Haloop: Efficient Iterative Data Processing on Large Clusters

2016104115 김원규

0. Background

PageRank: 각 웹사이트의 Rank(가치)값을 웹사이트 간의 링크 관계를 통해 반복적으로 알아내는 알고리즘
Descendant Query: 특정 사람과 본인이 몇 다리 건너 친구인지 알아내는 알고리즘.
K-Means: 각 데이터들의 cluster가 K개의 cluster 중 어디에 소속되는지 거리 계산을 통해 구하는 알고리즘
Livejournal dataset: 친구 관계를 나타내는 <key,value>로 이루어진 데이터셋
Triples dataset: RDF 형식으로 subject와 object의 관계를 특정 형태로 나타내는 데이터셋
Freebase dataset: concept_id1과 concept_id2의 관계를 특정 형태로 나타내는 데이터셋

1. Motivation

지금 수업에서 중점으로 배우고 있고, 현재 대규모 데이터 분산 처리의 기반이 되고 있는 Hadoop, MapReduce 프레임 워크가 있다. 그렇지만 현재 여러 분야에서 다루는 작업들 중 iterative program을 처리하는 경우가 많다. 즉, 반복적으로 동일한 데이터를 사용하거나, 현재 iteration의 output 데이터가 다음 iteration의 input으로 사용되는 경우의 program을 의미한다. Hadoop을 확장시킨 Haloop은 iterative program을 처리하는데 있어서 Hadoop보다 매우 빠르고, 효율적으로 처리한다.

2. Method

Haloop이 Hadoop과 다르게 작동하는 방식은 다음과 같이 이루어진다.

- Hadoop과 다른 아키텍쳐

  1. Haloop은 Hadoop과 달리 모든 job들을 통합하여 관리한다. 그로 인해 각 iteration이 이전 iteration의 데이터를 참고할 수 있다.
  2. Haloop은 'Loop Control'을 사용한다. 'Loop Control'이란 모든 Job들이 동일한 MapReduce Pair 함수를 사용하는 것이다. 반대로 Hadoop은 각 Job들이 다른 기능의 MapReduce Pair 함수를 사용한다.
  3. Haloop은 Cache와 Index를 사용한다. 이전 iteration에서 사용했던 데이터를 다시 사용하기 위해 이전 iteration에서는 각 iteration의 number를 index로 하고, key값을 통해 sorting하여 Cache에 저장한다. sort하여 저장하기 때문에 데이터를 불러올 때 속도가 빠르다.
  4. Haloop을 작동시키기 위한 Haloop만의 API를 제공하고 있다. 이로 인해 사용자는 Haloop을 만들기 위한 프로그래밍을 할 수 있다.

- Hadoop과 다른 Process

  1. Haloop은 각 iteration에서의 결과가 이전 iteration과의 결과와의 차이가 얼마 나지 않는다면, 또는 이미 마지막 iteration을 처리한다면 fixpoint 상태로 여겨 그 이후의 iteration은 반복하지 않고 program을 종료한다.
  2. Haloop은 Cache와 Index를 활용하여 이전 iteration에서 각 리듀서가 실행했던 task를 이

번 iteration에도 그 리듀서에게 할당하려고 하는 data locality의 성질을 최대화 한다. Master Node는 어떤 리듀서가 어떤 task를 수행했었는지 관계를 기억하고 있다.

- Caching과 Indexing
  총 세가지의 Cache가 존재한다.

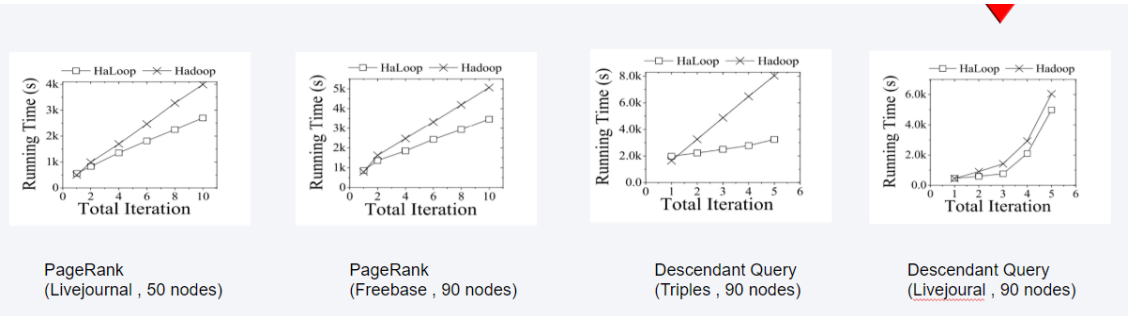
  1. Reducer Input Cache
  각 iteration에서는 Mapper의 output을 Reducer Input Cache에 저장한다. 이로 인해 반복적으로 사용되는 데이터가 Cache에 저장되고 다음 iteration에서 Reducer의 인풋으로 Mapping, Shuffling 과정 없이 바로 사용할 수 있다.
  2. Reducer Output Cache
  각 iteration의 최종 결과를 Reducer Output Cache에 저장한다. 이로 인해 fixpoint 상태를 매 iteration 마다 확인할 수 있으며, 다음 iteration의 input을 Reducer Output Cache에서 로드하여 사용한다.
  3. Mapper Input Cache
  Haloop은 Hadoop과 마찬가지로 data locality를 지키면서 map task를 slave node에 할당한다. 하지만 그렇지 못할 경우가 발생한다. 따라서 다른 node에서 해당 input 데이터가 저장된 node를 참고하여 데이터를 로드하게 된다. 이때 타겟 node의 data를 자신 node의 Mapper Input Cache에 복사하여 데이터를 사용하게 된다. 그렇게 되면 iteration 흐를수록 점차 성능이 좋아진다.

