Coded Terasort

Salman Avestimehr

USC

joint work with

Songze Li (USC), Sucha Supittayapornpong (USC), and Mohammad Maddah-Ali (Bell-Labs)

ITA
Feb. 2017
Distributed Sorting of Large Data

- Terasort MapReduce
Distributed Sorting of Large Data

- Terasort MapReduce

III. TeraSort

A. System Overview

B. Performance Evaluation

1) File Placement:

   - In File Placement, the entire KV pairs are split into disjoint files, and each file is placed on one node. For example, the domain of the keys can be 10-99.

2) Key Domain Partitioning:

   - Key Domain Partitioning is shown in Fig. 3. We next discuss each component in detail.

3) Map Stage:

   - Each node hashes each KV pair, denoted by key, and any corresponding node, which sorts all KV pairs in that partition locally.

4) Shuffle Stage:

   - A dotted box represents the final sorted list of a node. A dotted box is shown in the figure.

   - The partitions are created in the ascending order (uncode).

   - Since the partitions are created in the ascending order, the performance of TeraSort is viewed in Section II can be applied to significantly improve the performance of the total execution time achieved by the procedure on the file.

Table I

<table>
<thead>
<tr>
<th>K</th>
<th>F</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>47</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>F</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>87</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>F</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>78</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>90</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>F</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>99</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>F</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>87</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>F</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>78</td>
<td>90</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>F</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

98.4% of the total execution time was known all intermediate values from all nodes and 98.4% of the total execution time was calculated at Node 1. To understand the performance of the time spent in the Map stage, by the end of the Reduce stage, i.e., Sort, TeraSort is viewed in Section II can be applied to significantly improve the performance of the total execution time achieved by the procedure on the file.

1.86

2.35

10.47

961.25

12

87

52

72

39

87

16

64

30

80

47

99

[0,25)

[25,50)

[75,100]

[50,75)
Distributed Sorting of Large Data

- Terasort MapReduce
### Distributed Sorting of Large Data

- **Terasort MapReduce**

![Diagram of Terasort MapReduce](image)

**Algorithm Description**

1. **File Placement:** Each node hashes each file into 4 groups of key-value (KV) pairs, meaning each input KV pair consists of a key and a value. For example, the domain of the keys can be 10-byte integers, and the domain of the values can be arbitrary.

2. **Key Domain Partitioning:** The domain of the keys is split into 4-byte integer partitions, and the domain of the values can be arbitrary.

3. **Map Stage:** Each node hashes each KV pair in its locally stored file into one of the key domain partitions. In Key Domain Partitioning, the domain of the keys can be partitioned into 10-byte integers, and the domain of the values can be arbitrary.

4. **Shuffle Stage:** Each node sorts all KV pairs in a partition, for all 4 intermediate value partitions, for all the partitions. Each node sorts all KV pairs in any given partition, for all the domains, and for all the partitions.

5. **Reduce Stage:** Node 1 forms a sorted list of the entire input data. Each node sorts all intermediate values in the partition, for all the domains, and for all the partitions.
Distributed Sorting of Large Data

- Terasort MapReduce
Distributed Sorting of Large Data

- Terasort MapReduce

![Diagram of distributed sorting process]

**Key Domain Partitioning:**

- Each input file is hashed into 4 groups of KV pairs.
- A simple node sorts KV pairs belonging to its assigned partition.
- Key domain partitioning, map stage, shuffle stage.

**A. Algorithm Description**

1. **Hashing:**
   - The input data is split into disjoint files, and each file is placed on one of the 16 nodes.
   - The hashing is performed for distributed sorting over the 16 nodes.

2. **Key Domain Partitioning:**
   - Each file is hashed into 4 groups of KV pairs.
   - The 4 groups are split into 4 partitions, one for each partition.
   - For each of the 4 partitions, the 4 groups of KV pairs are delivered to the node that is responsible for sorting the KV pairs whose keys fall into that partition.

3. **Map Stage:**
   - Each node processes its locally stored file and generates intermediate values.

4. **Shuffle Stage:**
   - The intermediate values are shuffled across all nodes.
   - Redistribute the files across all nodes.

5. **Reduce Stage:**
   - Each node processes its intermediate values and outputs the final sorted result.

