CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

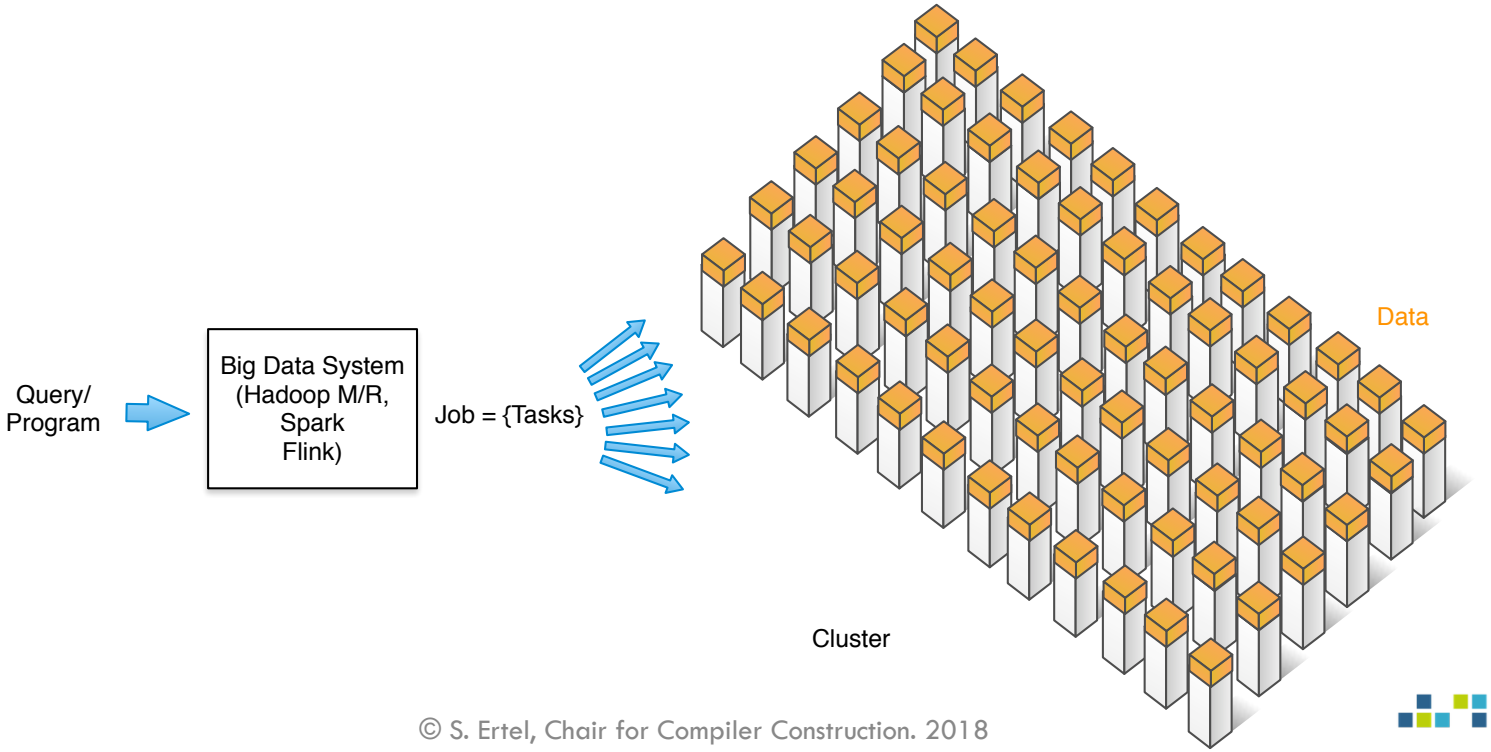
Supporting Fine-grained Dataflow Parallelism in Big Data Systems

Sebastian Ertel, Justus Adam and Jeronimo Castrillon

Chair for Compiler Construction
TU Dresden

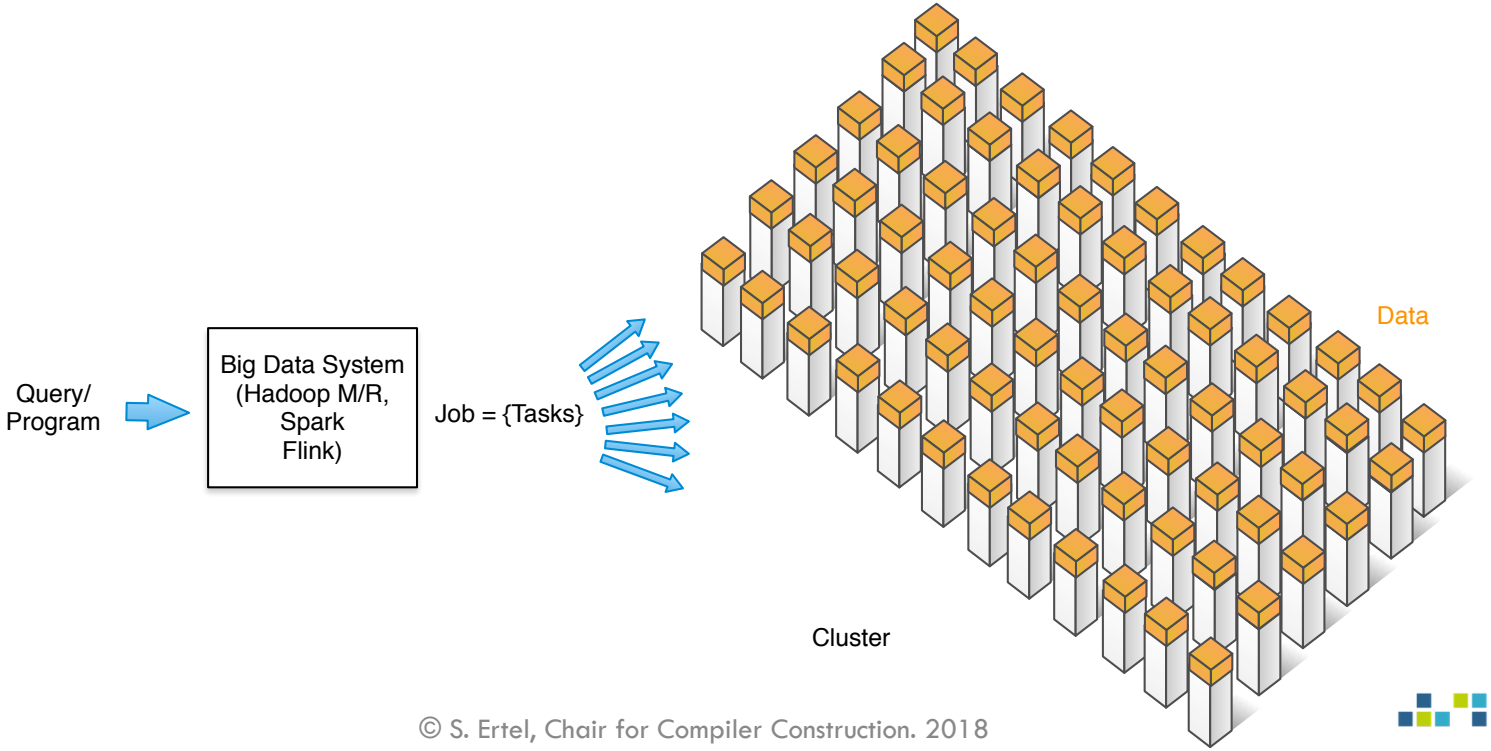
9th International Workshop on Programming Models and Applications for Multicores and Manycores
Vienna, 25.2.2018

