HaLoop: Efficient Iterative Data Processing
on Large Clusters

Yingyi Bu_ Bill Howe Magdalena Balazinska Michael D. Ernst
Department of Computer Science and Engineering
University of Washington, Seattle, WA, U.S.A.

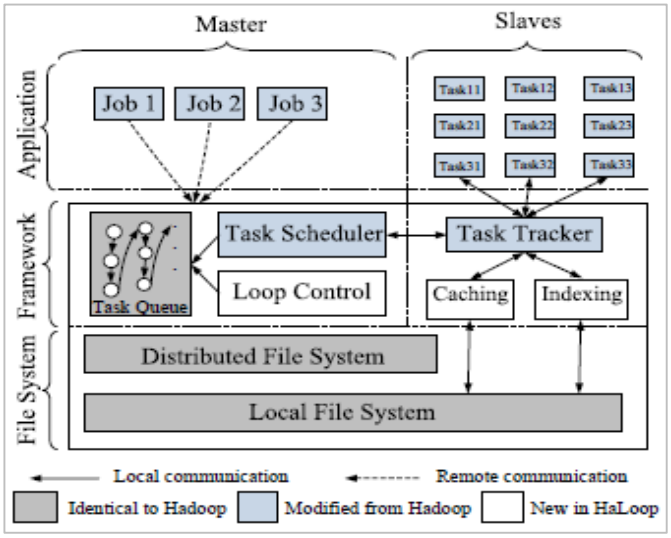
2013103891 김현하

1. Motivation

산업과 학업계에서 대용량 데이터 처리와 데이터 분석 어플리케이션에 대한 수요가 증가함에 따라 데이터를 분산처리 하기 위한 새로운 플랫폼들이 고안되었다. 그 중 하나가 MapReduce framework를 가지는 Hadoop이다. Hadoop은 cluster를 구축하여, 대용량 데이터 처리가 가능해 여러 분야에서 사용되고 있다.
하지만 Hadoop의 문제는 iterative data analysis application에 적합하지 않다는 것이다. 따라서, Page Rank, social network analysis, neural-network와 같이 반복적인 계산을 요구하는 data analysis techniques에 Hadoop을 사용하는 것은 그리 효율적이지 못하다.
따라서, iterative data analysis를 효율적으로 수행하기 위해서 HaLoop이 고안되었다. HaLoop은 기본적으로 Hadoop을 베이스로 하고, 여기에 loop-aware scheduling, Caching and Indexing을 추가하여 iterative data analysis를 지원할 수 있게 되었다.

2. Method or Solution

1) Architecture



위 그림은 HaLoop의 전반적인 구조를 보여준다. HaLoop은 Hadoop의 기본적인 distributed computing model과 architecture를 바탕으로 구성되어 있다. 위 그림에서 파란색 부분은 Hadoop과 비슷하게 작동하지만 수정된 항목들이다. Task Scheduler는 inter-iteration locality를 보장하기 위해 수정되었다. 또한, HaLoop은 iterative data analysis를 효과적으로 수행하기 위해서 Loop Control, Caching, Indexing 기능이 추가되었다.

2) Programming Model

하둡이 지원하는 iterative program은 다음과 같은 핵심 구조로 표현될 수 있다:

$$R_{i+1} = R_0 \cup (R_i \bowtie L)$$

여기서 R_0 는 초기 결과를 나타내고, L 은 invariant relation을 나타낸다. 이런 형식의 program은 fixpoint에 수렴하면 종료하게 된다. 즉, 다음 iteration에서도 결과가 변하지 않으면 종료하게 된다.
HaLoop은 loop control을 application에서 하는 Hadoop과 달리 framework 내에서 control함으로써 I/O cost를 줄일 수 있게 된다.

