Karlsruhe Institute of Technology

Thrill: High-Performance Algorithmic Distributed Batch Data Processing with C++

Timo Bingmann, Michael Axtmann, Peter Sanders, Sebastian Schlag, and 6 Students · 2019-08-28

INSTITUTE OF THEORETICAL INFORMATICS – ALGORITHMICS

Weak-Scaling Benchmarks

WordCountCC – $h \cdot 49\,\text{GiB}$ 222 lines

- Reduce text files from CommonCrawl web corpus.

PageRank – $h \cdot 2.7\,\text{GiB}$, $|E| \approx h \cdot 158\,\text{M}$ 410 lines

- Calculate PageRank using join of current ranks with outgoing links and reduce by contributions. 10 iterations.

TeraSort – $h \cdot 16\,\text{GiB}$ 141 lines

- Distributed (external) sorting of 100 byte random records.

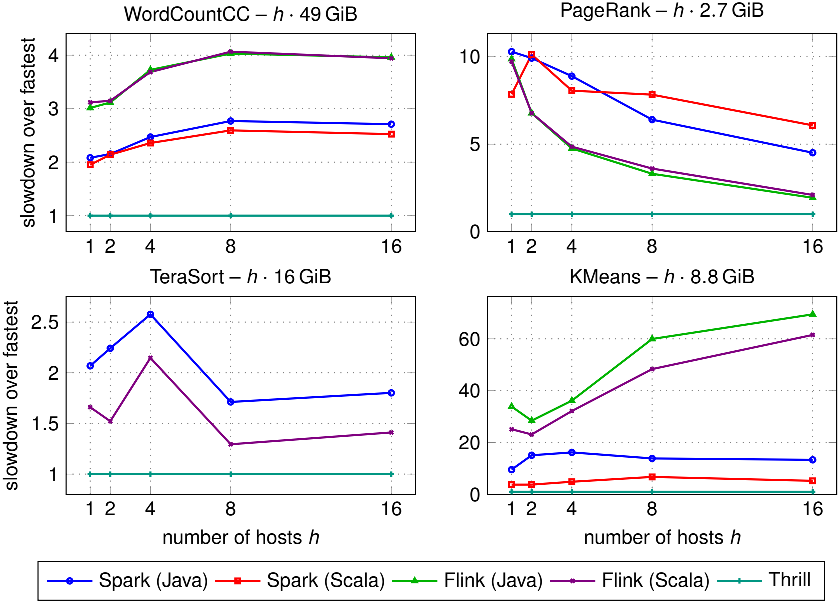
K-Means – $h \cdot 8.8\,\text{GiB}$ 357 lines

- Calculate K-Means clustering with 10 iterations.

Platform: $h \times$ r3.8xlarge systems on Amazon EC2 Cloud

- 32 cores, Intel Xeon E5-2670v2, 2.5 GHz clock, 244 GiB RAM, 2 x 320 GB local SSD disk, ≈ 400 MiB/s read/write Ethernet network ≈ 1000 MiB/s throughput, Ubuntu 16.04.

Experimental Results: Slowdowns



WordCountCC – $h \cdot 49$ GiB

PageRank – $h \cdot 2.7$ GiB

TeraSort – $h \cdot 16$ GiB

KMeans – $h \cdot 8.8$ GiB

Example $T = [\overset{0}{\texttt{t}}\overset{1}{\texttt{o}}\overset{2}{\texttt{b}}\overset{3}{\texttt{e}}\overset{4}{\texttt{o}}\overset{5}{\texttt{r}}\overset{6}{\texttt{n}}\overset{7}{\texttt{o}}\overset{8}{\texttt{t}}\overset{9}{\texttt{t}}\overset{10}{\texttt{o}}\overset{11}{\texttt{b}}\overset{12}{\texttt{e}}\overset{13}{\texttt{\$}}]$

SA_{i}	$\mathrm{LCP}_{\mathrm{i}}$	$T_{\mathrm{SA}_{\mathrm{i}}\ldots\mathrm{n}}$
13	-	$
11	0	be$
2	2	beornottobe$
12	0	e$
3	1	eornottobe$
6	0	nottobe$
10	0	obe$
1	3	obeornottobe$
4	1	ornottobe$
7	1	ottobe$
5	0	rnottobe$
9	4	tobe$
0	1	tobeornottobe$
8	0	ttobe$