Big Data Systems



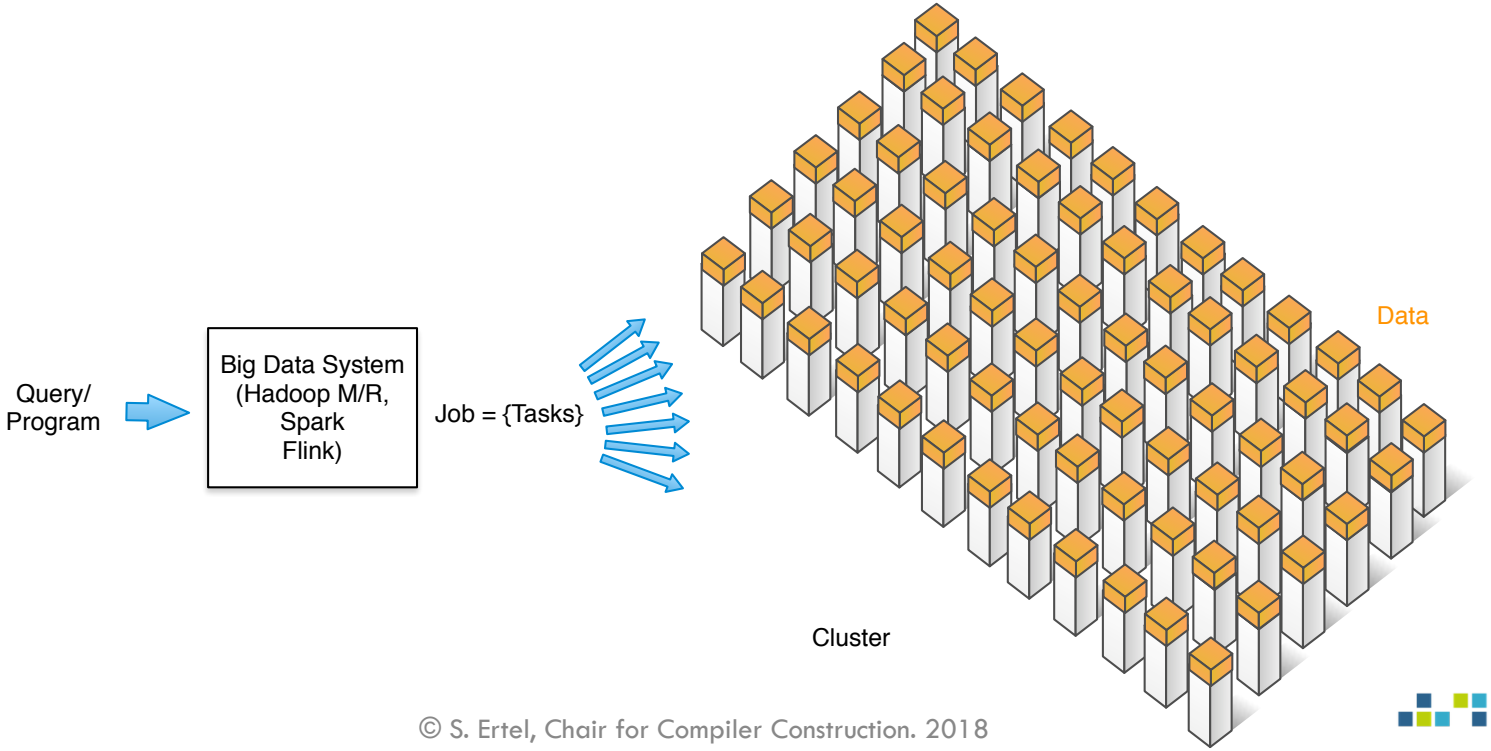
Big Data Systems

1. Big Data Systems (BDSs) scale with the number of cores in the cluster
2. The main bottleneck is I/O (disk and network)



Big Data Systems

1. Big Data Systems (BDSs) scale with the number of cores in the cluster for independent tasks.
2. The main bottleneck is I/O (disk and network) for simple data.



New Insights

1. Big Data Systems (BDSs) scale with the number of cores in the cluster for independent tasks.
2. The main bottleneck is I/O (disk and network) for simple data.

Jobs are CPU-bound!

WordCount, Sort		Analytics Queries (in Hive)
Simple Data Formats	vs.	Complex Data Formats (Parquet, Tables, JSON)
Uncompressed		Compressed

Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung-Gon Chun. 2015. Making sense of performance in data analytics frameworks.(NSDI'15).

New Insights

1. Big Data Systems (BDSs) scale with the number of cores in the cluster for independent tasks.
2. The main bottleneck is I/O (disk and network) for simple data.

Jobs are CPU-bound!

WordCount, Sort		Analytics Queries (in Hive)
Simple Data Formats	vs.	Complex Data Formats (Parquet, Tables, JSON)
Uncompressed		Compressed

Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung-Gon Chun. 2015. Making sense of performance in data analytics frameworks.(NSDI'15).

BDS cores/data processing pipelines do not scale on multicores!

- Local optimizations do not solve this problem.
- BDSs do not benefit from new network HW.

➡ Rewrite data processing cores!

Animesh Trivedi, Patrick Stuedi, Jonas Pfefferle, Radu Stoica, Bernard Metzler, Ioannis Koltsidas, and Nikolas Ioannou. 2016. On the [ir]relevance of network performance for data processing. (HotCloud'16).

