Coroutines & Data Stream Processing

An application of an almost forgotten concept in distributed computing

Zbyněk Šlajchrt, slajchrt@avast.com

"Strange" Iterator Example
Coroutines
MapReduce & Coroutines
Clockwork
Conclusion
Q&A

Joining Multiple Iterators To One

```
public class JoiningIterators {

    public static void main(String[] args) {

        Iterator<String> iterator1 = Arrays.asList("a", "b", "c").iterator();
        Iterator<String> iterator2 = Collections.<String>emptyList().iterator();
        Iterator<String> iterator3 = Arrays.asList("x").iterator();

        Iterator<String> joinedIterators = new JoinedIterators<String>(
                iterator1,
                iterator2,
                iterator3
        );

        while (joinedIterators.hasNext()) {
            String next = joinedIterators.next();
            System.out.println(next);
        }
    }

}
```

Joining Iterators - A Traditional Approach

```
public class JoinedIterators<T> implements Iterator<T> {
    private final Iterator<Iterator<T>> iteratorIterator;
    private Iterator<T> currentIterator;

    public JoinedIterators(Iterator<T>... iterators) {
        iteratorIterator = Arrays.asList(iterators).iterator();
    }

    public boolean hasNext() {
        if (currentIterator == null) {
            if (!iteratorIterator.hasNext()) {
                return false;
            } else {
                currentIterator = iteratorIterator.next();
            }
        }
        while (!currentIterator.hasNext()) {
            if (iteratorIterator.hasNext()) {
                currentIterator = iteratorIterator.next();
            } else {
                currentIterator = null;
                return false;
            }
        }
        return true;
    }

    public T next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        return currentIterator.next();
    }
```

```java
public class BufferedJoinedIterator<T> implements Iterator<T> {
    private final Iterator<T> joinedIterator;

    public BufferedJoinedIterator(Iterator<T>... iterators) {
        LinkedList<T> allElems = new LinkedList<T>();
        for (Iterator<T> iterator : iterators) {
            while (iterator.hasNext()) {
                T next = iterator.next();
                allElems.add(next);
            }
        }
        joinedIterator = allElems.iterator();
    }

    public boolean hasNext() {
        return joinedIterator.hasNext();
    }

    public T next() {
        return joinedIterator.next();
    }
```

Joining Iterators To One - A Pipe

```
public class PipeJoinedIterators<T> implements Iterator<T>, Runnable {
    private final Iterator<T>[] iterators;
    private final ArrayBlockingQueue<Object> pipe = new ArrayBlockingQueue<Object>(1);
    private static final Object STOP = new Object();

    public PipeJoinedIterators(Iterator<T>... iterators) {
        this.iterators = iterators;
        new Thread(this).start();
    }

    private void run_() throws InterruptedException {
        for (Iterator<T> iterator : iterators) {
            while (iterator.hasNext()) {
                T next = iterator.next();
                pipe.put(next);
            }
        }
        pipe.offer(STOP);
    }

    public boolean hasNext() {
        Object e;
        while ((e = pipe.peek()) == null);
        return e != STOP;
    }

    public T next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        try {
            return (T)pipe.take();
        } catch (InterruptedException e) {
            throw new IllegalStateException(e);
        }
    }
```

de.matthiasmann.continuations.CoIterator

```
public class CoJoinedIterators<T> extends CoIterator<T> {
    private final Iterator<T>[] iterators;

    public CoJoinedIterators(Iterator<T>... iterators) {
        this.iterators = iterators;
    }

    protected void run() throws SuspendExecution {
        for (Iterator<T> iterator : iterators) {
            while (iterator.hasNext()) {
                T next = iterator.next();
                produce(next);
            }
        }
    }
}
```

Coroutines

- A generalization of subroutines
- *AKA green threads, co-expressions, fibers, generators*
- Some may remember the old windows event loop
- Behavior similar to that of subroutines
- Coroutines can call other coroutines
- Execution may but may not later return to the point of invocation
- Often demonstrated on Produce/Consumer scenario

Coroutines Conceptual Example - Producer/Consumer (http://en.wikipedia.org/wiki/Coroutines)

```
var q := new queue

coroutine produce
    loop
        while q is not full
            create some new items
            add the items to q
        yield to consume

coroutine consume
    loop
        while q is not empty
            remove some items from q
            use the items
        yield to produce
```

- None of the top TIOBE languages (Java, C, C++, PHP, Basic)
- Go, Icon, Lua, Perl, Prolog, Ruby, Tcl, Simula, Python, Modula-2 …

- Buffers, phases (BufferedJoinedIterator)
- Emulations by threads (PipeJoinedIterator)
- Byte-code manipulation (CoJoinedIterator)
- A need for JVM support, some future JSR?
- For some implementations see the references
- In this presentation I use Continuations library developed by Matthias Mann (http://www.matthiasmann.de/content/view/24/26/)

See: http://ssw.jku.at/General/Staff/LS/coro/CoroIntroduction.pdf

- Iterators
- Producer/Consumer chains
- State machines
- Visitors with loops instead of callbacks
- Pull parsers
- Loggers
- Observers, listeners, notifications
- Generally capable of converting PUSH algorithms to PULL

