STXXL and Thrill :

(Distributed Big Data Batch Processing in C++)

Michael Axtmann, Timo Bingmann, Peter Sanders, Sebastian Schlag, and 6 Students | 2016-09-21

Karlsruhe Institute of Technology

INSTITUTE OF THEORETICAL INFORMATICS – ALGORITHMICS

Example $T = [\texttt{dbadcbccbabdcc\$}]$

(indices 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 above the characters)

SA_i	$T_{SA_i \ldots n}$
14	$
9	a b d c c $
2	a d c b c c b a b d c c $
8	b a b d c c $
1	b a d c b c c b a b d c c $
5	b c c b a b d c c $
10	b d c c $
13	c $
7	c b a b d c c $
4	c b c c b a b d c c $
12	c c $
6	c c b a b d c c $
0	d b a d c b c c b a b d c c $
3	d c b c c b a b d c c $
11	d c c $

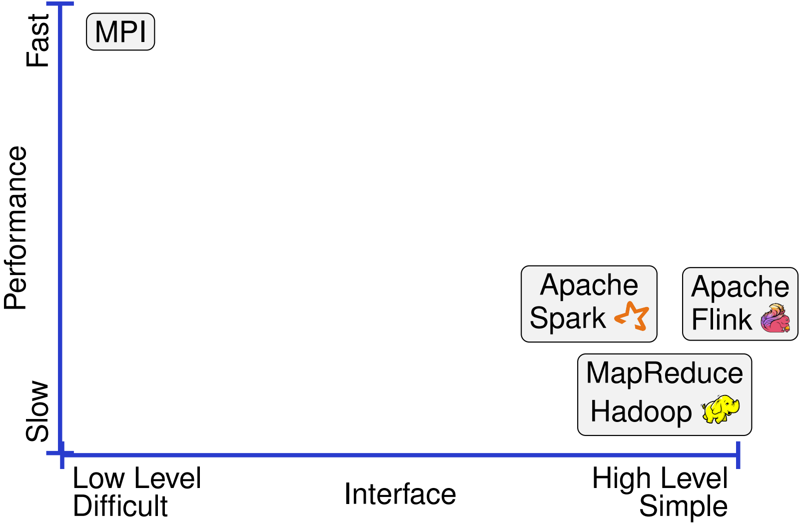

bwUniCluster
512 x 16 cores, 64 GB RAM
© KIT (SCC)

Flavours of Big Data Frameworks

Karlsruhe Institute of Technology

- 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.

Big Data Batch Processing



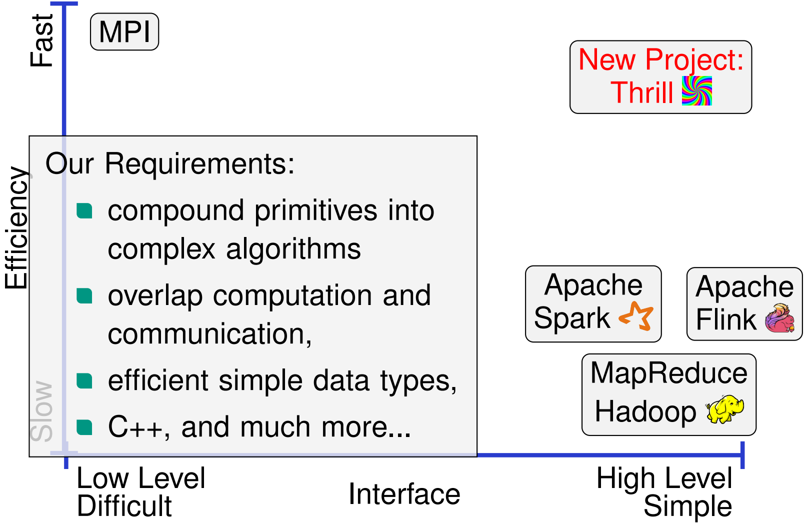
Karlsruher Institut für Technologie

Projektpraktikum: Verteilte Datenverarbeitung mit MapReduce

Timo Bingmann, Peter Sanders und Sebastian Schlag | 21. Oktober 2014 @ PdF Vorstellung