**Figure 3:** Illustration of conventional distributed sorting of a large amount of data. The input data for all files is hashed into 4 groups of KV pairs, one for each partition. For each of the 4 partitions, the 4 groups of KV pairs are delivered to the node that is responsible for sorting the KV pairs whose keys fall into that partition.
Distributed Sorting of Large Data

- Terasort MapReduce

Figure 3. Illustration of conventional sorted. They are split into disjoint files, and each file is placed on one of the partitions. In Map Stage, each node hashes each string. Let us consider a partition. In Reduce Stage, each node locally computes the entire input data.

Let \( p \) be a conventional algorithm for dis-sorting that partition. In Reduce Stage, each node sorts the collection of the intermediate values, e.g., sorting integers. Since the partitions are created in the ascending order of their keys, the hashing aims to sort the input data according to their keys.
Distributed Sorting of Large Data

- Terasort MapReduce

Three phases: Map-Shuffle-Reduce

1- Mapping each data point to an interval
2- Shuffling mapped data points among nodes
3- Reducing to sorted data
Terasort Performance

- Implementation over Amazon EC2 clusters

<table>
<thead>
<tr>
<th>Process</th>
<th>Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>1.86</td>
</tr>
<tr>
<td>Pack</td>
<td>2.35</td>
</tr>
<tr>
<td>Shuffle</td>
<td>945.72</td>
</tr>
<tr>
<td>Unpack</td>
<td>0.85</td>
</tr>
<tr>
<td>Reduce</td>
<td>10.47</td>
</tr>
<tr>
<td>Total</td>
<td>961.25</td>
</tr>
</tbody>
</table>

Performance of Terasort sorting 12GB data with $K = 16$ nodes and 100 Mbps network speed.
Terasort Performance

• Implementation over Amazon EC2 clusters

<table>
<thead>
<tr>
<th>Map (sec.)</th>
<th>Pack (sec.)</th>
<th>Shuffle (sec.)</th>
<th>Unpack (sec.)</th>
<th>Reduce (sec.)</th>
<th>Total (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.86</td>
<td>2.35</td>
<td>945.72</td>
<td>0.85</td>
<td>10.47</td>
<td>961.25</td>
</tr>
</tbody>
</table>

**Performance of Terasort sorting 12GB data with** $K = 16$ **nodes and 100 Mbps network speed**

- Process twice more data at the nodes
  - Map and Reduce increase by $\approx 100\%$
  - Shuffle reduces by $(1-1/8)/(1-1/16) \approx 7\%$
  - Overall speed up by $\approx 5\%$
Terasort Performance

• Implementation over Amazon EC2 clusters

<table>
<thead>
<tr>
<th>Step</th>
<th>Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>1.86</td>
</tr>
<tr>
<td>Pack</td>
<td>2.35</td>
</tr>
<tr>
<td>Shuffle</td>
<td>945.72</td>
</tr>
<tr>
<td>Unpack</td>
<td>0.85</td>
</tr>
<tr>
<td>Reduce</td>
<td>10.47</td>
</tr>
<tr>
<td>Total</td>
<td>961.25</td>
</tr>
</tbody>
</table>

Performance of TeraSort sorting 12GB data with $K = 16$ nodes and 100 Mbps network speed

Coded Terasort

- Process twice more data at the nodes
  - Map increase by $\approx 100\%$
  - Shuffle reduces by $\approx 56\%$
  - Overall speed up by $\approx 54\%$

\[
L(r) = \frac{7\%}{50\%} \times 2
\]
Terasort Performance

- Implementation over Amazon EC2 clusters

### Performance of Terasort

<table>
<thead>
<tr>
<th>Partition</th>
<th>Map (sec.)</th>
<th>Pack (sec.)</th>
<th>Shuffle (sec.)</th>
<th>Unpack (sec.)</th>
<th>Reduce (sec.)</th>
<th>Total (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-50</td>
<td>1.86</td>
<td>2.35</td>
<td>945.72</td>
<td>0.85</td>
<td>10.47</td>
<td>961.25</td>
</tr>
</tbody>
</table>

**Performance of Terasort** sorting 12GB data with \( K = 16 \) nodes and 100 Mbps network speed.

Coded Terasort

- Process twice more data at the nodes
  - Map increase by \( \approx 100\% \)
  - Shuffle reduces by \( \approx 56\% \)
  - Overall speed up by \( \approx 54\% \)
Terasort Performance