Suffix Sorting with DC3: Example

$$\begin{array}{rccccccccccccl} & & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & \\ T = & [& \mathtt{d} & \mathtt{b} & \mathtt{a} & \mathtt{c} & \mathtt{b} & \mathtt{a} & \mathtt{c} & \mathtt{b} & \mathtt{d} & \$ & \$ &] \quad = [\, t_i \,]_{i=0,\ldots,n-1} \end{array}$$

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

$$R = \boxed{1\ \ 1\ \ 2}\,\boxed{0\ \ 0\ \ 3}\ \$ \qquad \begin{array}{cccccc} r_1 & r_4 & r_7 & r_2 & r_5 & r_8 \end{array}$$

$$\mathrm{SA}_R = 3\ \ 4\ \ 0\ \ 1\ \ 2\ \ 5\ \ \$ \qquad \mathrm{ISA}_R = \boxed{2\ \ 3\ \ 4}\,\boxed{0\ \ 1\ \ 5}\ \$$$

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

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

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

$$\mathrm{SA}_T = \mathrm{Merge}(\mathrm{Sort}(S_0), \mathrm{Sort}(S_1), \mathrm{Sort}(S_2))$$

$$\boxed{\Theta(\mathrm{sort}(n))}$$

amazon web services

Microsoft Azure

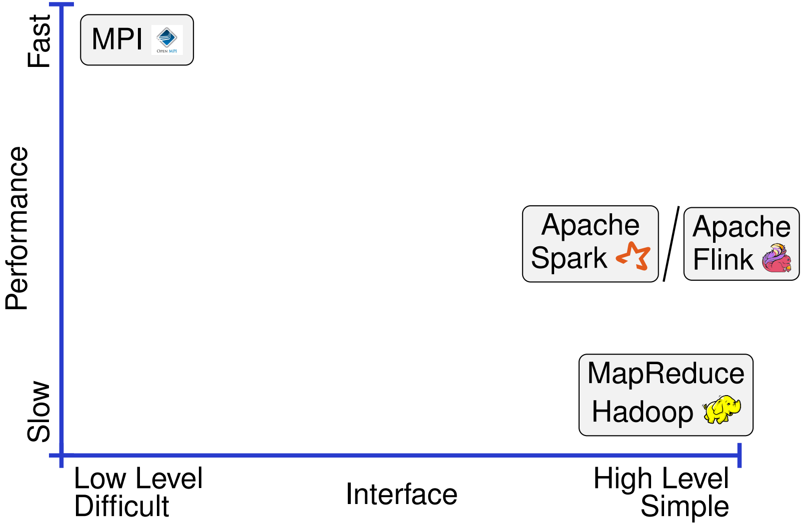
Google Cloud Platform

bwUniCluster
KIT (SCC)

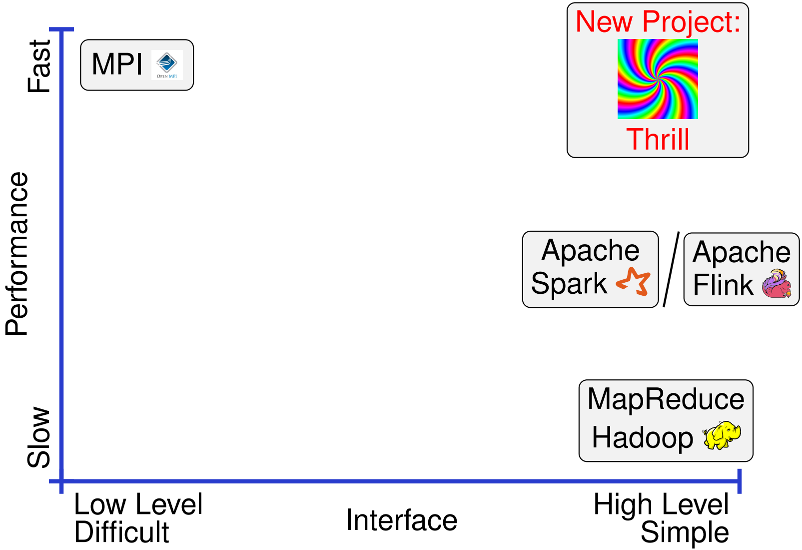
Flavours of Big Data Frameworks

- Batch Processing
  Google's MapReduce, Hadoop MapReduce, Apache Spark, Apache Flink (Stratosphere), Google's FlumeJava.
- High Performance Computing (Supercomputers)
  MPI
- 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



