SSD 캐시를 적용한 HDFS 의 I/O 비용 기반 데이터 선택 기법

김민경, 신민철, 박상현
연세대학교 컴퓨터과학과
e-mail : { goodgail, smanioso, sanghyun } @ cs.yonsei.ac.kr

Mechanism to Select the Data Source of HDFS with SSD Cache Based on Storage I / O Cost

Minkyung Kim, Mincheol Shin, Sanghyun Park
Dept. of Computer Science, Yonsei University

요 약

빅데이터 분석을 위한 Hadoop 환경에서 고성능 저장장치인 SSD 에 대한 중요성이 증가하면서 일반적으로 사용되는 저장장치인 HDD 와 혼합하여 사용하는 연구들이 주목 받고 있다. 특히 SSD 를 HDD 의 캐시로 사용했을 때 저장장치에 대한 I/O 성능을 향상할 수 있다는 연구 결과들이 있다. 본 연구는 이를 바탕으로 SSD 를 HDD 의 캐시로 사용한다. HDFS 는 저장장치에 접근하여 I/O 를 수행하는데 기존에는 로컬 서버에서 캐시 미스가 발생한 경우 로컬 HDD 로 접근한다. 이러한 방식은 접근하는 데이터에 따라 SSD 의 높은 Bandwidth 를 활용하지 못하게 되는 경우를 발생시키고 그 결과 특정 서버의 I/O 지연으로 전체 분산 처리의 성능을 저하시킬 수 있다. 이를 해결하기 위해 본 연구는 HDFS 레벨에서 로컬 서버의 HDD 와 데이터 복제본들이 저장된 원격 서버의 SSD 에서 I/O 를 수행하는 경우에 대해 수식을 통해 비용을 비교한다. 그 결과 항상 기대 성능이 높은 저장 장치를 선택하여 데이터를 읽어오게 함으로써 기존 방식보다 성능이 개선될 수 있음을 입증한다.

1. 서론

컴퓨터 기술 발달에 따라 데이터양은 매우 증가 하고 있으며 이를 다루는 연구들이 활발히 진행되고 있다. 특히 빅데이터의 저장과 처리 방식을 다루는 소프트웨어 프레임워크가 주목 받고 있는데 그 대표적인 예로는 Hadoop[1]이 있다.

Hadoop 은 분산처리시스템인 MapReduce 와 분산파일시스템인 HDFS(Hadoop Distributed File System)로 구성된다. 하지만 MapReduce 는 일괄처리에 적합하고 실시간 처리에는 부적합하다는 단점을 가진다. 최근에는 이러한 MapReduce 의 단점을 보완한 SQL-on-Hadoop 계열이 HDFS 와 결합하여 사용되고 있다.

빅데이터 환경에서 사용자의 요청을 처리하는 과정은 다음과 같다. 요청은 MapReduce, Hive[2], Impala[4], Tajo[5] 등의 분산처리시스템을 거쳐 HDFS 레벨로 전달된다. HDFS 는 데이터가 저장된 저장장치로 접근하여 I/O 를 수행한다.

HDFS 가 저장장치로 접근하는 과정에서 Hadoop 성능 향상을 위해 NAND 플래시 메모리를 기반으로 만들어진 SSD(Solid State Disk)를 이용하자는 연구들이 있다. 하지만 SSD 는 가격 대비 용량이 작고 NAND 플래시 메모리의 물리적 특성상 쓰기 횟수에 제한이 있다는 문제점이 있어서 아직 기존의 HDD(Hard Disk Drive)를 완전히 대체하지는 못하고 있다. 이러한 점을 보완하고자 SSD 와 HDD 를 혼합하여 사용하려는 시도들이 있다[6]. 혼합 환경은 두 가지로 나누어질 수 있는데 첫 번째는 SSD 와 HDD 를 독립적인 장치로 구성하여 서로 다른 종류의 I/O 를 처리하는 경우이다. 예로는 로컬 서버에서 발생한 중간 질의 처리 결과를 SSD 에 저장하고, HDFS 에 저장해야 하는 데이터는 HDD 에 저장하는 것이다. 두 번째는 SSD 를 HDD 의 캐시로 사용하는 경우이다. 본 연구는 후자에 초점을 맞춘다.

