Coded MapReduce

Salman Avestimehr

In Collaboration with: Songze Li (USC) and Mohammad Maddah-Ali (Bell Labs)
This talk is about ... The Tradeoff between Communication and Computing in Distributed Computing
Job processing is divided into **Map and Reduce tasks** across servers.

Data Shuffling (communication) is done after Map Execution to proceed to the reduction phase.

**Hadoop**: An open-source implementation of MapReduce.
Example: Word-Counting using MapReduce

- **Job:** Count the numbers of words in a book
  - Book has 6 chapters
  - Interested in 3 words, denoted as A, B, C
  - Computing resources: 3 servers interconnected with a multicast LAN network

- **Server 1:** count A
  - Chapter 1 → (a₁,b₁,c₁)
  - Chapter 2 → (a₂,b₂,c₂)

- **Server 2:** count B
  - Chapter 3 → (a₃,b₃,c₃)
  - Chapter 4 → (a₄,b₄,c₄)

- **Server 3:** count C
  - Chapter 5 → (a₅,b₅,c₅)
  - Chapter 6 → (a₆,b₆,c₆)

- **Each server processes 2 chapters** (minimum processing)
  - Intermediate value a₁ indicates that A appears a₁ times in Chapter 1

- **12 Intermediate values are exchanged for Reduce tasks**
Communication Load of MapReduce

- A general MapReduce job
  - Input file contains $N$ subfiles
  - $Q$ output keys: $W_1, \ldots, W_Q$
    - Each key needs to be evaluated using $N$ intermediate values, one from a subfile
  - $K$ available servers connected via a multicast LAN network
    - Network capacity: 1 intermediate value per channel use

- Communication Load of the Conventional MapReduce
  - Each server evaluates $Q/K$ output keys
  - Each server processes $N/K$ subfiles
    - Needs another $Q/K$ ($N-N/K$) intermediate values
  - Communication Load (# of channel uses during data shuffling):
    \[ L_{MR} = QN \left(1 - \frac{1}{K}\right) \]
    - Communication is a bottleneck!

Three parameters: $K$, $N$, $Q$
• How can we reduce the communication load?
  • Do more computation: no communication needed with K times more computation
• How to optimally trade communication with computing?
Example:

- Each server processes 4 chapters (2 times more) in the word-counting example.

Server 1: count A

Server 2: count B

Server 3: count C

- 6 intermediate values are exchanged.
- If each server does \( r \) times more processing, can save communication load by

\[
\frac{K - 1}{K - r} = 1 + \frac{r - 1}{K - r}
\]

- We call this repetition gain.
Uncoded Shuffling Scheme

Can we do better? Can we get a non-vanishing gain as $K$ gets large?

\[ \approx 1 \text{ (for large } K \text{ and fixed } r)! \]
Main Result: Coded MapReduce

- Slashing the communication load by coding
Main Result: Coded MapReduce

**Theorem:** Consider a job of using $K$ servers to evaluate $Q$ keys in an input file consisting of $N$ subfiles, where each subfile is repetitively assigned to $p (p \leq K)$ servers, and is randomly and uniformly mapped at $r (r \leq p)$ of those servers. Then the communication load of the proposed Coded MapReduce scheme $L_{CMR}(r)$ satisfies

$$\lim_{N \to \infty} \frac{L_{CMR}(r)}{L_{MR}} = \frac{K - r}{K - 1} \frac{1}{r}$$

(coding gain: scales linearly with # of repetitive processing)

**repetition gain**

N = 1200 subfiles, Q = 10 output keys, 
K = 10 servers

When doubling the processing load ($r$), Coded MapReduce can cut the communication load by more than 50%. Without coding, the reduction would have been only a factor of $(K-1)/(K-2)$, which vanishes as $K$ increases.
Another Interpretation of the Result

B. Tradeoff Between the Map Processing Time and the Communication Load

As demonstrated in Theorem 1, Coded MapReduce can cut off the communication load by a factor of \( \frac{1}{K} \frac{1}{r} \left( rK \right) \), where \( \frac{1}{K} \frac{1}{r} \) is the repetition gain from having the servers know the values from more subfiles locally by mapping each subfile at more than one server (\( rK \)), and a coding gain of \( rK \) on top of the repetition gain can be achieved by utilizing the multicast opportunities as described by the Coded MapReduce scheme presented in Section V-A. Both the repetition gain and the coding gain increase when the mapping of each subfile is repeated at more servers (larger \( r \)). However, the average overall processing time for the Map tasks becomes longer to wait for more servers to finish mapping the same subfile.

![Graph showing the tradeoff between the processing time and communication load](image)

- Conventional MapReduce
- Coded MapReduce

Fig. 5. The performance of the Coded MapReduce for a job with \( N = 1200 \) subfiles, \( Q = 10 \) keys, \( K = 10 \) servers and \( pK = 7 \) repetitive assignments of each subfile. The Map processing rate at each server \( \mu = 500 \). This figure demonstrates the tradeoff between the processing time to map one subfile at \( rK \) servers and the communication load in the following shuffling phase.

Fig. 6. The performance of the Coded MapReduce for a job with \( N = 1200 \) subfiles, \( Q = 10 \) keys, \( K = 10 \) servers and \( pK = 7 \) repetitive assignments of each subfile. The Map processing rate at each server \( \mu = 500 \). This figure demonstrates the tradeoff between the overall Map processing time and the communication load in the following shuffling phase.

