

Java Parallel Streams Internals: Demo'ing Collector Performance

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
 - Recognize key behaviors & differences of non-concurrent & concurrent collectors
 - Learn how to implement non-concurrent & concurrent collectors
 - Be aware of performance variance in concurrent & non-concurrent collectors

```
Starting collector tests for 1000 words..printing results
21 msec: sequential timeStreamCollectToSet()
30 msec: parallel timeStreamCollectToSet()
39 msec: sequential timeStreamCollectToConcurrentSet()
59 msec: parallel timeStreamCollectToConcurrentSet()

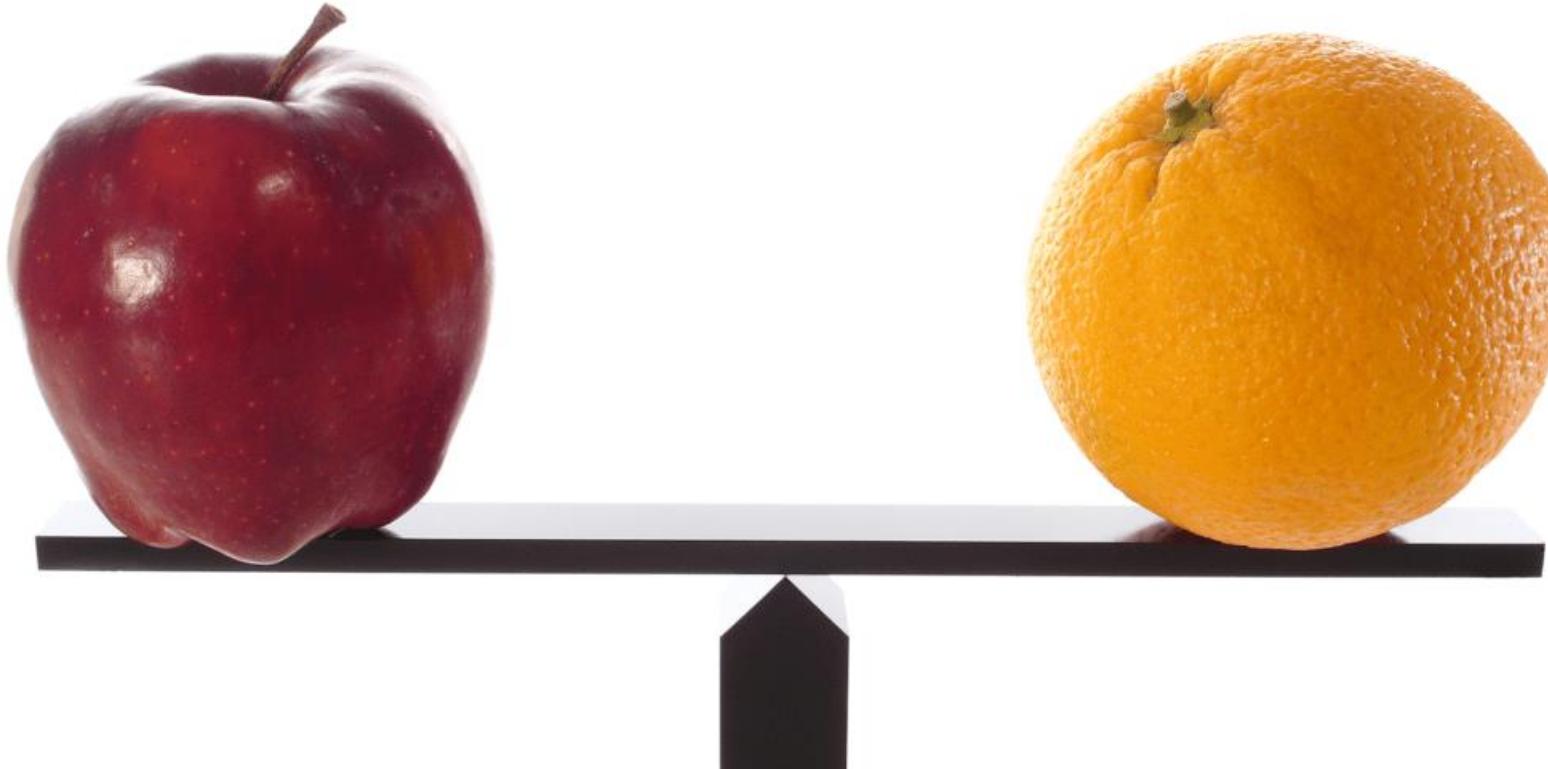
...
Starting collector tests for 100000 words..printing results
219 msec: parallel timeStreamCollectToConcurrentSet()
364 msec: parallel timeStreamCollectToSet()
657 msec: sequential timeStreamCollectToSet()
804 msec: sequential timeStreamCollectToConcurrentSet()

Starting collector tests for 883311 words..printing results
1782 msec: parallel timeStreamCollectToConcurrentSet()
3010 msec: parallel timeStreamCollectToSet()
6169 msec: sequential timeStreamCollectToSet()
7652 msec: sequential timeStreamCollectToConcurrentSet()
```