Big Data Batch Processing



Fast

Performance

Slow

MPI

New Project:

Thrill

Apache Spark / Apache Flink

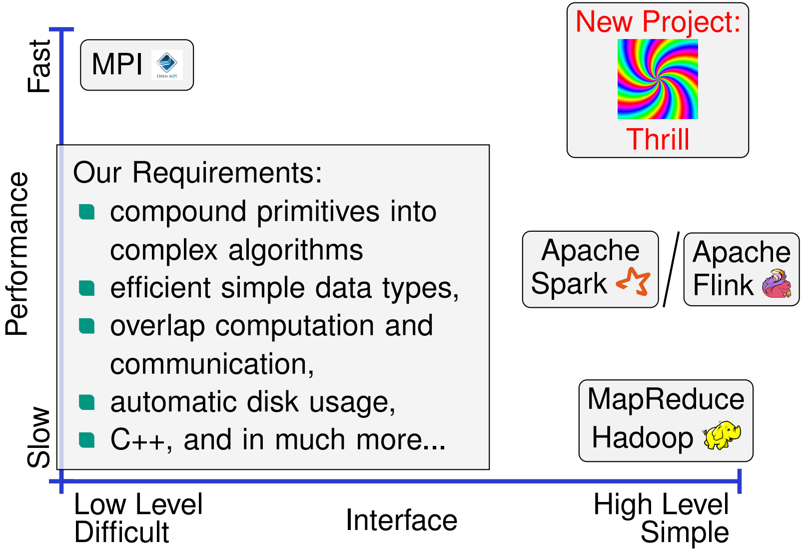
MapReduce Hadoop

Low Level
Difficult

Interface

High Level
Simple

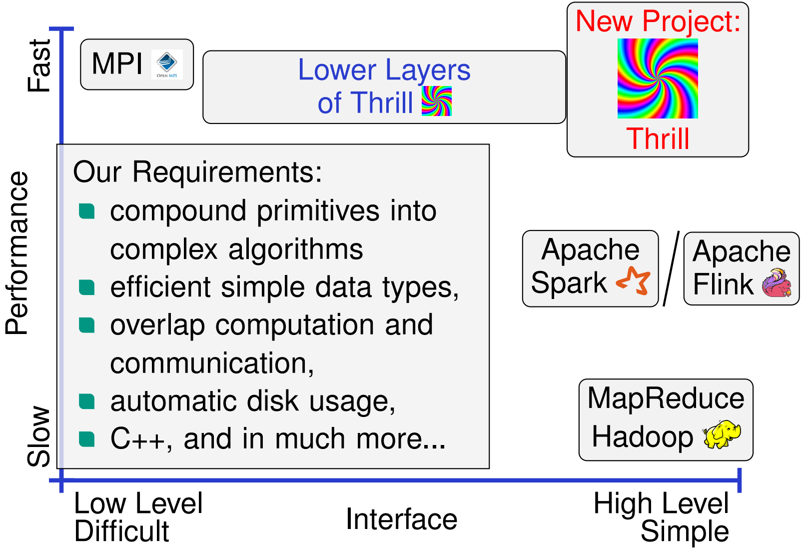
Big Data Batch Processing



Our Requirements:

- compound primitives into complex algorithms
- efficient simple data types,
- overlap computation and communication,
- automatic disk usage,
- C++, and in much more...

Big Data Batch Processing



Our Requirements:

- compound primitives into complex algorithms
- efficient simple data types,
- overlap computation and communication,
- automatic disk usage,
- C++, and in much more...

Thrill's Goal and Current Status

An easy way to program distributed external algorithms in C++.

Current Status:

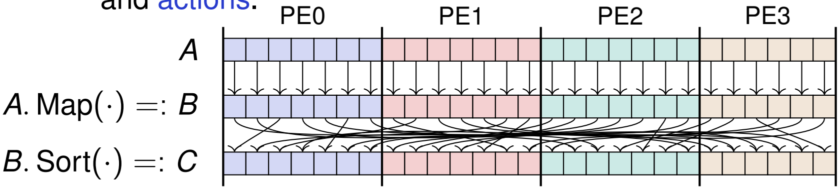
- Open-source prototype at `http://github.com/thrill/thrill`.
- $\approx$ 60 K lines of C++14 code, written by $\geq$ 12 contributors.
- Published at IEEE Conference on Big Data [B, et al. '16]
- Faster than Apache Spark and Flink on five micro benchmarks: WordCount1000/CC, PageRank, TeraSort, and K-Means.

