Thrill 🎉:

Distributed Big Data Batch Processing in C++

Michael Axtmann, Timo Bingmann, Peter Sanders, Sebastian Schlag, and 6 Students | 2016-01-19
Example $T = [\text{dbadcbcccbabdccc}$]

<table>
<thead>
<tr>
<th>$S_{A_i}$</th>
<th>$T_{S_{A_i}...n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>$$$</td>
</tr>
<tr>
<td>9</td>
<td>a b d c c $</td>
</tr>
<tr>
<td>2</td>
<td>a d c b c c b a b d c c $</td>
</tr>
<tr>
<td>8</td>
<td>b a b d c c $</td>
</tr>
<tr>
<td>1</td>
<td>b a d c b c c b a b d c c $</td>
</tr>
<tr>
<td>5</td>
<td>b c c b a b d c c $</td>
</tr>
<tr>
<td>10</td>
<td>b d c c $</td>
</tr>
<tr>
<td>13</td>
<td>c $</td>
</tr>
<tr>
<td>7</td>
<td>c b a b d c c $</td>
</tr>
<tr>
<td>4</td>
<td>c b c c b a b d c c $</td>
</tr>
<tr>
<td>12</td>
<td>c c $</td>
</tr>
<tr>
<td>6</td>
<td>c c b a b d c c $</td>
</tr>
<tr>
<td>0</td>
<td>d b a d c b c c b a b d c c $</td>
</tr>
<tr>
<td>3</td>
<td>d c b c c b a b d c c $</td>
</tr>
<tr>
<td>11</td>
<td>d c c $</td>
</tr>
</tbody>
</table>
Suffix Sorting with DC3: Example

$$T = \begin{bmatrix} d & b & a & c & b & a & c & b & d & $ & $ \end{bmatrix} = \begin{bmatrix} t_i \end{bmatrix}_{i=0,\ldots,n-1}$$

triples: $(bac,1), (bac,4), (bd,7), (acb,2), (acb,5), (d,8)$

sorted: $(acb,2), (acb,5), (bac,1), (bac,4), (bd,7), (d,8)$

equal 0/1: $0 \ 0 \ 1 \ 0 \ 1 \ 1$

prefix sum: $0 \ 0 \ 1 \ 1 \ 2 \ 3$

$$R = \begin{bmatrix} 1 & 1 & 2 & 0 & 0 & 3 & $ \end{bmatrix} \quad \begin{bmatrix} r_1 & r_4 & r_7 & r_2 & r_5 & r_8 \end{bmatrix}$$

$SA_R = \begin{bmatrix} 3 & 4 & 0 & 1 & 2 & 5 & $ \end{bmatrix} \quad ISA_R = \begin{bmatrix} 2 & 3 & 4 & 0 & 1 & 5 & $ \end{bmatrix}$

$S_0 = [(d, b, 2, 0, 0), (c, b, 3, 1, 3), (c, b, 4, 5, 6)] = (t_i, t_{i+1}, r_{i+1}, r_{i+2}, i)$

$S_1 = [(2, b, 0, 1), (3, b, 1, 4), (4, b, 5, 7)] = (r_{i+1}, t_{i+1}, r_{i+2}, i+1)$

$S_2 = [(0, a, c, 3, 2), (1, a, c, 4, 5), (5, d, $, 6, 8)] = (r_{i+2}, t_{i+2}, t'_{i+3}, r'_{i+4}, i+2)$

$SA_T = \text{Merge(Sort}(S_0), \text{Sort}(S_1), \text{Sort}(S_2)) \quad \Theta(\text{sort}(n))$
Flavours of Big Data Frameworks

- **High Performance Computing (Supercomputers)**
  - MPI

- **Batch Processing**
  - Google’s MapReduce, Hadoop MapReduce, Apache Spark, Apache Flink (Stratosphere), Google’s FlumeJava.

- **Real-time Stream Processing**
  - Apache Storm, Apache Spark Streaming, Google’s MillWheel.

- **Interactive Cached Queries**
  - Google’s Dremel, Powerdrill and BigQuery, Apache Drill.

- **Sharded (NoSQL) Databases and Data Warehouses**
  - MongoDB, Apache Cassandra, Apache Hive, Google BigTable, Hypertable, Amazon RedShift, FoundationDB.

- **Graph Processing**
  - Google’s Pregel, GraphLab, Giraph, GraphChi.

- **Time-based Distributed Processing**
  - Microsoft’s Dryad, Microsoft’s Naiad.
What is Map/Reduce?

