

Java Parallel Streams Internals: Combining Results (Part 2)

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

**Institute for Software
Integrated Systems**

**Vanderbilt University
Nashville, Tennessee, USA**



Learning Objectives in this Part of the Lesson

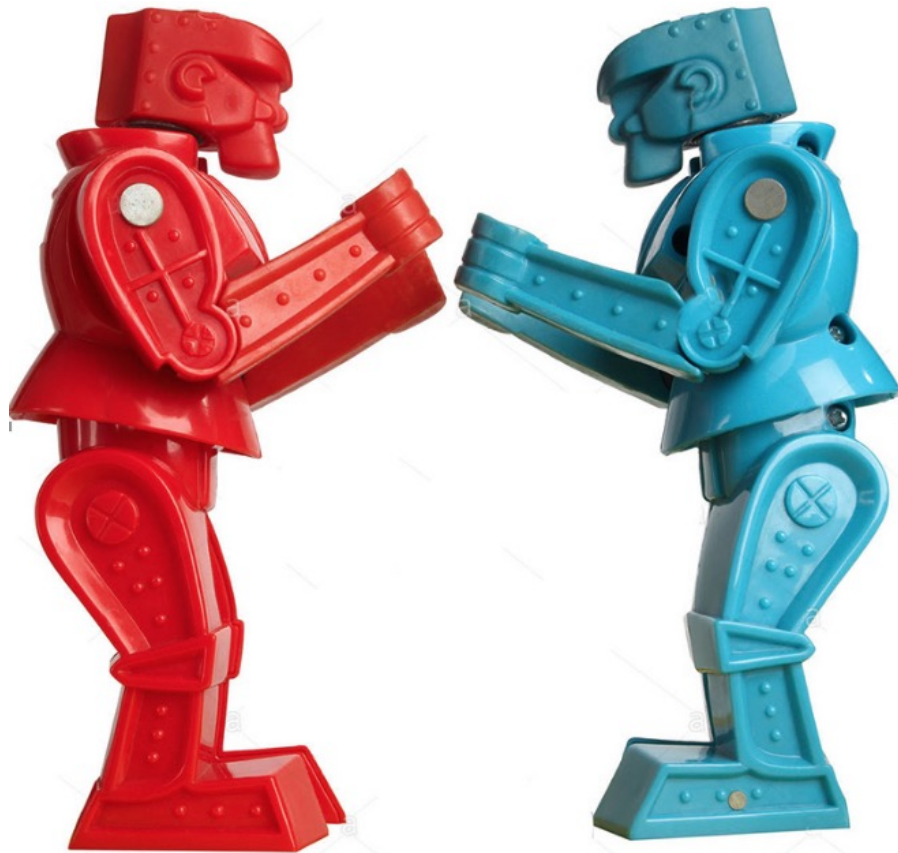
- Understand parallel stream internals, e.g.
 - Know what can change & what can't
 - Partition a data source into "chunks"
 - Process chunks in parallel via the common fork-join pool
 - Configure the Java parallel stream common fork-join pool
- Perform a reduction to combine partial results into a single result
 - Be aware of common traps & pitfalls with parallel streams & `reduce()`



Differences for collect() & reduce() in a Parallel Stream

Differences for collect() & reduce() in a Parallel Stream

- It's important to understand the semantic differences between collect() & reduce()



Differences for collect() & reduce() in a Parallel Stream

- It's important to understand the semantic differences between collect() & reduce(), e.g.
- Always test w/a parallel stream to detect mistakes wrt mutable vs. immutable reductions

```
void buggyStreamReduce3a
    (List<String> allStrings,
     boolean parallel) {
    ...
    Stream<String> stringStream =
        allStrings.stream();

    if (parallel)
        stringStream.parallel();

    String words = stringStream
        .reduce(new StringBuilder(),
                StringBuilder::append,
                StringBuilder::append)
        .toString();
}
```

