

MAPREDUCE’S ROLE IN MATHEMATICS FOR EXTREME-SCALE COMPUTING

Paul G. Constantine, David F. Gleich, and Joseph W. Nichols

Position summary. *Streaming MapReduce offers an appealing paradigm for designing and managing data-intensive in-situ processing of extreme-scale simulations.*

Motivation. Current computational fluid dynamics simulations produce over 100 TB of flow field data (see Figure 1) and next-generation extreme-scale simulations will produce 10s of PB. Given the expected constraints on future computing platforms, transferring the complete data set to disk will be infeasible. Therefore, common *post*-processing paradigms for analyzing and visualizing the simulation outputs must be reimaged. The *in-situ* paradigm [1] offers a promising alternative that mitigates the write-to-disk bottleneck. Processing of simulated quantities occurs on the fly with data that is local to the computation. The processed quantities occupy much less space, and they can be transferred through the network and written to disk without a large penalty on efficiency.

One of the primary practical challenges for the in-situ paradigm is that each analysis task must be efficiently implemented along with the physics solver. These analysis tasks—for example, computing a far-field spatial correlation with delay—often involve intricate data movement, local caching, and local storage. Efficiently implementing such tasks requires detailed knowledge of the network, memory hierarchy, and computing platform. Application scientists will need help designing in-situ processing for their simulations. *Thus, there is a need for an appropriate abstraction of the in-situ processing paradigm that is both accessible to scientists and permits efficient implementation by systems experts and computer scientists. Streaming or online MapReduce [2]—a realtime framework of the MapReduce data-intensive computing invented by Google [3]—is a promising candidate for such an abstraction.*

MapReduce naturally frames the scope of analyses that can be performed efficiently, and here we encounter a broad opportunity for

mathematics research. Just as matrix computations has provided the algorithmic framework for expressing scientific computations, in-situ processing tasks can be expressed as *map* and *reduce* functions applied to the simulated outputs. Once a task has been expressed in the MapReduce framework, one can invoke system-optimized “solvers” without custom implementation on new computing platforms.

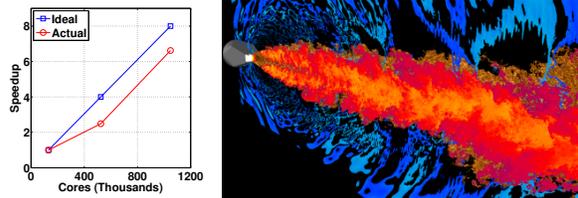


Figure 1: Nichols’ strong scaling study for large-eddy simulation of jet noise on more than one million processors from LLNL’s Sequoia BlueGene/Q with 83% efficiency [4].

Background. In 2004, Dean and Ghemawat published a description of the MapReduce model [3], which was developed by Google for data processing on large commodity clusters. MapReduce arose out of a desire to simplify writing codes for data analysis tasks at Google. The idea was to eliminate the need to write error-prone code for fault-tolerance in scalable data analysis programs. They devised the MapReduce computational model to fulfill the vast majority of their existing data analysis needs. The essential concept of the MapReduce model is shown in Figure 2. Users provide a *map* function that ingests a portion of the data, processes it, and emits an intermediate key/value pair. In the shuffle step, values with the same intermediate key are then collected and sent to a user-defined *reduce* function as a key/values pair, which processes the collected values and emits a final value. The advantage of this computational model is that parallelism and fault tolerance can be provided by the underlying implementation without any user code.

Google’s original MapReduce relies on disks to provide fault-tolerance, rendering it inappropriate for fast in-situ analyses. More recent on-

line or streaming MapReduce dynamically forwards data from the mappers to reducers. These data collection and forwarding procedures handle the task of moving relevant pieces of data from the computation to the statistical analysis and free users from writing error-prone implementations. Thus, the user implementing an algorithm need not redesign these same data movement tasks for each and every new simulation.

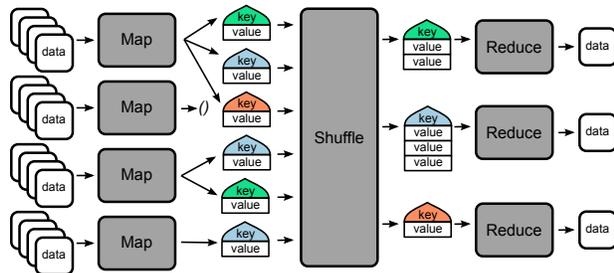


Figure 2: The MapReduce framework consists of three basic steps: map, shuffle, and reduce.

MapReduce for in-situ processing. We now describe how each component of MapReduce relates to in-situ processing; see Figure 3.

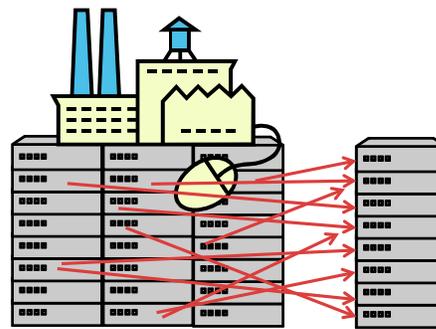
Map. This abstraction naturally expresses a user-defined function that operates on a subset of the outputs—e.g., a spatial partition of the velocity or pressure—as they are generated by the simulation. The *map* functions are instantiated and run alongside the simulation. The emitted value is derived from the subset of the outputs—e.g., a local spatial average—and the associated key identifies the value with a particular task, such as study over simulation parameters.

Shuffle. This step—invisible to the scientist—is where the tuned implementation seamlessly and efficiently forwards the emitted key/value pairs across the network. The implementation must be tuned once for each architecture, which then enables efficient processing of a wide variety of in-situ analyses.

Reduce. This user-defined function processes the set of values with a common key as its elements arrive from the *shuffle* step. Examples include averaging/interpolating across parameter values or computing two-point covariances.

Once the scientist expresses her in-situ analysis task as a sequence of *map* and *reduce* steps with the appropriate key/value inputs and outputs, then she may take advantage of the opti-

mized implementation with a few lines of code.



Map functions immediately view the simulation data on the supercomputer and forward relevant pieces for analysis.

Reduce functions process data from different portions of the simulation to compute statistics

Figure 3

Challenges for mathematics research.

There are two major opportunities here for mathematics research. The first is in the realm of the analysis, where an applied mathematician may attempt to answer: (i) Can one compute the desired quantities using an algorithm that performs efficiently within the constraints of the MapReduce framework? (ii) If such an algorithm is not straightforward, are there algorithms that compute an approximation that do perform efficiently? (iii) If such approximations exist, what is the error in these approximations? Similar questions abound in classical numerical analysis, e.g., how does one approximate the solution of a partial differential equation using matrix computations?

The second opportunity for mathematicians is to aid the systems specialists in implementing MapReduce on next-generation computing platforms. Network analysis will be a crucial component in load balancing and minimizing congestion as key/value pairs are transmitted over the network to reducers. The role of the mathematician will be to analyze and optimize communication patterns given network architectures.

- [1] K.-L. Ma, C. Wang, H. Yu, A. Tikhonova, *JPCS* **78**, 012043 (2007).
- [2] T. Condie, *et al.*, *NSDI2010* (2010), pp. 21–21.
- [3] J. Dean, S. Ghemawat, *OSDI2004* (2004), pp. 137–150.
- [4] <http://arstechnica.com/information-technology/2013/01/how-to-take-advantage-of-a-million-core-supercomputer/>.