Java StampedLock: Usage Considerations

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

• Understand the structure, functionality of the Java StampedLock class
• Know the key methods in Java StampedLock
• Recognize how to apply Java StampedLock in practice
• Appreciate Java StampedLock usage considerations

We’ll also compare/contrast StampedLock with other Java synchronizers
Java StampedLock
Usage Considerations
Java StampedLock Usage Conventions

- StampedLock often *much* faster than ReentrantReadWriteLock

<table>
<thead>
<tr>
<th>SYNCHRONIZED</th>
<th>OPTIMISTIC</th>
<th>RWLOCK</th>
<th>STAMPED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1996.6</td>
<td>1174</td>
<td>116393</td>
<td>64077</td>
</tr>
<tr>
<td>2312.7</td>
<td>1174</td>
<td>116617</td>
<td>47897</td>
</tr>
<tr>
<td>2100.9</td>
<td>1122</td>
<td>117746</td>
<td>65921</td>
</tr>
<tr>
<td>2285.1</td>
<td>1182.9</td>
<td>115605</td>
<td>73500</td>
</tr>
<tr>
<td>2173.6</td>
<td>1184.9</td>
<td>118346</td>
<td>32857</td>
</tr>
<tr>
<td><strong>2173.78</strong></td>
<td><strong>1167.56</strong></td>
<td><strong>116941.4</strong></td>
<td><strong>56850.4</strong></td>
</tr>
</tbody>
</table>

19 readers & 1 writer

Optimistic read mode works very well with little/no contention

Java StampedLock Usage Conventions

- StampedLock often *much* faster than ReentrantReadWriteLock

<table>
<thead>
<tr>
<th>Synchronized</th>
<th>Optimistic</th>
<th>RWLock</th>
<th>Stamped</th>
</tr>
</thead>
<tbody>
<tr>
<td>1996.6</td>
<td>1174</td>
<td>116393</td>
<td>64077</td>
</tr>
<tr>
<td>2312.7</td>
<td>1174</td>
<td>116617</td>
<td>47897</td>
</tr>
<tr>
<td>2100.9</td>
<td>1122</td>
<td>117746</td>
<td>65921</td>
</tr>
<tr>
<td>2285.1</td>
<td>1182.9</td>
<td>115605</td>
<td>73500</td>
</tr>
<tr>
<td>2173.6</td>
<td>1184.9</td>
<td>118346</td>
<td>32857</td>
</tr>
<tr>
<td>2173.78</td>
<td>1167.56</td>
<td><strong>116941.4</strong></td>
<td><strong>56850.4</strong></td>
</tr>
</tbody>
</table>

19 readers & 1 writer

-ReentrantReadWriteLock is very slow..

**Java StampedLock Usage Conventions**

- StampedLock often *much* faster than ReentrantReadWriteLock

<table>
<thead>
<tr>
<th>SYNCHRONIZED</th>
<th>OPTIMISTIC</th>
<th>RWLOCK</th>
<th>STAMPED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1996.6</td>
<td>1174</td>
<td>116393</td>
<td>64077</td>
</tr>
<tr>
<td>2312.7</td>
<td>1174</td>
<td>116617</td>
<td>47897</td>
</tr>
<tr>
<td>2100.9</td>
<td>1122</td>
<td>117746</td>
<td>65921</td>
</tr>
<tr>
<td>2285.1</td>
<td>1182.9</td>
<td>115605</td>
<td>73500</td>
</tr>
<tr>
<td>2173.6</td>
<td>1184.9</td>
<td>118346</td>
<td>32857</td>
</tr>
<tr>
<td>2173.78</td>
<td>1167.56</td>
<td>116941.4</td>
<td><strong>56850.4</strong></td>
</tr>
</tbody>
</table>

19 readers & 1 writer

*StampedLock with “reading mode” works better than ReentrantReadWriteLock*

Java StampedLock Usage Conventions

- StampedLock often *much* faster than ReentrantReadWriteLock

### Table: Performance Comparison

<table>
<thead>
<tr>
<th></th>
<th>Synchronized</th>
<th>Optimistic</th>
<th>RWLock</th>
<th>Stamped</th>
</tr>
</thead>
<tbody>
<tr>
<td>1996.6</td>
<td>1174</td>
<td>116393</td>
<td>47897</td>
<td>64077</td>
</tr>
<tr>
<td>2312.7</td>
<td>1174</td>
<td>116617</td>
<td>47897</td>
<td>64077</td>
</tr>
<tr>
<td>2100.9</td>
<td>1122</td>
<td>117746</td>
<td>65921</td>
<td>64077</td>
</tr>
<tr>
<td>2285.1</td>
<td>1182.9</td>
<td>115605</td>
<td>73500</td>
<td>64077</td>
</tr>
<tr>
<td>2173.6</td>
<td>1184.9</td>
<td>118346</td>
<td>32857</td>
<td>64077</td>
</tr>
<tr>
<td><strong>2173.78</strong></td>
<td><strong>1167.56</strong></td>
<td><strong>116941.4</strong></td>
<td><strong>56850.4</strong></td>
<td><strong>64077</strong></td>
</tr>
</tbody>
</table>

*Synchronized statements perform quite well*

Java StampedLock Usage Conventions

• StampedLock often *much* faster than ReentrantReadWriteLock

However, your mileage may vary!

