Journal of The Korea Society of Computer and Information
Vol. 20 No. 8, pp. 1-6, August 2015

www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2015.20.8.001

A Novel Method of Improving Cache Hit-rate in Hadoop MapReduce using SSD Cache

Jong-Chan Kim*, Jae-Hoon An**, Young-Hwan Kim***, Ki-Man Jeon****

Abstract

The MapReduce Program of Hadoop Distributed File System operates on any unspecified nodes due to distributed-parallel process and block replicate for data stability. Since it is difficult to guarantee the cache locality when a Solid State Drive is used as a cache in hadoop, cache hit-rate is decreased. In this paper, we suggest a method to improve cache hit rate by pre-loading the input data of the MapReduce onto the SSD cache. To perform this method, we estimated the blocks that are used on each node by using capacity scheduler and block metadata. Eventually we could increase the performance of SSD cache by loading the blocks onto SSD cache before the Map Task run.

▸ Keyword : Hadoop, SSD Cache, HDFS, Mapreduce, YARN

I. Introduction

솔리드 스테이트 드라이브(Solid State Drive, 이하 SSD)는 순수 전자식으로 동작하는 저장장치로써 기계식으로 동작하는 HDD에 비해 빠른 응답속도와 높은 대역폭, 낮은 실패율 그리고 낮은 전력 소모량을 보장하는 차세대 저장장치이다[2]. SSD 저장장치가 가지는 수많은 장점들은 하둡(Hadoop)[1]과 같은 분산 파일 시스템(Distributed File System)[3]을 사용하는 환경에서 특히 유용하기 때문에 일부 시스템에서 SSD를 사용하고 있다. 하지만 상대적으로 비싼 가격 때문에 높은 용량을 요구하는 환경에서 HDD를 완전하게 대체할 수 없어서 SSD를 기존 저장장치의 캐시로 설정하여 하이브리드 저장장치를 만들어 사용한다.

그러나 하둡 분산 파일 시스템(Hadoop Distributed File System, 이하 HDFS)에서 동작하는 맵리듀스(MapReduce)는 다음 두 가지 기능으로 인해 데이터 재사용성이 떨어져 캐시를 활용하기 어려운 상황이 발생된다. 첫째, Fig. 1. 에서 보이는 바와 같이 블록 리플리케이션(Replication)을 통해 여러 노드에 저장한다. 둘째, 스케줄링(Scheduling) 정책에 의해 동일

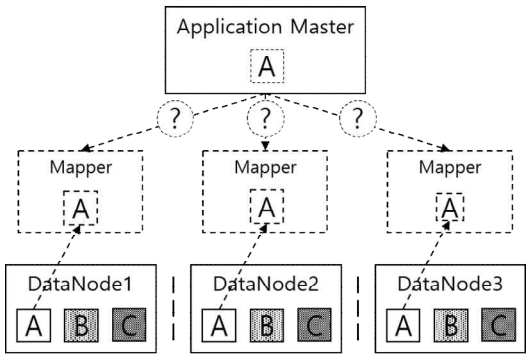

Fig. 1. Unpredictability of Locations Where The Block Loaded

• First Author: Jong-Chan Kim, Corresponding Author: Young-Hwan Kim
*Jong-Chan Kim (8eatles@keti.re.kr), Intelligent IDC Project Office, Korea Electronics Technology Institute
**Jae-Hoon An (corehun@keti.re.kr), Intelligent IDC Project Office, Korea Electronics Technology Institute
***Young-Hwan Kim (yhkim93@keti.re.kr), Intelligent IDC Project Office, Korea Electronics Technology Institute
****Ki-Man Jeon (kmjeon@keti.re.kr), Intelligent IDC Project Office, Korea Electronics Technology Institute
• Received: 2015. 06. 17, Revised: 2015. 06. 29, Accepted: 2015. 07. 31.
• This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No.10047088, Development of open type hadoop storage appliance to support more than 48TB per single data node).