INSTITUTE OF THEORETICAL INFORMATICS – ALGORITHMICS

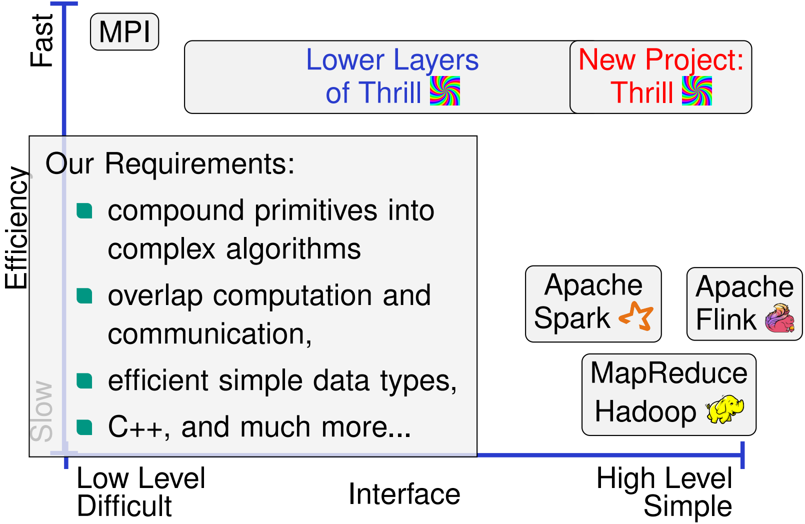
Big Data Batch Processing



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...

Thrill's Design Goals

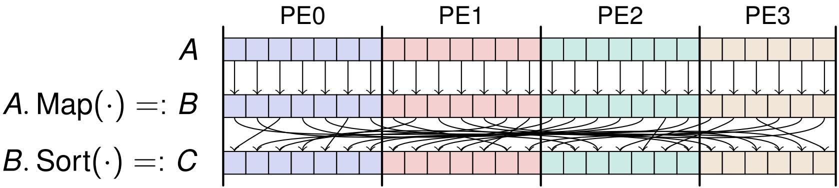
- A new and easier way to program distributed algorithms.
- Distributed arrays of small items (characters or integers).
- High-performance, parallelized C++ operations.
- Locality-aware, in-memory computation.
- Transparently use disk if needed
  $\Rightarrow$ external memory algorithms.
- Avoid all unnecessary round trips of data to memory (or disk).
- Optimize chaining/pipelining of local operations.

Current Status:

- Open-Source at `http://project-thrill.org` and Github.

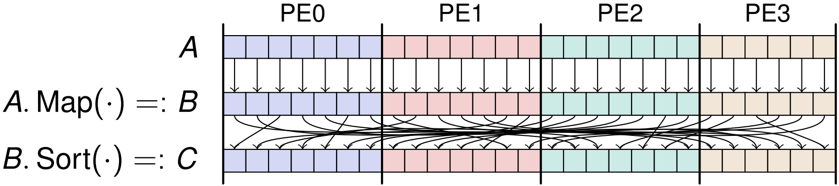
Distributed Immutable Array (DIA)

- 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 transformations and actions.



Distributed Immutable Array (DIA)

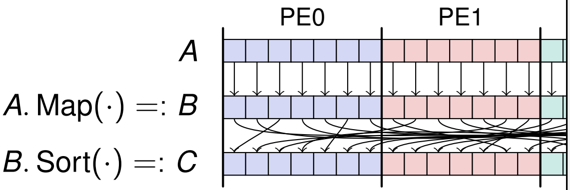
- User Programmer's View:
  - `DIA<T>` = result of an operation (local or distributed).
  - Model: distributed array of items T on the cluster
  - Cannot access items directly, instead use transformations and actions.