See [en.wiktionary.org/wiki/your_mileage_may_vary](en.wiktionary.org/wiki/your_mileage_may_vary)
Java StampedLock Usage Conventions

- StampedLock often *much* faster than ReentrantReadWriteLock

<table>
<thead>
<tr>
<th>RWLOCK</th>
<th>STAMPED</th>
<th>SYNCHRONIZED</th>
<th>OPTIMISTIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1960.8</td>
<td>165.1</td>
<td>177.4</td>
<td>387.9</td>
</tr>
<tr>
<td>1473.6</td>
<td>111.3</td>
<td>192.1</td>
<td>382.8</td>
</tr>
<tr>
<td>2119.7</td>
<td>216.8</td>
<td>173.3</td>
<td>403.6</td>
</tr>
<tr>
<td>2772.2</td>
<td>221.9</td>
<td>205.4</td>
<td>403.9</td>
</tr>
<tr>
<td>2721.4</td>
<td>189.3</td>
<td>181.2</td>
<td>394.2</td>
</tr>
<tr>
<td>2209.54</td>
<td>180.88</td>
<td>185.88</td>
<td><strong>394.48</strong></td>
</tr>
</tbody>
</table>

10 readers & 10 writers

*Optimistic read mode works less well with more contention*

Java StampedLock Usage Conventions

- StampedLock often *much* faster than ReentrantReadWriteLock

```
<table>
<thead>
<tr>
<th>RWLOCK</th>
<th>STAMPED</th>
<th>SYNCHRONIZED</th>
<th>OPTIMISTIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1960.8</td>
<td>165.1</td>
<td>177.4</td>
<td>387.9</td>
</tr>
<tr>
<td>1473.6</td>
<td>111.3</td>
<td>192.1</td>
<td>382.8</td>
</tr>
<tr>
<td>2119.7</td>
<td>216.8</td>
<td>173.3</td>
<td>403.6</td>
</tr>
<tr>
<td>2772.2</td>
<td>221.9</td>
<td>205.4</td>
<td>403.9</td>
</tr>
<tr>
<td>2721.4</td>
<td>189.3</td>
<td>181.2</td>
<td>394.2</td>
</tr>
<tr>
<td><strong>2209.54</strong></td>
<td><strong>180.88</strong></td>
<td><strong>185.88</strong></td>
<td><strong>394.48</strong></td>
</tr>
</tbody>
</table>
```

10 readers & 10 writers

However, ReentrantReadWriteLock is still *much* slower..

Java StampedLock Usage Conventions

- StampedLock often *much* faster than ReentrantReadWriteLock

<table>
<thead>
<tr>
<th>RWLOCK</th>
<th>STAMPED</th>
<th>SYNCHRONIZED</th>
<th>OPTIMISTIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1960.8</td>
<td>165.1</td>
<td>177.4</td>
<td>387.9</td>
</tr>
<tr>
<td>1473.6</td>
<td>111.3</td>
<td>192.1</td>
<td>382.8</td>
</tr>
<tr>
<td>2119.7</td>
<td>216.8</td>
<td>173.3</td>
<td>403.6</td>
</tr>
<tr>
<td>2772.2</td>
<td>221.9</td>
<td>205.4</td>
<td>403.9</td>
</tr>
<tr>
<td>2721.4</td>
<td>189.3</td>
<td>181.2</td>
<td>394.2</td>
</tr>
<tr>
<td>2209.54</td>
<td>180.88</td>
<td>185.88</td>
<td>394.48</td>
</tr>
</tbody>
</table>

10 readers & 10 writers

*StampedLock & synchronized statements both do quite well

Java StampedLock Usage Conventions

- Java StampedLock speedups are only fully realized under certain conditions
Java StampedLock Usage Conventions

- Java StampedLock speedups are only fully realized under certain conditions, e.g.:  
  - Frequency of reads to writes  
    - Ideally, *many* more reads than writes
Java StampedLock Usage Conventions

- Java StampedLock speedups are only fully realized under certain conditions, e.g.
  - Frequency of reads to writes
  - Duration of read & write operations
  - Ideally, read operations should be non-trivial or else locking costs may dominate
Java StampedLock Usage Conventions

- Java StampedLock speedups are only fully realized under certain conditions, e.g.
  - Frequency of reads to writes
  - Duration of read & write operations
  - “Contestation” for the data
    - Ideally, *many* concurrent readers
Java StampedLock Usage Conventions