3) Loop-aware Task Scheduling

```
Task Scheduling
Input: Node node
// The current iteration's schedule; initially empty
Global variable: Map⟨Node, List⟨Partition⟩⟩ current
// The previous iteration's schedule
Global variable: Map⟨Node, List⟨Partition⟩⟩ previous
 1: if iteration == 0 then
 2:     Partition part = hadoopSchedule(node);
 3:     current.get(node).add(part);
 4: else
 5:     if node.hasFullLoad() then
 6:         Node substitution = findNearestIdleNode(node);
 7:         previous.get(substitution).addAll(previous.remove(node));
 8:         return;
 9:     end if
10:     if previous.get(node).size()>0 then
11:         Partition part = previous.get(node).get(0);
12:         schedule(part, node);
13:         current.get(node).add(part);
14:         previous.remove(part);
15:     end if
16: end if
```

HaLoop의 task scheduler는 iterative program를 위해 Hadoop의 scheduler보다 더 좋게 schedule을 한다. 위 그림은 HaLoop의 task scheduling algorithm을 pseudo-code로 나타낸 것이다. 이것을 살펴보면, 첫 번째 iteration에서는 Hadoop이 하는 것과 똑같이 schedule한다(line 2). Scheduling이 끝난 뒤에는, master node가 data와 node 간의 관계를 기억하고 있다(line 3 and 13). 그 이후의 iteration들에서는, scheduler가 이전 data-node 관계를 저장하고 있는다(line 11 and 12). Load가 꽉 차 있다면, master는 해당 data를 다른 node와 연결시킨다(lines 6-8).
이 task scheduling을 통해서 inter-iteration locality를 보장하게 된다.

4) Caching and Indexing

HaLoop의 task scheduler에 의해서 Inter-iteration locality이 보장되기 때문에 특정 loop-invariant data partition에 접근하는 것은 단 하나의 physical node에 의해서만 이루어진다. I/O cost를 줄이기 위해서 HaLoop은 loop invariant data partition을 저장시켜 놓고 재사용한다.

· Reducer Input Cache

Reducer Input Cache는 loop-invariant data를 해당 reduce task node의 local disk에 저장시켜 놓음으로써 불필요한 map task과정과 shuffle이 수행되는 것을 막는다. 따라서, I/O cost를 줄일 수 있게 된다.

· Reducer Output Cache

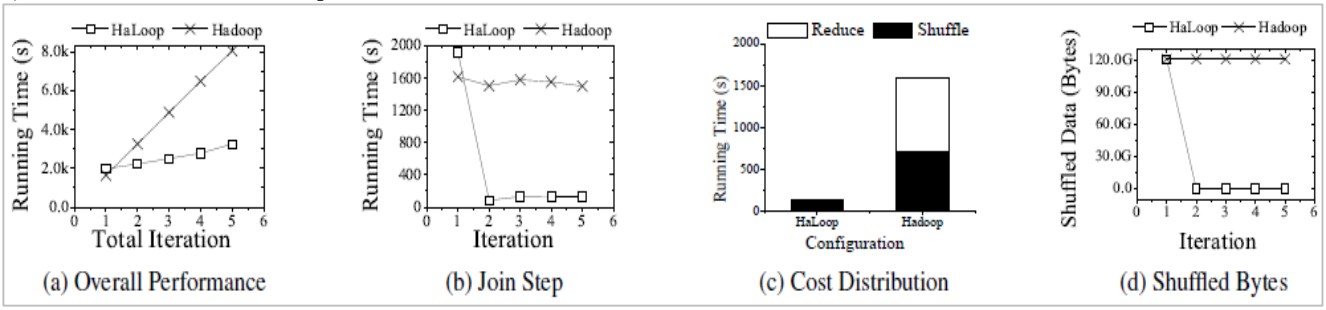
Reducer Output Cache는 가장 최근 iteration에서 발생한 reducer output을 해당 reduce task node에 저장시켜 놓는다. 그래서 현재 iteration에서 reducer output과 비교하여 fixpoint에 수렴하였는지 빠르게 비교할 수 있게 된다.

· Mapper Input Cache

Mapper Input Cache는 첫 번째 iteration에서 mapper가 data를 읽을 때 local에서 읽은 게 아니라면 그 mapper node local disk에 해당 data를 cache한다. 그래서 다음 iteration부터는 모든 mapper가 local에서 data를 읽게 된다.