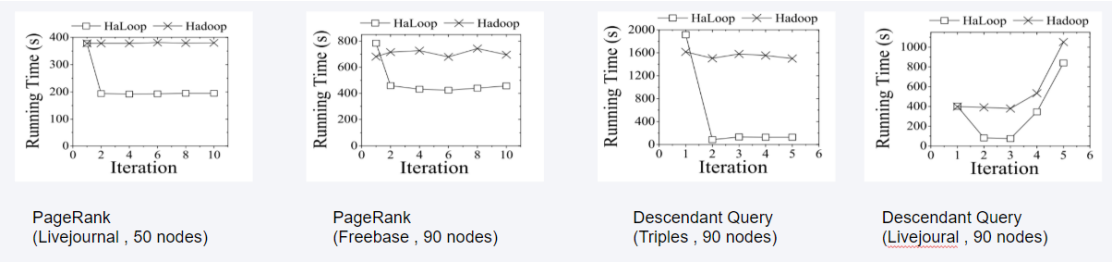
3. Analysis of experimental results

Hadoop과 Haloop의 성능을 비교하면서 설명할 것이다. 각 commodity의 세팅은 다음과 같다.
1) Livejournal -> 50개의 EC2 노드 사용, Freebase & Triples: 90개의 EC2 노드 사용.
2) PageRank -> Livejournal, Freebase 사용, Descendant Query -> Livejournal, Triples 사용

3-1. Reducer Input Cache

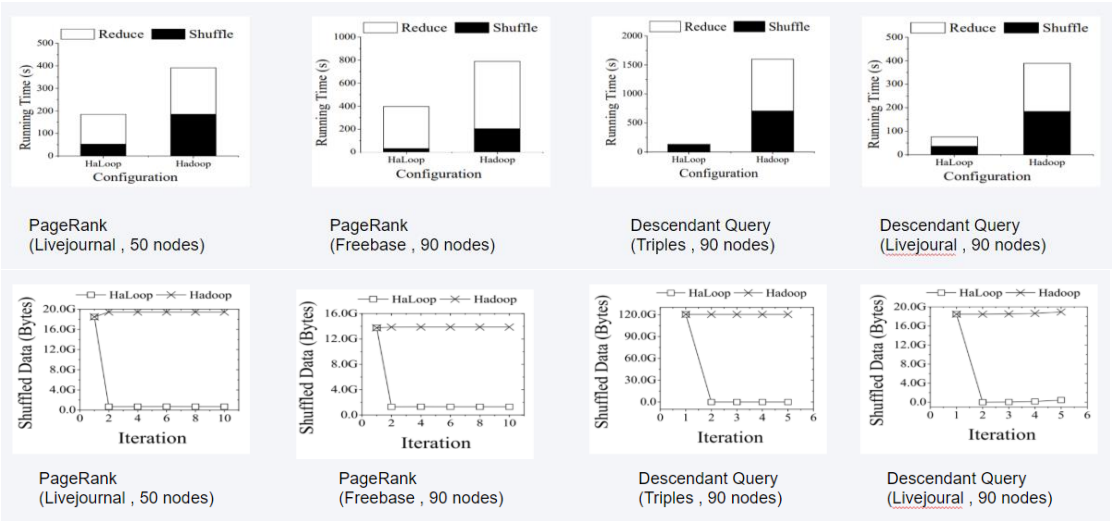
- Overall Run Time
  하지만 4번째 그래프의 경우 성능의 차이가 크지가 않다. 그 이유는 Livejournal의 데이터셋 특성에 따라 Reducer의 수행과정 중 이전 iteration의 결과들을 현재 산출하려는 결과로부터 중복 제거하려는 작업이 있어서 Reducer Input Cache를 사용한 이점이 줄어들기 때문이다.





- Join Step Run Time
  하지만 첫번째 iteration에서 Reducer Input Cache에 데이터를 저장하는 과정 때문에 느린 모습이 있다.
  또한 4번째 그래프의 경우, 위에서 언급한 중복 제거의 작업 때문에 Hadoop과 차이가 얼마 나지 않는 것을 볼 수 있다.

- Cost Distribution for Join Step & I/O in Shuffle Phase of Join Step



  Shuffle과 Reduce에 걸리는 시간이 확연이 줄어든 것, 그리고 Shuffled Data를 저장하는데 사용되는 Bytes가 Haloop이 훨씬 적음을 볼 수 있다.
  첫 그림의 3번째 그래프에서는 Haloop의 Reduce 작동이 거의 없는 것을 확인 할 수가 있다.(작동 안하는 것이 아니라 매우 조금만 작동한 것임). Dataset 특성상 Reduce가 생성해내는 새로운 output들이 거의 없기 때문에 Reduce가 그만큼 할 일이 없기 때문이다.

3-2) Reducer Output Cache: fixpoint를 찾아 program이 조기 종료되어 Running Time이 Hadoop보다 짧다.

3-3) Mapper Input Cache: 사실 Hadoop도 data locality를 매우 잘 만족하는 편이여서 Mapper Input Cache의 이점을 크게 보지 못했지만, 그래도 조금이나마 좋은 성능을 보인다.

4. Opinion or Improvement Point

Haloop은 Iterative Program에 대해 좋은 성능을 보인다. 하지만 현재 Haloop은 Hadoop의 1.x 의 확장버전으로만 존재하며, 많이 사용되고 있는 2.x의 확장버전은 존재하지 않는다. 또한, 현재 Haloop보다 더 뛰어난 플랫폼이 존재하고 있다. Haloop은 이 플랫폼의 기반이 되긴 하지만 현재 사용하긴 부적절하다고 생각한다.