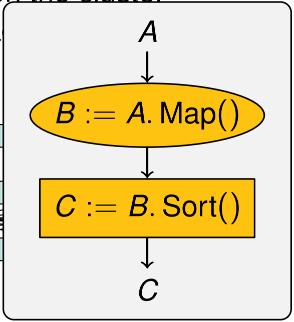


- Framework Designer's View:
  - Goals: distribute work, optimize execution on cluster, add redundancy where applicable. $\Longrightarrow$ build data-flow graph.
  - `DIA<T>` = chain of computation items

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, instea
    and actions.





- Framework Designer's View:
  - Goals: distribute work, optimize execution on cluster, add redundancy where applicable. $\Longrightarrow$ build data-flow graph.
  - `DIA<T>` = chain of computation items

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.
ReduceByKey()	Shuffle with Key Extractor, Hasher, and associative Reducer.
GroupByKey()	Like ReduceByKey, but with a general Reducer.
PrefixSum()	Compute (generalized) prefix sum on DIA.
Window_k()	Scan all k consecutive DIA items.
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.

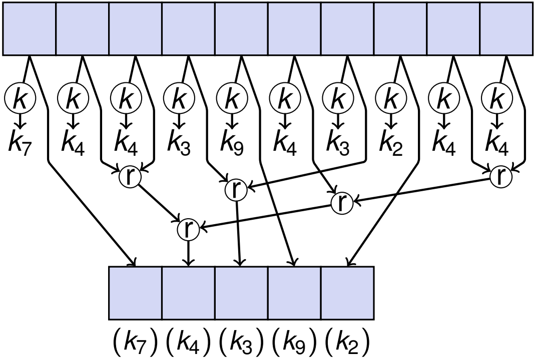
Example: WordCount in Thrill

```cpp
using Pair = std::pair<std::string, size_t>;
void WordCount(Context& ctx, std::string input, std::string output) {
    auto word_pairs = ReadLines(ctx, input)     // DIA<std::string>
    .FlatMap<Pair>(
        // flatmap lambda: split and emit each word
        [](const std::string& line, auto emit) {
            Split(line, ' ', [&](std::string_view sv) {
                emit(Pair(sv.to_string(), 1)); });
    });                                         // DIA<Pair>
    word_pairs.ReduceByKey(
        // key extractor: the word string
        [](const Pair& p) { return p.first; },
        // commutative reduction: add counters
        [](const Pair& a, const Pair& b) {
            return Pair(a.first, a.second + b.second);
    })                                          // DIA<Pair>
    .Map([](const Pair& p) {
        return p.first + ": " + std::to_string(p.second); })
    .WriteLines(output);                        // DIA<std::string>
}
```

