Haloop

Efficient Iterative Data Processing on Large Clusters

Bigdata 수업 보고서 컴퓨터 공학과 2 학년
2016104163 정의동
2019/10/20

1. Motivation

big data 분석에 있어서 반복되거나 비슷한 작업이 계속해서 수행되는 작업이 존재할 수밖에 없다. 이러한 작업을 최소화할 수 있다면 big data 분석을 더 효율적으로 할 수 있을 것이다. 기존의 MapReduce system 은 iteration 작업을 다음과 같이 수행하였다.

1. 반복과정에서 동일한 작업을 수행하여 자원을 낭비한다.
2. 더 이상 변동이 없는 작업임에도 계속해서 연산을 수행하는 비효율성을 보여준다.

따라서, Haloop 은 이러한 불필요한 반복 작업을 효율적으로 다루기 위한 고안되었다. Haloop 의 구상은 다음에서 시작되었다.

1. MapReduce 과정에서 첫 반복 시에 변하지 않는 데이터를 cache 에 저장하고 이를 재활용하자. (Ex. descendant query 에서 input table 의 변화가 없다.)
2. MapReduce 의 부가적인 작업없이 reducer 의 output 을 cache 해놓고 fixpoint 를 찾아 더 이상 reduce 과정을 수행하지 않도록 한다.

2. Background

- join step = 두 개 이상의 data set 을 하나의 data set 으로 합치는 작업.
- PageRank algorithm = 극한으로 계속해서 PageRank 를 구하다 보면 수렴 점이 나오고 이 부분이 우리가 얻고자 하는 PageRank 이다.
- Descendant query = 하나의 table 을 갖고 이를 이중 삼중으로 이용하여 원하는 데이터를 얻는 방식.
- K-mean = 방대한 데이터를 K 개의 그룹(클러스터)으로 분류하는 알고리즘이다.

3. Method or Solution

Architecture Haloop 은 Hadoop 의 architecture 를 상속하여 만들었다. Hadoop 에 추가적으로 더 구현한 부분을 살펴보면,

1) 반복 수행을 관리하기 위한 interface 를 MapReduce 프로그래밍에 추가하였다.
2) master node 에 생긴 loop control 이다. Loop control 는 사용자가 정의하는 종료시점을 만족하기 전까지 MapReduce 과정을 빠르게 동작하게 한다.
3) scheduler 들의 변화이다. Task scheduler 는 locality 에 더욱 알맞게 data 를 전달한다.
4) slave node 에 생긴 caching 과 indexing 이다. 이는 반복적의 data 를 임시 저장하고, 여기에 반복횟수를 추가하는 형태이다.
5) task tracker 의 변화이다. 기존에는 task 의 실행을 관리하는 데에서 그쳤다면, Haloop 에서는 cache 와 index 들을 관리하고 local file 에서의 cache 와 index 에 접근하여 redirect 하는 작업까지 수행한다.

이를 가능하게 하는 것은 Haloop 의 주요한 2 개의 주요 장치, scheduler 와 cache 이다.

<Scheduler>

Haloop scheduler 의 목적은 map 과 reduce 를 수행한 실제 machine 들에게 똑같은 일을 반복해서 시키는 것이다. (반복 차수는 다르지만, 똑같은 data 에 접근하는 경우) 이러한 접근법에 의해 data 는 쉽게 caching 되어 재사용이 용이하다. 첫 번째 반복 때에는 Hadoop 과 같은 작업을 수행한다. 그 후에 작업에서는 동일한 위치에서 작업을 하도록 scheduler 가 설정해준다. 이렇게 되면, loop 시에 불변하는 data 가 있을 경우 불필요한 작업을 시행하지 않을 수 있다.

이를 구현하기 위해 Scheduler 는 각각 자신의 machine 에서 실행한 map, reduce 로 생성된 data partition 의 track(기록)을 보관한다. 그리고 이것은 다음 반복에서 inter iteration locality 를 보장하는 scheduling 을 하기 위한 정보로 사용한다. master node 는 각 slave node 가 이전 반복에서 사용했던 data partition 과의 mapping 을 유지한다. 만약 slave node 가 이미 부하 상태라면 인접 노드에게 이를 맡긴다

<Cache>

이러한 inter iteration locality 덕분에 하나의 machine 에 접근하면 바로 cache 된 data 를 얻을 수 있다. Haloop 은 이러한 data 를 재활용하기 위해서 해당 node 의 local 에 저장한다. 다음은 Haloop 의 3 가지 cache 의 내용이다.

1) Reducer Input cache intermediate table 이 loop-invariant 하고 reducer input cache 가 사용 가능하다면, reducer 의 input 값을 caching 하고 해당 cache 에 대한 local index 를 생성한다. reducer input 은 reduce function 이 실행되기 전에 caching 된다. 이를 통해 reducer 는 sorting 과 grouping 하는 작업을 하지 않아도 되고, 해당 데이터를 처리하던 mapper node 는 실행되지 않아도 된다.

2) Reducer Output Cache Reducer Output Cache 는 각 각의 reducer node 의 가장 최근 output 을 가르키고 저장한다. 프로그램이 수렴하는 지를 확인하기 위해서는 현재 output 과 이전의 output 을 비교해야 하는데 해당 cache 가 분산 환경에서도 이를 가능하게 한다. 모든 reduce function 의 처리 후에 reducer process 안에서 fixpoint 를 평가하고 이를 master node 에게 보고한다.