한 데이터를 사용하는 애플리케이션이 재실행되더라도 상황에 따라 다른 노드에서 동작하는 상황이 빈번하게 발생한다. 따라서 맵리듀스 작업 시 필요한 데이터 블록을 어떤 노드에서 가져올지, 또 어느 노드에서 연산을 수행할지 예측하기 어렵다.

본 논문에서는 SSD를 캐시로 사용하는 하둡 환경에서 맵리듀스를 통한 데이터 I/O 수행 시 동작하는 데몬 프로세서(Daemon Processor)를 설계하여 맵태스크(Map Task)단위로 사용 예정인 블록들을 캐시로 적재한다. 설계한 데몬 프로세서는 맵리듀스에 필요한 데이터 블록의 예측을 위해 HDFS의 블록 메타데이터를 관리한다. 그리고 HDFS 파일시스템 메타데이터와 맵리듀스 과정에서 실행된 맵태스크에 배정된 입력 데이터 파일의 블록 ID를 이용하여 연관된 블록의 I/O 요청이 발생하기 전에 캐시에 적재함으로써 캐시 적중률을 높일 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 하둡에서 맵리듀스를 통한 I/O 동작과 관련 연구 및 사용기술에 대해 소개한다. 3장에서는 제안한 메커니즘의 구조와 구현방법에 대해 기술하며, 4장에서는 제안한 메커니즘을 적용한 시스템과 기존 시스템의 성능평가를 보인다. 마지막으로 5장에서는 결론 및 향후 연구에 대해 기술한다.

II. Related works

1. MapReduce I/O on HDFS

HDFS는 구글 파일 시스템(Google File System)을 기반으로 설계되었으며 마스터 노드(Master Node)가 존재하는 분산 파일 시스템으로써 하둡에서 사용하는 기본 파일시스템이다. HDFS는 입력된 파일을 일정 크기의 블록으로 분할하여 다수의 노드에 분산 저장하고, 각 블록단위로 복제본(Replicas)을 생성하여 데이터 손실과 같은 위험에 대비한다. HDFS에서의 마스터 노드는 네임노드(NameNode)이며 네임노드를 보조하는 세컨더리 네임노드(Secondary NameNode)가 존재한다. 세컨더리 네임노드는 네임노드에서 관리하는 파일 시스템 이미지를 백업하고 변경기록(EditLog)을 통합하여 네임노드로 전달하는 역할을 한다. 또한 HDFS에서의 종속 노드(Slave Node)인 데이터 노드(DataNode)는 실제 블록 저장 및 복제, 스트리밍을 수행한다.

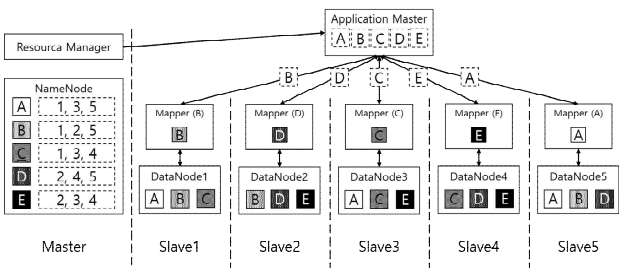

Fig. 2. MapReduce I/O Process

Fig. 2. 는 HDFS에서 수행되는 맵리듀스의 과정을 보인다. 맵리듀스 애플리케이션 실행이 요청되면 마스터 노드의 리소스 매니저(Resource Manager, 이하 RM)가 종속 노드에 애플리케이션 마스터(Application Master, 이하 AM)를 실행시킨다. AM은 입력 데이터(Input Data)를 설정된 크기로 분할(Split)하고 분할된 데이터의 수만큼 맵태스크를 생성하여 처리한다. 입력 데이터의 경우 블록 단위로 나뉘어 각 종속 노드 저장되며 AM에 의해 분할된 입력 데이터 또한 블록으로 구성되어진다.