Challenge Accepted!

BDS core study

(Trivedi et.al., 2016) results mostly experimental.

↪ What is the inherent pattern that all these BDS cores suffer from?

Code study: Hadoop M/R (HMR), Spark, Flink

Challenge Accepted!

BDS core study ➡ Implicit Parallel Programming

(Trivedi et.al., 2016) results mostly experimental.

↪ What is the inherent pattern that all these BDS cores suffer from?

Code study: Hadoop M/R (HMR), Spark, Flink

BDS cores are already complex:

Framework	Java	Scala	Python
HMR 2.7.3	1402899	0	406
Spark 2.11	99118	344613	28079
Flink 2.10	491339	81549	2814

↪ Threads and Co. make them even more complex and harder to maintain!

Challenge Accepted!

BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

(Trivedi et.al., 2016) results mostly experimental.

↪ What is the inherent pattern that all these BDS cores suffer from?

Code study: Hadoop M/R (HMR), Spark, Flink

BDS cores are already complex:

Framework	Java	Scala	Python
HMR 2.7.3	1402899	0	406
Spark 2.11	99118	344613	28079
Flink 2.10	491339	81549	2814

↪ Threads and Co. make them even more complex and harder to maintain!

If all BDS cores are similar ↪ Representative rewrite(s) for the HMR (map task) core.

Inherent Parallelism in BDS cores

BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

```java
public class Mapper<KEYIN, VALUEIN,
                    KEYOUT, VALUEOUT> {
 /* The default implementation
    is the identity function. */
 protected
 void map(KEYIN key, VALUEIN value,
          Context ctxt) {
  ctxt.write((KEYOUT) key,
             (VALUEOUT) value);
 }

 public
 void run(Context ctxt) {
  while (ctxt.nextKeyValue())
    map(ctxt.getCurrentKey(),
        ctxt.getCurrentValue(),
        ctxt);
 }}
```

(a) Hadoop

```scala
private[spark] class
ShuffleMapTask(partitionId: Int,
               partition: Partition)
 extends Task[MapStatus] {

 override
 def runTask(ctxt: TaskContext)
 :MapStatus = {
 /* Deserialization and init of
    variables omitted for brevity. */
  var writer: ShuffleWriter[Any, Any] =
   manager.getWriter[Any, Any](
        dep.shuffleHandle,
        partitionId, ctxt)
 writer.write(
  rdd.iterator(partition, ctxt)
     .asInstanceOf[
      Iterator[_<:Product2[Any, Any]]])
 writer.stop(success = true).get}}
```

(b) Spark

```java
public class DataSourceTask<OT>
 extends AbstractInvokable {
 private
  InputFormat<OT, InputSplit> format;
 private Collector<OT> output;

 @Override public
 void invoke() throws Exception {
  OT reuse =
     serializer.createInstance();
  while (!this.taskCanceled &&
         !format.reachedEnd()) {
   OT returned;
   if((returned =
         format.nextRecord(reuse))
       != null)
     output.collect(returned);
  }}}
```

(c) Flink

Inherent Parallelism in BDS cores

BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

```java
public class Mapper<KEYIN, VALUEIN,
                    KEYOUT, VALUEOUT> {
 /* The default implementation
    is the identity function. */
 protected
 void map(KEYIN key, VALUEIN value,
          Context ctxt) {
  ctxt.write((KEYOUT) key,
             (VALUEOUT) value);
 }

 public
 void run(Context ctxt) {
  while (ctxt.nextKeyValue())
    map(ctxt.getCurrentKey(),
        ctxt.getCurrentValue(),
        ctxt);
 }}
```

(a) Hadoop

```scala
private[spark] class
ShuffleMapTask(partitionId: Int,
               partition: Partition)
 extends Task[MapStatus] {

 override
 def runTask(ctxt: TaskContext)
 :MapStatus = {
 /* Deserialization and init of
    variables omitted for brevity. */
  var writer: ShuffleWriter[Any, Any] =
   manager.getWriter[Any, Any](
        dep.shuffleHandle,
        partitionId, ctxt)
 writer.write(
  rdd.iterator(partition, ctxt)
     .asInstanceOf[
      Iterator[_<:Product2[Any, Any]]])
 writer.stop(success = true).get}}
```

(b) Spark

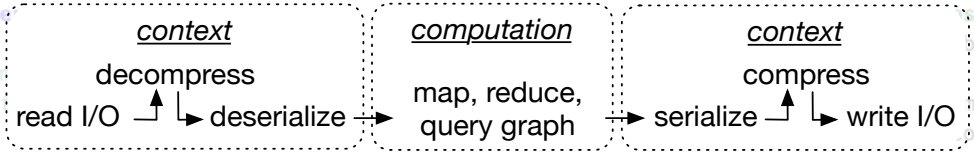
```java
public class DataSourceTask<OT>
 extends AbstractInvokable {
 private
  InputFormat<OT, InputSplit> format;
 private Collector<OT> output;

 @Override public
 void invoke() throws Exception {
  OT reuse =
     serializer.createInstance();
  while (!this.taskCanceled &&
         !format.reachedEnd()) {
   OT returned;
   if((returned =
         format.nextRecord(reuse))
       != null)
     output.collect(returned);
  }}}
```

(c) Flink

Inherent Parallelism in BDS cores

BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite



Data processing pipeline: Iterator or Observer

➡ Duals of each other!

➡ Pattern for pipeline parallelism.

More parallelism:

Keys vs. values ➡ Task-level parallelism.

Stateless iterators ➡ Data parallelism.

Erik Meijer. Subject/Observer is Dual to Iterator. 2010. PLDI, Fun Ideas and Thoughts Session.

Ohua

BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

Implicit = No concurrency/parallelism abstractions, only functions/algorithms and variables.

DSL

Dataflow Runtime

Ohua

BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

Implicit = No concurrency/parallelism abstractions, only functions/algorithms and variables.

DSL

t ::= v
| (algo [v] t)
| (t t)
| (let [v t] t)
| (if t t t)
| (sf_{JVM} v_1 … v_n)
| (seq t t)

v ::= $v \in V_{JVM}$
| [v_1 … v_n]

(smap (algo [v] t) [v_1 … v_n])

Dataflow Runtime

Ohua

BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

Implicit = No concurrency/parallelism abstractions, only functions/algorithms and variables.