Differences for collect() & reduce() in a Parallel Stream

- It's important to understand the semantic differences between collect() & reduce(), e.g.
- Always test w/a parallel stream to detect mistakes wrt mutable vs. immutable reductions

Convert a list of words into a stream of words

```
void buggyStreamReduce3a
    (List<String> allStrings,
     boolean parallel) {
    ...
    Stream<String> stringStream =
        allStrings.stream();

    if (parallel)
        stringStream.parallel();

    String words = stringStream
        .reduce(new StringBuilder(),
               StringBuilder::append,
               StringBuilder::append)
        .toString();
}
```

Naturally, this call doesn't really do any work since streams are "lazy"

Differences for collect() & reduce() in a Parallel Stream

- It's important to understand the semantic differences between collect() & reduce(), e.g.
- Always test w/a parallel stream to detect mistakes wrt mutable vs. immutable reductions

```
void buggyStreamReduce3a
    (List<String> allStrings,
     boolean parallel) {
    ...
    Stream<String> stringStream =
        allStrings.stream();

    if (parallel)
        stringStream.parallel();

    String words = stringStream
        .reduce(new StringBuilder(),
                StringBuilder::append,
                StringBuilder::append)
        .toString();
}
```

A stream can be dynamically switched to "parallel" mode!

Differences for collect() & reduce() in a Parallel Stream

- It's important to understand the semantic differences between collect() & reduce(), e.g.
- Always test w/a parallel stream to detect mistakes wrt mutable vs. immutable reductions

```
void buggyStreamReduce3a
    (List<String> allStrings,
     boolean parallel) {
    ...
    Stream<String> stringStream =
        allStrings.stream();

    if (parallel)
        stringStream.parallel();

    String words = stringStream
        .reduce(new StringBuilder(),
                StringBuilder::append,
                StringBuilder::append)
        .toString();
}
```

The "last" call to .parallel() or .sequential() in a stream "wins"

Differences for collect() & reduce() in a Parallel Stream

- It's important to understand the semantic differences between collect() & reduce(), e.g.
- Always test w/a parallel stream to detect mistakes wrt mutable vs. immutable reductions

```
void buggyStreamReduce3a
    (List<String> allStrings,
     boolean parallel) {
    ...
    Stream<String> stringStream =
        allStrings.stream();

    if (parallel)
        stringStream.parallel();

    String words = stringStream
        .reduce(new StringBuilder(),
               StringBuilder::append,
               StringBuilder::append)
        .toString();
}
```



This code works when parallel is false since the StringBuilder is only called in a single thread

Differences for collect() & reduce() in a Parallel Stream

- It's important to understand the semantic differences between collect() & reduce(), e.g.
- Always test w/a parallel stream to detect mistakes wrt mutable vs. immutable reductions

```
void buggyStreamReduce3a
    (List<String> allStrings,
     boolean parallel) {
    ...
    Stream<String> stringStream =
        allStrings.stream();

    if (parallel)
        stringStream.parallel();

    String words = stringStream
        .reduce(new StringBuilder(),
               StringBuilder::append,
               StringBuilder::append)
        .toString();
}
```



This code fails when parallel is true since reduce() expects to do an "immutable" reduction

Differences for collect() & reduce() in a Parallel Stream

- It's important to understand the semantic differences between collect() & reduce(), e.g.
- Always test w/a parallel stream to detect mistakes wrt mutable vs. immutable reductions



There's a race condition here since `StringBuilder` is not thread-safe..

```
void buggyStreamReduce3a
    (List<String> allStrings,
     boolean parallel) {
    ...
    Stream<String> stringStream =
        allStrings.stream();

    if (parallel)
        stringStream.parallel();

    String words = stringStream
        .reduce(new StringBuilder(),
               StringBuilder::append,
               StringBuilder::append)
        .toString();
}
```