Computation model popularized in 2004 by Google with the name MapReduce.
Why Map/Reduce?

- Changes the perspective from the number of processors to how data is processed.

- A simple algorithmic and programming abstraction with
  - automatic parallelization of independent operations (map) and aggregation (reduce),
  - automatic distribution and balancing of data and work,
  - automatic fault tolerance versus hardware errors.

⇒ MapReduce framework
Where is Map/Reduce?

- Programming model for (highly) distributed system
- Implementations, e.g., in *hadoop* and *mongoDB* and other experimental frameworks (some also in C++): (Boost.MapReduce, Sector/Sphere, mapreduce-lite).
- for calculations like: PageRank (sparse matrix multiplication), parallel image processing, aggregation of statistics, machine learning, etc.
- **NOT the same** as distributed file systems or (distributed) databases
- And now, our’s: **Thrill**.
## Why another Big Data Framework?

<table>
<thead>
<tr>
<th></th>
<th>Hadoop World Record</th>
<th>Spark 100 TB</th>
<th>Spark 1 PB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Size</td>
<td>102.5 TB</td>
<td>100 TB</td>
<td>1000 TB</td>
</tr>
<tr>
<td>Elapsed Time</td>
<td>72 mins</td>
<td>23 mins</td>
<td>234 mins</td>
</tr>
<tr>
<td># Nodes</td>
<td>2100</td>
<td>206</td>
<td>190</td>
</tr>
<tr>
<td># Cores</td>
<td>50400</td>
<td>6592</td>
<td>6080</td>
</tr>
<tr>
<td># Reducers</td>
<td>10 000</td>
<td>29 000</td>
<td>250 000</td>
</tr>
<tr>
<td>Rate</td>
<td>1.42 TB/min</td>
<td>4.27 TB/min</td>
<td>4.27 TB/min</td>
</tr>
<tr>
<td>Rate/node</td>
<td>11.2 MB/sec</td>
<td>345 MB/sec</td>
<td>375 MB/sec</td>
</tr>
<tr>
<td>Daytona Rules</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Environment</td>
<td>dedicated</td>
<td>EC2 (i2.8xlarge)</td>
<td></td>
</tr>
</tbody>
</table>

source: [http://databricks.com/blog/2014/10/10/spark-petabyte-sort.html](http://databricks.com/blog/2014/10/10/spark-petabyte-sort.html)
Our Requirements:
- Compound primitives into complex algorithms
- Overlap computation and communication,
- Efficient simple data types,
- C++, and much more...

Lower Layers of Thrill

New Project: Thrill

Michael Axtmann, Timo Bingmann, Peter Sanders, Sebastian Schlag, and 6 Students – Thrill: Distributed Big Data Batch Processing in C++

Institute of Theoretical Informatics – Algorithmics
January 19nd, 2016
Our Requirements:
- compound primitives into complex algorithms
- overlap computation and communication,
- efficient simple data types,
- C++, and much more...
Big Data Batch Processing

Our Requirements:
- compound primitives into complex algorithms
- overlap computation and communication,
- efficient simple data types,
- C++, and much more...

New Project: Thrill

Lower Layers of Thrill

Apache Spark
Apache Flink
MapReduce
Hadoop

Institute of Theoretical Informatics – Algorithmics  
Michael Axtmann, Timo Bingmann, Peter Sanders, Sebastian Schlag, and 6 Students – Thrill: Distributed Big Data Batch Processing in C++  
January 19th, 2016
Big Data Batch Processing

Our Requirements:
- compound primitives into complex algorithms
- overlap computation and communication,
- efficient simple data types,
- C++, and much more...

Lower Layers of Thrill

New Project: Thrill

Lower Level Difficult

Interface

High Level Simple

Efficiency

Fast

MPI

Apache Spark

Apache Flink

MapReduce

Hadoop

Low Level Difficult
Thrill’s Design Goals

- Distributed arrays of small items (characters or integers).
- High-performance, parallelized C++ operations.
- Locality-aware, in-memory computation.
- Transparently use disk if needed
  ⇒ external memory algorithms.
- Avoid all unnecessary round trips of data to memory (or disk).
- Optimize chaining of local operations.

Current Status:

- Status: prototypes of many DOps work reasonably well.
- Near future: extension to distributed LCP array construction.
Distributed Immutable Array (DIA)

User Programmer’s View:
- \( \text{DIA}<T> = \text{result} \) of an operation (local or distributed).
- Model: \text{distributed array} of items \( T \) on the cluster
- Cannot access items directly, instead use \text{actions}.