Demonstrating Collector Performance

Demonstrating Collector Performance

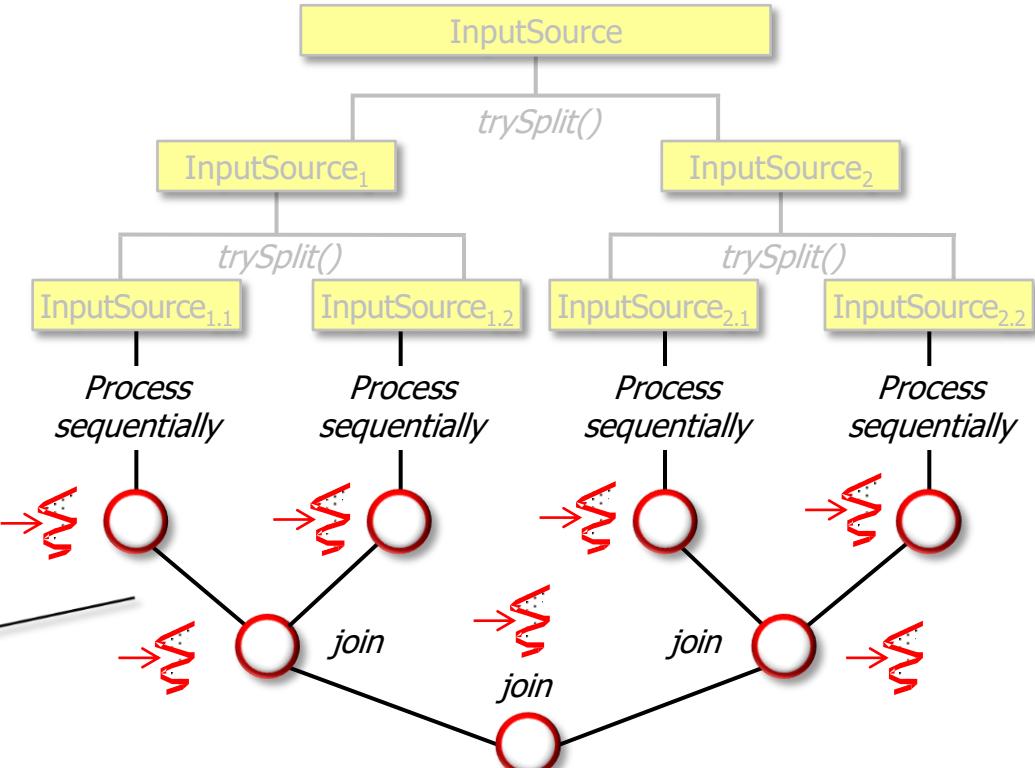
- Concurrent & non-concurrent collectors perform differently when used in parallel & sequential streams on different input sizes



See prior lessons on "Java Parallel Streams Internals: Non-Concurrent and Concurrent Collectors"