MapReduce & Coroutines

- PULL, batch processing, offline
- Two-phase computational mode: Map and Reduce
- Map - filters, cleans or parses the input records
- Reduce - aggregates the records obtained from Map
- Easily distributable
- Inspired by functional programming
- Benefits - scalable, thread-safe (no race conditions), simple computational model
- Rich libraries of algorithms - e.g. machine learning (Mahout)

Map-Reduce Overview - Word Count example (http://en.wikipedia.org/wiki/MapReduce)

```
function map(String name, String document):
  // name: document name
  // document: document contents
  for each word w in document:
    emit (w, 1)

function reduce(String word, Iterator partialCounts):
  // word: a word
  // partialCounts: a list of aggregated partial counts
  sum = 0
  for each pc in partialCounts:
    sum += ParseInt(pc)
  emit (word, sum)
```

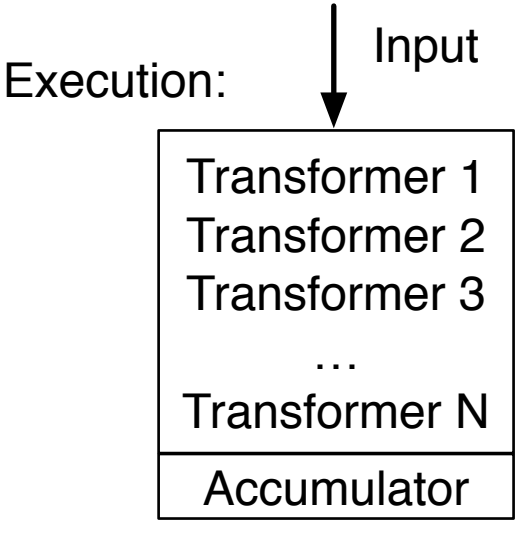
- Could we write the same code for data streams?
- We could keep thinking in MR paradigm
- Perhaps, it would inherit the nice MR properties
- Sadly, streaming algorithms are inherently PUSH, incremental or event-driven, i.e. callbacks instead of loops
- WordCount MR solution has 2 simple loops
- It is a typical Producer/Consumer problem
- Coroutines should help us!

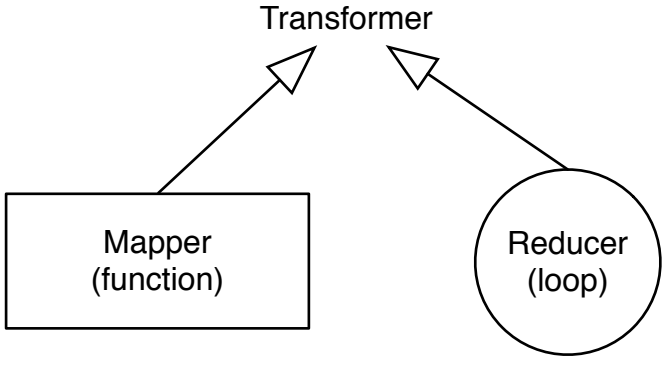
Clockwork

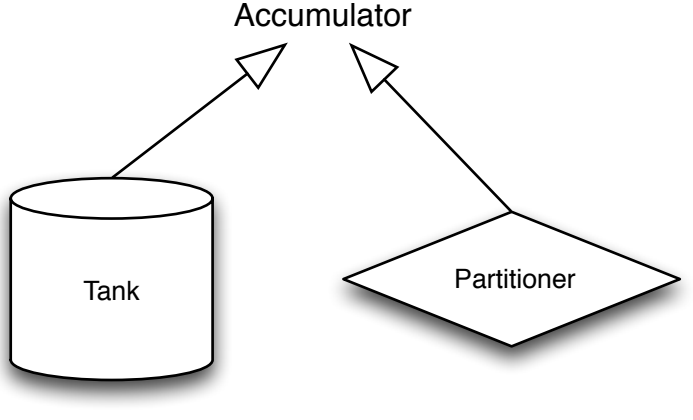
Adoption of MR to stream data processing

See https://github.com/avast-open/clockwork

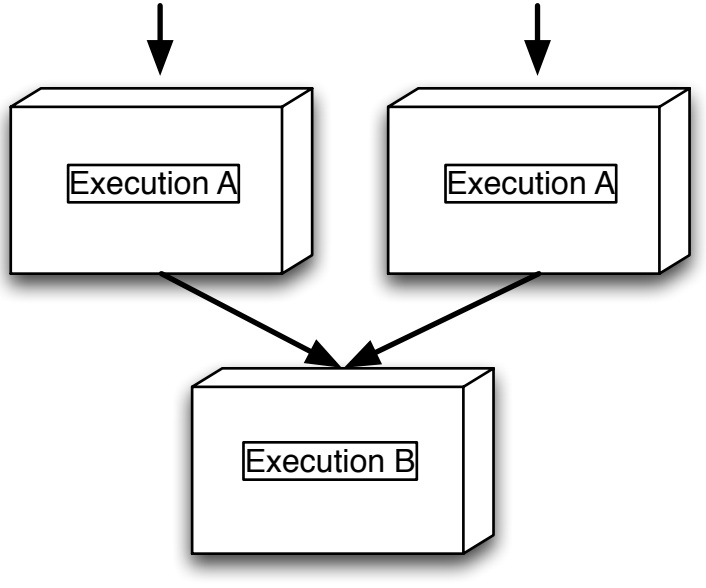
- Adoption of MR paradigm to data stream processing
- Built on top of coroutines (Producer/Consumer)
- Goes further than the original MR concepts
- Easy for anyone familiar with Hadoop (or other)
- Open-sourced recently by AVAST
- The core is practically production-ready (RC)
- A lot of work to be done (networking, RPC)