3. Analysis of Experimental results

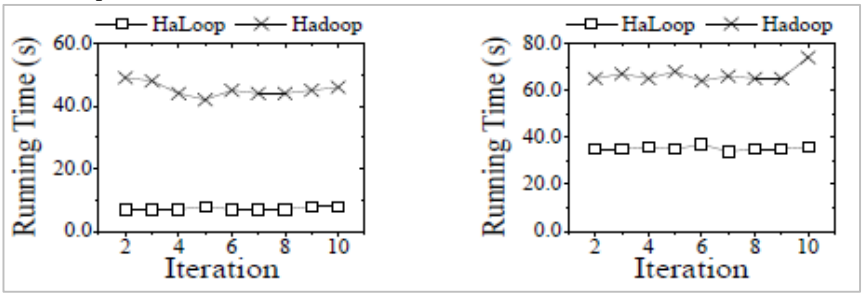
1) Evaluation of Reducer Input Cache



Descendant Query Performance : HaLoop vs. Hadoop (Triples Dataset, 90 nodes)

위의 결과는 Reducer Input Cache를 사용할 경우, 성능이 가장 효과적으로 향상된 경우의 결과를 나타낸다. Reducer Input Cache를 사용하는 HaLoop의 overall runtime과 join step time이 훨씬 더 적게 걸린다. 다만 첫 번째 iteration에서는 HaLoop이 Hadoop이 하는 작업 외에 추가적으로 cache data를 저장하고, index를 저장하는 등의 overhead가 발생하여 run time이 더 걸리게 된다. 또한, loop invariant data의 불필요한 작업을 줄임으로써 shuffle running time과 shuffle되는 data의 양을 많이 줄일 수 있다.

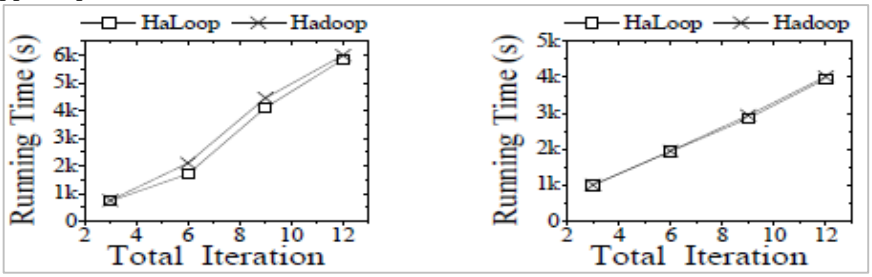
2) Evaluation of Reducer Output Cache



Fixpoint Evaluation Overhead in PageRank : HaLoop vs. Hadoop

Reducer output cache를 사용하면 fixpoint에 수렴하였는지 확인하기 위한 추가적인 map/reduce 작업을 할 필요가 없어지기 때문에 fixpoint evaluation cost가 약 40% 줄어들게 된다. 그에 따라서, 위 결과와 같이 HaLoop의 running time이 적게 걸리는 것을 확인할 수 있다.

3) Evaluation of Mapper Input Cache



Performance of k-means: HaLoop vs. Hadoop

PageRank와 descendant query application은 매 iteration마다 mapper input이 변하기 때문에 mapper input cache를 제대로 활용할 수 없다. 따라서, mapper input cache 평가는 k-means clustering 알고리즘을 사용한다. 다소 미미하지만 mapper input cache를 사용하여 HaLoop의 성능이 더 우수한 것을 확인할 수 있다. 차이가 미미한 이유는 Hadoop도 mapper의 local read를 약 70~95%로 잘 지원해주기 때문이다.

4. Opinion or Improvement point

이 논문을 통해서 대용량 데이터를 처리하기 위해 최적화 되어 있다고 생각했던 Hadoop의 문제점과 한계를 느낄 수 있었다. 또한, HaLoop과 Hadoop의 방식을 비교함으로써 Hadoop의 전반적인 구조도 이해할 수 있게 되었다. HaLoop이 iterative data analysis에 있어서 Hadoop에 비해서 성능이 향상이 되었지만, 이는 HaLoop이 loop invariant data를 갖는 iterative data analysis application에만 효과적이라는 한계 또한 내포하고 있다. 따라서, HaLoop과 같이 특정 application에만 국한되지 않고 범용적으로 general하게 지원할 수 있도록 Hadoop을 개선해나갈 필요성을 느끼게 되었다.
그리고 HaLoop에서는 index file 또한 key file과 함께 local disk에 저장을 하는데, index file은 selectivity가 낮은 data에 대해서만 큰 효과를 발휘하기 때문에 index를 사용할 것인지에 대한 API도 추가를 한다면 좋을 것 같다는 생각이 들었다.