SSD 를 캐시로 사용했던 기존의 많은 연구는 캐시 힛(Cache Hit)의 경우 SSD 를 읽고, 캐시 미스(Cache Miss)인 경우에는 로컬 서버의 HDD 로 접근하여 데이터를 읽어온다. 그러므로 캐시 미스가 많이 발생한다면 HDD 에서 계속 읽게 되므로 SSD 의 높은 Bandwidth 를 활용하지 못한다. 이는 특정 서버의 I/O 지연을 발생시키고 전체적인 성능 저하를 일으키므로 비효율적이다.

이에 본 연구는 로컬 서버에서 캐시 미스가 발생했을 경우에 대해 집중적으로 다뤄보고자 한다. 수식을 통해 로컬의 HDD 와 복제본들이 저장된 원격 서버의

SSD 에서 읽는 경우에 대해 각각 I/O 비용을 계산한다. 계산된 I/O 비용은 저장장치를 선택하기 위해 사용된다. 최종적으로는 데이터의 복제본들이 저장되어 있는 저장장치 중에서 I/O 성능이 높은 저장장치를 선택하여 필요한 데이터에 대한 I/O 를 수행하도록 함으로써 기존 방식보다 성능이 개선될 수 있음을 입증하는 것을 목표로 한다.

이 논문의 구조는 다음과 같다. 2 장에서는 본 연구의 바탕이 되는 Hadoop 과 SSD 에 대해 살펴보고, 3 장에서는 성능 비교를 위한 수식과 HDFS 레벨에서 개선된 데이터 선택 기법에 관해 설명한다. 마지막 4 장에서는 본 논문에서의 결론과 앞으로의 연구 발전 방향을 제시한다.

2. 배경

2.1 Hadoop

Hadoop 은 오픈 자바 소프트웨어 프레임워크로서 대용량 데이터를 다수의 컴퓨터 클러스터에서 동시에 동작시켜 부하를 분산 처리하는 것으로 MapReduce(분산처리시스템)와 HDFS(분산파일시스템)로 구성되어 있다.

MapReduce 는 대용량의 데이터를 병렬처리하기 위한 분산 처리 프로그래밍 모델이다. 본 논문은 HDFS 에 초점을 맞추기 때문에 MapReduce 에 대한 자세한 설명은 생략한다.

HDFS 는 데이터를 여러 서버에 분산시켜 저장하는 분산 파일 시스템이다. 기본 구성은 단일 Master 역할을 하는 NameNode 와 다수의 Slaves 인 DataNode 로 이루어져 있다. NameNode 는 대용량 파일을 블록(기본 64MB) 단위로 나누고 내고장성(Fault Tolerance)을 지원하기 위해 각 블록을 복제 후 여러 서버에 분배하여 저장한다. DataNode 는 분산 파일에 대해 실제 I/O 작업을 수행하는 역할을 한다.

2.2 SQL-on-Hadoop

최근에는 Hadoop 의 MapReduce 를 대체 할 수 있는 SQL-on-Hadoop 계열의 분산처리시스템들이 개발되고 있다. SQL-on-Hadoop 은 HDFS 에 저장된 데이터를 전통 데이터베이스 진영에서 사용해 왔던 표준 SQL 언어를 사용하여 다룰 수 있게 해준다. 그 예로는 Hive, Tajo, Impala 등이 있다. 하지만 Hive 의 경우 MapReduce 기반이므로 처리 속도가 느리다. 이를 해결하고자 MapReduce 를 대체하는 Tajo 와 Impala 등이 개발되었다. 이들은 MapReduce 가 아닌 자체분산처리 시스템을 사용하여 SQL 질의를 처리하므로 Hive 보다 성능이 좋다. 그러므로 Tajo 와 Impala 를 HDFS 와 결합함으로써 실시간 데이터 처리 성능을 향상하고자 하는 연구가 활발히 진행되고 있다.