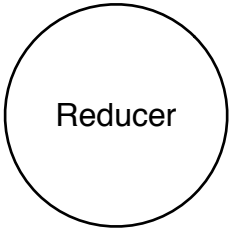




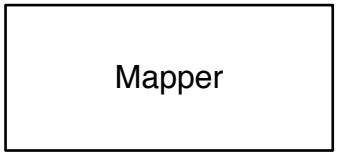


Clockwork Distributed Execution

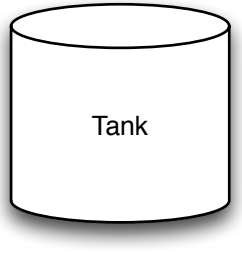




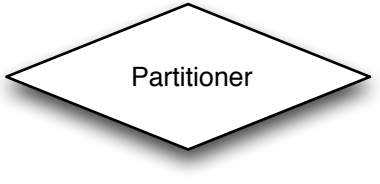
- aggregations
- a coroutine component (loop)



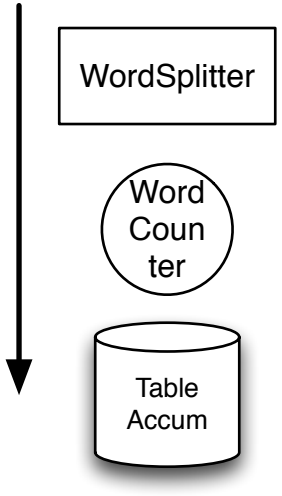
- filtering
- transforming
- expanding



- key-value storage
- key-values storage
- flushing buffer



- routing output to another nodes
- partitioning
- broadcasting



Construction:

```
Execution<Long, String, String, Long> execution = Execution.newBuilder()
        .mapper(new WordSplitter())
        .reducer(new WordCounter())
        .accumulator(new TableAccumulator<String, Long>()).build();
```

Feeding:

```
long counter = 0;
BufferedReader reader =
        new BufferedReader(new FileReader(fileName));
String line;
while ((line = reader.readLine()) != null) {
    execution.emit(counter++, line);
}
```

Note: The program must run with `-javaagent:Continuations.jar`
or transformed by means of the Continuation ANT task. See http://www.matthiasmann.de/content/view/24/26/

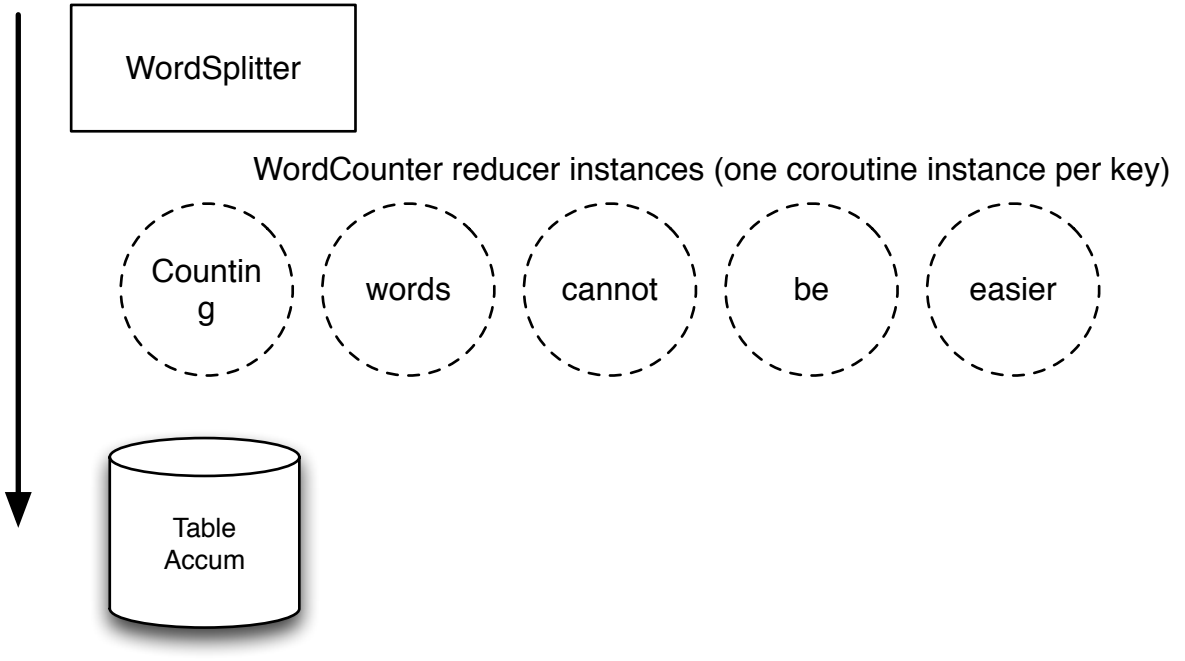
Word Counter Mapper And Reducer

```java
public class WordSplitter extends Mapper<Long, String, String, Long> {
    @Override
    protected void map(Long inputKey, String inputValue, Context context) throws Exception {
        Iterable<String> splits = Splitter.on(" ").trimResults().split(inputValue);
        for (String split : splits) {
            emit(split, 1L);
        }
    }
}
```