Case Studies:

- Five suffix sorting algorithms [B, Gog, Kurpicz, BigData'18]
- Louvain graph clustering algorithm [Hamann et al. Euro-Par'18]
- Process scientific data on HPC (poster) [Karabin et al. SC'18]
- More: stochastic gradient descent, triangle counting, etc.
- Future: fault tolerance, scalability, predictability, and more.

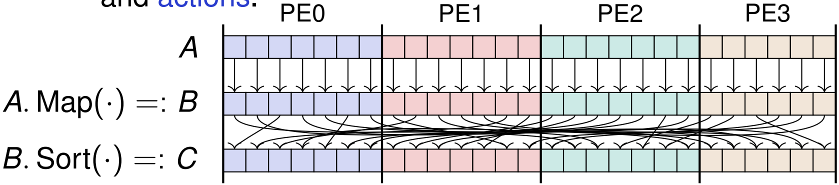
Distributed Immutable Array (DIA)

- 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)

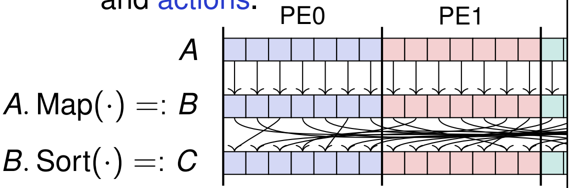
- User Programmer's View:
  - `DIA<T>` = result of an operation (local or distributed).
  - Model: distributed array of items T on the cluster
  - Cannot access items directly, instead use transformations and actions.

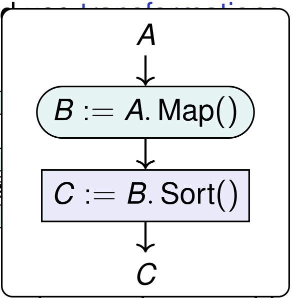


- Framework Designer's View:
  - Goals: distribute work, optimize execution on cluster, add redundancy where applicable. $\Longrightarrow$ build data-flow graph.
  - `DIA<T>` = chain of computation items
  - Let distributed operations choose "materialization".

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
    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
  - Let distributed operations choose "materialization".

List of Primitives (Excerpt)

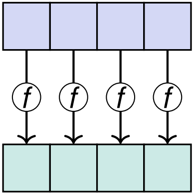
- Local Operations (LOp): input is one item, output $\geq$ 0 items.
  Map(), Filter(), FlatMap().
- Distributed Operations (DOp): input is a DIA, output is a DIA.

Sort()	Sort a DIA using comparisons.
ReduceBy()	Shuffle with Key Extractor, Hasher, and associative Reducer.
GroupBy()	Like ReduceBy, 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.
Union()	Combine equal typed DIAs in arbitrary order.
InnerJoin()	Join items from two DIAs by key.

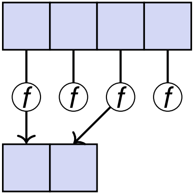
- Actions: input is a DIA, output: $\geq$ 0 items on every worker.
  Sum(), Min(), ReadLines(), WriteLines(), pretty much open.

Local Operations (LOps)

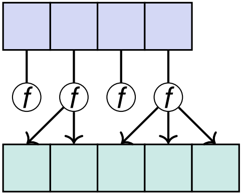
$$\textbf{Map}(f) : \langle A \rangle \to \langle B \rangle$$
$$f : A \to B$$



$$\textbf{Filter}(f) : \langle A \rangle \to \langle A \rangle$$
$$f : A \to \{\mathit{false}, \mathit{true}\}$$



$$\textbf{FlatMap}\langle B \rangle(f) : \langle A \rangle \to \langle B \rangle$$
$$f : A \to \text{array}(B)$$

