 가능케 하는 하둡 플랫폼의 최적화 정도를 계량화하는 방법을 고려하였다. 제안한 방법은 하둡 플랫폼의 병목현상이 하드웨어 리소스들을 효과적으로 사용하지 못함에 따른다는 사실에 착안하여 자원의 불균형 정도를 나타내는 로렌츠 커브를 이용하였다. 하지만, 이미 개발된 로렌츠 커브를 사용하여 불균형을 재는 지니 계수를 직접 사용할 수는 없었다. 그 근본적인 이유는 실제 하둡 시스템에서는 우리가 통제할 수는 없는 물리적 시스템에 기인한 오차가 존재하지만, 이러한 오차를 정확히 추정하기 매우 어렵기 때문이다. 설사 가능하다 할지라도 정확한 추정을 위해 필요한 매우 세밀한 프로파일링, 개별 레코드에 대한 정보,를 추출하기에는 그 소모되는 시간이 매우 방대하여 오히려 병목현상을 초래하기 때문이다. 따라서 이 논문에서는 개별 레코드가 아닌 1버킷을 처리하는데 걸리는 시간을 측정하고 병목현상의 시작점을 변화점 추정량으로 추정한 뒤 변화점 전/후의 로렌츠 커브의 차이를 통하여 최적화 지수를 산출하였다. 또 실험 데이터 및 시뮬레이션을 통해 제안한 방법의 유용성을 검토하였다. 우리가 제안한 최적화 지수는 하둡 플랫폼이 얼마만큼의 성능 향상이 가능하며, 또 그에 따른 비용 분석(cost analysis)을 연구하는데 필요한 기초 자료를 제공한다는 점에서 매우 유용하다. 예를 들어 $H$-index를 토대로 Hadoop에 관여하는 많은 내부 파라미터들 값들의 최적화값을 추정할 수 있고, SSD 드라이브 사용과 같은 하드웨어적의 개선이 얼마만큼의 성능 향상을을 초래하는지 계량화할 수 있으며, 궁극적으로 하둡 시스템을 통해 빅데이터를 분석하였을 때 얼마나 많은 시간이 소요되는지에 대한 대기 시간(waiting time) 및 가격결정(pricing policy) 등의 연구에 매우 중요한 실마리를 제공하였다는 점에서 큰 의의를 찾을 수 있다.

## References

Bai, J. and Perron, P. (1998). Estimating and testing linear models with multiple structural changes, *Econometrica*, **66**, 47–78.

Dean, J. and Ghemawat, S. (2004). MapReduce: simplified data processing on large clusters, In Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 10.

Embrechts, P., Klüppelberg, C. and Mikosch, T. (1997). *Modelling Extremal Events: For Insurance and Finance*, Springer.

Ghemawat, S., Gobioff, H. and Leung, S.-T. (2003). The Google file system, In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, 29–43.

Herodotou, H. and Babu, S. (2011). Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs, In *Proceedings of the VLDB Endowment*, **4**, 1111–1122.

Jiang, D., Ooi, B. C., Shi, L. and Wu, S. (2010). The performance of MapReduce: an in-depth study, In *Proceedings of the VLDB Endowment*, **3**, 472–483.

Khoussainova, N., Balazinska, M. and Suciu, D. (2012). PerfXplain: debugging MapReduce job performance, In *Proceedings of the VLDB Endowment*, **5**, 598–609.

Lee, K.-H., Lee, Y.-J., Choi, H., Chung, Y. D. and Moon, B. (2012). Parallel data processing with MapReduce: A survey, *SIGMOD Record*, **40**, 11–20.

Shafer, J., Rixner, S. and Cox, A. L. (2010). The Hadoop Distributed Filesystem: Balancing Portability and Performance, In *Proceedings of 2010 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 122–133.

Yitzhaki, S. and Schechtman, E. (2012). *The Gini Methodology: A Primer on a Statistical Methodology*, Springer series in statistics, Springer.

# 로렌츠 커브를 이용한 하둡 플랫폼의 최적화 지수

김우철[a] · 백창룡[b,1]

[a]모비젠, [b]성균관대학교 통계학과

(2013년 12월 18일 접수, 2014년 2월 10일 수정, 2014년 3월 11일 채택)

**요 약**

최근 큰 관심을 받는 빅데이터는 분산처리를 통해서만 효과적으로 처리할 수 있다. 분산처리란 주어진 쿼리를 여러 대의 컴퓨터로 분할하고 각 분할된 데이터의 계산 결과를 취합하는 과정으로, 주어진 하드웨어 리소스를 효과적으로 최대한 사용하는 것이 중요하다. 하둡은 이러한 분산처리를 가능하게 하는 플랫폼 중의 하나로 분산처리에 사용된 컴퓨터의 개수만큼 성능 향상을 기대할 수 있는 확장성을 최대한 보장하는 매우 성공적인 플랫폼이다. 이 논문에서는 하둡 플랫폼이 얼마나 최적화 되어있는지에 대한 객관적이고 계량적인 지수를 제공함으로써 주어진 하둡 플랫폼의 효율성을 측정한다. 방법론적으로는 로렌츠 커브를 이용하여 하드웨어 리소스들이 얼마나 잘 균등히 배분되어 있는지 살펴보고 CPU, 디스크 일기/쓰기 및 네트워크 병목현상에 따른 비용을 감안한 최적화된 로렌츠 커브를 찾음으로써 최적화 지수를 산출한다. 바꾸어 말하면, 이러한 최적화 지수는 주어진 하둡 플랫폼이 얼마만큼의 성능 향상이 가능한지 알려주는 척도로 오랜 시간을 필요로 하는 빅테이터의 처리 속도 개선을 위한 중요한 정보를 제공한다. 실험 자료 및 모의실험을 통해 본 논문에서 제안된 방법을 검증하였다.

주요용어: 로렌츠 커브, 빅데이터, 하둡, 지니 계수.

[1]교신저자: (110-745) 서울특별시 종로구 성균관로 25-2, 성균관대학교 통계학과. E-mail: crbaek@skku.edu