Revision Response

Modification List

1. We add a new subsection about fault tolerance in the Implementation Section.
2. We extend the related work Section and add more discussion about shuffle optimization and DAG scheduling.
3. We add new content and modify each subsection in Section II to better detail the shuffle optimization.
4. We modify some paragraphs and add some new ones in Section 5 to better explain the FRQ model and Figure 5.
5. We update the description of Algorithm 1&2 and also discuss their complexity in Section II.
6. We modify Section V to explain how and why we choose TPC-DS as a standard benchmark to simulate the realistic industrial workload.
7. We modify Subsection V.C to discuss the different performance gain between Spark and Hadoop MapReduce.
8. In addition to the above modifications, we also paraphrase all other parts of the paper to ensure the similarity is low enough.

Reviewer 1

6. Discuss about fault tolerant:

    - What happens if some workers fail before reading the shuffle data by getBlock API?
    - Do you need to change the code of MapReduce for guaranteeing the fault tolerance?

    Answer: We add a new subsection about fault tolerance in the Implementation Section. SCache only restarts failed workers without recovering the data. If a failure happens, the $getBlock$ API called by DAG frameworks will return a data not found error. This will cause the DAG frameworks to re-compute the current stage. We leave the fault handling to the DAG frameworks. As a result, we do not need any changes in MapReduce for guaranteeing the fault tolerance.

7. we read: “After scheduling, if the new assigned id of a reduce task did not equal the original one, a re-shuffle will be triggered to transfer data to the new host.”.Why the assigned if of a reduce task may be different than the original one? If happened, what is the impact of this re-shuffleing on performance?

    Answer:

    1. In the multi-shuffle dependencies, when a new shuffle starts, the predicted parameters are accumulated with previous shuffles. And then SCache use these parameters to re-schedule the assignment. Therefore, the new assignment maybe different from the original ones.
    2. If re-shuffle happens, it will cause extra data shuffle overhead. But this re-shuffle is rare and can only affect the performance slightly.

8. Why SCache obtains better performance gains in Spark than MapReduce?

    Answer: The main reason causes the different performance gain in Spark and MapReduce is that they have quite different DAG workflows. Unlike Spark's complex DAG computing, Hadoop MapReduce has a simple DAG computing workflow which has only one Map phase and one Reduce phase. Such simple

DAG of MapReduce alleviates the performance gain of SCache's shuffle data prediction. Besides, the Terasort we used generates input data by its own. This means that heavy data skew is unlikely to happen. Therefore, the performance gain of in MapReduce is mainly rely on SCache' pre-fetching strategy and this is also why we choose Terasort benchmark which is a shuffle-heavy task. We also modify Subsection V.C to discuss the different performance gain between Spark and Hadoop MapReduce.

9. Move "Model evaluation" to the "Evaluation" section.

   Answer: OK, we move the subsection into the Evaluation.

Reviewer 2

6. Add related work:

   - "LEEN: Locality/Fairness-Aware Key Partitioning for MapReduce in the Cloud," 2010
   - Transformation-Based Monetary CostOptimizations for Workflows in the Cloud
   - the origin paper in PPoPP

   Answer: We extend the related work Section and add more discussion about shuffle optimization and DAG scheduling. We add a new subsection which names *DAG Optimization*. In the new subsection, we discuss the related work in shuffle optimization, including slow-start, Starfish, DynMR, iHadoop, iShuffle and so on. We also discuss some related work about DAG scheduling, including Skewtune, LIBRA, and ToF.

7. About experiments:

   - version of Hadoop and Spark
   - evaluate the scalability of the proposed approach with the number of nodes varied.
   - evaluate your approach with more realistic industrial workload besides benchmarks like TPC-DS

   Answer: SQL-Based Big Data Systems are widely used on industrial application, such as data mining, predictive analytics, text analytics, and statistical analysis. Because of the limited time and resource, we use standard benchmark instead of realistic industrial workload to evaluate SCache. TPC-DS models queries and data maintenance of a retail product supplier and provide a representative evaluation of performance. We believe using TPC-DS is able to evaluate the performance of SCache in industry. We also modify the first paragraph of Section V to explain how and why we choose TPC-DS as a standard benchmark to simulate the realistic industrial workload.

Reviewer 3

1. Provide a detailed mathematical optimization framework in Section II regarding the shuffle optimization, e.g., by specifying the detailed objective function and the constraints

   Answer: We add new content and modify each subsection in Section II to better detail the shuffle optimization. We first discuss the reason which causes the shuffle overhead. Then we present how we use proposed methodologies to mitigate the overhead, so that gain the optimization.

2. Discuss the complexity of Algorithm 1 and Algorithm 2.

   Answer: We update the description of Algorithm 1&2 and also discuss their complexity in Section II. For Algorithm 1, the algorithm needs $O(n)$ operations because it maintains a min-heap and traverses

$reduce$ for swapping. For Algorithm 2, the algorithm also needs $O(n)$ operations because it only traverses $reduce$ for accumulating previous $shuffle$ and re-shuffling data.

3. Figure 5 needs more explanations:

    - what the rationale behind the proposed FRQ model?
    - How can it fit the requirements for evaluating the proposed SCache?

    Answer: As discussed in paper, the FRQ model focuses on describing the overhead caused by the shuffle process. This overhead is not only depends on the input data size, computation speed and other parameters but also on which scheduling strategy it takes. By using the FRQ model, we can analyze the advantages of each strategy in various situations. Take the pre-fetching strategy in SCache as example, the FRQ model shows that pre-fetching can get better performance than the original Hadoop MapReduce strategy by reducing $T_{P\_Shuffle}$. We also modify some paragraphs and add some new ones in Section 5 to better explain the FRQ model and Figure 5.

Reviewer 4

6. Similarity score is too high.

    Answer: To ensure the similarity is low enough, we paraphrase all other parts of the paper, including Introduction, Optimization, and Implementation.