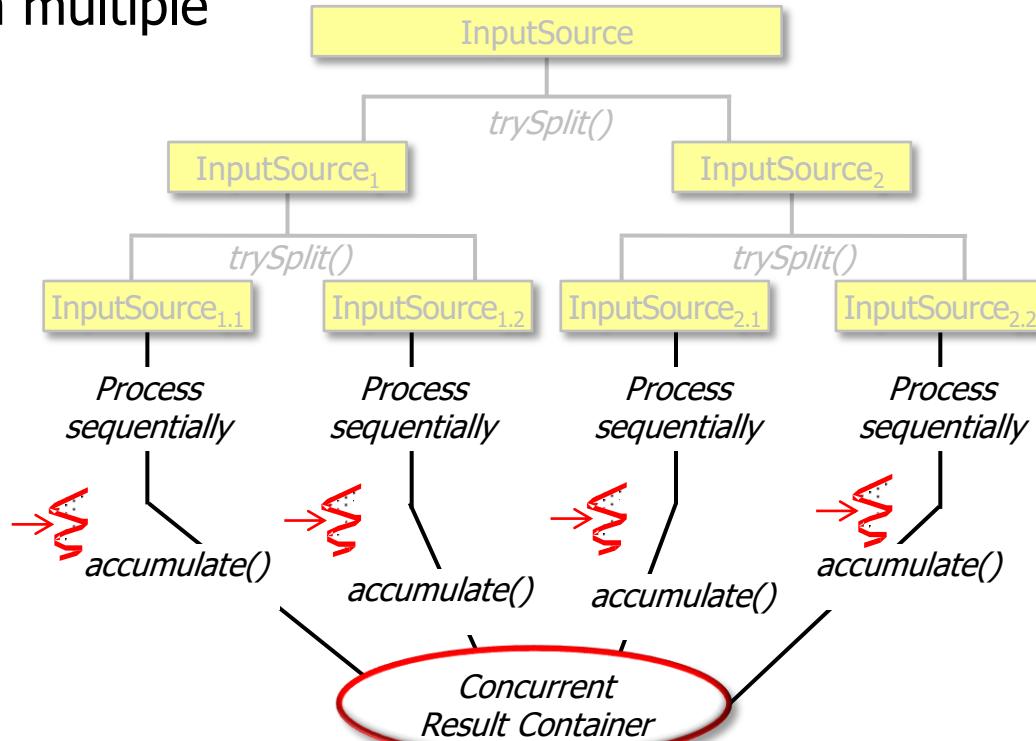
Demonstrating Collector Performance

- A non-concurrent collector operates by merging sub-results



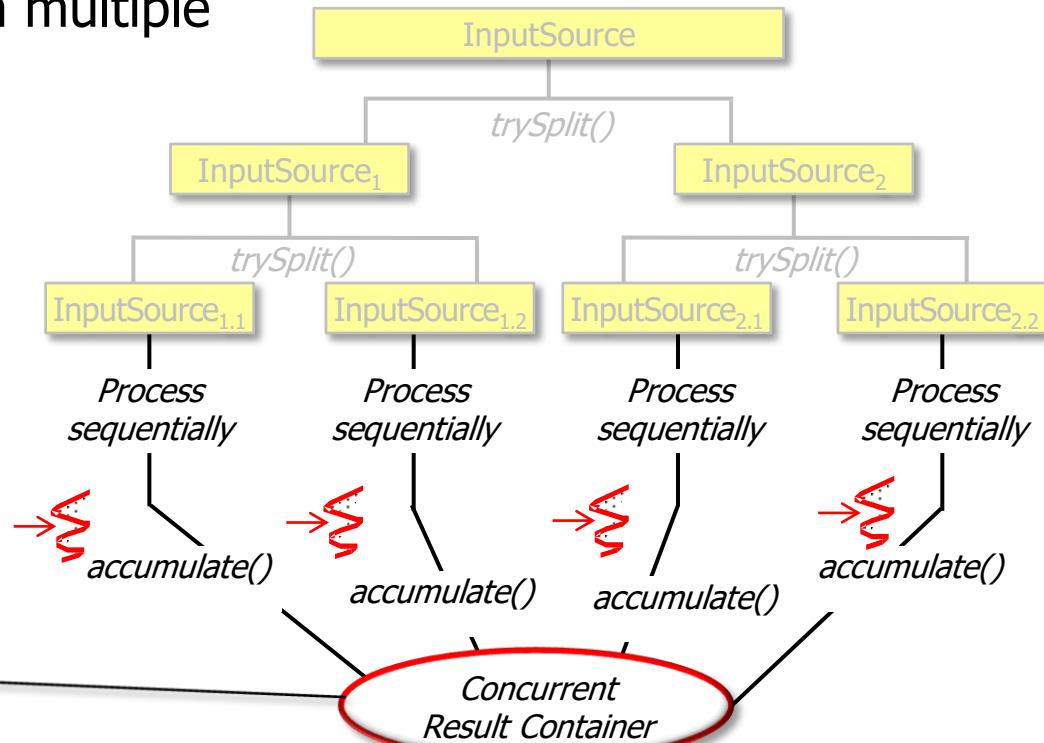
Demonstrating Collector Performance

- A concurrent collector creates one concurrent mutable result container & accumulates elements into it from multiple threads in a parallel stream



Demonstrating Collector Performance

- A concurrent collector creates one concurrent mutable result container & accumulates elements into it from multiple threads in a parallel stream



Demonstrating Collector Performance

- Results show collector differences become more significant as input grows