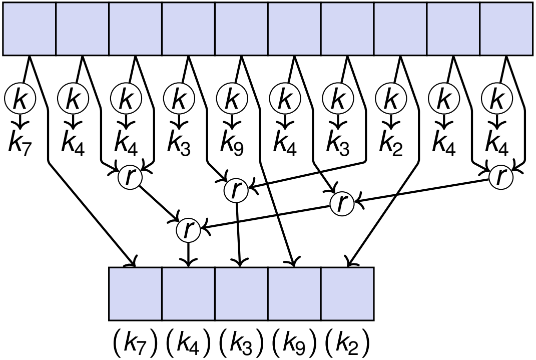


DOps: ReduceByKey

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

$k : A \to K$ key extractor

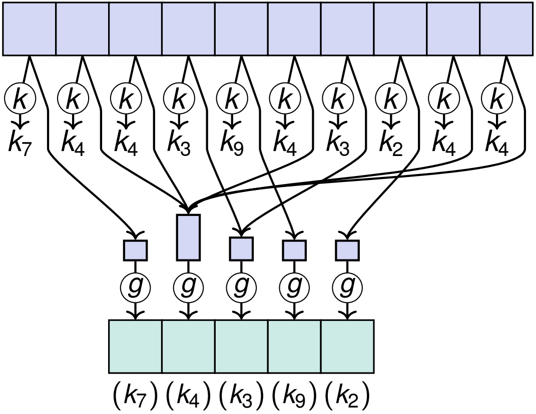
$r : A \times A \to A$ reduction



DOps: GroupByKey

$$\textbf{GroupByKey}(k, g) : \langle A \rangle \to \langle B \rangle$$

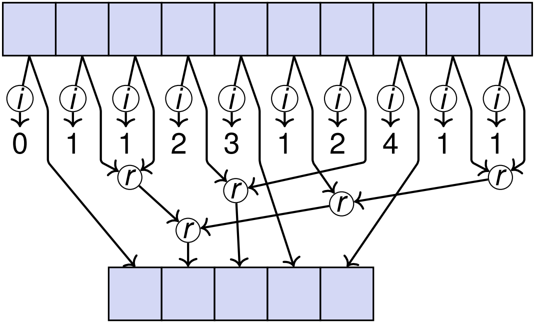
$k : A \to K$	key extractor
$g : iterable(A) \to B$	group function



DOps: ReduceToIndex

$\textbf{ReduceToIndex}(i, n, r) : \langle A \rangle \to \langle A \rangle$

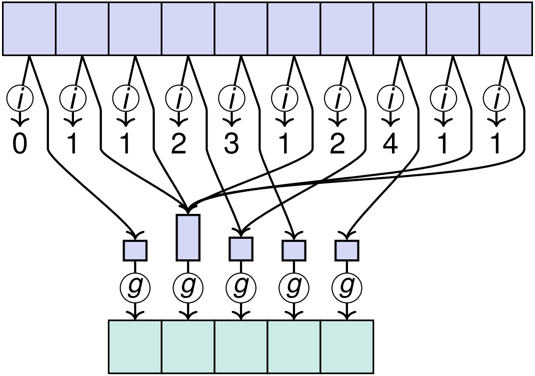
$i : A \to \{0..n-1\}$	index extractor
$n \in \mathbb{N}_0$	result size
$r : A \times A \to A$	reduction



DOps: GroupToIndex

$\textbf{GroupToIndex}(i, n, g) : \langle A \rangle \to \langle B \rangle$

$i : A \to \{0..n-1\}$	index extractor
$n \in \mathbb{N}_0$	result size
$g : \mathit{iterable}(A) \to B$	group function

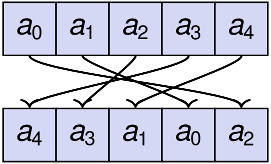


DOps: Sort and Merge

Sort$(o) : \langle A \rangle \to \langle A \rangle$

$o : A \times A \to \{\mathit{false}, \mathit{true}\}$

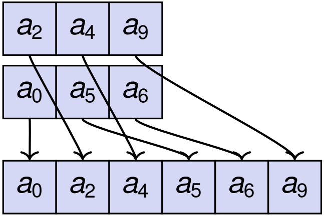
(less) order relation



Merge$(o) : \langle A \rangle \times \langle A \rangle \cdots \to \langle A \rangle$

$o : A \times A \to \{\mathit{false}, \mathit{true}\}$

(less) order relation

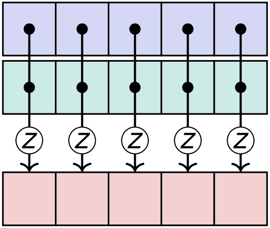


DOps: Zip and Window

Zip$(z) : \langle A \rangle \times \langle B \rangle \cdots \to \langle C \rangle$