Differences for collect() & reduce() in a Parallel Stream

- It's important to understand the semantic differences between collect() & reduce(), e.g.
- Always test w/a parallel stream to detect mistakes wrt mutable vs. immutable reductions
 - One solution use reduce() with string concatenation

```
void streamReduceConcat
    (List<String> allStrings,
     boolean parallel) {
    ...
    Stream<String> stringStream =
        allStrings.stream();

    if (parallel)
        stringStream.parallel();

    String words = stringStream
        .reduce(new String(),
                (x, y) -> x + y);
```

Differences for collect() & reduce() in a Parallel Stream

- It's important to understand the semantic differences between collect() & reduce(), e.g.
- Always test w/a parallel stream to detect mistakes wrt mutable vs. immutable reductions
 - One solution use reduce() with string concatenation

```
void streamReduceConcat
    (List<String> allStrings,
     boolean parallel) {
    ...
    Stream<String> stringStream =
        allStrings.stream();

    if (parallel)
        stringStream.parallel();

    String words = stringStream
        .reduce(new String(),
                (x, y) -> x + y);
```

This simple fix is inefficient due to string concatenation overhead

Differences for collect() & reduce() in a Parallel Stream

- It's important to understand the semantic differences between collect() & reduce(), e.g.
- Always test w/a parallel stream to detect mistakes wrt mutable vs. immutable reductions
 - One solution use reduce() with string concatenation
 - Another solution uses collect() with the joining collector

```
void streamCollectJoining
    (List<String> allStrings,
     boolean parallel) {
    ...
    Stream<String> stringStream =
        allStrings.stream();

    if (parallel)
        stringStream.parallel();

    String words = stringStream
        .collect(joining());
}
```

Differences for collect() & reduce() in a Parallel Stream

- It's important to understand the semantic differences between collect() & reduce(), e.g.
- Always test w/a parallel stream to detect mistakes wrt mutable vs. immutable reductions
 - One solution use reduce() with string concatenation
 - Another solution uses collect() with the joining collector

```
void streamCollectJoining
    (List<String> allStrings,
     boolean parallel) {
    ...
    Stream<String> stringStream =
        allStrings.stream();

    if (parallel)
        stringStream.parallel();

    String words = stringStream
        .collect(joining());
}
```

This is a much better solution!!



Differences for collect() & reduce() in a Parallel Stream

- Also beware of issues related to associativity & identity with reduce()

```
void testDifferenceReduce(...) {  
    long difference = LongStream  
        .rangeClosed(1, 100)  
        .parallel()  
        .reduce(0L,  
            (x, y) -> x - y);  
}
```

```
void testSum(long identity, ...) {  
    long sum = LongStream  
        .rangeClosed(1, 100)  
        .reduce(identity,  
            // Could use (x, y) -> x + y  
            Math::addExact);  
}
```


Differences for collect() & reduce() in a Parallel Stream

- Also beware of issues related to associativity & identity with reduce()

```
void testDifferenceReduce(...) {  
    long difference = LongStream  
        .rangeClosed(1, 100)  
        .parallel()  
        .reduce(0L,  
            (x, y) -> x - y);  
}
```

```
void testSum(long identity, ...) {  
    long sum = LongStream  
        .rangeClosed(1, 100)  
        .reduce(identity,  
            // Could use (x, y) -> x + y  
            Math::addExact);  
}
```

This code fails for a parallel stream since subtraction is not associative

Differences for collect() & reduce() in a Parallel Stream

- Also beware of issues related to associativity & identity with reduce()

```
void testDifferenceReduce(...) {  
    long difference = LongStream  
        .rangeClosed(1, 100)  
        .parallel()  
        .reduce(0L,  
            (x, y) -> x - y);  
}
```

```
void testSum(long identity, ...) {  
    long sum = LongStream  
        .rangeClosed(1, 100)  
        .reduce(identity,  
            // Could use (x, y) -> x + y  
            Math::addExact);  
}
```

This code fails if identity is not 0L

The "identity" of an OP is defined as "identity OP value == value" (& inverse)