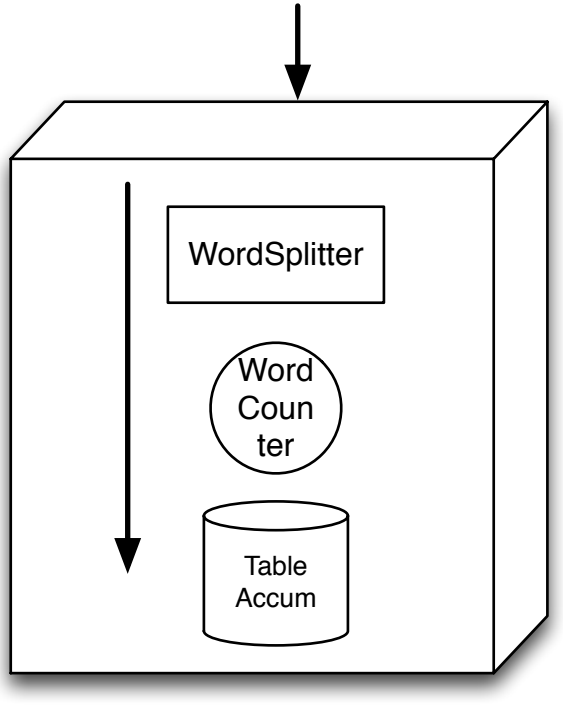
```java
public class WordCounter extends Reducer<String, Long, String, Long> {
    @Override
    protected void reduce(String inputKey, SuspendableIterator<Long> inputValues, Context context)
            throws SuspendExecution, Exception {
        long counter = 0;
        while (inputValues.hasNext()) {
            counter += inputValues.next();
        }
        emit(inputKey, counter);
    }
}
```

Reducer Per-Key Instances

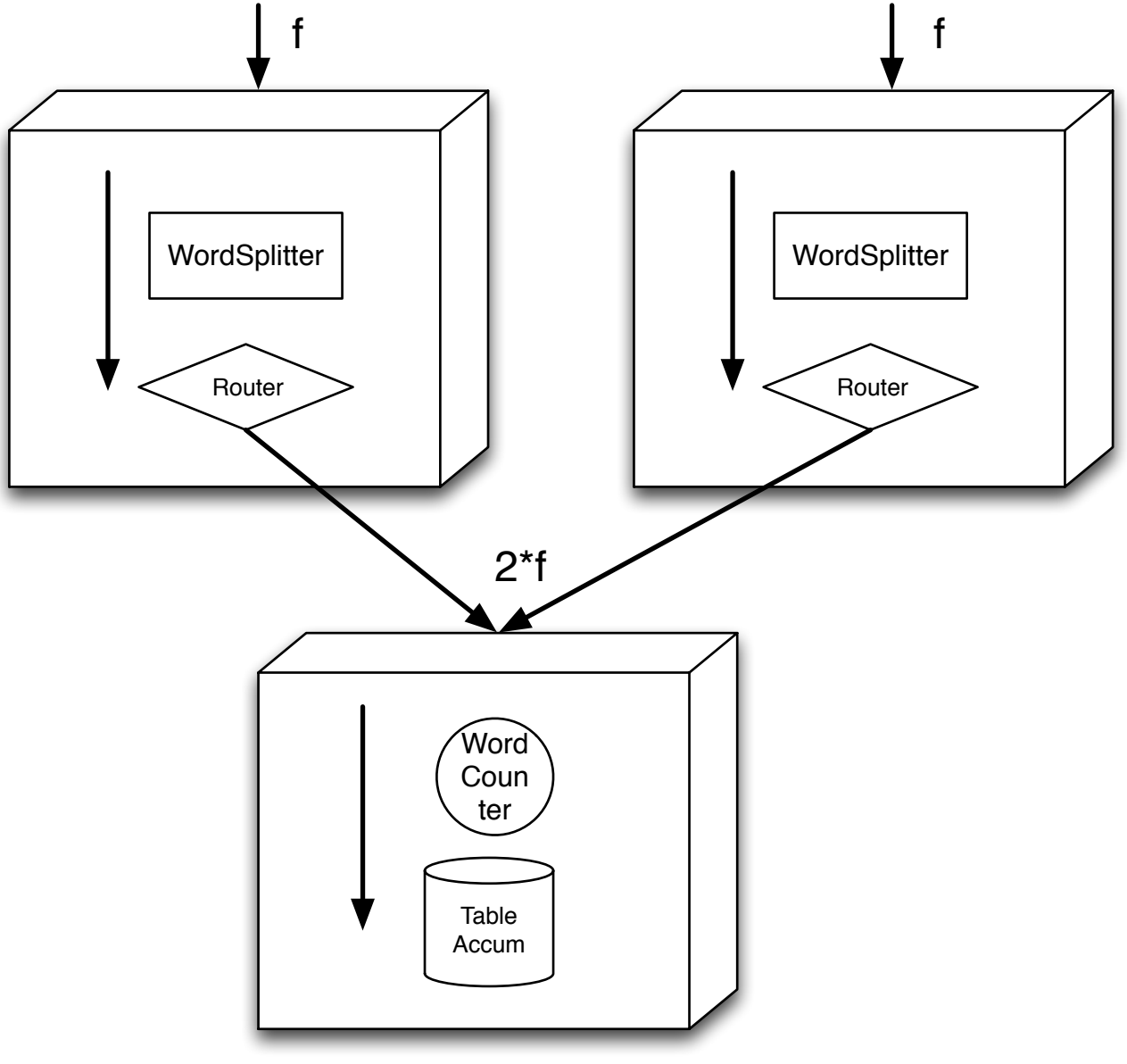
"Counting words cannot be easier"