DSL

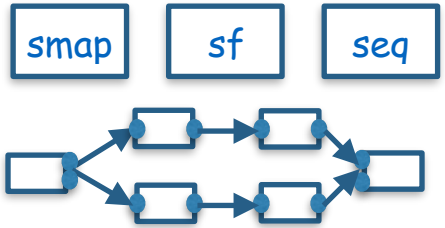
t ::= v
| (algo [v] t)
| (t t)
| (let [v t] t)
| (if t t t)
| (sf_{JVM} v_1 … v_n)
| (seq t t)

v ::= $v \in V_{JVM}$
| [v_1 … v_n]

(smap (algo [v] t) [v_1 … v_n])

Dataflow Runtime

t ::= → (arc)
| ● (port)
| □ (node)



- Threads/actors/…
- Queues/channels/…
- Scheduler

Rewriting the HMR Map Task

BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

Ready! Set! Go!

```
1  public class Mapper<KEYIN, VALUEIN,
2                      KEYOUT, VALUEOUT> {
3   /* The default implementation
4      is the identity function. */
5   protected
6   void map(KEYIN key, VALUEIN value,
7            Context ctxt) {
8    ctxt.write((KEYOUT) key,
9               (VALUEOUT) value);
10  }
11
12  public
13  void run(Context ctxt) {
14   while (ctxt.nextKeyValue())
15     map(ctxt.getCurrentKey(),
16         ctxt.getCurrentValue(),
17         ctxt);
18  }}
```

(a) Hadoop

Rewriting the HMR Map Task

BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

Ready! Set! Go!

```
1  public class Mapper<KEYIN, VALUEIN,
2                      KEYOUT, VALUEOUT> {
3   /* The default implementation
4      is the identity function. */
5   protected
6   void map(KEYIN key, VALUEIN value,
7            Context ctxt) {
8    ctxt.write((KEYOUT) key,
9               (VALUEOUT) value);
10  }
11
12  public
13  void run(Context ctxt) {
14   while (ctxt.nextKeyValue())
15     map(ctxt.getCurrentKey(),
16         ctxt.getCurrentValue(),
17         ctxt);
18  }}
```

(a) Hadoop

Rewriting the HMR Map Task

BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

Ready! Set! Go!

```
public class Mapper<KEYIN, VALUEIN,
                    KEYOUT, VALUEOUT> {
```

```
(defn coarse
  [^org.apache.hadoop.mapreduce.Mapper$Context reader
   ^org.apache.hadoop.mapreduce.Mapper mapper
   ^org.apache.hadoop.mapreduce.Mapper$Context writer]
  (let [records-on-disk (new InputIterator reader)]
    (ohua
      (smap
        (algo compute-and-output  [ [line content] ]
          (let [kv-pairs (hmr-map line content mapper)]
            (smap
              (algo output-side [ [k v] ] (output k v writer))
               kv-pairs)))
         records-on-disk))))
```

```
 }}
```

(a) Hadoop

Rewriting the HMR Map Task

BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

Ready! Set! Go!

Clojure function

```
public class Mapper<KEYIN, VAL
                    KEYOUT, ...
```

```
(defn coarse
  [^org.apache.hadoop.mapreduce.Mapper$Context reader
   ^org.apache.hadoop.mapreduce.Mapper mapper
   ^org.apache.hadoop.mapreduce.Mapper$Context writer]
  (let [records-on-disk (new InputIterator reader)]
    (ohua
      (smap
        (algo compute-and-output  [ [line content] ]
          (let [kv-pairs (hmr-map line content mapper)]
            (smap
              (algo output-side [ [k v] ] (output k v writer))
               kv-pairs)))
         records-on-disk))))
```

```
}}
```

(a) Hadoop

Rewriting the HMR Map Task

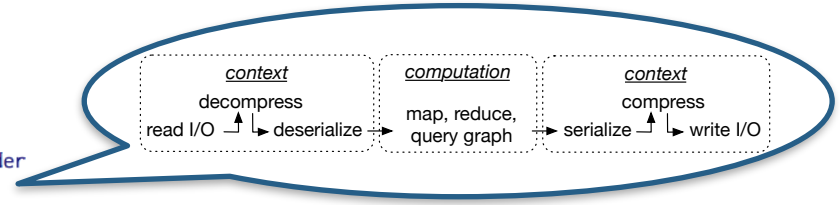
BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

Ready! Set! Go!

```
public class Mapper<KEYIN, VALUEIN,
                    KEYOUT, VALUEOUT> {
```

```
(defn coarse
  [^org.apache.hadoop.mapreduce.Mapper$Context reader
   ^org.apache.hadoop.mapreduce.Mapper mapper
   ^org.apache.hadoop.mapreduce.Mapper$Context writer]
  (let [records-on-disk (new InputIterator reader)]
    (ohua
      (smap
        (algo compute-and-output  [ [line content] ]
          (let [kv-pairs (hmr-map line content mapper)]
            (smap
              (algo output-side [ [k v] ] (output k v writer))
               kv-pairs)))
         records-on-disk))))
 }}
```

(a) Hadoop



Rewriting the HMR Map Task

BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

Ready! Set! Go!

```
public class Mapper<KEYIN, VALUEIN,
                    KEYOUT, VALUEOUT> {
```

```
(defn coarse
  [^org.apache.hadoop.mapreduce.Mapper$Cont
   ^org.apache.hadoop.mapreduce.Mapper mapp
   ^org.apache.hadoop.mapreduce.Mapper$Cont ext writer]
  (let [records-on-disk (new InputIterator reader)]
    (ohua
      (smap
        (algo compute-and-output  [ [line content] ]
          (let [kv-pairs (hmr-map line content mapper)]
            (smap
              (algo output-side [ [k v] ] (output k v writer))
               kv-pairs)))
         records-on-disk))))
 }}
```

BDS core written in Ohua

(a) Hadoop

Rewriting the HMR Map Task

BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

Ready! Set! Go!

```
public class Mapper<KEYIN, VALUEIN,
                    KEYOUT, VALUEOUT> {
```

```
(defn coarse
  [^org.apache.hadoop.mapreduce.Mapper$Context reader
   ^org.apache.hadoop.mapreduce.Mapper mapper
   ^org.apache.hadoop.mapreduce.Mapper$Context writer]
  (let [records-on-disk (new InputIterator reader)]
    (ohua
      (smap
        (algo compute-and-output  [ [line content] ]
          (let [kv-pairs (hmr-map line content mapper)]
            (smap
              (algo output-side [ [k v] ] (output k v writer))
               kv-pairs)))
         records-on-disk))))
```

```
 }}
```