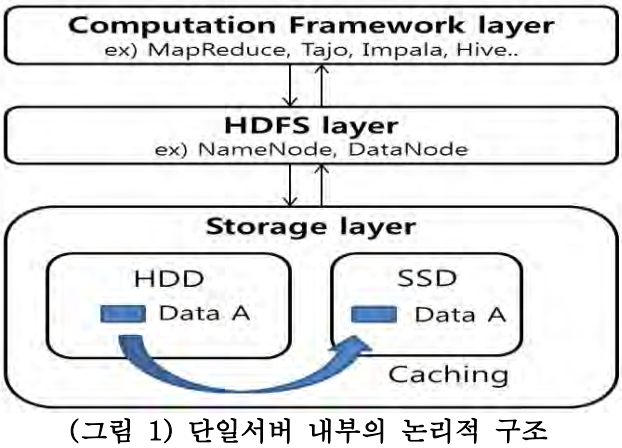
2.3 SSD 를 활용한 HDFS I/O 관련 연구

HDFS 를 사용하여 빅데이터를 처리할 때 발생하는 I/O 는 크게 네트워크 I/O 와 저장장치 I/O 로 나눌 수 있다. 네트워크 I/O 예로는 DataNode 가 NameNode 로부터 관리 정보를 받아오거나 중간질의 결과를 DataNode 간에 전달하는 경우에 발생한다. 일반적으로 네트워크는 외부망일 경우에 100Mbps(100 Mega bit per second), 클러스터에서는 1Gbps(1 Giga bit per second), 고성능 클러스터는 10Gbps 를 많이 사용한다. 10Gbps 는 초당 10 기가비트의 데이터 속도를 제공함을 의미한다. 저장장치 I/O 는 HDD 나 SSD 와 같은 물리적 저장장치에 저장된 데이터를 읽고 쓰는 것을 의미한다. HDD 의 경우 랜덤 I/O 보다는 순차 I/O 에 빠른 처리 속도를 가진다. 이는 헤더를 움직이며 데이터를 찾아야 하는 HDD 의 물리적 특성에서 기인한다. 반대로 SSD 의 경우 HDD 보다 매우 짧은 시간 안에 원하는 데이터로 접근하므로 Access time 이 매우 적게 걸린다. 또한, 시간당 데이터 전송량을 의미하는 Bandwidth 가 HDD 에 비해 높으므로 뛰어난 I/O 성능을 보장한다. 하지만 가격 대비 용량이 작다는 것이 단점이다.

최근 성능 향상을 위해 HDFS 내의 HDD 를 SSD 로 대체하려는 시도들이 있다. 그러나 위에서 언급한 바와 같이 SSD 의 높은 가격대로 인하여 수십 수백만 대의 HDD 를 모두 교체하기란 쉽지 않다. 그러므로 bcache[3]와 같은 오픈소스 캐시 계층(Cache Layer)을 활용하여 SSD 를 HDD 의 캐시 용도로 사용하고 성능 향상을 이루려는 방향으로 연구가 진행되고 있다.

3. 개선된 HDFS 레벨의 데이터 선택 방법

[그림 1]은 클러스터를 구성하는 각 서버의 논리 계층 구조를 나타낸 것이다. 사용자는 MapReduce 와 같은 Computation Framework Layer 를 통해 데이터를 요청한다. Computation Framework Layer 는 사용자가 요청한 데이터를 처리하기 위해 기존 데이터를 가공하며, 이때 필요한 데이터는 HDFS 저장장치에 접근하여 요청한다. HDFS 는 NameNode 정보를 이용하여 클러스터 내의 특정 서버로 접근한 뒤 요청 받은 데이터를 물리적 저장장치에서 읽어온다.



(그림 1) 단일서버 내부의 논리적 구조

SSD 를 HDD 의 캐시로 사용했을 경우 기존 방식은 캐시 힛 일 때 SSD 를 우선으로 읽는다. 반대로 캐시 미스의 경우 로컬 HDD 에서 데이터를 읽는다. 하지만 HDD 는 SSD 보다 I/O 성능이 낮으므로 특정 서버의 I/O 지연을 발생시켜 전체 분산 처리 성능을 저하할 수 있다.

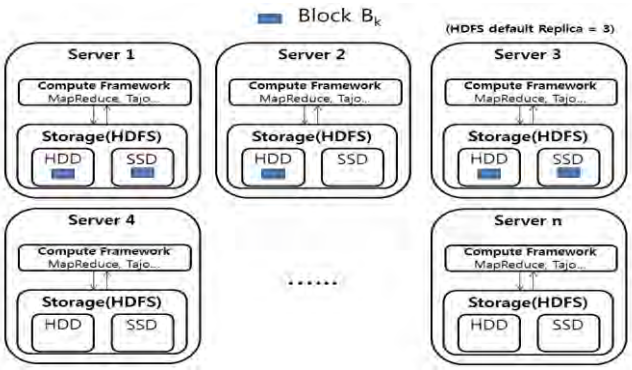
3.1 아이디어

위와 같은 문제를 해결하고자 SSD 캐시를 적용한 HDFS 의 I/O 비용 기반 데이터 선택 기법을 제안한다. 세부 아이디어는 다음과 같다. [그림 2]와 같이 SSD 를 HDD 캐시로 사용한 환경에서 HDFS 에 블록 B_k가 저장되어 있다. 블록 B_k를 읽기 위해 다음과 같은 경우를 생각해 볼 수 있다.