$z : A \times B \to C$

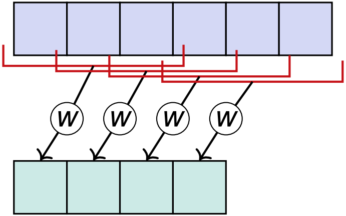
zip function



Window$(k, w) : \langle A \rangle \to \langle B \rangle$

$k \in \mathbb{N}$ window size

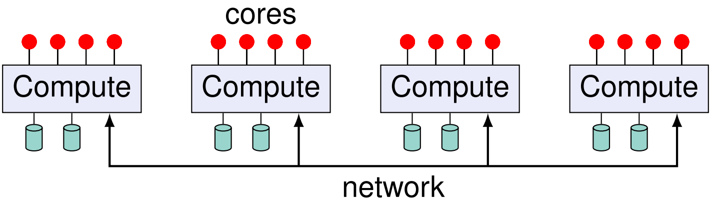
$w : A^k \to B$ window function



Example: WordCount in Thrill

```
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) {
            tlx::split_view(' ', line, [&](tlx::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>
}
```

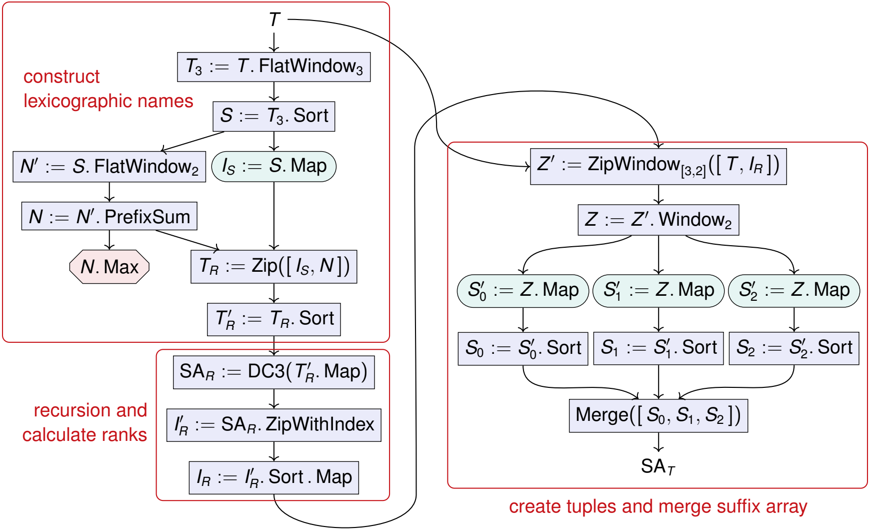
Execution on Cluster



- 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.