AM이 생성하는 맵태스크들은 RM에 있는 커패시티 스케줄러(Capacity Scheduler)에 의해 분할 데이터를 구성하는 블록을 가진 종속 노드에 최우선적으로 배치된다. 만약 구성하는 블록이 존재하는 종속 노드에 여유 리소스가 없는 경우에는 네트워크상 거리가 짧은 노드로 배치된다. 따라서 각 노드의 상황에 따라서 유연하게 리소스를 활용하는 것이 가능하지만, 반대로 언제 어디서 해당 블록을 처리하는지 알 수 없고 상황에 따라 블록 재사용 시 동일한 노드에서 실행됨을 보장할 수 없다. 이는 SSD를 캐시로 사용하는 환경에서 심각한 캐시 지역성(Locality) 문제로 연결되어 진다.

2 Flashcache

페이스북(Facebook)에서 개발한 플래시캐시(Flashcache)[5]는 SSD를 기존의 HDD 저장장치에 하나로 묶어 라이트백(Write-Back) 방식의 캐시로 동작 가능하게 해주는 블록 캐시 리눅스 커널 모듈이다. 디스크 블록 숫자(Disk Block Number, 이하 DBN)를 이용한 집합 연관 해시(Set Associative Hash)를 이용해 읽기 작업 요청 시 DBN에 근거한 캐시 셋의 주소를 계산하여 선형 시간 내 검색이 가능하도록 설계되었다. sysctl 설정에 따라 FIFO(First-In First-Out)와 LRU(Least Recently Used) 두 가지의 교체정책을 지원하는데[6], 두 교체정책 모두 캐시 지역성이 성능에 미치는 영향이 크다. 따라서 같은 데이터를 사용해도 상황에 따라 다른 노드에서 실행될 가능성이 있는 하둡 시스템에서는 플래시캐시의 효율이 떨어질 수밖에 없다.

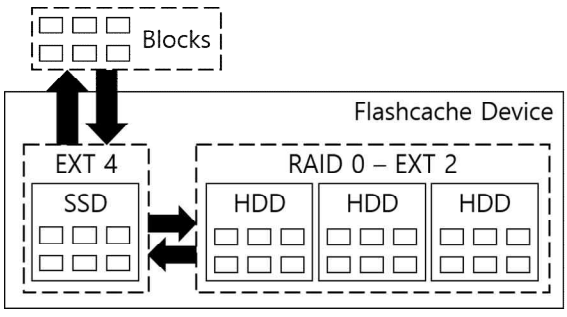

Fig. 3. Flashcache I/O Process

현재의 실험 환경은 Fig. 3. 과 같이 각 종속 노드에 레이드(RAID)로 묶인 3개의 HDD와 1개의 SSD를 이용해 캐시 디바

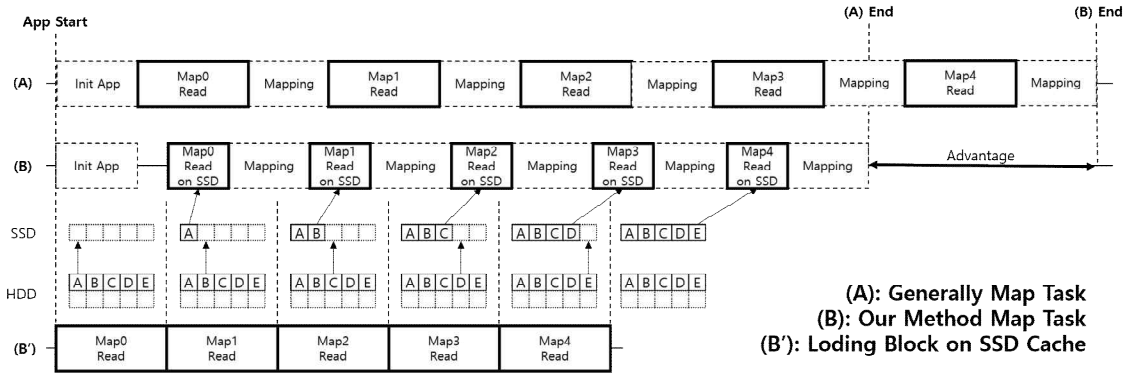

Fig. 4. Block Pre-Load Architecture

이스를 생성하여 사용하기 위해 플래시캐시 모듈을 적용한다.