- **Case 1. 서버 1 에서 B_k 를 읽는다.**
- **Case 2. 서버 2 에서 B_k 를 읽는다.**

Case 1 의 경우 B_k가 로컬 SSD 에 저장되어 있으므로 HDFS 는 SSD 에서 원하는 데이터를 빠르게 읽는다. Case 2 의 경우 B_k 는 로컬 HDD 에만 저장되어 있다. B_k 를 읽기 위해 본 아이디어는 다음과 같은 두 가지의 선택지를 제시한다.

- **Case 2-1. (로컬 HDD 에서 읽음)**
  ex) 서버 2 의 HDD 에서 읽는다.
- **Case 2-2. (원격 SSD 에서 읽음)**
  ex)복제 본이 저장되어 있는 원격 서버 SSD 에 B_k가 있다면(e.g. 서버 1 or 서버 3) 읽는다.



(그림 2) n 개의 서버로 구성된 클러스터에서 블록 B_k의 분포 예시

3.2 성능 비교 방법

Case 2 에 대해 조금 더 다뤄보고자 한다. Case 2 는 로컬 서버 2 의 SSD 에 필요로 하는 데이터가 없는 경우이다. 이때 로컬 서버의 HDD 에서 읽는 비용과 복제본들이 저장된 원격 서버의 SSD 에서 읽는 비용을 계산한다. 만약 원격 서버의 SSD 에서 가져오는 경우가 로컬 HDD 에서 가져오는 것보다 비용이 적은 경우 네트워크를 통해 데이터를 전달받는다. 본 아이디어는 I/O 비용을 비교하여 복제본들 중에서 최종적으로 접근할 데이터를 선택할 수 있는 것이 기존 방식과 차별화된 점이다.

알고리즘 1 은 서버에서 새로운 블록을 읽을 때마다 반복적으로 수행되어야 하는 과정이다. 블록 B_k 를 읽기 위해 HDFS 레벨에서 데이터 복제본들에 대한 I/O 비용 검사를 실시하고 데이터를 선택하면 이를 반영하여 최종 I/O 를 수행하는 과정을 보여준다. 1~2 번 라인은 현재 서버 i 의 SSD 에 B_k가 존재하여 SSD 에 있는 데이터를 바로 읽는 경우이다.

알고리즘 1
HDFS 레벨에서 I/O 비용 비교를 통한 데이터 선택 알고리즘
상세설명
로컬 서버 SSD 에 데이터가 없는 경우, 복제본들이 저장된 각 서버 저장장치의 I/O 비용을 계산하여 성능이 좋은 데이터 선택
입력
i : 현재 서버의 번호
B_k : 필요한 데이터 블록 정보

출력
j : 최종 I/O 를 수행 할 서버의 번호
Output_Block B_k : 데이터 블록에 대한 I/O 수행 결과
HDDi_Cost : 현재 서버 i 의 HDD 에서 I/O 비용

```
1.  if (Server i SSD 에 Bk 가 존재)
2.    then Output Bk = scan Bk from Server i SSD; return Output Bk;
3.  else (Server i SSD 에 Bk 가 존재하지 않음)
4.  {
5.    HDDi_Cost = Server i HDD 에서 I/O 비용 저장;
6.    j = NameNode(HDDi_Cost);
7.    if (i!=j) // 현재 서버와 NameNode 에 의해 선정된 서버가 다름
8.      then Output Bk = scan Bk from Server j SSD;
9.    else // 현재 서버와 NameNode 에 의해 선정된 서버가 같음
10.     then Output Bk = scan Bk from Server i HDD;
11.   return Output Bk;
12. }
```