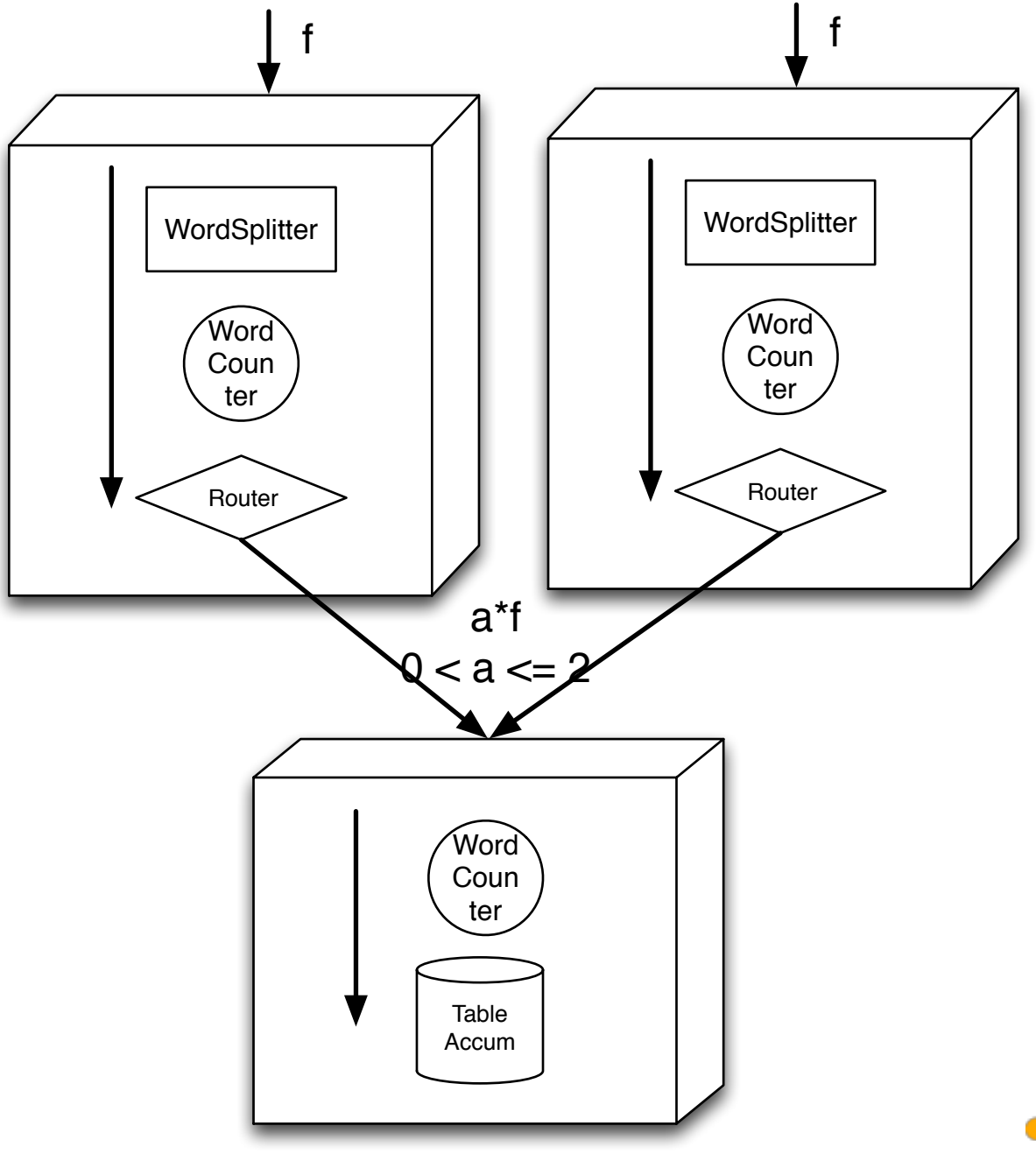
Word Counter on One Node



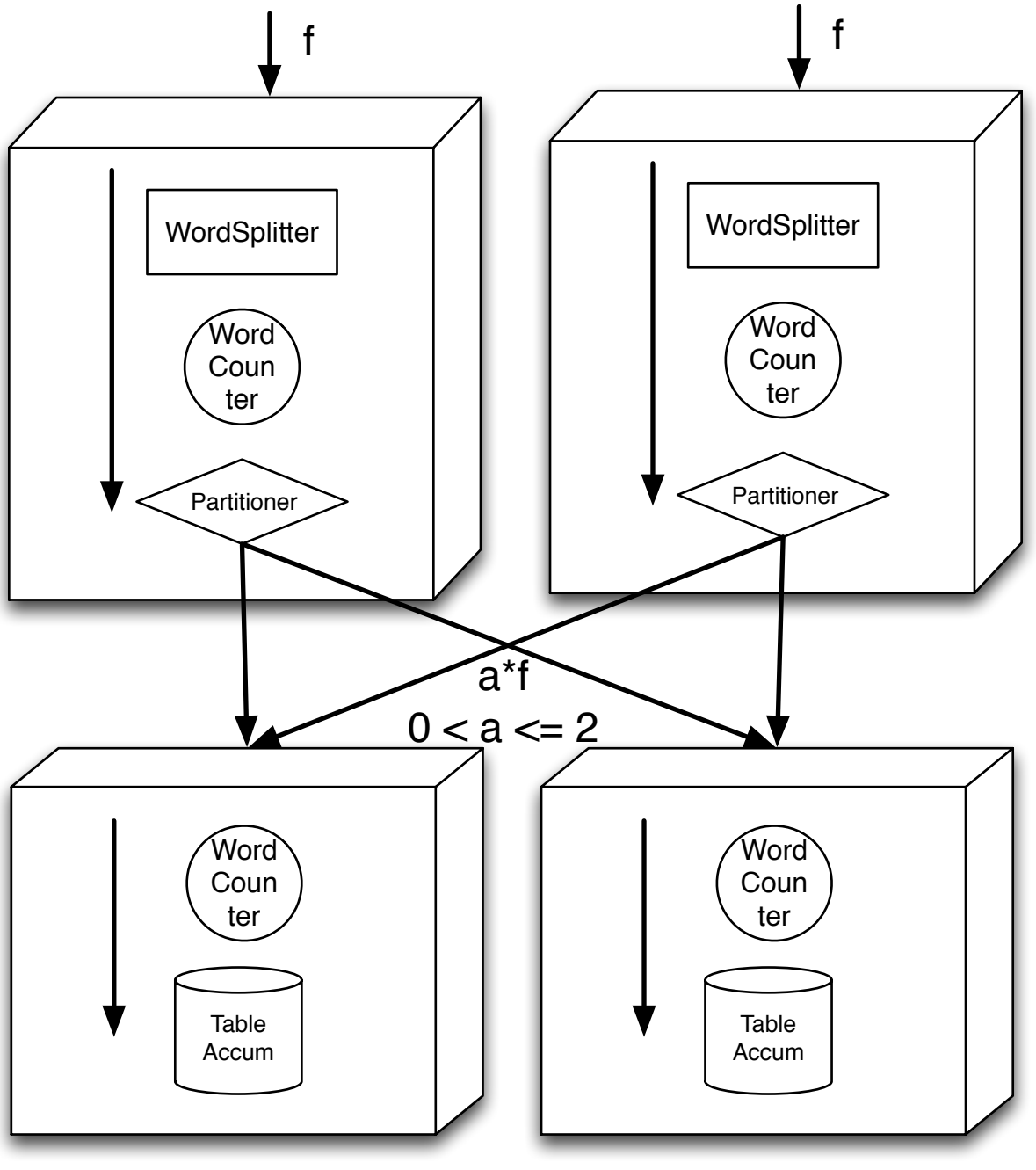
Distributed Word Counter- Many Mappers One Reducer



Distributed Word Counter- Many Mappers with Combiners One Reducer

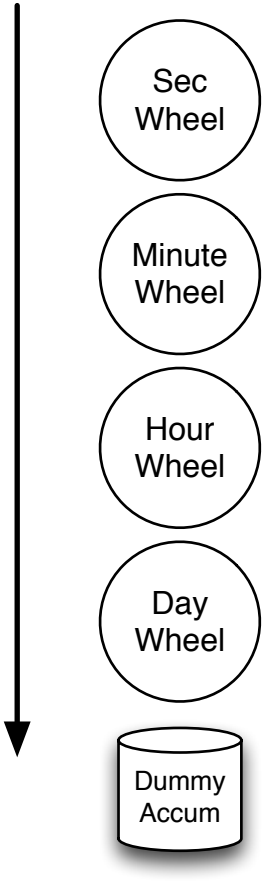


Distributed Word Counter- Many Mappers with Combiners Many Reducers



Reduce-only Setup - The "Nerdiest" clock in the world

1-msec ticks



Construction:

```
Clock clock = new Clock();
Execution<Long, Long, Long, Long> clockWork = Execution.newBuilder()
        .reducer(new Wheel(ClockPart.SEC, clock))
        .reducer(new Wheel(ClockPart.MIN, clock))
        .reducer(new Wheel(ClockPart.HOUR, clock))
        .reducer(new Wheel(ClockPart.DAY, clock))
        .accumulator(new DummyAccumulator<Long, Long>())
        .build();
```

Feeding:

```
for (; ; ) {
    clockWork.emit(0L, 0L);
    Thread.sleep(1000);
    System.out.println(clock);
}
```