VIII. Conclusion

In this paper, we propose Coded MapReduce, a joint framework that aims to improve the performance of data shuffling in a MapReduce environment. We demonstrate through analytical analysis that, by carefully assigning repetitive Map tasks onto different servers and smartly coding the transmitted message bits across keys and data blocks, Coded MapReduce can slash the inter-server communication load by a factor that grows linearly with number of servers in the system. An interesting future direction is to consider a software implementation of Coded MapReduce in Hadoop clusters to demonstrate the coding gain for reducing the inter-server communication load of the shuffling phase in such clusters.
Key Coding Technique: Coded Multicast

- Sending bit-wise XORs of the required intermediate values
  - Each server cancels the undesired messages using local processing results
  - Simultaneously satisfies multiple data requests in a single channel use

Using Coded MapReduce, data shuffling can be finished in 3 channel uses
Key Challenge

- Careful assignment of MapTasks to servers, such that multicast coding opportunity of size $r$ arises in the shuffling phase
- Example: $K=Q=4$, $N=12$, $r=2$
Key Challenge

- Careful assignment of MapTasks to servers, such that multicast coding opportunity of size $r$ arises in the shuffling phase
- Example: $K=Q=4$, $N=12$, $r=2$
• Careful assignment of MapTasks to servers, such that multicast coding opportunity of size $r$ arises in the shuffling phase
• This problem resembles the prefetching design in cache networks!

- One major difference: once a sub-file is assigned to a server, all keys for that sub-file should be calculated at that server
- A random (uniform) assignment approach still works!
Assign the task of mapping each subfile to \( r \) servers

1. Partition the \( N \) subfiles into \( \binom{K}{r} \) batches, each contains \( g \) subfiles
   - Suppose \( N = g \binom{K}{r} \) for some integer \( g \)
2. Assign each batch to a unique subset of \( r \) servers

Word-Counting example:
- \( N = 6, K = 3, r = 2 \) \( \Rightarrow \) 3 batches, each contains 2 subfiles
We show that for each subset $S$ of $r+1$ devices:

- Each device can satisfy the demand of the remaining $r$ devices with only a single coded multicast transmission.

Word-Counting example: $S = \{1,2,3\}$

$$L_{CMR}(r) = \lim_{N \to \infty} \frac{L_{CMR}(r)}{L_{MR}} = \left(1 - \frac{r - 1}{K - 1}\right) \cdot \frac{1}{r}$$
Coded MapReduce: Communication Load

Is this the optimal tradeoff?

\[ \lim_{{N \to \infty}} \frac{L_{\text{CMR}}(r)}{L_{\text{MR}}} = \left( 1 - \frac{r - 1}{K - 1} \right) \cdot \frac{1}{r} \]
Cut-set Lower Bounds on Communication Load

- When mapping a subfile, the server knows the intermediate values of all keys in that subfile
- Communication Load is independent of key distribution
- First cut

\[
3(R_2 + R_3) + 12 \geq 18
\]

Map outcomes of Server 1:
\[
(A,a_1)[1], (B,b_1)[1], (C,c_1)[1]
(A,a_2)[2], (B,b_2)[2], (C,c_2)[2]
(A,a_3)[3], (B,b_3)[3], (C,c_3)[3]
(A,a_4)[4], (B,b_4)[4], (C,c_4)[4]
\]

\[
4 \times 3 = 12F \text{ bits}
\]

- Repeat for Server 2 and 3
- Communication Load \( L = R_1 + R_2 + R_3 \geq 3 = L_{CMR} \)
- In general: \( L(r) \geq \frac{QN}{K} \cdot \frac{K - r}{K - 1} \)
• Second cut
  • Focus on the first s servers
  • Consider key distributions on these s servers
  • For example K = Q = 4, choose s = 2

\[ 2L + 4(|M_1| + |M_2|) \geq 4N \]

• In general:
Approximate Optimality of Coded MapReduce

- Using cut-set type bounds, we obtain

\[ L^*(r) \geq \frac{QN}{K} \cdot \max \left\{ \frac{K - r}{K - 1}, \max_{s \in \{1, \ldots, K\}} s \left(1 - \frac{r}{\left\lfloor \frac{K}{s} \right\rfloor}\right) \right\} \]

- Communication load of Coded MapReduce:

\[ L_{CMR}(r) = \frac{QN}{K} \cdot \frac{K - r}{r} + o(N) \]

**Theorem**: For any MapReduce job of using \( K \) servers to evaluate \( Q \) keys in an input file consisting of \( N \) subfiles, where each subfile is repetitively assigned to \( p (p \leq K) \) servers, and is randomly and uniformly mapped at \( r (r \leq p) \) of those servers, the proposed Coded MapReduce scheme achieves the minimum communication load up to a constant multiplicative factor. More precisely:

\[ \lim_{N \to \infty} \frac{L_{CMR}(r)}{L^*(r)} < 3 + \sqrt{5} \]
Concluding Remarks

- Coded MapReduce provides a near optimal framework for trading “computing” with “communication” in distributed computing

- Many future directions:
  - Impact of Coded MapReduce on the overall run-time of MapReduce
  - General server topologies
  - Multi-phase MapReduce