분산 하둡 시스템의 성능 비교 분석

배병진* · 김영주** · 김영국***

*한국기계연구원, **한국전자통신연구원, ***충남대학교 컴퓨터공학과

Performance Analysis of Distributed Hadoop Systems

Byoung-Jin Bae* · Young-Joo Kim** · Young-Kuk Kim***

*Korea Institute of Machinery & Materials, **Electronics and Telecommunications Research Institute,

***Dept. of Computer Science & Engineering, Chungnam National University

E-mail : bjbae@kimm.re.kr, kr.yjkim@etri.re.kr, ykim@cnu.ac.kr

요 약

오늘날 급증하는 빅데이터를 효율적으로 관리하기 위해 오픈소스인 하둡을 많이 사용한다. 하둡은 분산 파일 처리 시스템인 HDFS(Hadoop Distributed File System)와 분산 병렬 처리 시스템인 맵리듀스(MapReduce)로 구성되어 있다. 하둡의 맵리듀스 프레임워크에서는 빅데이터를 HDFS에서 읽어들이고 분석 처리된 결과를 다시 HDFS에 쓴다. 이러한 분산 병렬 처리 방식은 하둡 버전에 따라 다른 시스템 구조를 가진다. 따라서 본 논문에서는 하둡 버전에 따른 빅데이터 처리 시에 동작하는 하둡 시스템들의 내부 성능을 비교 분석한다. 이를 위해서 하둡 시스템을 감시할 수 있는 방법을 고안하여 내부적으로 생성되는 프로세스 및 스레드들과 변수들의 발생빈도를 측정하여 분석 지표로 사용한다.

ABSTRACT

Nowadays open-source hadoop systems have been using widely to efficiently manage a fast-growing big data. Hadoop systems consist of distributed file processing system called HDFS (Hadoop Distributed File System) and distributed parallel processing system called MapReduce. The MapReduce reads and processes big data from HDFS and then processed results are written in HDFS again by the MapReduce. Such a processing method has different system structure respectively according to hadoop version. Therefore, this paper shows analysis results for performance of hadoop systems. For this, we devise a way which monitors hadoop systems and measure occurrence frequency of processes, threads, and variables generated in hadoop system itself using the devised way. So, by using the measured results as analysis indicator, we help the indicator predict inner performance of hadoop systems.

키워드

하둡 시스템, 빅데이터, HDFS, MapReduce

Ⅰ. 서 론

모바일 기기 보급의 확산에 따른 소셜 네트워크 서비스 이용과 실시간 데이터 처리 및 수집을 위한 클라우드 서비스의 증가, 기업 정보시스템의 고도화 등으로 인하여 빅데이터가 기하급수적으로 발생되고 있다[1]. 이러한 빅데이터를 분석하고 활용하기 위해서 오프 소스인 하둡(Hadoop)을 많이 사용하고 있다.

하둡은 클러스터 컴퓨팅 환경에서 맵리듀스 프레임워크[2]와 하둡 분산 파일 시스템(HDFS, Hadoop Distributed File System)을 사용함으로써 빅데이터를 효율적으로 분산 처리 및 저장 관리한다[3]. 하둡의 맵리듀스 프레임워크를 이용한 분산 처리 방식은 하둡 버전에 따라 다른 시스템 구조로 구성되어 있다[4].

본 논문에서는 맵리듀스 분산 처리 구조가 하둡 버전에 따른 다른 시스템 구조인 것을 경험적

방법으로 검증하고, 빅데이터 처리에서 하둡 프레임워크를 감시하며 발생되는 스레드와 변수의 발생빈도를 분석지표로 사용하여 하둡 시스템의 내부 성능을 비교 분석한다.

Ⅱ. 비교 분석

1. 하둡 시스템 구성

그림 1과 그림 2와 같이 하둡 1.2.2와 하둡 2.2.0의 맵리듀스 분산 처리 구조는 서로 다르게 구성되어 있다.