- Implementation over Amazon EC2 clusters

<table>
<thead>
<tr>
<th></th>
<th>Map (sec.)</th>
<th>Pack (sec.)</th>
<th>Shuffle (sec.)</th>
<th>Unpack (sec.)</th>
<th>Reduce (sec.)</th>
<th>Total (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.86</td>
<td>2.35</td>
<td>945.72</td>
<td>0.85</td>
<td>10.47</td>
<td>961.25</td>
</tr>
</tbody>
</table>

Performance of Terasort sorting 12GB data with $K = 16$ nodes and 100 Mbps network speed

Coded Terasort

- Process five times more data at the nodes
  - Map increase by $\approx 400\%$
  - Shuffle reduces by $\approx 80\%$
  - Overall speed up by $\approx 80\%$
How Does Coded Terasort Work?

Empirical Evaluation of Coded Terasort?
Coded Distributed Computing

MapReduce-Type Frameworks

\[ L(r) \]

Computation Load

\[ \times K \]

\[ \times r \]

state-of-the-art (uncoded)

Coded Distributed Computing

average # times each data point is mapped

# of exchanged intermediate values
Key Idea

- Careful assignment of Map tasks to servers, such that multicast coding opportunity of size $r$ arises in the data shuffling phase.
Example: Coded Distributed Computing

6 Inputs, 3 Servers, 3 Functions, r=2
Example: Coded Distributed Computing

6 Inputs, 3 Servers, 3 Functions, $r=2$
Example: Coded Distributed Computing

6 Inputs, 3 Servers, 3 Functions, r=2

Each coded packet is useful for two servers

$L_{uncoded} = 6$

$L_{coded} = 3$
Illustration of Coded Terasort

- Consider r=2, meaning that each data point is stored at two servers.
Illustration of Coded Terasort

- Consider \( r=2 \), meaning that each data point is stored at two servers

![Diagram showing the Coded Terasort process with nodes numbered 1 to 4 and data ranges partitioned into [0,25), [25,50), [50,75), and [75,100]. Each node multicasts its data ranges to other nodes.]

In this stage, each node hashes each KV pair in the locally stored file to the partition its key belongs to. The intermediate values generated from the file are then distributed among the nodes. Each node sorts KV pairs belonging to its assigned partition. A simple distribution of the input files such that every subset of \( S \) forms an experiment on Amazon EC2 to sort 12GB of the entire input data.

Let \( S \) be a set of \( |S| \) strings. Each node, \( i \), sorts all KV pairs in its assigned partition. The algorithms reduce the total execution time. We generate an intermediate value for all \( K \) pairs in the Partition \( P \) with \( I \), for all \( K \) as follows:

Let us consider an example illustrating for sorting that partition. In Reduce Stage, each node locally sorts the assigned partition. Each node multicasts each of its \( S \) nodes, whose indices are denoted by a set corresponding node, which sorts all KV pairs in that partition locally. In this stage, each node repeatedly performs the Map stage, i.e.,

- CodedTeraSort mapped on CMR in (4), when executing the same sorting job using Decoding.
Illustration of Coded Terasort

- Consider $r=2$, meaning that each data point is stored at two servers.
Illustration of Coded Terasort

- Consider r=2, meaning that each data point is stored at two servers
Illustration of Coded Terasort

• Consider r=2, meaning that each data point is stored at two servers.
Objective: Each server can code intermediate values that are simultaneously useful for $r$ other servers.

We assign the files such that:

- for every subset $S$ of $r+1$ servers,
- and for every subset $T$ of $S$ with $r$ servers,
- Servers in $T$ share an intermediate value needed by the server $S \setminus T$.

**How to Do the Task Assignments?**

$N$ Files, $K$ Servers, Comp. Load $r$
Connection with Coded Caching

This task assignment is similar to caching strategy in “coded caching”.

- In coded caching, in placement phase, the demand of the each user is not known.
- In task assignment for distributed computing, the server that reduces a key is known!
Theorem:

Coded Distributed Computing achieves communication load of

\[ L_{\text{coded}} = \frac{L_{\text{uncoded}}}{r} = \left(1 - \frac{r}{K}\right) \frac{1}{r} \]
Can we do better?