3) Mapper Input Cache Hadoop 에서는 input data 가 위치하는 곳에서 map task 를 수행하도록 한다. 실제로는 이를 70~95% 정도 보장한다. Haloop 의 mapper input cache 는 반복이 시작되는 않는 동안 mapper 에서 local 에 없는 데이터를 읽는 것을 피하는데 목적을 둔다. 예를 들어, 첫 번째 반복 시에 mapper 가 local 에 있지 않은 데이터를 읽으려고 시도하면, mapper node 의 disk 에 caching 된다.

4) Cache Reloading cache 가 reloading 되어야 할 상황이 때때로 발생한다. 이러한 상황이 발생하더라도 Haloop 에서는 이를 지원하는 알고리즘이 존재한다.

4. Analysis of experimental results

Haloop 과 Hadoop 의 성능비교를 시행할 것이다. Haloop 의 성능 지표를 보기 위한 각 각의 cache 를 독립적으로 사용하여 성능 향상을 살펴볼 것이다.

(공간 관계상 그래프는 논문의 Figure 9~14 까지를 참고해주시기바랍니다.
https://pdfs.semanticscholar.org/8a3c/09414e8037dcf61a243af3f1c71e5c9a9ea8.pdf)

1) Reducer Input Cache

Haloop 은 항상 Hadoop 보다 성능이 좋다. 하지만, reduce 함수의 작업에 의한 cost 가 크다면, 효율성이 감소한다. 그 이유는 reducer input cache 는 데이터를 불러오는 과정을 줄이고 이로 인한 overhead 를 최소한 시키는 것인데, reduce 함수 자체의 처리 시간이 길다면, 이에 의한 이득이 희미해질 수 밖에 없다. 이러한 영향 때문에 figure 9, 10 에서 Haloop 의 효율성이 점차 낮아지는 것을 볼 수 있다. 또한, figure10 을 보게 되면 전체적으로 첫 번째 시행 때는, Haloop 이 Hadoop 보다 느린 것을 알 수 있다. 이는 Haloop 이 caching 작업을 하기 때문에 조금 더 느린 것이다. 하지만, 이후에는 더 높은 효율을 보여준다. Haloop 이 줄이는 시간은 총 2 개이고, 이는 각 각 shuffle time 과 reduce time 이다. shuffle time 은 shuffle time(reducer 에 mapper 의 input 이 들어온 순간부터 sorting 을 시작하는 순간까지의 시간) + 첫 번째 mapper 의 수행 시간(매우 작은 시간이기 때문에 통념적으로 shuffle time 에 더하여 본다.)이고, reduce time 은 sorting, grouping, reducing 작업을 하는데 걸리는 시간을 말한다. 따라서, reducer time 에서는 sorting 과 grouping 의 시간을 줄일 수 있기 때문에 figure11 에서 봤을 때 많은 시간을 절약하는 것을 볼 수 있다. 특히, 3 번째 그래프에서는 마치 reduce time 이 0 인 것처럼 보이기도 한다. figure12 에서 볼 수 있듯이 Shuffle time 역시 작업 시에 mapper 의 output 데이터의 개수를 세는 과정, mapper 를 처리하는 과정에서 발생하는 overhead 를 없앨 수 있다.

2) Reducer Output Cache reduce 작업 시에 추가적인 작업을 수행하는 것과 Haloop 의 reducer output cache 를 쓰는 것 사이의 비용차이를 살펴봤을 때, 더 효율적인 모습을 보여줬다. (figure13 을 참고.)

3) Evaluation of Mapper Input Cache slave node 간의 데이터 교환의 감소를 목표로 하는 mapper input cache 는 반복과 반복 사이에 관계가 없기 때문에 효과를 보기 어렵다. 따라서, 반복 시에 효과가 있는 데이터를 사용하게 될 경우 96%까지 효율을 올리는 모습을 볼 수 있었다.(figure14 를 참고)

5. Opinion or improvement point

해당 논문이 발표된 지는 벌써 9년이 넘었고, Hadoop의 버전은 벌써 3.x에 도달했다. Haloop 이후로 twister iMapreduce 등 여러 가지 iteration MapReduce framework들도 많이 나왔다. 하지만, Haloop은 여전히 1.x 버전에서만 시행 되고 있다. 그래서 조금은 구식이라는 생각에 조사해 보았을 때, 하나의 논문을 찾을 수 있었다. 결론은 아직도 Haloop이 좋음을 알 수 있었다. (https://dm.kaist.ac.kr/lab/papers/cikm16_expr.pdf) Haloop의 성능이 좋은 것은 Hadoop을 확장하여 만들었기 때문에 Hadoop의 위에서 진행하는 다른 iteration MapReduce 작업들보다 효율적인 모습을 보여줄 거라는 개발자들의 말이 기억이 났고, 원리를 이해하고 밑에서부터 작업하는 과정이 엄청난 효율을 보여준다는 것을 깨달았다. 또한, 이러한 방식으로 Hadoop 2.x 3.x에도 돌아갈 수 있도록 Haloop을 개발하는 것도 좋을 것 같다고 생각하였다.