Differences for collect() & reduce() in a Parallel Stream

- Also beware of issues related to associativity & identity with reduce()

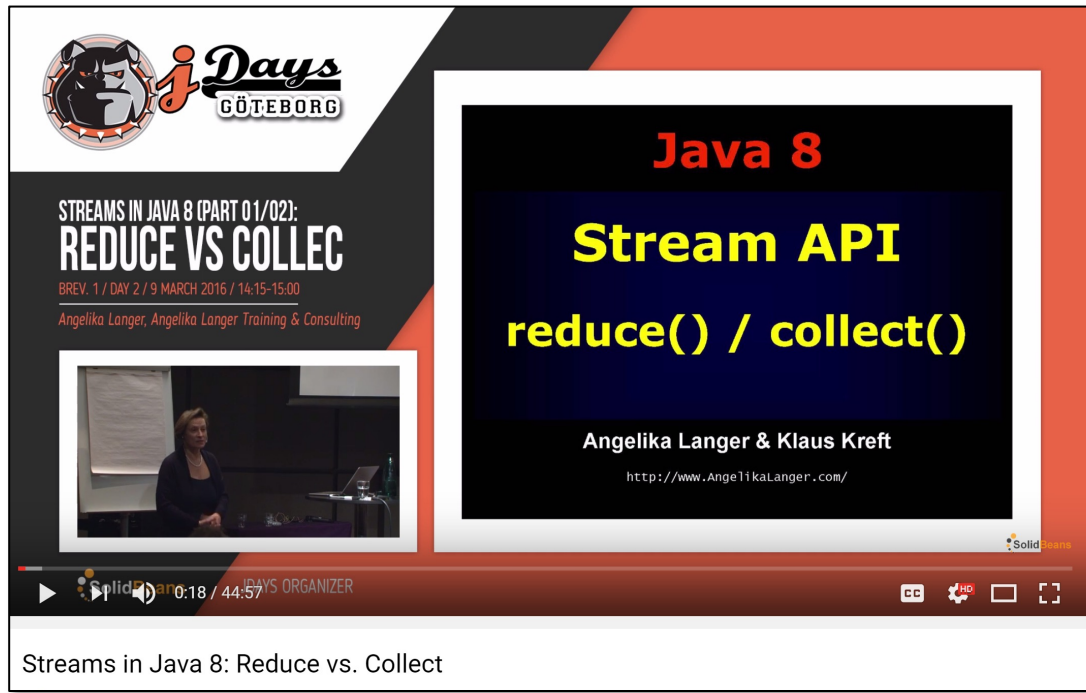
```
void testDifferenceReduce(...) {  
    long difference = LongStream  
        .rangeClosed(1, 100)  
        .parallel()  
        .reduce(0L,  
            (x, y) -> x - y);  
}
```

```
void testProd(long identity, ...) {  
    long sum = LongStream  
        .rangeClosed(1, 100)  
        .reduce(identity,  
            (x, y) -> x * y);  
}
```

This code fails if identity is not 1L

Differences for collect() & reduce() in a Parallel Stream

- More good discussions about reduce() vs. collect() appear online



The image shows a video player interface. The main content is a presentation slide with a dark blue background and yellow and red text. The slide title is "Java 8 Stream API reduce() / collect()". Below the title, it says "Angelika Langer & Klaus Kreft" and provides a URL: "http://www.AngelikaLanger.com/". The video player's control bar at the bottom shows a play button, a progress bar at 0:18 / 44:57, and a "SLIDESHOW ORGANIZER" button. The video title below the player is "Streams in Java 8: Reduce vs. Collect".

Java 8
Stream API
reduce() / collect()

Angelika Langer & Klaus Kreft
<http://www.AngelikaLanger.com/>

Streams in Java 8: Reduce vs. Collect

See www.youtube.com/watch?v=oWIWEK5Aw

End of Java Parallel Streams Internals: Combining Results (Part 2)