- Java StampedLock speedups are only fully realized under certain conditions, e.g.
  - Frequency of reads to writes
  - Duration of read & write operations
  - “Contention” for the data
  - Number of processor cores
    - Ideally, *many* cores
Java StampedLock Usage Conventions

- StampedLock can be harder to use than ReentrantReadWriteLock
Java StampedLock Usage Conventions

- StampedLock can be harder to use than ReentrantReadWriteLock
- Many more methods
Java StampedLock Usage Conventions

- StampedLock can be harder to use than ReentrantReadWriteLock
  - Many more methods
  - More intricate semantics & usage patterns

```java
void moveIfAtOrigin(double newX, double newY) {
    long stamp = sl.readLock();
    try {
        while (x == 0.0 && y == 0.0) {
            long ws =
                sl.tryConvertToWriteLock(stamp);
            if (ws != 0L) {
                stamp = ws;
                x = newX; y = newY;
                break;
            } else {
                sl.unlockRead(stamp);
                stamp = sl.writeLock();
            }
        }
    } finally {
        sl.unlock(stamp);
    }
    ...
}
```

See www.techevents.online/using-java-8-lambdas-stampedlock-manage-thread-safety
Java StampedLock Usage Conventions

- StampedLock can be harder to use than ReentrantReadWriteLock
  - Many more methods
  - More intricate semantics & usage patterns
  - Invariants are tricky with optimistic read locks

```java
class Boooom {
    StampedLock mS = new StampedLock();
    int mX = 0;
    int mY = 1;
    ...

    // Thread T1
    while (true) {
        mS.writeLock();
        mX++;
        mY++;
        mS.writeUnlock();
    }

    // Thread T2
    do {
        stamp = mS.tryOptimisticRead();
        z = 1 / (mX - mY);
    } while (mS.validate(stamp));
}
```

See concurrencyfreaks.blogspot.com/2013/11/stampedlocktryoptimisticread-and.html
Java StampedLock Usage Conventions

- StampedLock can be harder to use than ReentrantReadWriteLock
- Many more methods
- More intricate semantics & usage patterns
- Invariants are tricky with optimistic read locks

```java
class Booooom {
    StampedLock mS =
        new StampedLock();
    int mX = 0;
    int mY = 1;
    ...

    // Thread T1
    while (true) {
        mS.writeLock();
        mX++;
        mY++;
        mS.writeUnlock();
    }

    // Thread T2
    do {
        stamp = mS.tryOptimisticRead();
        z = 1 / (mX - mY);
    } while (mS.validate(stamp));
}
```

Create a StampedLock to protect two fields.
Java StampedLock Usage Conventions

- StampedLock can be harder to use than ReentrantReadWriteLock
  - Many more methods
  - More intricate semantics & usage patterns
- Invariants are tricky with optimistic read locks
- Fields read in optimistic mode may be inconsistent since their values can change unpredictably

```java
class Boooom {
    StampedLock mS = new StampedLock();
    int mX = 0;
    int mY = 1;
    ...

    // Thread T1
    while (true) {
        mS.writeLock();
        mX++;
        mY++;
        mS.writeUnlock();
    }

    // Thread T2
    do {
        stamp = mS.tryOptimisticRead();
        z = 1 / (mX - mY);
    } while (mS.validate(stamp));
}
```

Want to establish the invariant \( mX = mY - 1 \)
Java StampedLock Usage Conventions

- StampedLock can be harder to use than ReentrantReadWriteLock
  - Many more methods
  - More intricate semantics & usage patterns
- Invariants are tricky with optimistic read locks
- Fields read in optimistic mode may be inconsistent since their values can change unpredictably

```java
class Boooom {
    StampedLock mS =
        new StampedLock();
    int mX = 0;
    int mY = 1;
    ...

    // Thread T1
    while (true) {
        mS.writeLock();
        mX++;
        mY++;
        mS.writeUnlock();
    }

    // Thread T2
    do {
        stamp = mS.tryOptimisticRead();
        z = 1 / (mX - mY);
    } while (mS.validate(stamp));
}
```

*Since no read lock is held, mX & mY may be reordered, such that invariant mX == mY – 1 may not hold*

See concurrencyfreaks.blogspot.com/2013/11/stampedlocktryoptimisticread-and.html
Java StampedLock Usage Conventions

- StampedLock can be harder to use than ReentrantReadWriteLock
  - Many more methods
  - More intricate semantics & usage patterns
  - Invariants are tricky with optimistic read locks
- Non-reentrant

```java
class SomeComponent {
    private StampedLock sl = new StampedLock();

    public void someMethod1() {
        long stamp = sl.readLock();
        someMethod2();
        ...
    }

    private void someMethod2() {
        long stamp = sl.readLock();
        ...
    }
}
```
• StampedLock is usually the best choice for readers-writer locks in Java 8+!
• Despite its complexity & lack of reentrant semantics

End of Java StampedLock: Usage Considerations