Framework Designer’s View:
- Goals: distribute work, optimize execution on cluster, add redundancy where applicable.

\[
\begin{align*}
A & \quad \text{PE0} & \quad \text{PE1} & \quad \text{PE2} & \quad \text{PE3} \\
A. \text{Map}(\cdot) & =: B \\
B. \text{Sort}(\cdot) & =: C
\end{align*}
\]
Distributed Immutable Array (DIA)

User Programmer’s View:
- DIA\(<T>\) = result of an operation (local or distributed).
- Model: distributed array of items T on the cluster
- Cannot access items directly, instead use actions.

```
A  A
A. Map(·) =: B
B. Sort(·) =: C
```

Framework Designer’s View:
- Goals: distribute work, optimize execution on cluster, add redundancy where applicable. \(\Rightarrow\) build data-flow graph.
- DIA\(<T>\) = chain of computation items
- Let distributed operations choose “materialization”.

<table>
<thead>
<tr>
<th>PE0</th>
<th>PE1</th>
<th>PE2</th>
<th>PE3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A. Map(·) =: B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B. Sort(·) =: C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Distributed Immutable Array (DIA)

- **User Programmer’s View:**
  - \( \text{DIA}\langle T \rangle = \text{result} \) of an operation (local or distributed).
  - Model: distributed array of items \( T \) on the cluster.
  - Cannot access items directly, instead use actions.

\[
A \quad \text{Map}() =: B \\
B \quad \text{Sort}() =: C \\
\]

- **Framework Designer’s View:**
  - Goals: distribute work, optimize execution on cluster, add redundancy where applicable. \( \implies \text{build data-flow graph} \).
  - \( \text{DIA}\langle T \rangle = \text{chain of computation items} \)
  - Let distributed operations choose “materialization”.

Diagram:
- PE0 and PE1
- Arrows indicating data flow
- Boxes labeled with operations
- Resulting computation graph

Example:
- \( A \rightarrow B \rightarrow C \)
List of Primitives

- **Local Operations (LOp):** input is one item, output $\geq 0$ items. Map(), Filter(), FlatMap().

- **Distributed Operations (DOp):** input is a DIA, output is a DIA.
  - Sort() Sort a DIA using comparisons.
  - ShuffleReduce() Shuffle with Key Extractor, Hasher, and associative Reducer.
  - PrefixSum() Compute (generalized) prefix sum on DIA.
  - Window$_k()$ Scan all $k$ consecutive DIA items.
  - Concat() Concatenate two or more DIAs of equal type.
  - Zip() Combine equal sized DIAs item-wise.
  - Merge() Merge equal typed DIAs using comparisons.

- **Actions:** input is a DIA, output: $\geq 0$ items on master.
  - At(), Min(), Max(), Sum(), Sample(), pretty much still open.
Exert of DC3’s Data-Flow Graph

\[ T = (t) \]
\[ R := (r, i) \]

Window\(_3\)(\( t_i \rightarrow (t_i, t_{i+1}, t_{i+2}) \))

\[ A := R. \text{Sort(by } r) \]

Filter\((i < n_{mod1})\)

\[ R_1 := \text{Map}((r, i) \rightarrow (i)) \]

Filter\((i \geq n_{mod1})\)

\[ R_2 := \text{Map}((r, i) \rightarrow (i)) \]

Zip\(((t_i, t_{i+1}, t_{i+2}), (r_1), (r_2))\)

\[ T = (t_i, t_{i+1}, t_{i+2}) \]

\[ M = (T, r_1, r_2) \]

Window\(_2\)((T, r_1, r_2)_i, (T', r'_1, r'_2)_{i+1})\)

Map\(((t_i, t_{i+1}, r_1, r_2, i))\)

Map\(((r_1, t_{i+1}, r_2, i + 1))\)

Map\(((r_2, t_{i+2}, t'_i, r'_1, i + 2))\)

Sort(by \((t_i, r_1))\))

Sort(by \((r_1))\))

Sort(by \((r_2))\))
A Suffix Sorting Algorithm: DC3
Compile program into \textbf{one binary}, running on all nodes.

\textbf{Collective} coordination of work on compute nodes, like MPI.

\textbf{Control flow} is decided on by using C++ statements.