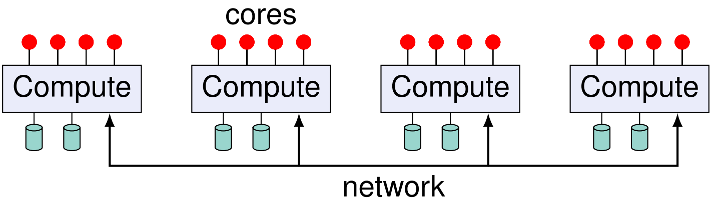
DOps: ReduceByKey

$$\textbf{ReduceByKey}(k, r) : \langle A \rangle \to \langle A \rangle$$

$k : A \to K$	key extractor
$r : A \times A \to A$	reduction



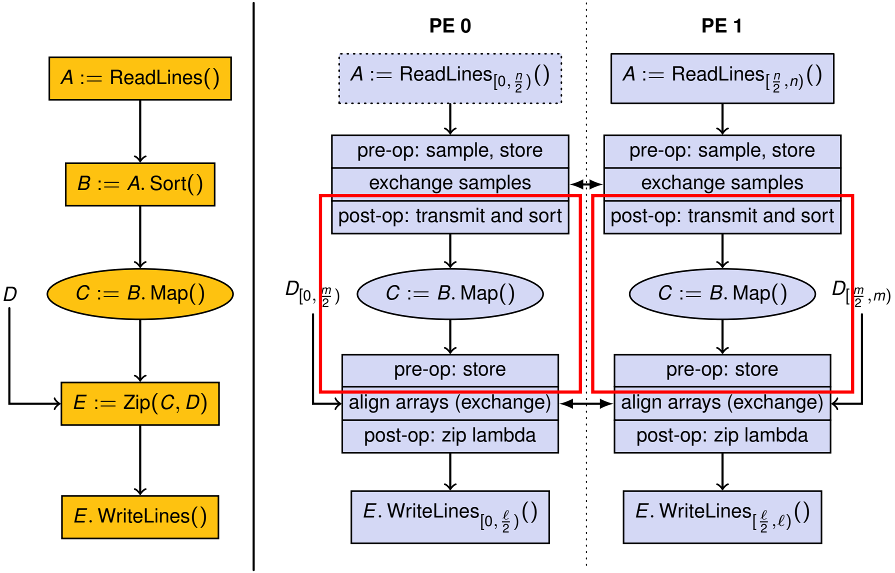
Execution on Cluster



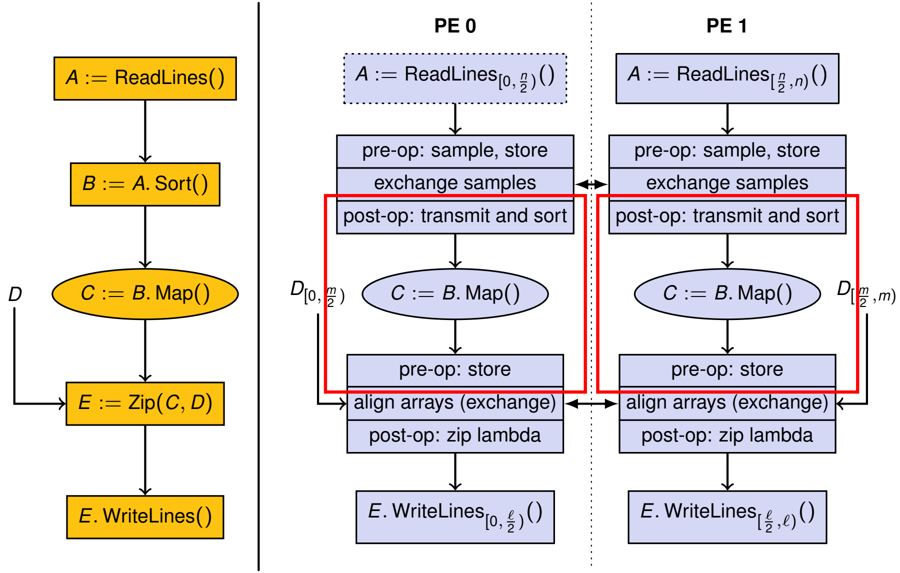
- Compile program into one binary, running on all hosts.
- Collective coordination of work on compute hosts, like MPI.
- Control flow is decided on by using C++ statements.

- Runs on MPI HPC clusters and on Amazon's EC2 cloud.

Pipelining Stages in Thrill



Pipelining Stages in Thrill



Layers of Thrill

KIT Karlsruhe Institute of Technology

api: High-level User Interface
DIA<T>, Map, FlatMap, Filter, Reduce, Sort, Merge, ...

core: Internal Algorithms
reducing hash tables (bucket and linear probing), multiway merge, stage executor

data: Data Layer
Block, File, BlockQueue, Reader, Writer, Multiplexer, Streams, BlockPool (paging)