3~12 번 라인은 현재 서버 i 의 SSD 에 B_k가 존재 하지 않는 경우이며 5 번 라인으로 이동한다. 로컬 서버의 HDD 에서 B_k를 읽을 경우에 대하여 비용을 계산하고 HDDi_Cost 변수에 저장한다. 6 번 라인은 계산된 HDDi_Cost 를 NameNode 호출 시에 전달하는 것을 의미한다. NameNode 는 전달받은 HDDi_Cost 를 복제본들이 저장된 원격 서버의 SSD 에서 데이터를 읽을 경우의 비용과 비교한다. 이를 통해 블록 B_k 를 읽을 최종 저장 장치가 속해있는 서버의 번호를 반환하고 이는 j 에 저장된다. 7~10 번 라인은 NameNode 로부터 전달받은 서버와 현재 로컬 서버 정보를 비교하여 저장장치를 선정한 후에 블록 B_k를 읽는 것을 의미한다. 7, 8 번 라인은 로컬 서버와 NameNode 로부터 전달받은 서버의 번호가 다른 경우이다. 이는 원격 서버의 SSD 가 성능이 높다고 해석할 수 있다. 즉 서버 j 의 SSD 에서 블록 B_k를 읽는다. 반대로 9, 10 번 라인은 로컬 서버와 NameNode 로부터 전달받은 서버의 번호가 같은 경우이다. 즉 로컬 HDD 의 성능이 더 높다는 것을 의미한다. 그러므로 i 의 HDD 에서 블록 B_k를 읽는다.

위의 코드를 기반으로 그림 2 를 다시 해석하면 서버 2 에서 블록 B_k를 읽는 경우에 대해 B_k는 HDD 에만 저장되어 있으므로 3 번 라인에 해당한다. 5, 6 번 라인에 의해 현재 서버 2 의 HDD 에서 읽을 때 드는 비용을 계산하고 NameNode 로 전달한다. 이후 NameNode 는 복제본들이 저장된 서버 1 과 서버 3 의 SSD 에서 블록 B_k 를 읽어오는 경우의 비용을 구하여 전달받은 HDDi_Cost 와 비교한다. 마지막으로 NameNode 가 최종 접근 할 저장장치를 결정하면 해당 위치에서 블록을 읽는다.

저장장치에서 데이터를 읽을 때 I/O 비용은 Access time 과 Transfer time 의 영향을 받는다. 64MB 의 데이터를 1MB 씩 저장한다면 64 번의 Access time 이 발생

한다고 말할 수 있다. 하지만 주로 빅데이터를 다루는 HDFS 의 경우 하나의 블록에 대한 I/O 는 기본 64MB 의 chunk 단위로 실행되므로 1 번의 Access time 으로 충분한 크기의 I/O 를 처리할 수 있다고 판단된다. 이에 본 연구는 Access time 은 배제하고 비용에 큰 영향을 미치는 Transfer time 에 집중한다.

Transfer time 이란 읽은 데이터를 전송하는 데 걸리는 시간으로 전달해야 하는 총 데이터 크기에 비례하고 Bandwidth 에 반비례한다. Bandwidth 는 시간 당 전송할 수 있는 데이터의 양이므로 다음과 같이 표현된다.

$$BW = \frac{D}{T}$$

위의 식에서 BW 는 Bandwidth, D 는 읽을 데이터의 양, T 는 Transfer time 을 의미한다.

$$T_{2,HDD} = T_{HDD}$$

$$T_{3,SSD} = T_{SSD} + T_{Network}$$

Case 2 에 대해 서버 2 에서 읽을 때와 서버 3 에서 읽을 때 Transfer time 을 비교해보자. $T_{i,HDD}$ 는 서버 i 의 HDD 에서 Transfer time 을, $T_{i,SSD}$ 는 서버 i 의 SSD 에서 Transfer time 을 의미한다. Case 2-1 의 경우는 $T_{2,HDD,}$ Case 2-2 는 $T_{3,SSD}$ 이다. $T_{2\text{-}HDD}$ 인 경우는 로컬 서버의 HDD 에서 읽으므로 T_{HDD} 만 고려하면 된다. 하지만 $T_{3,SSD}$ 의 경우 원격 서버에서 데이터를 가져오므로 네트워크의 Transfer time 이 추가로 고려되어야 한다. Bandwidth 식을 기반으로 위의 식을 변형하면 다음과 같다.

$$T_{2,HDD} = T_{HDD} = \frac{D}{BW_{HDD}}$$

$$T_{3,SSD} = T_{SSD} + T_{Network} = \frac{D}{BW_{SSD}} + \frac{D}{BW_{Network}}$$