Starting collector tests for 1000 words..printing results

```
21 msec: sequential timeStreamCollectToSet()  
30 msec: parallel timeStreamCollectToSet()  
39 msec: sequential timeStreamCollectToConcurrentSet()  
59 msec: parallel timeStreamCollectToConcurrentSet()  
...
```

Starting collector tests for 100000 words....printing results

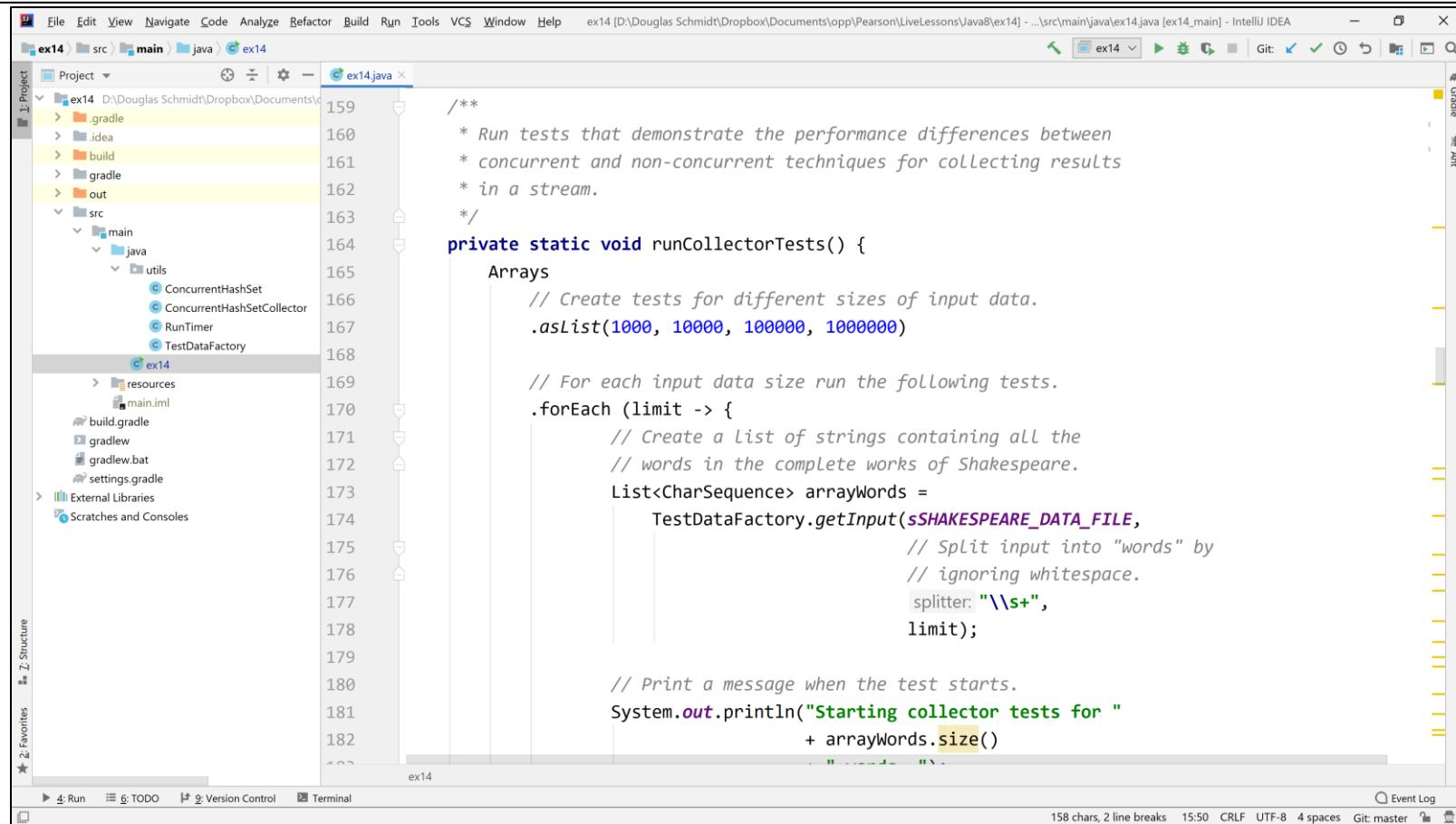
```
219 msec: parallel timeStreamCollectToConcurrentSet()  
364 msec: parallel timeStreamCollectToSet()  
657 msec: sequential timeStreamCollectToSet()  
804 msec: sequential timeStreamCollectToConcurrentSet()
```