(a) Hadoop

Rewriting the HMR Map Task

BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

Ready! Set! Go!

```
public class Mapper<KEYIN, VALUEIN,
                    KEYOUT, VALUEOUT> {
```

```
(defn coarse
  [^org.apache.hadoop.mapreduce.Mapper$Context reader
   ^org.apache.hadoop.mapreduce.Mapper mapper
   ^org.apache.hadoop.mapreduce.Mapper$Context writer]
  (let [records-on-disk (new InputIterator reader)]
    (ohua
      (smap
        (algo compute-and-output  [ [line content] ]
          (let [kv-pairs (hmr-map line content mapper)]
            (smap
              (algo output-side [ [k v] ] (output k v writer))
               kv-pairs)))
          records-on-disk))))
 }}
```

(a) Hadoop

Rewriting the HMR Map Task

BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

Ready! Set! Go!

```
public class Mapper<KEYIN, VALUEIN,
                    KEYOUT, VALUEOUT> {
```

```
(defn coarse
  [^org.apache.hadoop.mapreduce.Mapper$Context reader
   ^org.apache.hadoop.mapreduce.Mapper mapper
   ^org.apache.hadoop.mapreduce.Mapper$Context writer]
  (let [records-on-disk (new InputIterator reader)]
    (ohua
      (smap
        (algo compute-and-output  [ [line content] ]
          (let [kv-pairs (hmr-map line content mapper)]
            (smap
              (algo output-side [ [k v] ] (output k v writer))
               kv-pairs)))
         records-on-disk))))
 }}
```

(a) Hadoop

```
public class Output {
  @defsfn public
  void output(Object key, Object value, Context ctxt) {
    ctxt.write(key, value); }}
```

Rewriting the HMR Map Task

BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

Ready! Set! Go!

```
public class Mapper<KEYIN, VALUEIN,
                    KEYOUT, VALUEOUT> {
```

```
(defn coarse
  [^org.apache.hadoop.mapreduce.Mapper$Context reader
   ^org.apache.hadoop.mapreduce.Mapper mapper
   ^org.apache.hadoop.mapreduce.Mapper$Context writer]
  (let [records-on-disk (new InputIterator reader)]
    (ohua
      (smap
        (algo compute-and-output  [ [line content] ]
          (let [kv-pairs (hmr-map line content mapper)]
            (smap
              (algo output-side [ [k v] ] (output k v writer))
               kv-pairs)))
         records-on-disk))))
```

(a) Hadoop

```
public class Output {
  @defsfn public
  void output(Object key, Object value, Context ctxt) {
    ctxt.write(key, value); }}
```

Java	Ohua
o.output(k, v, writer);	(output k v writer)

Rewriting the HMR Map Task

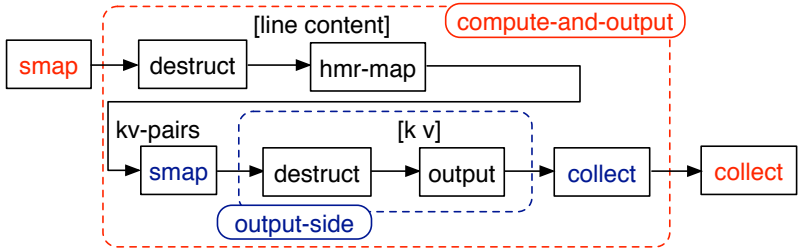
BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

Ready! Set! Go!

```
public class Mapper<KEYIN, VALUEIN,
                    KEYOUT, VALUEOUT> {
```

(a) Hadoop

```
(defn coarse
  [^org.apache.hadoop.mapreduce.Mapper$Context reader
   ^org.apache.hadoop.mapreduce.Mapper mapper
   ^org.apache.hadoop.mapreduce.Mapper$Context writer]
  (let [records-on-disk (new InputIterator reader)]
    (ohua
      (smap
        (algo compute-and-output  [ [line content] ]
          (let [kv-pairs (hmr-map line content mapper)]
            (smap
              (algo output-side [ [k v] ] (output k v writer))
               kv-pairs)))
         records-on-disk))))
```




```
public class Output {
  @defsfn public
  void output(Object key, Object value, Context ctxt) {
    ctxt.write(key, value); }}
```

Java

```
o.output(k, v, writer);
```

Ohua

```
(output k v writer)
```

Rewriting the HMR Map Task

BDS core study ➡ Implicit Parallel Programming ➡ BDS core rewrite

Ready! Set! Go!

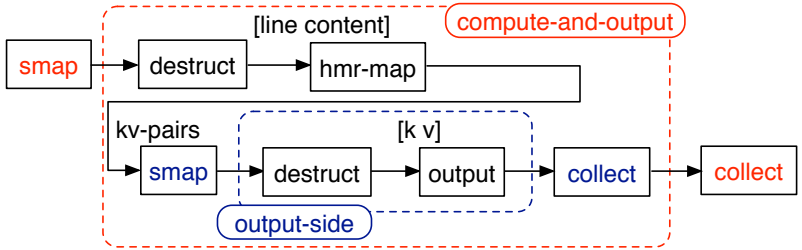
```
public class Mapper<KEYIN, VALUEIN,
                    KEYOUT, VALUEOUT> {
```

```
(defn coarse
  [^org.apache.hadoop.mapreduce.Mapper$Context reader
   ^org.apache.hadoop.mapreduce.Mapper mapper
   ^org.apache.hadoop.mapreduce.Mapper$Context writer]
  (let [records-on-disk (new InputIterator reader)]
    (ohua
      (smap
        (algo compute-and-output  [ [line content] ]
          (let [kv-pairs (hmr-map line content mapper)]
            (smap
              (algo output-side [ [k v] ] (output k v writer))
               kv-pairs)))
         records-on-disk))))
```

```
}}
```

(a) Hadoop

4 variants: Coarse (C) - Coarse Input Fine Output (CIFO) - Fine Input Coarse Output (FICO) - Fine (F)




```
public class Output {
  @defsfn public
  void output(Object key, Object value, Context ctxt) {
    ctxt.write(key, value); }}
```