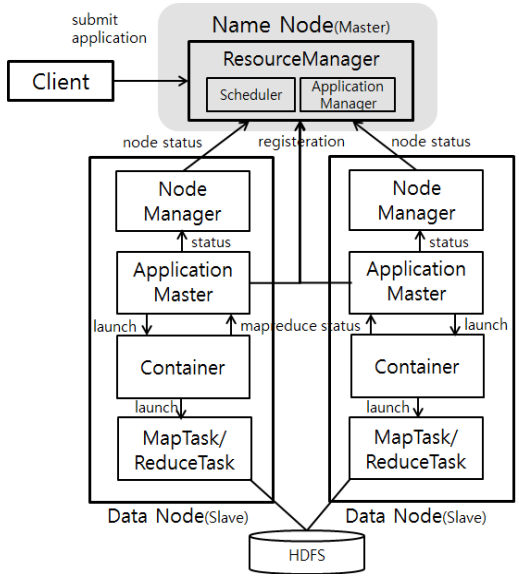

그림 1. 하둡 1.2.2 맵리듀스 분산 처리 구조

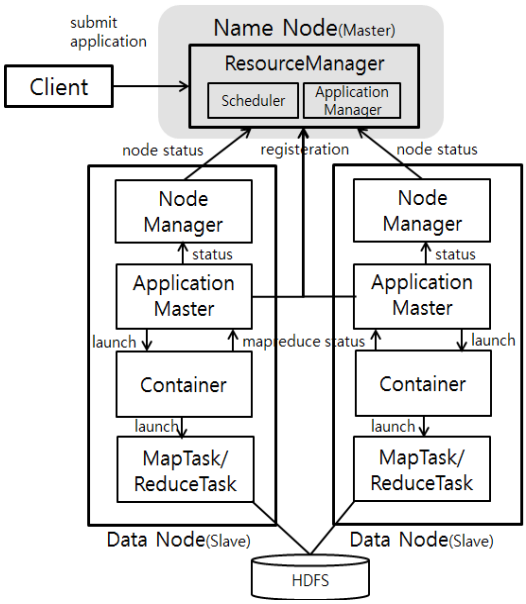

그림 2 하둡 2.2.0 맵리듀스 분산 처리 구조

그림 1에서 하둡 1.2.2는 클라이언트가 제출한 잡(Job)을 잡 트래커(JobTracker)에서는 실행계획을 수행한다. 즉, 잡을 큐에 저장 및 관리하고 잡 초기화 과정을 실행한다. 또한, 분할된 잡을 실행하기 위해 맵과 리듀스 태스크를 생성한다. 태스크 트래커(TaskTracker)는 잡 트래커로부터 할당받은 맵과 리듀스 태스크를 실행하고, 하트비트를 통해 태스크 상태 정보를 잡 트래커에게 전송한다.

그림 2에서 하둡 2.2.0는 클라이언트가 잡에 대한 응용을 리소스 매니저(ResourceManager)에게 제출한다. 하둡 클러스터의 마스터 역할을 하며 1대의 서버에서 실행되는 리소스 매니저는 스케줄러(Scheduler)와 애플리케이션 매니저(ApplicationManager)으로 구성되어 있다. 스케줄러는 애플리케이션의 상태 변화를 추적하거나 모니터링을 하지 않는 순수 스케줄러로써, 애플리케이션의 리소스 요청 처리 후 할당하는 역할을 한다. 애플리케이션 매니저는 클라이언트에서 수행한 잡 제출을 받아 새로운 애플리케이션 식별자를 부여하고 스케줄러에게 잡 요청을 전달한다. 또한, 애플리케이션 마스터를 실행하기 위해 컨테이너(Container)의 리소스 할당을 처리하고, 컨테이너가 중지되면 재시작을 수행한다.

하둡 2.2.0에서는 하둡 1.2.2의 잡 트래커가 광범위한 클러스터 컴퓨팅 환경에서 맵리듀스를 실행할 때에 발생되는 병목현상을 해결하기 위해서, 리소스 관리와 잡 스케줄링 및 모니터링의 기능으로 분리하였다. 이러한 분리된 기능을 수행하는 프로세스는 리소스 매니저, 노드매니저, 애플리케이션 마스터 등이 있다.

2. 실험 시나리오

하둡의 성능 비교 분석을 위해서 표 1과 같이 3대의 컴퓨터를 이용하여 2개의 하둡 버전 (하둡 1.2.2와 하둡 2.2.0)을 설치하였고, 하둡 모드는 멀티 노드 클러스터로 구축하였다. 실험 범위는 그림 1과 그림 2에서 회색 영역으로 표시된 하둡 1.2.2의 잡 트래커 프로세스와 하둡 2.2.0의 리소스 매니저 프로세스를 대상으로 한다. 실험 방법은 실행하는 하둡 내부 구조를 파악하기 위해 스레드와 변수의 발생빈도를 측정하며 이를 위해서 상위 레벨의 자바 API로 구성된 JDI (Java Debug Interface)[5]를 이용한다. 그리고 맵리듀스 프로그램은 하둡에서 제공해주는 wordcount를 사용한다. 실험에 사용한 빅데이터는 위키미디어에서 제공하는 4개의 공인된 범용 데이터(User group assignments(400KB), List of all page titles(621MB), Wiki media/files usage records(2.2GB), Wiki external URL link records(11GB))를 사용한다[6].

