23 November 2010

Response to reviewers:

We thank the reviewers for their careful reading of our paper. Their suggestions helped us improve the clarity of what we discuss, and even sparked some new ideas for improving our algorithms. Below we list the Reviewers' comments; our responses are shown in red.

Steve Plimpton and Karen Devine

Reviewer #1: The submitted paper features a description and analysis of a new MapReduce implementation based on MPI. The features and limitations of the implementation are covered with respect to other MapReduce implementations. The implementation is applied to several graph-theoretic algorithms, and the performance of these applications is compared with applications from an established library (Trilinos).

The introductory and supporting text are, overall, very good. The paper places MapReduce in its historical context and motivates its usage. The outline of strengths is helpful- important features include support for out-of-core operations that other libraries (e.g., Trilinos) often cannot handle and the ability to use MPI directly in addition to the MapReduce abstraction. The paper then provides a good overview of the composition of operations in a MapReduce application and a summary of the out-of-core mechanisms available. The figure (Figure 1) used to provide a high-level schematic of MapReduce data flows could be enhanced- for example, it should indicate node-local and communication intensive operations.

Figure 1 is actually describing an on-processor rearrangement of data that each processor performs on its local data. Thus there is no inter-processor communication involved. We added text to the caption to clarify that all operations are done on-processor.

All performance results are shown as scaling studies for a range of problem sizes for the six graph algorithms. The results show that the MapReduce applications do not compete well with the reference implementation (Trilinos), but the authors make clear the limitations involved in using this high-level library which trades some performance for usability. Additionally, the implementation supports very large data sets which can only be processed using its out-of-core feature set. The results are presented well.

Since improved performance for small and mid-range problems is not the contribution of the paper, it could be improved by stressing other aspects of the system. Real-world out-of-core problems could be surveyed. Code size and complexity could be considered with usability as a motivation. Is the MapReduce-based graph algorithm library easier to use than the Trilinos library? Should higher-level user applications be considered? Can the MapReduce-based inefficiencies be isolated more concisely?

We added some of these comparison points to a new section 5.7 and to the final "Lessons Learned" section. In particular we highlight the relatively small line count of the MapReduce algorithms and the natural out-of-core ability it provides, as compared for example to Trilinos. The comment with respect to MR inefficiencies is addressed below.

The paper does not have a concise "Conclusion", only a "Lessons Learned" section.

Reviewer #2: This is a nice paper addressing an important current topic with a good quality open-source implementation. The overall presentation is good: the description of algorithms and implementations is clear and the performance evaluation is detailed. As the authors pointed

out, the performance of the MR-MPI implementations of some graph algorithms falls behind the matrix-based counterparts so it would be interesting to further investigate where the performance descrepency comes from and if it's possible to optimize the algorithm and implementation to bridge the performance gap.

We instrumented our MR-MPI and Trilinos implementations of PageRank to show where time is spent during the PageRank iterations. These results were added in Table 2. We further instrumented the two most expensive steps of the PageRank iterations, as shown in Tables 3 and 4. These experiments showed that considerable time is spent in reorganizing data from key-value pairs to key-multivalue pairs. We added text in sections 4.1 and 5.2 to describe alternative data layouts that could reduce the number of times the data needs to be reorganized to improve performance.

Reviewer #3: I appreciated the clear exposition of your map/reduce framework, its decomposition into a number of independent and composable stages, and the explanation of how out-of-core computations are handled.

I also liked that you presented the map/reduce implementation of a number of different graph algorithms. However, I thought the presentation of these algorithms could be improved in two ways: - The text tends to just explaining what the code does, not why, nor why this solves the specific problem (4.2 was the worst culprit here - f I understood correctly, the vertices get labeled by edge counts that are used to give the edges a direction that helps identify triangles: if this is correct a little diagram showing this would help...) - The detailed algorithms (Figures 4-9) are a little too imprecise to follow at times. In particular, when multiple map/reduces are involved the lack of names for the map/reduce operations and their inputs makes things confusing. Please be very explicit (after all the code itself is), and Figure 4 at least actually had names for the map-reduce objects.

We agree that our notation in Section 4 was deficient. I couldn't understand some of the details when I re-read it! So we re-worked and standardized the notation used in Section 4 to hopefully describe all the graph algorithms more precisely. In particular we noted which operations are performed with which MapReduce objects and their associated key/value data, so that the flow of data through each algorithm should be more clear. A discussion of the MR-MPI library's ability to allow creation of multiple MapReduce objects was added to Section 2.

Your performance is quite low when compared to non map/reduce implementations of the same algorithm. It would be interesting to know the breakdown of execution time between map, shuffle and reduce phases to get a better understanding of where your system spends its time (not necessarily for every single performance measurement, but some representative or overall measurements would be interesting).

Reviewer #2 made this same suggestion; please see the response above.

Minor: - 5.4: In Figure *13* (not 6)...

Fixed this; good catch.