Java

```
o.output(k, v, writer);
```

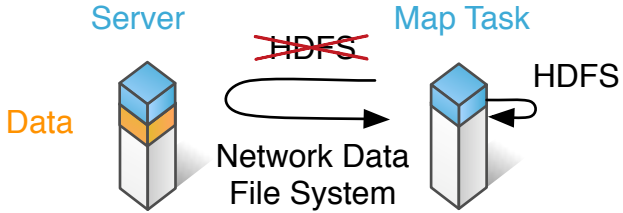
Ohua

```
(output k v writer)
```

Evaluation – Setup

Intel NUMA 2.6 GHZ

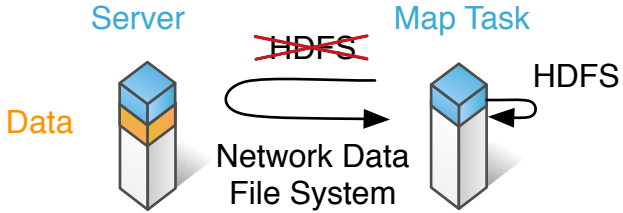
- 2 CPU Sockets
- 12 cores (24 HW threads)
- 128 GB RAM



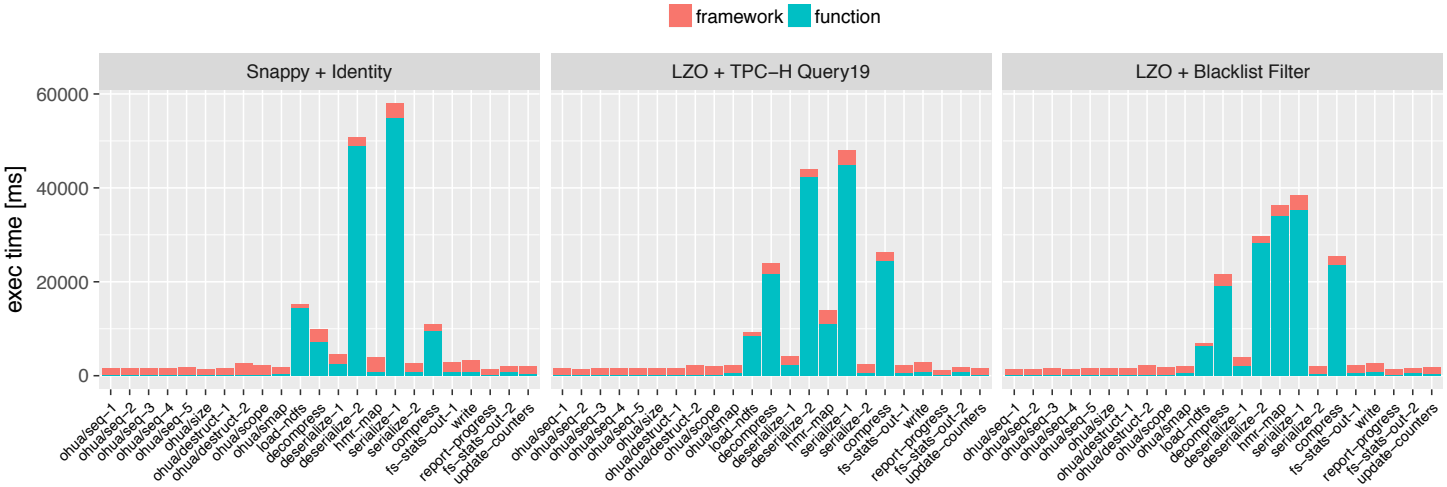
- SequenceFile
- JSON
- Snappy, LZO
- TPC-H Parts Table

Evaluation - Execution Breakdown

Intel NUMA 2.6 GHZ
- 2 CPU Sockets
- 12 cores (24 HW threads)
- 128 GB RAM

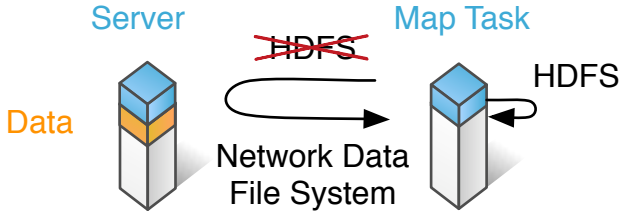


- SequenceFile
- JSON
- Snappy, LZO
- TPC-H Parts Table

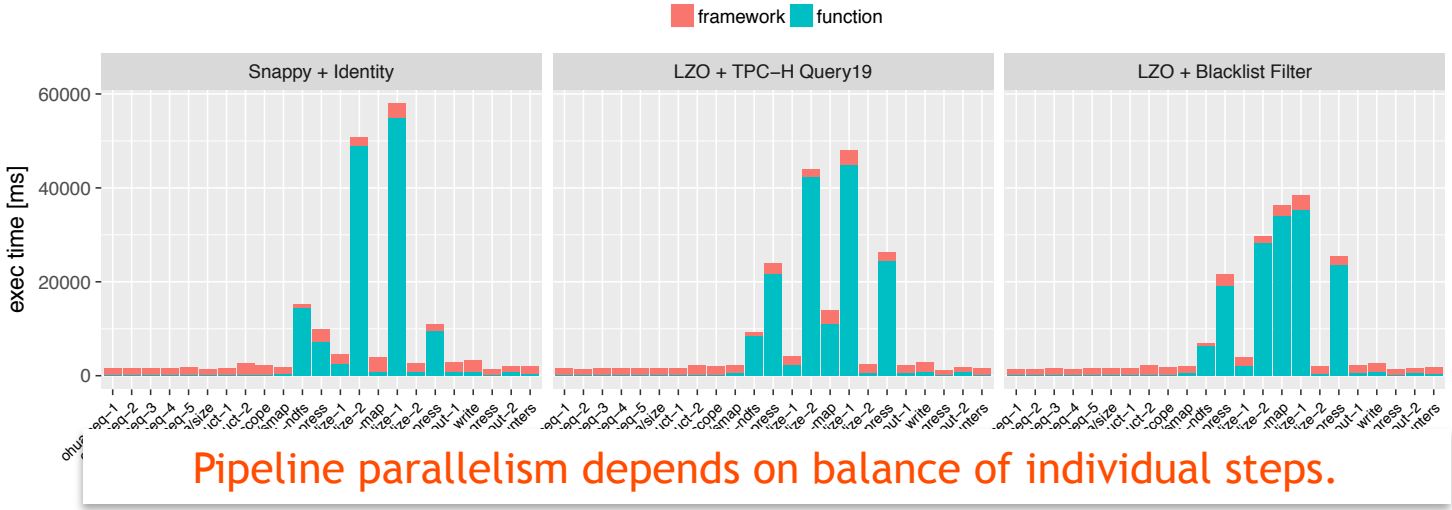


Evaluation - Execution Breakdown

Intel NUMA 2.6 GHZ
- 2 CPU Sockets
- 12 cores (24 HW threads)
- 128 GB RAM



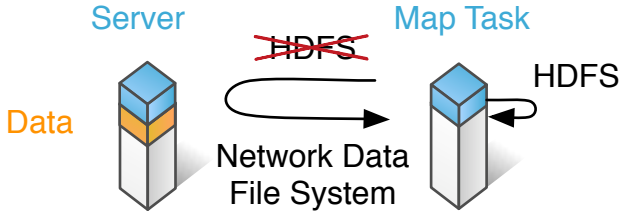
- SequenceFile
- JSON
- Snappy, LZO
- TPC-H Parts Table



