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 System (Hadoop M/R, Spark, Flink)

Query/Program

Job = \{Tasks\}

Cluster

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

Analytics Queries (in Hive)  
Complex Data Formats (Parquet, Tables, JSON)  
Compressed

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!

<table>
<thead>
<tr>
<th>Simple Data Formats</th>
<th>WordCount, Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Uncompressed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Complex Data Formats (Parquet, Tables, JSON)</th>
<th>Analytics Queries (in Hive)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compressed</td>
<td>vs.</td>
</tr>
</tbody>
</table>


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!

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:

<table>
<thead>
<tr>
<th>Framework</th>
<th>Java</th>
<th>Scala</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMR 2.7.3</td>
<td>1402899</td>
<td>0</td>
<td>406</td>
</tr>
<tr>
<td>Spark 2.11</td>
<td>99118</td>
<td>344613</td>
<td>28079</td>
</tr>
<tr>
<td>Flink 2.10</td>
<td>491339</td>
<td>81549</td>
<td>2814</td>
</tr>
</tbody>
</table>

↪ 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:

<table>
<thead>
<tr>
<th>Framework</th>
<th>Java</th>
<th>Scala</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMR 2.7.3</td>
<td>1402899</td>
<td>0</td>
<td>406</td>
</tr>
<tr>
<td>Spark 2.11</td>
<td>99118</td>
<td>344613</td>
<td>28079</td>
</tr>
<tr>
<td>Flink 2.10</td>
<td>491339</td>
<td>81549</td>
<td>2814</td>
</tr>
</tbody>
</table>

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

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

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

private[spark] class ShuffleMapTask(partitionId: Int, partition: Partition) extends Task[MapStatus] {
    override def runTask(ctx: TaskContext) : MapStatus = {
        /* Deserialization and init of variables omitted for brevity. */
        var writer: ShuffleWriter[Any, Any] =
            manager.getWriter[Any, Any](
                dep.shuffleHandle, partitionId, ctx)
        writer.write(
            rdd.iterator(partition, ctx).asInstanceOf[
                Iterator[_:<Product2[Any, Any]]].value)
        writer.stop(success = true).get
    }

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);
        }
    }
```

(a) Hadoop  
(b) Spark  
(c) Flink
Inherent Parallelism in BDS cores

BDS core study ➔ Implicit Parallel Programming ➔ BDS core rewrite

(a) Hadoop

```java
public class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
    // The default implementation
    Map<KEYOUT, VALUEOUT> map(KEYIN key, VALUEIN value, Context ctx) {
        // Deserialization and init of variables omitted for brevity.
        var writer: ShuffleWriter<Any, Any> = manager.getWriter<Any, Any>[
            dep.shuffleHandle,
            partitionId, ctx]
        writer.write[
            rdd.iterator(partition, ctx),
            .asInstanceOf[
                Iterator[<Product2<Any, Any>]]]
        writer.stop(success = true).get}
    }
}
```

(b) Spark

```java
private[spark] class ShuffleMapTask(partitionId: Int, partition: Partition) extends Task[MapStatus] {
    override def runTask(ctx: TaskContext): MapStatus = {
        var writer: ShuffleWriter<Any, Any> = manager.getWriter<Any, Any>[
            dep.shuffleHandle, partitionId, ctx]
        writer.write[
            rdd.iterator(partition, ctx),
            .asInstanceOf[
                Iterator[<Product2<Any, Any>]]]
        writer.stop(success = true).get}
    }
}
```

(c) Flink

```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);
            }
        }
    }
}
```
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.
Implicit = No concurrency/parallelism abstractions, only functions/algorithms and variables.

**DSL**

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

**DSL**

\[
\begin{align*}
\tau & ::= \nu \\
& \quad | \ (\text{algo } [\nu] \ \tau) \\
& \quad | \ (\tau \ \tau) \\
& \quad | \ (\text{let } [\nu \ \tau] \ \tau) \\
& \quad | \ (\text{if } \tau \ \tau \ \tau) \\
& \quad | \ (\text{sf}_{\text{JVM}} \ v_1 \ \cdots \ v_n) \\
& \quad | \ (\text{seq}_{\text{JVM}} \ v_1 \ \cdots \ v_n) \\
& \quad | \ (\text{smap} \ (\text{algo} \ [\nu] \ \tau) [v_1 \ \cdots \ v_n])
\end{align*}
\]

**Dataflow Runtime**

\[
v ::= \nu \in V_{\text{JVM}} \\
\]
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 v1 … vn)`
- `(seq t t)`
- `(smap (algo [v] t) [v1 ... vn])`

**Dataflow Runtime**

- `t ::= (arc)`
- `(port)`
- `(node)`

- `v ::= v ∈ V_{JVM}`
- `[v1 … vn]`

- `smap`  `sf`  `seq`

- Threads/actors/…
- Queues/channels/…
- Scheduler
Rewriting the HMR Map Task

Ready! Set! Go!

```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
Rewriting the HMR Map Task

BDS core study ➔ Implicit Parallel Programming ➔ BDS core rewrite

Ready! Set! Go!

```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
```

© S. Ertel, Chair for Compiler Construction. 2018
Rewriting the HMR Map Task

BDS core study ➔ Implicit Parallel Programming ➔ BDS core rewrite

Ready! Set! Go!

```java
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)]
            (huya
                (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

Ready! Set! Go!

```clojure
(defn coarse
  [:org.apache.hadoop.mapreduce.Mapper$Context reader
   :org.apache.hadoop.mapreduce.Mapper$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

Ready! Set! Go!

```java
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

Ready! Set! Go!

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

    (define coarse
        [^org.apache.hadoop.mapreduce.Mapper$Context
            ^org.apache.hadoop.mapreduce.Mapper mapper
                ^org.apache.hadoop.mapreduce.Mapper$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

Ready! Set! Go!

```java
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

Ready! Set! Go!

```java
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)]
            (hhua
                (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

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); 
  }
```

© S. Ertel, Chair for Compiler Construction. 2018
Rewriting the HMR Map Task

Ready! Set! Go!

```java
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
        )
    )
}
```

```java
public class Output {
    @defsfm 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

Ready! Set! Go!

```java
class Mapper<KEYIN, VALUEIN, 
   KEYOUT, VALUEOUT> {
    @defn coarse
    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); }
}
```

(a) Hadoop
Rewriting the HMR Map Task

Ready! Set! Go!

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

Java
```java
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)
    }
```

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

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

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

---

![Diagram showing data flow from Server to Map Task through HDFS]

<table>
<thead>
<tr>
<th>Data Source</th>
<th>File Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Data</td>
<td>SequenceFile, JSON, Snappy, LZO</td>
</tr>
<tr>
<td>HDFS</td>
<td>TPC-H Parts Table</td>
</tr>
</tbody>
</table>

---

![Graphs showing throughput and speedup for different data formats and configurations]

# Threads

## Throughput - Speedup

- Snappy + Identity
- LZO + Identity
- LZO + TPC-H Query19
- LZO + Blacklist Filter

© S. Ertel, Chair for Compiler Construction. 2018
Evaluation - Throughput Analysis

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

Throughput - Speedup analysis

- 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

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

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

© S. Ertel, Chair for Compiler Construction. 2018

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