비교를 위한 최종식은 다음과 같다.

$$\frac{D}{BW_{HDD}} \; vs \; \frac{D}{BW_{SSD}} + \frac{D}{BW_{Network}}$$

각 서버를 대역폭이 100MB/S 인 HDD 한 개와 1GB/S 인 SSD 한 개로 구성한 후에 10Gbps 네트워크를 사용한다고 가정하자. 64MB 인 데이터 블록 한 개를 읽을 경우에 대해 위의 식을 적용한 결과는 다음과 같다.

$$\frac{64MB}{100MBps} > \frac{64MB}{1000MBps} + \frac{64MB}{10000Mbps}$$

하나의 블록을 HDD 에서 읽을 경우에는 0.64s, 원격 서버의 SSD 에서 읽어올 경우에는 0.0704s 이다. 후자의 경우가 빠른 Transfer time 을 제공하고 있다. 위와 같은 상황에서 만약 기존의 방식대로 로컬 HDD 로만 접근하여 데이터를 읽어온다면 원격 서버의 SSD 를 활용할 기회를 놓치게 된다. 하지만 본 연구에서 제시하는 HDFS 의 I/O 비용 기반 데이터 선택 기법을 적용한다면 항상 기대 성능이 높은 저장장치에서 데이터를 읽어오기 때문에 원격 서버의 SSD 를 활용할 수 있을 뿐만 아니라 저장장치의 I/O 성능도 향상할 수 있다.

3.3 기존 연구들과의 차별성

Hadoop 환경에서 SSD 와 HDD 를 혼합하여 저장장치를 구성함으로써 I/O 의 성능을 향상하고자 하는 연구들이 활발하게 진행되고 있다. 하지만 SSD 에 저장되어 있지 않은 데이터에 대해서는 곧바로 로컬 HDD 로 접근하여 데이터를 읽어온다. 본 연구는 기존 연구와는 달리 원격 서버 SSD 의 높은 성능을 최대한 활용하기 위해 I/O 성능이 가장 높은 곳에서 데이터를 읽어올 수 있도록 한다. 이를 위해서 로컬 HDD 와 원격 SSD 의 I/O 비용을 수식을 통해 계산하고 비교하였다. 그 결과 항상 기대 성능이 높은 저장장치에서 데이터를 읽게 함으로써 기존 방식보다 성능이 개선될 수 있음을 입증하였다. 이러한 기능이 실제로 추가된다면 HDFS 내부의 간단한 수정만으로 전체 I/O 성능을 크게 향상할 수 있다. 또한, 오픈소스 캐시 계층의 수정 없이도 SSD 캐시를 최대한 활용할 수 있다는 점에서도 장점을 가진다.

4. 결론 및 향후 연구방향

Hadoop, SQL-on-Hadoop 환경에서 HDD 의 한계를 극복하기 위해 SSD 캐시를 적용한 HDFS I/O 비용 기반 데이터 선택 기법을 제시하였다.

오픈소스 캐시 계층을 활용하여 SSD 를 HDD 의 캐시 용도로 사용하였다. 로컬 HDD 와 원격 서버의 SSD 의 I/O 수행 비용을 비교 후 최종 I/O 를 수행할 데이터를 선택하도록 하였다. 이를 통해 전체 성능 향상에 이바지할 수 있음을 입증하였다.

앞으로는 예측된 결과를 기반으로 본 아이디어를 구현함을 목표로 한다. 이를 위해서는 HDFS 레벨에서 고려되어야 하는 추가적인 요소들에 대한 검토가 필요하다.

참고문헌

[1] Apache Hadoop. http://hadoop.apache.org/

[2] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff and Raghotham Murthy “Hive - A Warehousing Solution Over a Map-Reduce Framework”, Proceedings of the VLDB

[3] bcache http://bcache.evilpiepirate.org/

[4] Cloudera Impala.http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html

[5] Hyunsik Choi, Jihoon Son, Haemi Yang, Hyoseok Ryu, Byungnam Lim, Soohyung Kim, Yon Dohn Chung “Tajo: A Distributed Data Warehouse System on Large Clusters”, ICDE 2013

[6] Karthik Kambatla,Yanpei Chen, “The Truth About MapReduce Performance on SSDs”, LISA