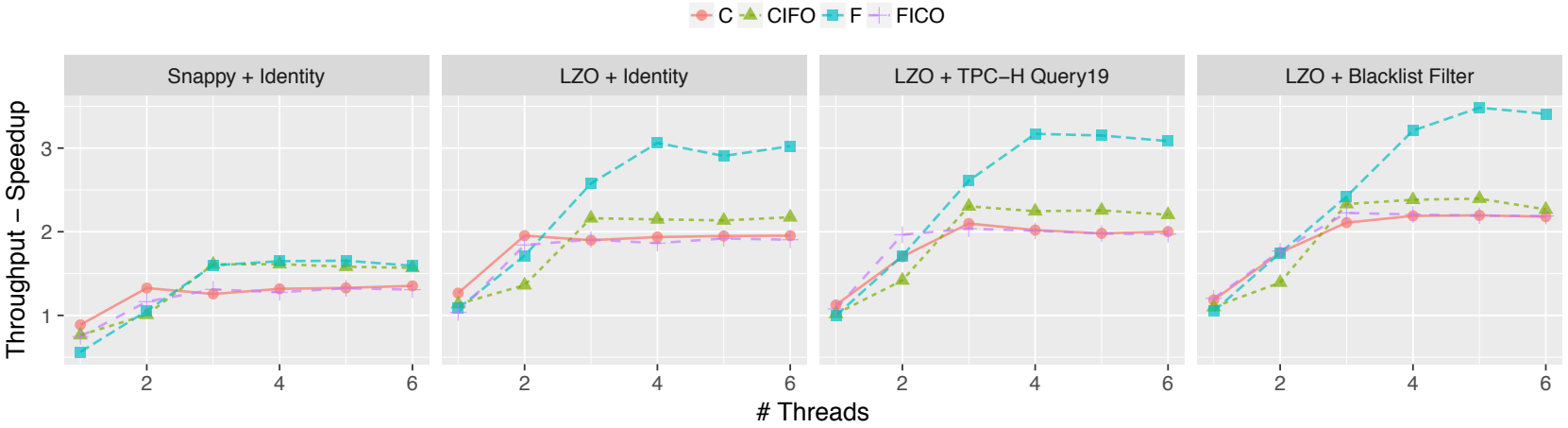
Pipeline parallelism depends on balance of individual steps.

Evaluation - Throughput Analysis

Intel NUMA 2.6 GHZ
- 2 CPU Sockets
- 12 cores (24 HW threads)
- 128 GB RAM

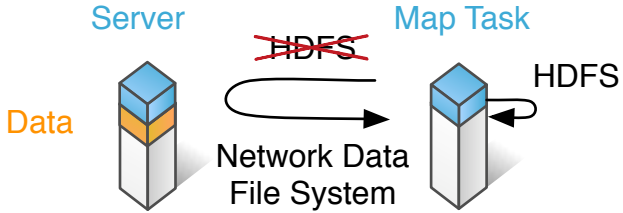


- SequenceFile
- JSON
- Snappy, LZO
- TPC-H Parts Table

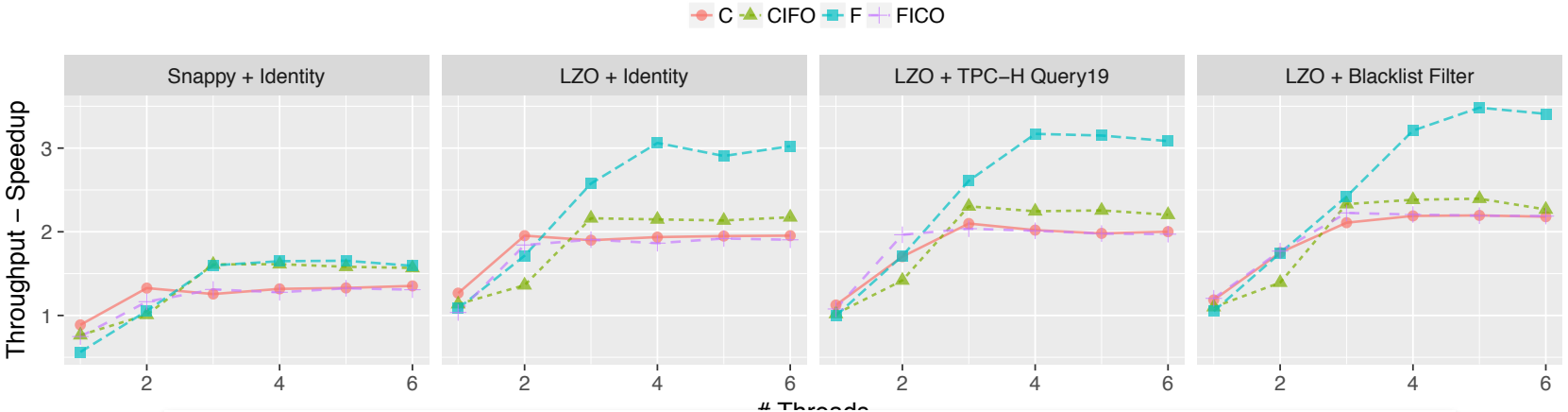


Evaluation - Throughput Analysis

Intel NUMA 2.6 GHZ
- 2 CPU Sockets
- 12 cores (24 HW threads)
- 128 GB RAM



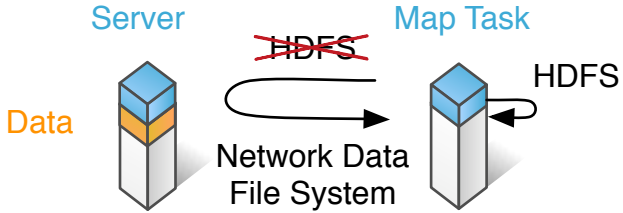
- SequenceFile
- JSON
- Snappy, LZO
- TPC-H Parts Table