표 1. 하둡 멀티 노드 클러스터 환경

운영체제	하둡		노드명	CPU	RAM (GB)
	버전	모드			
우분투 12.04 LTS	1.2.2 and 2.2.0	Multi cluster mode	Name Node	Xeon 3.1GHz×4	8
			Data Node1	Pentium4 3.0GHz	3
			Data Node2	E4500 2.2GHz×2	2

3. 실험 결과

감시를 하지 않는 하둡(Non-Monitoring Hadoop: NMH)과 JDI를 이용한 감시 하둡(JDI-Monitoring Hadoop: JMH) 환경에서 실행한 맵 리듀스 프로그램은 네임노드(마스터노드)의 콘솔에서 wordcount를 실행하였다.

맵리듀스 처리의 실행 시간은 태스크 진행률과 시간 등을 콘솔에 출력하는 하둡의 기본 기능을 이용하여 측정하였다.

JDI는 하둡 시스템 시작시에 생성되는 프로세스 중에서 하둡 1.2.2의 잡 트래커와 하둡 2.2.0의 리소스 매니저 프로세스와 함께 시작된다. 즉, 두 프로세스가 맵과 리듀스 태스크 등을 수행할 때 JDI에서 제공하는 각 이벤트 메소드에서 변수의 접근 횟수와 스레드의 발생 수를 측정한다. JDI의 이벤트는 상위 레벨의 자바 API이므로, 맵 리듀스 처리 시간에 영향을 미친다. 따라서, 맵 리듀스 태스크와 관련성이 낮고, 불필요한 시간을 감소시키게 위해 일부 자바 패키지와 클래스는 JDI 감시 대상에서 제외하였다. 제외한 패키지와 클래스로는 java.*, javax.*, com.sun.*, org.xml.*, org.apache.log4j.*,org.apache.hadoop.ipc.*, org.apache.commons.* 등이다.

NMH과 JMH에서 입력 데이터를 맵 리듀스로 분석 처리한 실행 시간은 표 2와 같다. 하둡 1.2.2와 하둡 2.2.0의 NMH 및 JMH는 입력 데이터의 크기가 클수록 시간이 증가하였고 같은 크기의 데이터에 대한 시간은 하둡 1.2.2에 비해서 하둡 2.2.0이 다소 소요되었다. 한편, 하둡 2.2.0 JMH가 2.2GB와 11GB의 데이터에 대해서 맵 태스크를 수행 중에 컨테이너 시작 오류 및 네임노드와 데이터노드와의 소켓 통신시 시간 초과 오류가 순차적으로 발생하였다. 이러한 오류의 발생 원인은 하둡 2.2.0 JMH에서 하둡 1.2.2 JMH보다 많은 클래스 감시로 인하여 추적파일이 커짐에 따라 맵 태스크의 처리에 영향을 준 것으로 판단하였다. 그 영향중의 하나가 맵 태스크의 처리 지연 시간으로 유추할 수 있다. 향후에 이 문제에 대해서 파악할 예정이다.

표 2. 하둡 1.2.2와 하둡 2.2.0의 맵리듀스 분산 처리 시간 비교 (단위 h:시간, m:분, s: 초)

데이터 크기	하둡 1.2.2		하둡 2.2.0		하둡 2.2.0 - 하둡 1.2.2	
	NMH	JMH	NMH	JMH	NMH	JMH
400 KB	21s	8m 37s	23s	45m 3s	2s	36m 26s
621 MB	4m 22s	14m 38s	22m 55s	30m 59s	18m 33s	16m 21s
2.2 GB	8m 2s	22m 28s	10m 30s	N/A	2m 28s	N/A
11 GB	53m 53s	1h 34m 36s	57m 52s	N/A	3m 59s	N/A

표 3은 하둡 1.2.2 JMH에서 발생된 스레드 발생개수와 Static 변수 및 Non-static 변수의 접근 횟수를 기준으로 하여 하둡 2.2.0 JMH와의 발생 비율을 비교한 표이다. 각 셀의 계산식은 “하둡 2.2.0JMH 발생 수/하둡1.2.2JMH 발생 수*100” 이다. 계산값이 100%일 때 하둡 1.2.2 JMH와 하둡 2.2.0 JMH의 스레드 및 변수의 발생비율은 동일하며, 100%보다 작을 때는 하둡 1.2.2 JMH에서 발생비율이 높고 100%보다 클 때는 하둡 2.2.0 JMH에서 발생비율이 높다는 것을 의미한다. 수식으로 간략하게 표현하면 아래와 같다.

