Group assignment

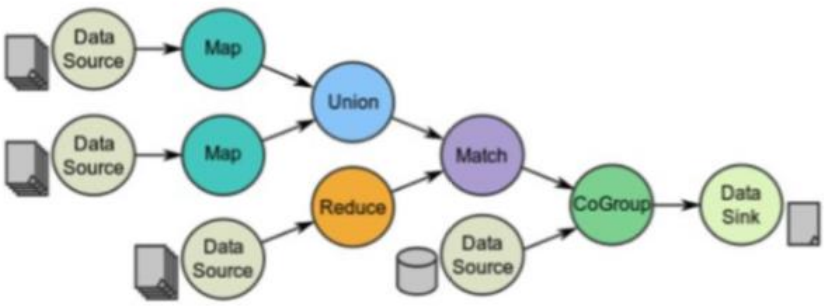
-RDD vs DataFrame vs SparkSQL queries-

Group2

주영훈,류재훈,오영택,김영민

Summary

수업시간에서 배운 내용을 토대로 spark 분산 처리 환경에서 data processing 형태에 따른 비교를 진행하였다. 분산 처리 환경에 대해서 RDD, DataFrame, SparkSQL Query의 성능 차이에 대해서 알아보고자 하여, 각 처리 방법에 대해서 처리되는 Dataset을 동일하게 하고, 각각의 속도와 코드의 라인 수, 그리고 replication의 개수와 block size에 따라서 어떻게 처리 성능에 영향을 미치는지에 대해서 서술하고자 한다. 우리는 Dataset을 가지고 4가지 주제에 대해서 그 성능을 알아보고자 하였다. 먼저, 대략적인 single node(Replication 1개)에서의 각 분산 처리 방법 별 코드 line 개수와 그 성능을 알아보고, 더 나아가 hdfs 환경에서 replication의 개수와 block size에 따른 분산 처리 성능에 대해서도 알아본다.



Analysis 1

1. Retrieve customer orders

1.1. RDD way

Code line: 20 line (involved visualization code)

Time: CPU times: user 596 ms, sys: 180 ms, total: 776 ms, Wall time: 1.28 s

1.2. DataFrame way

Code line: 8 line

Time: CPU times: user 80 ms, sys: 16 ms, total: 96 ms, Wall time: 12.4 s

1.3. SparkSQL queries way

Code line: 3 line

Time: CPU times: user 12 ms, sys: 8 ms, total: 20 ms, Wall time: 1.46 s

2. Retrieve customer orders with addresses

2.1. RDD way

Code line: 10 line

Time: CPU times: user 48 ms, sys: 12 ms, total: 60 ms, Wall time: 1.47 s

2.2. DataFrame way

Code line: 5 line

Time: CPU times: user 56 ms, sys: 16 ms, total: 72 ms, Wall time: 7.64 s

2.3. SparkSQL queries way

Code line: 3 line

Time: CPU times: user 28 ms, sys: 0 ms, total: 28 ms, Wall time: 1.99 s

3. Retrieve a list of all customers and their orders

3.1. RDD way

Code line: 12 line

Time: CPU times: user 32 ms, sys: 0 ms, total: 32 ms, Wall time: 150 ms

➔ No visualization so, it has faster than below measure.

3.2. DataFrame way

Code line: 5 line

Time: CPU times: user 64 ms, sys: 8 ms, total: 72 ms, Wall time: 2.18 s

3.3. SparkSQL queries way

Code line: 4 line

Time: CPU times: user 20 ms, sys: 12 ms, total: 32 ms, Wall time: 1.84 s

4. Retrieve a list of customers with no address

4.1. RDD way

Code line: 7 line

Time: CPU times: user 24 ms, sys: 0 ms, total: 24 ms, Wall time: 251 ms

4.2. DataFrame way

Code line: 4 line

Time: CPU times: user 32 ms, sys: 4 ms, total: 36 ms, Wall time: 525 ms

4.3. SparkSQL queries way

Code line: 3 line

Time: CPU times: user 0 ms, sys: 4 ms, total: 4 ms, Wall time: 347 s

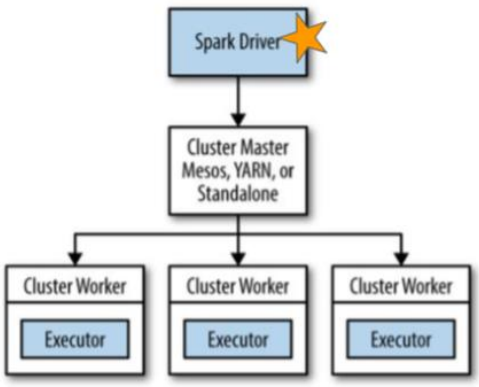
Result 1

모든 분석 방법이 시간 성능으로 보았을 때, lazy evaluation을 지원하기 때문에 좋은 결과를 낸다. 하지만 이 세가지 방법의 결과에는 차이가 있다. 위에 나타난 결과를 보았을 때, 왜 이런 결과가 나왔을까. RDD의 경우에는 분석에 있어서 어느 정도의 Lazy evaluation을 지원한다. 하지만 DataFrame은 Lazy evaluation과 transformation 방법의 최적화를 모두 진행하여 훨씬 더 좋은 결과를 낼 수 있다. RDD는 "어떻게 분석할 것인가"에 대해서 초점을 맞추고 있는 분석방법이라면, DataFrame은 "무엇을 분석할 것인가"에 대한 분석방법이다. 그래서 DataFrame은 최적화된 분석 과정을 거친다. 코드 라인 수에 비교하여 볼 때, DataFrame은 분석을 위해 새로운 schema를 만든다. 이 과정을 진행하면서 코드 라인 수가 추가된다. 그리고 결과적으로 시간을 소비하게 된다. 3가지 방법 중 가장 좋은 성과를 낸 것은 SparkSQL을 통해 query를 날리는 것이다. 이것은 SQL query를 날리는 것과 동일한 방법을 제공한다. 우리가 데이터에 query를 날릴 때, query는 자동으로 최적화가 진행된다. 그리고 데이터로 날아간다. 쿼리를 날리는 함수밖에 없기 때문에 Code line적으로 가장 좋은 성능을 자랑했다.