3 Block Metadata

HDFS에서 데이터를 가진 모든 파일은 블록 단위로 블록 풀에 저장된다. HDFS의 파일 메타데이터에는 파일을 구성한 블록들이 배열로 저장되어 있으며 모든 블록들은 블록 풀이라는 가상의 저장소에 보관된다.

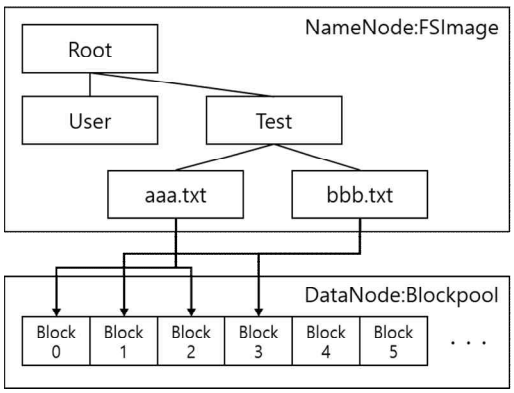

Fig. 5. Relation of File System Metadata and Blocks in HDFS

블록 풀은 모든 종속노드의 데이터 저장소들의 합이며 설정된 복제 수에 따라 동일한 블록이 서로 다른 종속노드에 존재하게 된다. 만약 종속 노드의 수와 설정된 복제 수가 같다면 모든 종속노드의 각 데이터 저장소에 블록 풀에 존재하는 모든 블록들이 저장된다.

Fig. 4. 와 같이 파일시스템 메타데이터에 저장된 블록 배열의 각 항목은 블록 ID와 블록 풀에 존재하는 블록의 위치를 가지고 있다. 본 논문에서는 파일시스템 메타데이터를 이용하여 맵태스크가 처리하는 첫 블록의 정보로 이후 처리될 블록들의 블록 ID를 얻어온다.

4 Capacity Scheduler

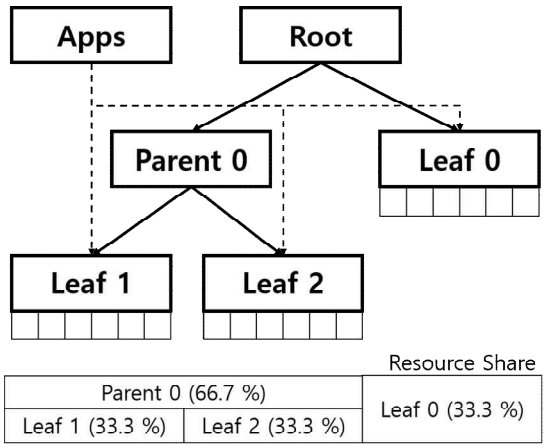

Fig. 6. Capacity Scheduler

커패시티 스케줄러는 하둡에서 사용하는 스케줄러로 Fig. 6. 에서 보이는 바와 같이 계층형 큐를 이용하여 전체 클러스터의 리소스에 각 리프 큐의 영역을 분할하여 사용한다[7]. 커패시티 스케줄러의 특징 중 하나는 사용자 정의 지역성을 지원하여 [8] 애플리케이션에 밀접한 노드 혹은 랙을 지정할 수 있다. 이런 특성 때문에 노드별로 1번씩 맵태스크를 생성하는 FIFO와 달리 지정한 커패시티가 모두 사용될 때까지 동일한 노드에서 맵태스크를 계속 생성한다. 커패시티 스케줄러를 이용하면 최초 블록 요청이 있은 후 입력 데이터의 블록 배열을 참조하여 앞으로 사용될 블록들의 예측이 가능하다.[11]

III. The Proposed Scheme

1. Architecture of The Proposed Method

1.1 Design

II. 1 MapReduce I/O on HDFS 절에서 제시한 캐시 지역성 문제를 해결하기 위해 맵리듀스 애플리케이션이 동작하는 동안 각 종속 노드에서 동작하는 맵태스크의 HDFS 읽기 작업에서

파일을 읽기 전에 블록 단위로 SSD 캐시에 데이터를 미리 적재해놓는 메커니즘을 사용한다. 따라서 어떤 블록을 사용할지 미리 예측하는 과정이 필요하다. 본 논문에서는 커패시티 스케줄러의 리소스 선점방식과 입력 데이터의 블록 배열을 이용하여 각 노드별로 사용될 블록을 예측하고 예측된 블록이 사용되기 전에 미리 캐시에 적재하여 적중률을 향상시키는 메커니즘을 설계하였다.