Data-Flow Graph of DC3



T

$T_3 := T.\text{FlatWindow}_3$

construct
lexicographic names

$S := T_3.\text{Sort}$

$N' := S.\text{FlatWindow}_2$

$I_S := S.\text{Map}$

$N := N'.\text{PrefixSum}$

$N.\text{Max}$

$T_R := \text{Zip}([\,I_S, N\,])$

$T'_R := T_R.\text{Sort}$

$\text{SA}_R := \text{DC3}(T'_R.\text{Map})$

recursion and
calculate ranks

$I'_R := \text{SA}_R.\text{ZipWithIndex}$

$I_R := I'_R.\text{Sort}.\text{Map}$

$Z' := \text{ZipWindow}_{[3,2]}([\,T, I_R\,])$

$Z := Z'.\text{Window}_2$

$S'_0 := Z.\text{Map}$

$S'_1 := Z.\text{Map}$

$S'_2 := Z.\text{Map}$

$S_0 := S'_0.\text{Sort}$

$S_1 := S'_1.\text{Sort}$

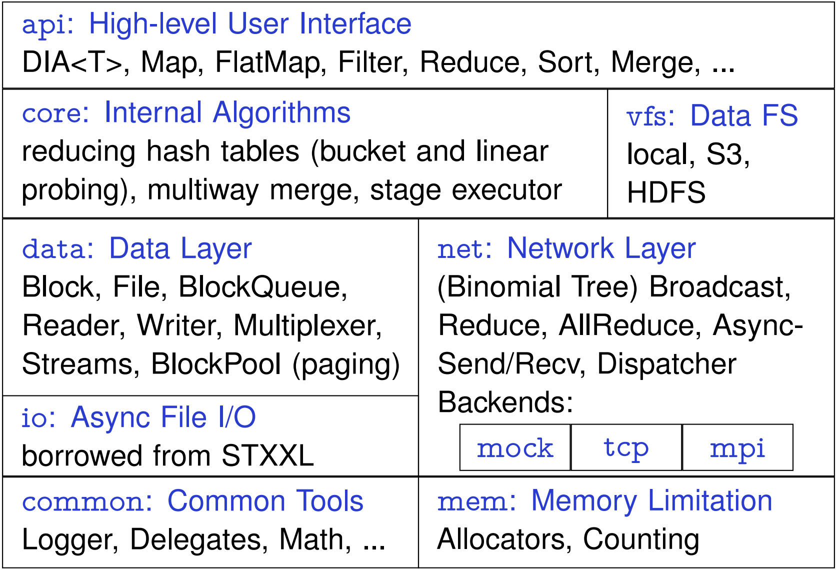
$S_2 := S'_2.\text{Sort}$

$\text{Merge}([\,S_0, S_1, S_2\,])$

SA_T

create tuples and merge suffix array

Layers of Thrill



`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

`vfs`: Data FS
local, S3, HDFS

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

`io`: Async File I/O
borrowed from STXXL

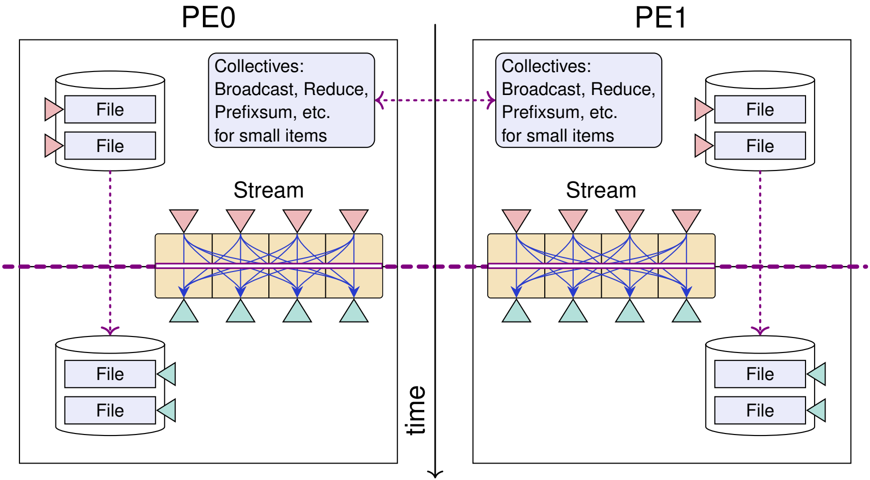
`net`: Network Layer
(Binomial Tree) Broadcast, Reduce, AllReduce, Async-Send/Recv, Dispatcher
Backends:

`mock` `tcp` `mpi`

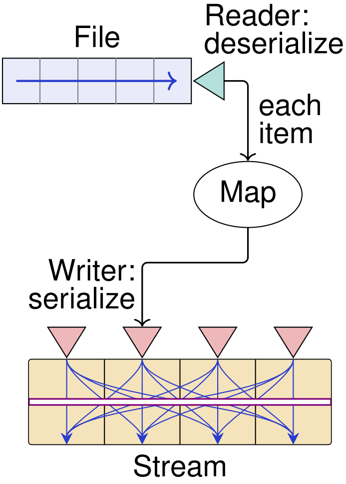
`common`: Common Tools
Logger, Delegates, Math, ...

`mem`: Memory Limitation
Allocators, Counting

Thrill's Communication Abstraction



Thrill's Data Processing Pipelines



File

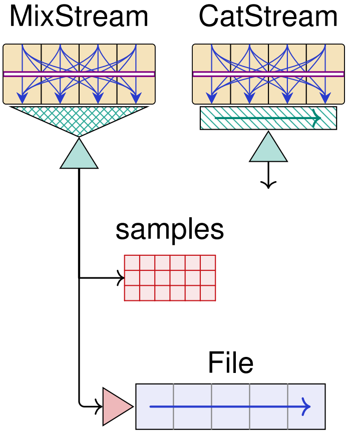
Reader: deserialize

each item

Map

Writer: serialize

Stream



MixStream

CatStream

samples

File

Weak-Scaling Benchmarks

WordCountCC – $h \cdot 49\,\text{GiB}$ 222 lines

- Reduce text files from CommonCrawl web corpus.

