TSQR, A^tA, Cholesky QR, and Iterative Refinement with Hadoop on Magellan

Austin R. Benson
University of California, Berkeley | CS 267 Spring 2011 | arbenson@berkeley.edu

The problem

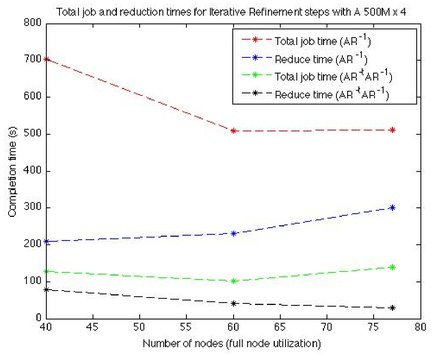
We have two main methods for computing the QR factorization of a TS matrix A:

- The communication-avoiding parallel algorithm explained by Demmel et al. [2]
- Cholesky QR by taking R = Chol(A^tA)

Cholesky QR is known to be faster but less numerically stable [3]. MapReduce has been shown to effectively compute the R factor [1]. Naively, we can compute Q by $Q = AR^{-1}$. However, this computation can have roundoff issues [3]. To maintain the orthogonality of the matrix Q, we can again compute the QR decomposition of Q. This method is called an iterative refinement of Q. We may perform several refinements of Q.

R/W Bandwidth Communication Costs

- Magellan uses SATA drives which can achieve bandwidth up to 3 GB/s.
- Experimental results show bandwidth ≈ 315 MB/s for reads and ≈ 157 MB/s writes.
- 2.5 B x 50 matrix ≈ 1 TB on disk; 1 node → 6 map, 2 reduce tasks
- 10 (80) nodes → 52.9 (6.61) seconds to read, 318.47 (39.81) seconds to write
- With Cholesky QR, number of reducers is bounded by the number of columns of A
- By combining algorithm steps, we only need to read/write $O(ncols^2)$



Future work and optimizations

- SYRK implementation of A^tA (cvxopt package or more advanced matrix algebra)
- Optimize reduce tasks so that nreducers is not limited by ncols for A^tA.
- Improve data structures used for A^tA to avoid string parsing
- Use the Hadoopy wrapper (written in Cython) instead of Dumbo
- Try other cloud environments (e.g., EC2) and infrastructures (e.g., Spark)

References

[1] Paul G. Constantine and David F. Gleich. *Tall and Skinny QR factorizations in MapReduce architectures*. MAPREDUCE 2011.

[2] James Demmel, Laura Grigori, Mark F. Hoemmen, and Julien Langou. *Communication-optimal parallel and sequential QR and LU factorizations*. UCB/EECS-2008-89. August 2008.

[3] James Demmel. Applied Numerical Linear Algebra. SIAM. 1997.

[4] Magellan at NERSC. ⟨ http://magellan.nersc.gov ⟩

MapReduce Schemes

The three main schemes used were TSQR, A^tA, and AR^{-1}. Cholesky QR is computed by feeding Q to the A^tA scheme and then performing Cholesky on the resulting matrix. I did not write the TSQR code, see [1] for that. The naive iterative refinement scheme must store the large matrix at each step. By combining the AR^{-1} and A^tA steps to form a new scheme, we can avoid the communication of writing a large matrix to disk at each time step.

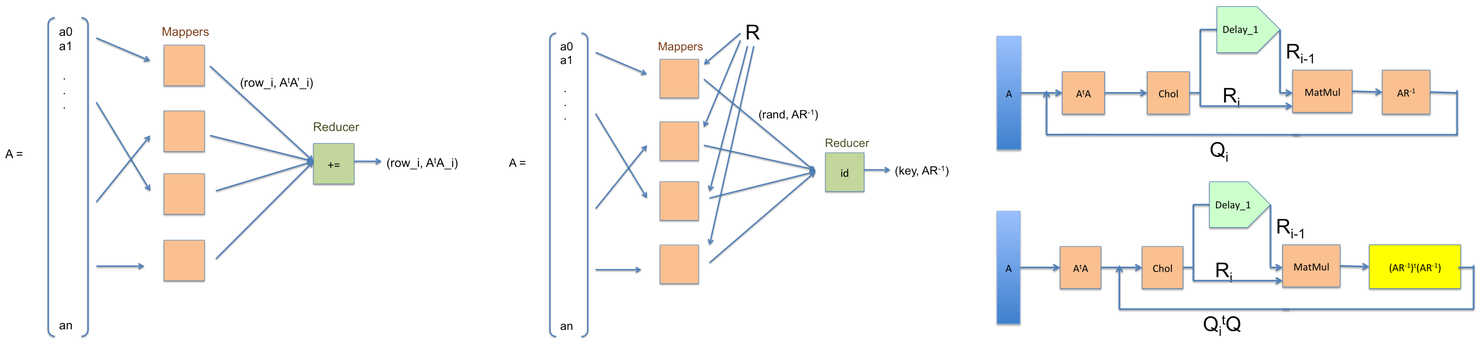
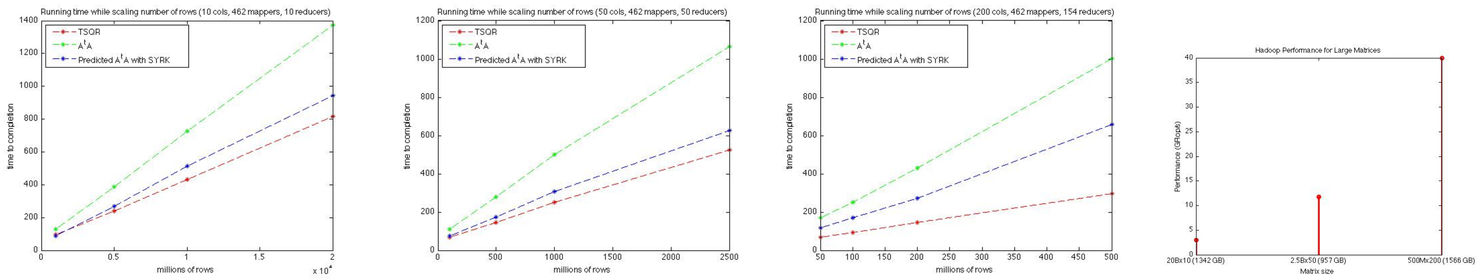


Figure: L: Scheme for computing A^tA. M: Scheme for computing AR^{-1}. R: (top) Iterative refinement pipeline, (bottom) communication-avoiding version

Experiments on NERSC's Magellan

Magellan is a cloud testbed, which typically has around 80 nodes available. Each node supports 6 map tasks and 2 reduce tasks running on dual quad-core 2.67 GHz Intel Nehalems [4]. The first set of graphs shows how the TSQR, A^tA compare, and predicted A^tA with SYRK compare. Unfortunately, the A^tA scheme runs quite slowly (possibly due to how my code uses NumPy).



Iterative Refinement Norms and Optimization Performance Modelling

- We can measure iterative refinement success by $||Q^tQ - I||_1$ (below left). Note that the 0→1 refinement dominates.
- If ncols ≤ nnodes*2 = nreducers, the Flop count per reducer is O(nrows*ncols). Let p = nreducers/ncols. We can reroute key-value pairs with key k uniformly to keys $k_0 \dots k_{p-1}$. Reducers perform row sums and then reducer k_j re-emits (k, partial row sum) to another row sum reducer (below middle).
- We introduce extra overhead by adding another job iteration, but the payoff is big for smaller numbers of columns (below, right).