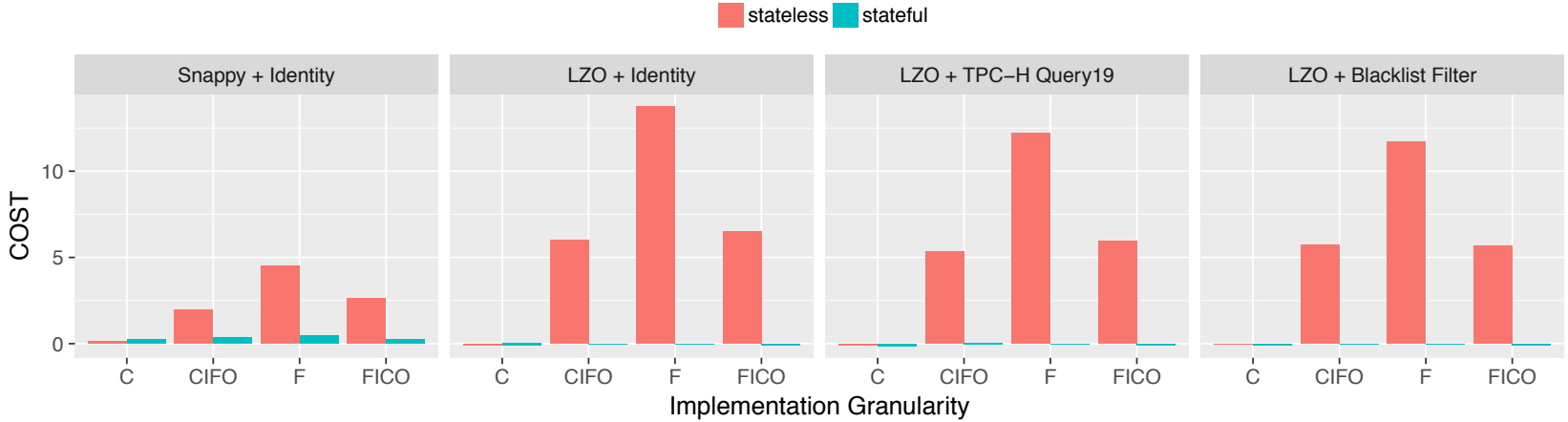
Speedup of up to 3.5x for compute-intensive configurations!

Evaluation - COST Analysis

Intel NUMA 2.6 GHZ
- 2 CPU Sockets
- 12 cores (24 HW threads)
- 128 GB RAM



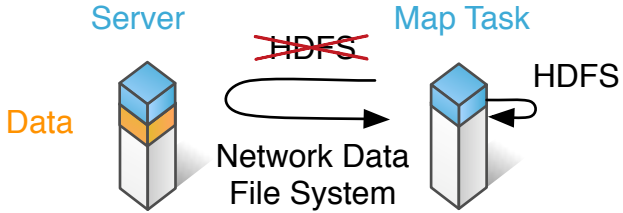
- SequenceFile
- JSON
- Snappy, LZO
- TPC-H Parts Table



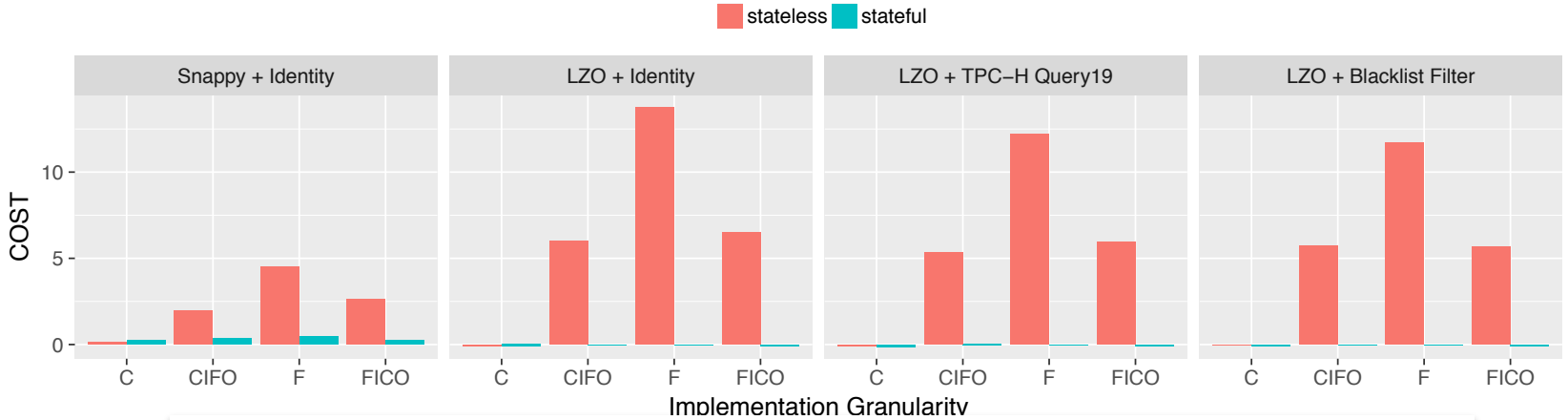
Frank McSherry, Michael Isard, and Derek G. Murray. 2015. Scalability! but at what cost? (HotOS'15).

Evaluation - COST Analysis

Intel NUMA 2.6 GHZ
- 2 CPU Sockets
- 12 cores (24 HW threads)
- 128 GB RAM



- SequenceFile
- JSON
- Snappy, LZO
- TPC-H Parts Table



Processing is inherently stateful! ➡ Cache for (de)compression.

Frank McSherry, Michael Isard, and Derek G. Murray. 2015. Scalability! but at what cost? (HotOS'15).

Conclusion

- Data processing cores of state-of-the-art BDS are all similar,
- and be rewritten with low effort to scale on multi- and manicures.

If you do or (write a new BDS):
Consider using an implicit parallel programming language such as Ohua!

☑ Clean, concise and modular code structure,
☑ without concurrency abstractions.
☑ Use the associated compiler and runtime system to adapt to scale to new/heterogenous HW