계산값 : Z, 하둡 1.2.2 JMH : X, 하둡 2.2.0 JMH : Y

$$Z = \begin{cases} = 100, & X = Y \text{ 일 때} \\ < 100, & X > Y \text{ 일 때} \\ > 100, & X < Y \text{ 일 때} \end{cases}$$

표 3. 하둡 1.2.2 JMH와 하둡 2.2.0 JMH 맵리듀스 분산 처리 시 생성되는 스레드 수, Staic 변수와 Non-Staitc 변수의 접근 횟수 비교(단위 %)

데이터 크기	스레드	Static 변수		Non-Static 변수	
		Read	Write	Read	Write
400 KB	55.4%	52.6%	438.6%	7.3%	33.9%
621 MB	52.6%	91.6%	440.4%	12.4%	57.1%
2.2 GB	N/A	N/A	N/A	N/A	N/A
11 GB	N/A	N/A	N/A	N/A	N/A

실험결과를 요약하면, 하둡 1.2.2가 하둡 2.2.0 보다 시간적 성능면에서는 우수하다는 것을 알 수 있다. 하지만 이 결과는 하둡 전체 시스템의 성능결과를 보인 것이 아니라, 하둡의 네임노드(마스터노드)에 대한 성능이다. 또한, JDI를 이용한 하둡 시스템를 감시하는데 걸리는 시간에 있어서 하둡 2.2.0는 입력 데이터가 2.2GB 이상에서는 측정을 할 수 없는 상태였다. 이것은 향후에 해결해야 될 문제이다. 그리고 표 3과 같이, 데이터의 크기가 400KB와 621MB 인 경우에 스레드 생성비율, Non-Static 변수의 접근사건들의 생성비율, 그리고 Static 변수의 Read 접근사건의 생성비율이 하둡 1.2.2가 더 커다는 것을 각각의 수치로써 보여주고 있다. 하지만 표 2에서 하둡 1.2.2가 수행 속도가 더 빠른 이유는 Static 변수의 Write 접근사건이 하둡 2.2.0가 400% 이상 (즉 4배 이상) 증가했기 때문이다. 즉, I/O 발생 빈도수가 더 많다는 의미이다. 이러한 이유는 하둡 1.2.2와 하둡 2.2.0의 내부 구조가 달라졌고 처리하는 방식이 달라졌다는 것을 의미한다. 데이터 2.2GB와 11GB의 계산값은 NMH과 JMH에서 입력 데이터를 맵 리듀스로 실행한 시간 측정 실험과 같은 오류가 발생되므로 표 2와 동일하게 표기하였다. 본 실험에서 알 수 있는 중요한 사실은 하둡 2.2.0는 Static 변수의 Write 접근사건이 하둡 1.2.0 보다 4배 이상 발생하므로 I/O에 대한 부담이 있는 것으로 판단된다.

Ⅲ. 결론 및 향후 연구 방향

본 논문에서는 하둡 1.2.2의 잡 트래커와 하둡 2.2.0 리소스매니저 프로세스를 대상으로 성능 평가를 하였다. 입력 데이터의 크기가 증가할수록 맵리듀스를 분석 처리하는 시간도 증가하였고, NMH와 JMH는 하둡 1.2.2가 하둡 2.2.0 보다 네임노드(마스터노드)에서 성능이 다소 뛰어남을 확인하였다. 이러한 결과는 하둡 1.2.2에서 네임노드(마스터노드)의 실행과 처리 부담이 낮은 것으로 판단된다. 그러나, 본 연구에서는 하둡 전체 시스템에 대해서 성능평가를 하지 않았으므로 하둡 1.2.2가 하둡 2.2.0보다 성능이 우수하다는 판단을 내리기는 쉽지 않다.

향후 연구에서는 데이터 노드와 노드 매니저 등을 포함한 하둡 전체 시스템을 비교분석을 할 것이다. 또한, 빅데이터에 대한 JMH의 수행 속도를 개선하여, 하둡 시스템을 투명하게 감시하고 시각적으로 확인 가능한 도구를 연구할 예정이다.

참고문헌

[1] N. Laptev, K. Zeng, and C. Zaniolo, “Very Fast Estimation for Result and Accuracy of Big Data Analytics: the EARL System,“ in Proceedings of the 29th IEEE International Conference on Data Engineering, Brisbane, pp.1296-1299, Apr. 2013.

[2] 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.

[3] http://hadoop.apache.org/

[4] http://hadoop.apache.org/docs/r2.2.0/hadoop-yarn/hadoop-yarn-site/YARN.html

[5] http://docs.oracle.com/javase/7/docs/technotes/guides/jpda/architecture.html#jdi

[6] http://dumps.wikimedia.org/enwiki