Theorem:

The proposed Coded Distributed Computing is **optimal**.

\[ L^*(r) = L_{\text{coded}}(r) = \frac{L_{\text{uncoded}}(r)}{r} \]

Communication Load \( \times r \) x Computation Load \( \times K \) \( \sim \) constant
Impact of Coded Distributed Computing

- We can reduce the total computation time by trading Map time with Shuffle time

\[ T_{\text{total}} = \mathbb{E}[T_{\text{Map}} + T_{\text{Shuffle}} + T_{\text{Reduce}}] \]

\[ T_{\text{total, CDC}} = \min_r \mathbb{E}[rT_{\text{Map}} + \frac{T_{\text{Shuffle}}}{r} + T_{\text{Reduce}}] \]

- E.g., by choosing \( r=5 \) in this Terasort scenario, Coded Terasort promises to provide 80% speed up!

<table>
<thead>
<tr>
<th></th>
<th>Map (sec.)</th>
<th>Pack (sec.)</th>
<th>Shuffle (sec.)</th>
<th>Unpack (sec.)</th>
<th>Reduce (sec.)</th>
<th>Total (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.86</td>
<td>2.35</td>
<td>945.72</td>
<td>0.85</td>
<td>10.47</td>
<td>961.25</td>
</tr>
</tbody>
</table>

**Performance of TeraSort sorting 12GB data with \( K = 16 \) nodes and 100 Mbps network speed**
Practical Challenges

• Multicast capability
  ➢ Coded Distributed Computing takes advantage of coded multicast to reduce comm. load
  ➢ Network-layer multicast is disabled in most data center networks like Amazon EC2
  ➢ We can instead use application-layer multicast by using Massage Passing Interface (MPI)

• Encoding/Decoding Complexity
  ➢ The delays caused by Encoding/Decoding cannot be ignored
  ➢ We limit ourselves to small coding gains

• Storage Optimization
  ➢ Coded Distributed Computing optimizes input file placement to enable coded multicast
  ➢ Data is often stored without prior knowledge of the computation tasks (e.g., HDFS, GFS)
  ➢ We can instead use random placement!
Experiments over Amazon EC2

• We have implemented Coded Terasort over Amazon EC2

<table>
<thead>
<tr>
<th></th>
<th>Map (sec.)</th>
<th>Pack (sec.)</th>
<th>Shuffle (sec.)</th>
<th>Unpack (sec.)</th>
<th>Reduce (sec.)</th>
<th>Total (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TeraSort</td>
<td>1.86</td>
<td>2.35</td>
<td>945.72</td>
<td>0.85</td>
<td>10.47</td>
<td>961.25</td>
</tr>
</tbody>
</table>

**Performance of TeraSort Sorting 12GB Data with K = 16 Nodes and 100 Mbps Network Speed**

• Results:

<table>
<thead>
<tr>
<th></th>
<th>CodeGen (sec.)</th>
<th>Map (sec.)</th>
<th>Pack/Encode (sec.)</th>
<th>Shuffle (sec.)</th>
<th>Unpack/Decode (sec.)</th>
<th>Reduce (sec.)</th>
<th>Total Time (sec.)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>TeraSort</td>
<td>–</td>
<td>1.86</td>
<td>2.35</td>
<td>945.72</td>
<td>0.85</td>
<td>10.47</td>
<td>961.25</td>
<td></td>
</tr>
<tr>
<td>CodedTeraSort: r = 3</td>
<td>6.06</td>
<td>6.03</td>
<td>5.79</td>
<td>412.22</td>
<td>2.41</td>
<td>13.05</td>
<td>445.56</td>
<td>2.16×</td>
</tr>
<tr>
<td>CodedTeraSort: r = 5</td>
<td>23.47</td>
<td>10.84</td>
<td>8.10</td>
<td>222.83</td>
<td>3.69</td>
<td>14.40</td>
<td>283.33</td>
<td>3.39×</td>
</tr>
</tbody>
</table>

Coded Terasort provides 50% - 70% speed up
Conclusions and Future Directions

- Coding plays a fundamental role in distributed computing by enabling optimal tradeoffs between resources.

  - Coded Terasort empirically demonstrates 50%-70% speedup.
  - Many interesting directions.