Fig. 5. 는 맵리듀스의 각 맵태스크에서 예측한 블록의 사전 적재를 통해 얻는 시간적 이점을 시각적으로 나타내고 있다. 블록 A~E로 이루어진 입력 데이터를 이용하여 맵리듀스 작업을 진행하는 과정에서 (A)는 기존의 맵태스크들을 수행하는 상황을 나타낸 것이고, (B)는 본 논문에서 제안하는 메커니즘인 사전 로딩(B')을 백그라운드에서 수행하여 캐시 적중률을 향상시킨 상황을 나타낸다. (B)에서 애플리케이션이 준비되는 동안 (B')에선 사용할 블록들을 미리 로딩하기 시작하여 실제 맵태스크에서 블록을 불러오는 순간에 HDD가 아닌 SSD에서 블록을 가져온다. 이때 (B')에서 사용할 블록들을 미리 알기 위해 종속 노드의 데이터 노드 로그를 모니터링 하고, 현재 잔여 리소스의 (설정된 RAM중 사용되지 않는) 크기를 통해 실행될 맵태스크의 수를 구한다. 그리고 "hadoop fsck" 명령어를 통해 얻은 HDFS의 전체 블록 메타데이터에 데이터 노드 로그를 모니터링 하여 캡처된 블록 아이디를 대조하여 해당 파일의 블록 배열에서 실행될 맵태스크 수만큼의 블록을 미리 읽기 요청을 통해 블록을 사전 로딩 한다.

1.2 Implementation

본 논문에서 제안한 메커니즘을 위해 캐시 가속기(Cache Accelerator) 데몬 프로세스를 설계했다. 설계된 캐시 가속기는 주기적으로 마스터 노드의 HDFS를 구성하는 블록들의 메타데이터와 현재 노드의 리소스 상태 정보를 갱신하고 있다. 또한 잔여 리소스 정보와 데이터 노드 로그를 분석하여 사용할 블록을 예측해 캐시에 적재한다.

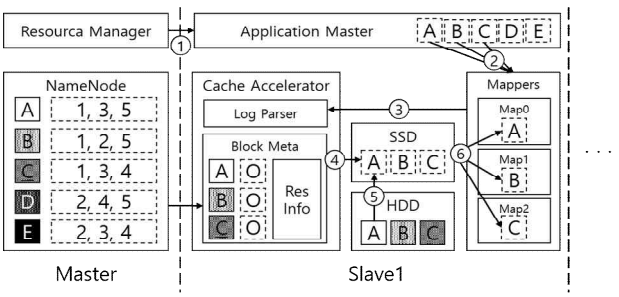

Fig. 7. Cache Accelerator

Fig. 7. 은 새 맵리듀스 애플리케이션이 실행되었을 때의 캐시 가속기의 동작을 나타내며 세부 과정은 다음과 같다.

① RM이 실행된 애플리케이션의 AM을 생성함
② 생성된 AM에서 종속 노드에 맵태스크를 생성함
③ 데이터 노드 로그를 통해 맵태스크의 블록 정보가 출력됨
캐시 가속기에서 로그로 출력된 블록 정보를 분석함
리소스 상태정보와 블록 정보로 필요한 블록을 예측함
④ CLI 명령을 통해 SSD 캐시로 적재 요청을 보냄
⑤ 블록 적재 CLI 명령이 실행되고 블록이 캐시로 적재됨
⑥ 맵태스크에서 필요한 블록을 SSD 캐시에서 읽음