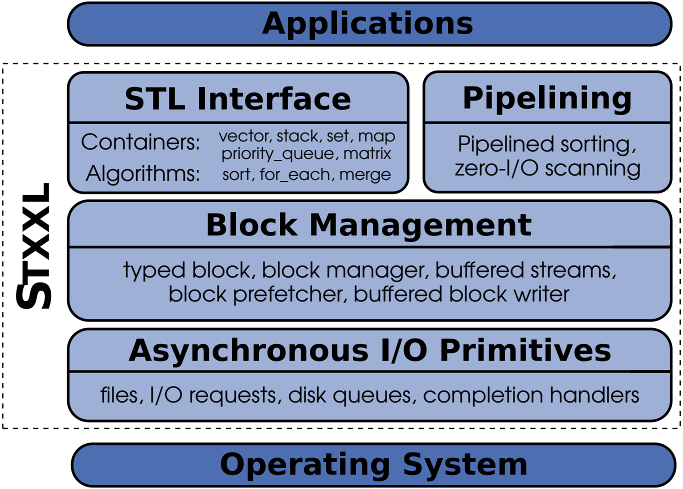
io: Async File I/O
borrowed from STXXL

net: Network Layer
(Binomial Tree) Broadcast, Reduce, AllReduce, Async-Send/Recv, Dispatcher
Backends: mock, TCP, MPI

common: Common Tools
Logger, Delegates, Math, ...

mem: Memory Limitation
Allocators, Counting

Layers of STXXL



STXXL and Thrill

	STXXL	Thrill
Model	external	distributed external
Shared Memory	partially parallelized	inherently parallel
External Memory	explicit	via swapping
Items	fixed size	variable length
High-level API	containers, streams	DIA operations
Programming	mix of imperative and functional parts	
Pipelining via	nested template streams	functional templating, consume in pipeline
RAM Limit	manual (parameters)	automatic (stages)
Code Base	C++98	very modern C++14

STXXL and Thrill

STXXL API: vector (Paging), sorter (Sort), sequence (Scan), map (B-Tree), unordered_map (Hash), priority_queue (PQ), parallel_priority_queue (PPQ), matrix (Block Matrix),

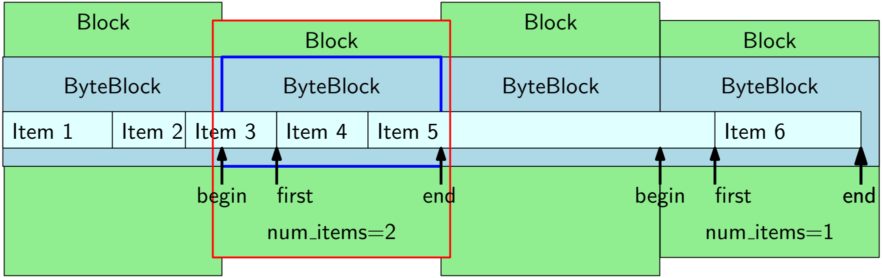
Stream/Pipelining: stream::sort, stream::runs_creator, stream::runs_merger, ...

Lower Layers: BID, typed_block, block_manager, read_write_pool, buf_istream, buf_ostream, async I/O impl.

Thrill API: DIA.Generate, DIA.Map, DIA.ReduceByKey, DIA.Window, DIA.Sort, DIA.Union, DIA.Zip, DIA.ReadBinary, ...

Lower Layers: Block, ByteBlock, File, BlockQueue, BlockReader, BlockWriter, BlockPool (Paging), STXXL's BlockManager and async I/O implementations.

File and Blocks in Thrill



Future of STXXL and Thrill?

On STXXL:

- Pre-C++11 old-style code. But good architecture.
- Many simplifications possible with C++11.
- Shared memory parallelism is critically important.
- Missing some important convenience features:
  variable length items, memory management.

Thrill and STXXL:

- Share much in common code layers, esp. C++ tooling.
- Thrill is distributed and inherently parallel, but not the API itself.

Thank you for your attention!

Questions? ... Discussion?