PageRank – $h \cdot 2.7\,\text{GiB}$, $|E| \approx h \cdot 158\,\text{M}$ 410 lines

- Calculate PageRank using join of current ranks with outgoing links and reduce by contributions. 10 iterations.

TeraSort – $h \cdot 16\,\text{GiB}$ 141 lines

- Distributed (external) sorting of 100 byte random records.

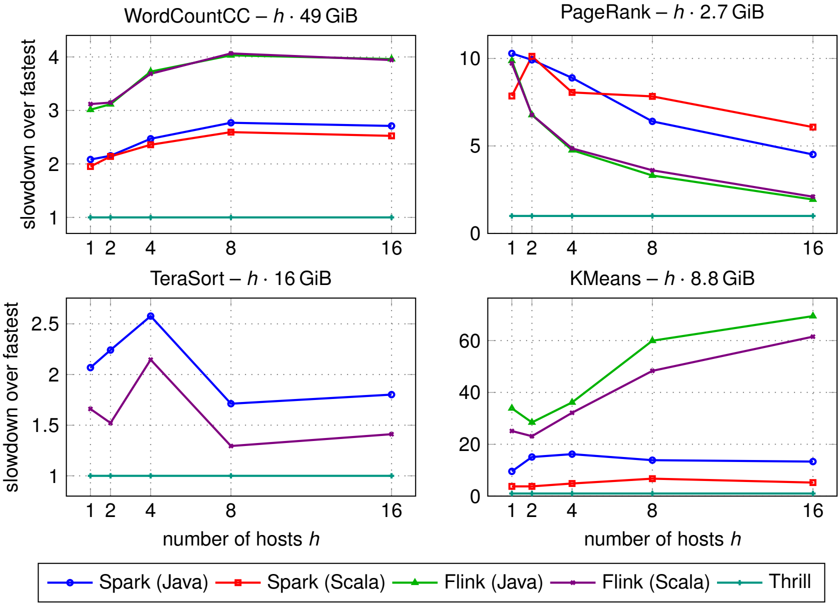
K-Means – $h \cdot 8.8\,\text{GiB}$ 357 lines

- Calculate K-Means clustering with 10 iterations.

Platform: $h \times$ r3.8xlarge systems on Amazon EC2 Cloud

- 32 cores, Intel Xeon E5-2670v2, 2.5 GHz clock, 244 GiB RAM, 2 x 320 GB local SSD disk, ≈ 400 MiB/s read/write Ethernet network ≈ 1000 MiB/s throughput, Ubuntu 16.04.

Experimental Results: Slowdowns



Thoughts on the Architecture

Thrill's Sweet Spot

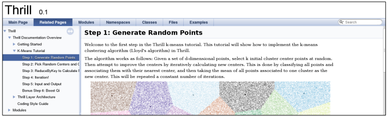
- C++ toolkit for implementing distributed algorithms quickly.
- Platform to engineer and evaluate distributed primitives.
- Efficient processing of small items and pipelining of primitives.
- Platform for implementing on-the-fly compiled queries?

Open Questions

- Compile-time optimization only – no run-time algorithm selection or (statistical) knowledge about the data.
- Assumes *h* identical hosts constantly running, (the old MPI/HPC way, Hadoop/Spark do block-level scheduling).
- Memory management
- Predictability and scalability to 1 million cores

Current and Future Work

- Open-Source at `http://project-thrill.org` and Github.
- High quality, very modern C++14 code.
- A K-Mean tutorial is available!



Step 1: Generate Random Points

Welcome to the first step in the Thrill k-means tutorial. This tutorial will show how to implement the k-means clustering algorithm (Lloyd's algorithm) in Thrill.

The algorithm works as follows: Given a set of d-dimensional points, select k initial cluster center points at random. Then attempt to improve the centers by iteratively calculating new centers. This is done by classifying all points and associating them with their nearest center, and then taking the mean of all points associated to one cluster as the new center. This will be repeated a constant number of iterations.

Ideas for Future Work:

- Distributed rank()/select() and succinct bit vectors?
- Beyond `DIA<T>`? `Graph<V,E>`? `Matrix<T>`?
- Fault tolerance in the algorithms and scalability to large clusters.
- Stream/Micro-batch or Query processing framework?

Thank you for your attention!

Questions?