2. Performance Evaluation

2.1 Environment

Table 1. System Environment

	Master	Slave
Processor	Intel Xeon 32Core	Intel Atom 8Core
Memory	48GB	32GB
OS Storage	HDD 2TB	HDD 2TB
Hadoop Storage		HDD 4TB * 3
Cache Device	None	SSD 1TB

본 논문에서 사용된 하둡 시스템의 실험환경은 1개의 마스터 노드와 3개의 종속 노드로 구성되어있으며 사용된 하드웨어 정보는 Table. 1.과 같다. 종속 노드의 하둡 저장소를 구성하는 4TB HDD 3개는 RAID 0 로 구성되어있다.

2.2 The Method of Performance Evaluation

테스트는 하둡의 기본적인 벤치마크 툴인 TestDFSIO, Terasort로 관련연구 2.2의 Flashcache를 이용한 환경과 캐시 가속기를 추가 적용한 환경, 총 2가지 환경에서 진행하였다. 보다 정확한 SSD 캐시의 속도를 측정하기 위해 하둡 기본 설정에서 캐시 관련 설정을 off 하고 진행하였다. 테스트는 각 항목별로 10회씩 5세트를 진행하였고, 각 세트가 끝날 때마다 캐시 장치를 초기화하였다. 결과 값은 세트별로 최대, 최솟값을 제외한 8개 값의 평균을 냈다.

2.3 Performance Evaluation Using TestDFSIO

TestDFSIO는 분산파일시스템에서의 순수한 I/O 속도를 측정하기 위한 툴로써 I/O 작업 외에 맵태스크나 리듀스태스크(Reduce Task)에서 데이터를 처리 혹은 가공하지 않는다. 본 논문에서 제안한 구조는 Write가 아닌 Read에서의 성능을 최대로 끌어올리기 위한 메커니즘 Write 테스트는 생략하였다. TestDFSIO는 입력 옵션에 따라 각 맵태스크에서 처리하는 블록의 수를 조절할 수 있다. 각 테스트별로 40GB의 데이터를 Read 하였으며, 맵태스크의 수를 조절하며 테스트하였다. Table 2, 3은 TestDFSIO의 결과 값(처리속도)을 나타낸다.

Table 2. Flashcache Env. TestDFSIO Throughput (MB/s)

	1Set	2Set	3Set	4Set	5Set
20 Tasks	14.67	14.87	15.16	14.75	14.33
40 Tasks	10.42	9.94	10.22	10.30	10.01
80 Tasks	8.49	8.88	8.32	8.21	8.09
160 Tasks	7.53	7.64	7.51	6.98	7.27

Table 3. Flashcache and Cache Accelerator Env. TestDFSIO Throughput (MB/s)

	1Set	2Set	3Set	4Set	5Set
20 Tasks	15.04	14.83	14.98	16.04	15.11
40 Tasks	10.28	10.56	10.44	10.72	10.93
80 Tasks	8.8	8.79	9.02	9.23	9.12
160 Tasks	7.52	7.7	8.31	8.22	8.28

맵태스크의 수가 많을수록 각 맵에서 처리하는 블록의 수가 많아진다. 본 논문의 HDFS 블록 사이즈는 128MB이며, 각 맵 태스크에서 처리하는 데이터 크기는 전체 데이터 크기인 40GB에 맵태스크 숫자를 나누어 구한다.

Table 2, 3에서 보이는 처리량은 맵태스크 개별 처리량의 평균을 나타낸 값으로 태스크의 수가 많아질수록 처리량 수치가 줄어든다. 이는 동시에 많은 수의 태스크를 처리 하는 과정에서 프로세서와 메모리 리소스를 최대로 활용하기 때문에 맵 태스크 개별 처리량이 줄어드는 것을 나타낸다.