Reduce-only Setup - Reducer as a Clock Wheel

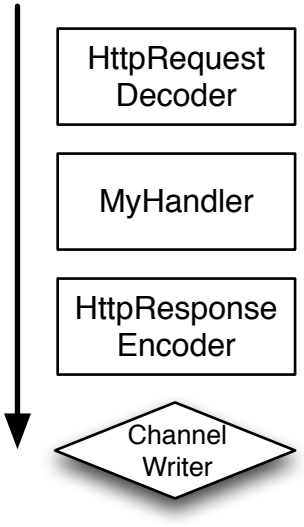
```
public static class Wheel extends Reducer<Long, Long, Long, Long> {

    final ClockPart clockPart;
    final Clock clock;

    public Wheel(ClockPart clockPart, Clock clock) {
        this.clockPart = clockPart;
        this.clock = clock;
    }

    @Override
    protected void reduce(Long inputKey, SuspendableIterator<Long> inputValues, Context context)
            throws SuspendExecution, Exception {
        long counter = 0;
        long timeHand = 0;
        while (inputValues.hasNext()) {
            inputValues.next();
            counter++;
            if (counter % clockPart.period == 0) {
                clockPart.inc(clock);
                emit(0L, timeHand++);
            }
        }
    }
}
```

Map-only Setup - HTTP pipeline



Construction:

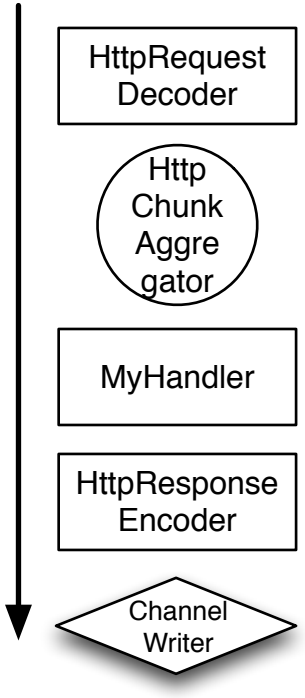
```
Execution<Long, ByteBuffer, Long, ByteBuffer> pipeline =
        Execution.newBuilder()
                .mapper(new HttpRequestDecoder())
                .mapper(new MyHandler()) // custom business logic
                .mapper(new HttpResponseEncoder())
                .accumulator(new ChannelWriter<Long>())
                .build();
```

Feeding:

```
WritableChannelContext channelContext = createContext(socket);
pipeline.emit(rqCnt, inputBuffer, channelContext);
```

avast! be free

Many-Maps-One-Reduce Setup - HTTP pipeline



Construction:

```
int maxContentLength = 10000;
Execution<Long,ByteBuffer, Long, ByteBuffer> pipeline =
        Execution.newBuilder()
        .mapper(new HttpRequestDecoder())
        .reducer(new HttpChunkAggregator(maxContentLength))
        .mapper(new MyHandler())
        .mapper(new HttpResponseEncoder())
        .accumulator(new ChannelWriter<Long>())
        .build();
```

```
int maxContentLength = 10000;
Execution<Long, ByteBuffer, Long, ByteBuffer> pipeline =
        Execution.newBuilder()
                .mapper(new HttpMessageDecoder())
                .reducer(new HttpChunkAggregator(maxContentLength))
                .mapper(new MyHandler())
                .mapper(new HttpResponseEncoder())
                .accumulator(new ChannelWriter<Long>())
                .build();
```

Feeding:

```
WritableChannelContext channelContext = createContext(socket);
pipeline.emit(rqCnt, inputBuffer, channelContext);
```

Naive Bayes Classification Map Reduce Job (http://nickjenkin.com/blog/?p=85)

```
def map(key, instance):
    i = 0
    for attribute in instance.attributes:
        collect(instance.class + "_" + i, attribute)
        i++

    collect("target_" + instance.class, 1) # class distribution

def reduce(key, values):
    if key.startsWith("target_"): # reduce class dist keys
        sum = 0
        for v in values:
            sum += v
                collect(key,sum)

    else: # reduce attribute/class keys
        sum=0
        sumSq = 0
        count = 0
        for v in values:
            sum += v
            sumSq += v*v
            count++

        mean = sum/count
        collect(key + "_mean", mean)
        collect(key + "_stddev", sqrt(abs(sumSq - mean * sum) / count))
```

Naive Bayes Classification - Mapper

```
public class InstanceMapper extends Mapper<Long, String, String, Double> {

    private final Splitter splitter = Splitter.on(CharMatcher.WHITESPACE).omitEmptyStrings().trimResults();

    @Override
    protected void map(Long rowNumber, String instanceRow, Context context) throws Exception {
        int attrCnt = 0;
        String target = null;
        for (String cell: splitter.split(instanceRow)) {
            if (target == null) {
                target = cell;
                emit("target_" + target, 1d);
            } else {
                double attr = Double.parseDouble(cell);
                emit(target + "_" + attrCnt, attr);
                attrCnt++;
            }
        }
    }
}
```