Runs on MPI clusters and on Amazon’s EC2 cloud.
# Layers of Thrill

<table>
<thead>
<tr>
<th>api: High-level User Interface</th>
<th>core: Internal Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIA&lt;T&gt;, Map, FlatMap, Filter, Reduce, Sort, Merge, ...</td>
<td>reducing hash tables (bucket and linear probing), multiway merge, stage executor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data: Data Layer</th>
<th>net: Network Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block, File, BlockQueue, Reader, Writer, Multiplexer, Streams, BlockPool (paging)</td>
<td>(Binomial Tree) Broadcast, Reduce, AllReduce, Async-Send/Recv, Dispatcher</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>io: Async File I/O</th>
<th>mem: Memory Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>borrowed from STXXL</td>
<td>Allocators, Counting</td>
</tr>
</tbody>
</table>
Thrill contains many things you learned in this lecture:

- DOps are **BSP**-style communications primitives. LOps are inlined into them.
- Network layer’s Broadcast(), Reduce(), etc are **binomial tree** algorithms. Prefixsum() is a **hypercube** algorithm.
- Sort() is a **distributed external sample sort**.
- Merge() uses distributed **multisequence selection** to balance result.

In future (like bachelor/master theses): add more advanced things like distributed hashing, Bloom filters, etc to Thrill.
Mapping Data-Flow Nodes to Cluster

Master

\[ A := \text{Read}() \]

\[ B := A.\text{Sort}() \]

\[ C := B.\text{Map}() \]

\[ E := \text{Zip}(C, D) \]

\[ E.\text{WriteFs}() \]

PE 0

\[ A := \text{Read}_{[0, \frac{n}{2}]}() \]

\[ \text{pre-op: sample, store} \]

\[ \text{exchange samples} \]

\[ \text{post-op: transmit and sort} \]

\[ D_{[0, \frac{m}{2}]} \]

\[ C := B.\text{Map}() \]

\[ \text{pre-op: store} \]

\[ \text{align arrays (exchange)} \]

\[ \text{post-op: zip lambda} \]

\[ E.\text{WriteFs}_{[0, \frac{\ell}{2}]}() \]

PE 1

\[ A := \text{Read}_{[\frac{n}{2}, n]}() \]

\[ \text{pre-op: sample, store} \]

\[ \text{exchange samples} \]

\[ \text{post-op: transmit and sort} \]

\[ D_{[\frac{m}{2}, m]} \]

\[ C := B.\text{Map}() \]

\[ \text{pre-op: store} \]

\[ \text{align arrays (exchange)} \]

\[ \text{post-op: zip lambda} \]

\[ E.\text{WriteFs}_{[\frac{\ell}{2}, \ell]}() \]
Sorting DOp

pre-op: sample
save to disk when/if full
broadcast splitters, build tree
in parallel: distribute to PEs, sort and save to disk if full
external multiway merge
output

input

samples

in parallel: distribute to PEs, sort and save to disk if full
external multiway merge
output

input

samples

pre-op: sample
save to disk when/if full
broadcast splitters, build tree
in parallel: distribute to PEs, sort and save to disk if full
external multiway merge
output

input

samples
Timing: Word-Count Weak-Scaling

Benchmarks: Weak-Scaling

Sebastian Lamm – Thrill - Chaining and Applications
Institute of Theoretical Informatics – Algorithmics II 23. September 2015

19 / 23
Speedup: Word-Count Weak-Scaling

**Weak-Scaling**

- framework=Spark
- framework=Thrill

![Graph showing speedup comparison between Thrill and Spark](image)

Michael Axtmann, Timo Bingmann, Peter Sanders, Sebastian Schlag, and 6 Students – Thrill: Distributed Big Data Batch Processing in C++

Institute of Theoretical Informatics – Algorithmics January 19nd, 2016
Timing: Word-Count Strong-Scaling

Strong-Scaling (128GB total)

- framework=Spark
- framework=Thrill

Time per Element [ns]

#Cores

Michael Axtmann, Timo Bingmann, Peter Sanders, Sebastian Schlag, and 6 Students – Thrill: Distributed Big Data Batch Processing in C++
Institute of Theoretical Informatics – Algorithmics
January 19nd, 2016
Speedup: Word-Count Strong-Scaling

Strong-Scaling (128GB total)

- framework=Spark
- framework=Thrill

#Cores

Speedup Thrill over Spark

Michael Axtmann, Timo Bingmann, Peter Sanders, Sebastian Schlag, and 6 Students – Thrill: Distributed Big Data Batch Processing in C++
Institute of Theoretical Informatics – Algorithmics
January 19nd, 2016
Current and Future Work

- High quality, very modern C++14 code.

Some Master thesis ideas:

- Distributed rank()/select() and wavelet tree construction.
- Distributed query processing.
- Communication efficient distributed operations for Thrill.

Thank you for your attention!

Questions?