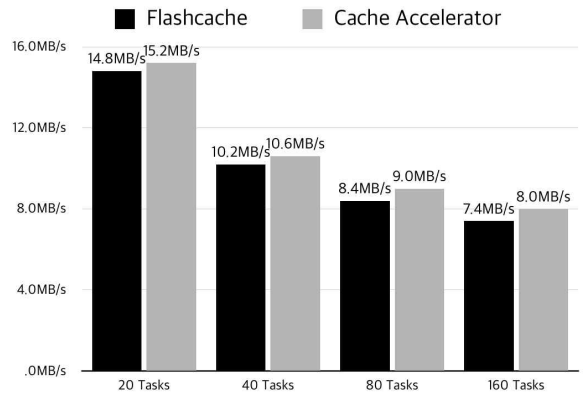

Fig. 8. TestDFSIO Throughput Result

Fig. 8. 은 맵태스크 숫자가 많을수록 본 논문에서 제시한 메커니즘이 최대 8.1%의 성능이 높아짐을 보이고 있다. 따라서 캐시 가속기에 의해 예측 가능한 블록의 숫자가 맵태스크의 수에 영향을 받는 것을 알 수 있다.

2.4 Performance Evaluation Using Terasort

Terasort는 I/O 작업 외에도 입력 데이터의 정렬 작업을 진행하므로 실제 맵리듀스 작업과 유사한 테스트 프로그램이다. Terasort는 TestDFSIO와는 달리 모든 설정이 현재 환경에 설정된 그대로를 따라가므로 맵태스크의 수를 별도로 지정할 수가 없다. 다만 입력 데이터의 크기가 일정 크기 이상이 되면 현재 환경에서의 모든 종속노드의 자원을 사용하므로, 실험 결과에 크게 영향을 받지 않는다. Table 4, 5 는 Terasort의 결과 값(수행시간)을 나타낸다.

Table 4. Flashcache Env. Terasort Running Time (Sec)

	1Set	2Set	3Set	4Set	5Set
10GB	1990	1958	2028	1935	1929

Table 5. Flashcache and Cache Accelerator Env.Terasort Running Time (Sec)

	1Set	2Set	3Set	4Set	5Set
10GB	1778	1822	1801	1789	1830

Terasort는 로그나 결과를 통해 성능을 보여주지 않는다. 따라서 Terasort 수행 성능을 측정하려면 별도로 수행 시간을 측정해 주어야 한다. 측정된 수행시간이 짧을수록 맵리듀스 처리 속도가 빠른 것을 나타내므로 Fig. 9.에서 보이는 바와 같이 수행 시간이 약 8.3% 감소한 것을 통해 본 논문에서 제시한 메커니즘이 실제 맵리듀스 작업에서도 성능 향상으로 이루어지는 것을 확인할 수 있다.

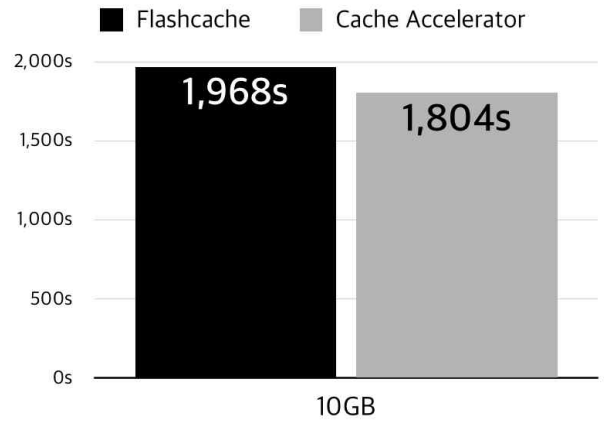

Fig. 9. Terasort Running Time Result

IV. Conclusion

SSD 캐시를 사용하는 하둡 클러스터 시스템에서는 맵리듀스 작업 시 캐시 지역성을 보장받지 못해 적중률이 떨어지는 문제점이 있다. 본 논문에서는 제시한 문제점을 보완하기 위해 캐시 적중률을 향상시키는 메커니즘을 제안했다. 제안된 메커니즘은 마스터 노드가 관리하는 HDFS 블록들의 메타데이터와 종속노드의 리소스 현황을 수집하는 데몬 프로세스를 개발하여, 첫 번째 맵태스크에서 사용하는 블록 ID를 받아 나머지 맵

태스크가 필요로 하는 블록들의 정보를 알아낸다. 그리고 남은 맵태스크들이 실행될 때 블록이 SSD 캐시에 적재되어 있도록 실행 전 미리 적재하여 SSD 캐시의 적중률을 높였다. 제안된 메커니즘을 적용한 환경의 성능평가에서 약 8%의 성능 향상을 보임으로써 전체적인 맵리듀스 수행 시간을 단축시키는데 효과가 있음을 알 수 있다.