Naive Bayes Classification - Reducer

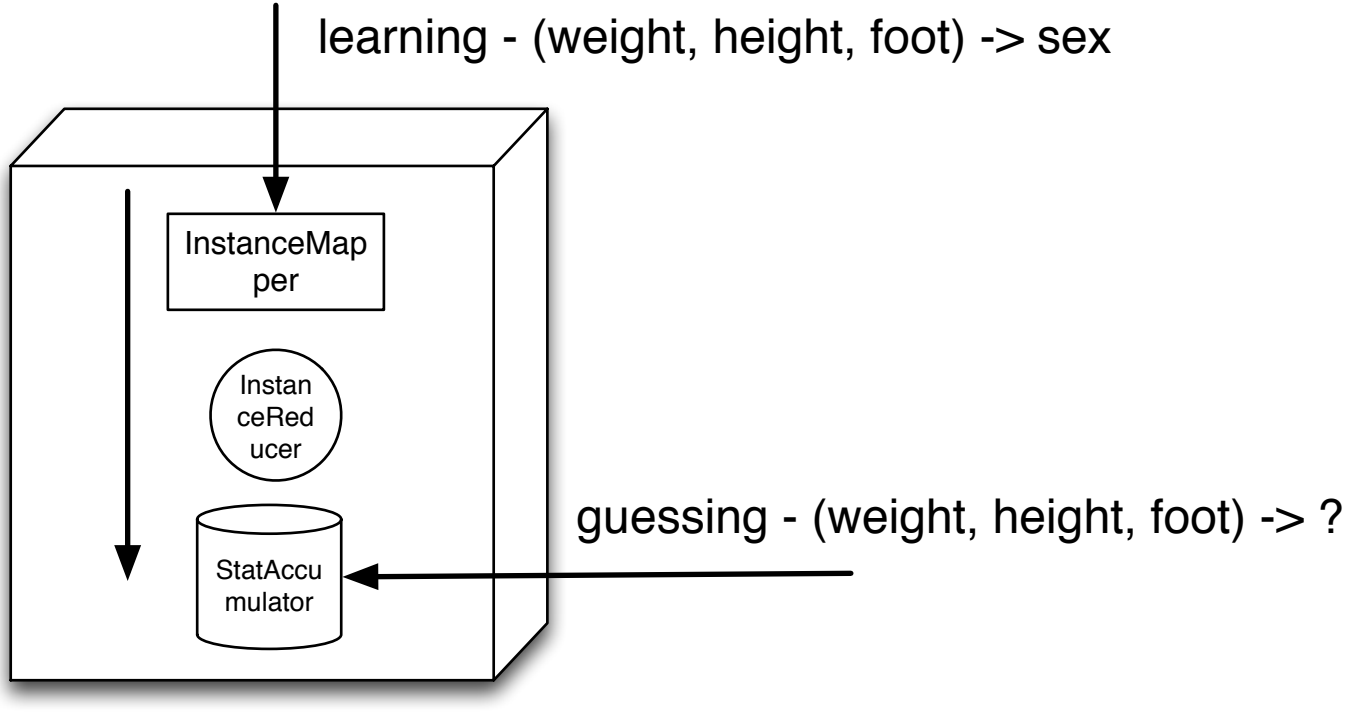
```
public class InstanceReducer extends Reducer<String, Double, String, BaseStat> {

    @Override
    protected void reduce(String inputKey, SuspendableIterator<Double> inputValues, Context context)
            throws SuspendExecution, Exception {
        if (inputKey.startsWith("target_")) {
            int targetTotal = 0;
            while (inputValues.hasNext()) {
                Double value = inputValues.next();
                int targetCounter = value.intValue();
                targetTotal += targetCounter;
            }
            emit(inputKey, new BaseStat(targetTotal));

        } else {
            float sum = 0;
            float sumSq = 0;
            int count = 0;
            while (inputValues.hasNext()) {
                double attr = inputValues.next();
                sum += attr;
                sumSq += attr * attr;
                count++;
            }

            emit(inputKey, new AttrStat(sum, sumSq, count));

        }
    }
}
```



Conclusion

- Distributed stream processing easier
- Some techniques known from offline MR can be adopted more or less directly
- Requires incremental algorithms and models
- Many deployment options, flushing strategies
- A lot of work to be done: communication protocol, machine learning and statistics algorithms, management tools, documentation ...

Thanks for your attention!

Q&A

Abstract

This presentation deals with the concept of coroutines and its applicability in the world of stream data processing. Although it is rarely used in the todays applications, the coroutines have been here since the early days of digital computing. Surprisingly, coroutines can be nicely combined with the map-reduce paradigm that is used frequently in the world of cloud computing and big data processing. In contrast to the traditional map-reduce concept, which is designed for offline job processing, the coroutines&map-reduce hybrid is primarily targeted at real-time event processing. Clockwork, an open-source library developed at Avast, combines these two concepts and allows a programmer to write a real-time stream analysis as if he wrote a traditional map-reduce job for Hadoop, for instance. The presentation is focused mainly on coding and samples and will show how to program applications ranging from simple real-time statistics to more advanced tasks.

1. http://www.matthiasmann.de/content/view/24/26/
2. http://ssw.jku.at/General/Staff/LS/coro/CoroIntroduction.pdf
3. http://en.wikipedia.org/wiki/Coroutine
4. http://nickjenkin.com/blog/?p=85
5. http://code.google.com/p/moa/
6. http://www2.research.att.com/~marioh/papers/vldb08-2.pdf
7. http://www.slideshare.net/mgrcar/text-and-text-stream-mining-tutorial-15137759