Starting collector tests for 883311 words....printing results

```
1782 msec: parallel timeStreamCollectToConcurrentSet()  
3010 msec: parallel timeStreamCollectToSet()  
6169 msec: sequential timeStreamCollectToSet()  
7652 msec: sequential timeStreamCollectToConcurrentSet()
```

See upcoming lessons on “*When [Not] to Use Parallel Streams*”

Demonstrating Collector Performance



The screenshot shows the IntelliJ IDEA IDE interface with the following details:

- File Bar:** File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help.
- Project Bar:** ex14 > src > main > java > ex14.
- Toolbars:** Standard, Version Control, Terminal, Event Log.
- Left Sidebar:** Project (ex14), Favorites, Structure.
- Right Sidebar:** Gradle, Ant.
- Code Editor:** The file ex14.java contains Java code for demonstrating collector performance. The code includes comments explaining the purpose of the tests and the use of arrays and parallel streams to collect results.

```
159     /**
160      * Run tests that demonstrate the performance differences between
161      * concurrent and non-concurrent techniques for collecting results
162      * in a stream.
163     */
164     private static void runCollectorTests() {
165         Arrays
166             // Create tests for different sizes of input data.
167             .asList(1000, 10000, 100000, 1000000)
168
169             // For each input data size run the following tests.
170             .forEach (limit -> {
171                 // Create a list of strings containing all the
172                 // words in the complete works of Shakespeare.
173                 List<CharSequence> arrayWords =
174                     TestDataFactory.getInput(sSHAKESPEARE_DATA_FILE,
175                         // Split input into "words" by
176                         // ignoring whitespace.
177                         splitter: "\\\s+",
178                         limit);
179
180                 // Print a message when the test starts.
181                 System.out.println("Starting collector tests for "
182                     + arrayWords.size())
183             })
184     }
185 }
```

See github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex14

End of Java Parallel Streams Internals: Demo'ing Collector Performance