본 실험을 위해 개발한 캐시 가속기 프로그램은 종속 노드에 존재하지 않는 블록들에 대한 처리가 되어있지 않다. 향후 연구 방향으로 종속노드에 필요하다고 예측된 블록이 존재하지 않는 상황에서 맵태스크가 준비되는 동안 해당되는 블록들을 사전에 스트리밍 받아와 기존 맵리듀스의 동작에서 블록 스트리밍으로 인한 지연시간을 줄여 맵리듀스의 성능을 향상시키는 기법 연구를 진행할 것이다.

REFERENCE

[1] "Hadoop.", from http://hadoop.apache.org

[2] "Solid-state drive, Wikipedia", https://en.wikipedia.org/wiki/Solid-state_drive

[3] Shvachko K., Kuang H., Radia S., and Chansler R, "The hadoop distributed file system.", In Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium, pp. 1-10, May, 2010

[4] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters." In Communications of the ACM, Vol.51, No.1, pp. 107-113, Jan, 2008

[5] "Flashcache.", https://wiki.archlinux.org/index.php/Flashcache

[6] "Flashcache project", https://github.com/facebook/flashcache

[7] "Hadoop's Capacity Scheduler", http://hadoop.apache.org/core/docs/current/capacity_scheduler.html

[8] Arun C. Murthy. "Apache Hadoop YARN, Moving beyond MapReduce and Batch Processing with Apache Hadoop 2", Pearson Education, pp. 153-170, 2014

[9] S. H. Kang, D. H. Koo, W. H. Kang, and S. W. Lee, "A case for flash memory ssd in hadoop applications." International Journal of Control and Automation, Vol.6, No.1, pp. 201-210, Feb, 2013

[10] T. H. Keum, W. J. Lee, and C. H. Jeon, "A Performance Analysis Based on Hadoop Application's Characteristics in Cloud Computing", Journal of The Korea Society of Computer and Information, Vol.15, No.5, pp.49-56, May, 2010

[11] J. S. Kim, C. H. Kim, W. J. Lee, and C. H. Jeon "A Block Relocation Algorithm for Reducing Network Consumption in Hadoop Cluster", Journal of The Korea Society of Computer and Information, Vol.19, No.11, pp.9-15, Nov ,2014

Authors

Jong Chan Kim has completed in Computer Engineering from Korea Polytechnic University, in 2014
He is now assistant researcher in Korea Electronics Technology Institute, Korea. He is interested in distributed systems.

Jae-hoon An received the B.S. degree in the Department of Computer Engineering from Kwangwoon University, Seoul, Korea in 2007. He also obtained the MS and PhD degrees in the Computer Engineering from Soongsil University, Korea in 2009 and 2014 respectively.
Dr. An is now senior researcher in Korea Electronics Technology Institute, Korea. His research interests include embedded systems, fault tolerance computing systems, real-time systems.

Young-Hwan, Kim received the B.S degree in the department of Computer Engineering from Pukyong National University in 2001. He received his M.S degree in the department of Computer Science and Engineering from Sungkyunkwan University in 2003.
He also obtained PhD degree in the Computer Engineering from Soongsil University, Korea in 2013. Dr. Kim is now managerial researcher in Korea Electronics Technology Institute, Korea. His research interests include embedded systems, storage.

Ki-Man, Jeon received the M.S. and Ph.D. in the department of electrical and biomedical engineering from Hanyang University, Korea in 2000, 2015, respectively. Dr. Jeon is now managerial researcher in Korea Electronics Technology Institute, Korea. His research interests include vdi system, system hardware.