Analysis 2

이번에는 replication 개수와 block size에 따른 분산 처리 성능에 대해서 비교하여 본다.

우선, 한가지 주제(**Retrieve customer orders**)에 대해서 동일한 block size에서 replication 개수에 따른 분산 처리 성능에 대하여 본다.



1. Block size = 64

   block size가 64로 같을 때, replication의 개수가 많아질수록, 처리하는데 들어가는 시간이 점차 줄어드는 것을 발견하였다. 기대한 결과이다. 하지만, 특정 코드에 따라서 줄어드는 구간도 있고, replication의 개수가 2개나 3개일 때 동일한 속도를 보이는 구간도 존재하였다. 특정 코드는 replication을 할수록 더 높은 성능을 내는 반면, 그저 그렇게 큰 영향을 주지 않는 코드들도 있다는 사실을 발견하게 되었다.

2. Block size = 128

   Block size가 128로 동일할 때, replication의 개수가 많아질수록, cpu user time(사용자 입장에서 진행되는 작업)이 작게 소요되는 것을 알게 되었다. 왜냐하면 처리하는 시간은 데이터의 양과 정비례하기 때문이다. 그렇지만, replication이 점점 늘어날수록 cpu system time(system 입장에서 진행되는 작업)이 크게 소요되는 것을 발견하였다. Block size가 어느 정도 크다면, 적당한 사이즈의 데이터 작업일 때, replication에 따라서 영향을 받지 않았다. 오히려 replication을 1개로 하여 분산 처리를 진행하지 않는 것이 replication이 많을 때 들어가는 cpu system time을 줄이면서, 사용되는 node(slave 비용)또한 줄일 수 있는 방법이라고 생각한다.

3. Block size = 256

   Block size가 256으로 동일할 때, replication의 개수가 많아질수록 자연스럽게 cpu user time이 적게 소요되었다. 반면 cpu sys time은 늘어났다. 어느 정도 block size가 충분하다면, cpu system time이라는 overhead를 피하기 위해서 따로 replication을 늘리기 않고 단일 노드로 처리하는 것이 낫다.

Result 2

최종적으로 Replication을 3개로 놓고, block size의 크기는 64, 128, 그리고 256으로 변경시키면서 그 성능 추이를 관찰하였다. 블록 사이즈가 64일때는 24ms 가 들었던 작업이, block size가 128이 되면서, 20ms로 4ms가 줄었다. 하지만, block size가 256이 되어도 그 속도는 그대로 20ms였다. 적당한 크기의 block size가 있을 때에는 그 성능이 크게 변하지 않을 것을 알았다. Block size가 256일 때는 replication이 1개,2개일 때 24ms로 똑같았고, 3개일 때 20ms 로 4ms가 줄어들었다. 비용적인 측면과, 결과적인 측면을 모두 고려하여 생각하면 replication을 마냥 늘리기만 해서는 효율적인 분산 처리가 아니다.

Conclusion

데이터 분산 처리에서 여러 방법(RDD way, DataFrame, SparkSQL query)들을 실제로 코드로 돌려보면서 각각의 방법들을 더 이해할 수 있었다. 모든 방법이 적당한 Lazy evaluation을 지원하고 있었고, dataframe 경우 transformation의 최적화의 과정도 포함하고 있어 더 속도가 빨랐다는 것을 알 수 있었다. 그리고 spark에서 sql query를 날릴 수 있는 통로역할을 하는 SparkSQL에 대해서 코드 라인과 속도 성능 차이에 대해서 비교해 볼 수 있었다. Line & time의 효율성으로 보았을 때, 가장 좋았던 것은 spark sql이다. 더 나아가 분산처리 시스템에서 hdfs에 독립변인을 주어 또 다른 방면의 분석을 진행하였다. 성능에 대해서 알아보았다. Block size를 64, 128, 256으로 놓고 각각의 block size에서 replication의 개수는 1개, 2개, 그리고 3개일 때, 얼마나 데이터 분석 성능의 차이가 생기는지 알아보았다. Replication은 그 개수가 많아질수록 cpu system에서 분산을 관리하고 제어하는 시간이 소요되었다. 오히려 replication의 개수가 1개일 때, cpu에서 처리되는 system time이 없었고, node 1개에서 들어가는 cpu user time이 소요되어 사실 상 동일한 결과를 내었다. 또한 적당히 큰 block size에서는 replication의 개수가 1개, 2개일 때 큰 성능의 차이가 없었다. 거의 동일한 결과를 내었다. 이런 결과들을 보았을 때, dataset의 따라서, 분석하고자 하는 주제에 따라서 그리고 분석할 때 드는 비용에 따라서 최적의 분석 환경을 구성하는 것이 좋다.