Java Readers/Writer Locks

(Part 3)

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 benefits that readers-writer synchronizers provide to concurrent programs

• Know the key methods in Java 8 StampedLock

• Recognize how to apply Java 8 StampedLock in practice

class Point { ...
    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();
                }
            }
        ...
    }
Applying StampedLock in Practice
Applying StampedLock in Practice

• The Point class shows how to program with StampedLock

```java
class Point {

    private double x;
    private double y;

    private final StampedLock sl =
        new StampedLock();

    ...

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/StampedLock.html
```
Applying StampedLock in Practice

- The Point class shows how to program with StampedLock

```java
class Point {
    private double x;
    private double y;

    private final StampedLock sl = new StampedLock();
    ...
}
```

Maintains two-dimensional points
The Point class shows how to program with StampedLock

class Point {
    private double x;
    private double y;

    private final StampedLock sl =
        new StampedLock();

    ...

    State that must be protected

Applying StampedLock in Practice

- The Point class shows how to program with StampedLock

```java
class Point {
    private double x;
    private double y;

    private final StampedLock sl =
        new StampedLock();
    ...
}
```
Applying StampedLock in Practice

- The Point class shows how to program with StampedLock

```java
class Point {
    ...
    An exclusively locked method
    void move(double deltaX, double deltaY) {
        long stamp = sl.writeLock();
        try {
            x += deltaX;
            y += deltaY;
        } finally {
            sl.unlockWrite(stamp);
        }
    }
    ...
}
```
Applying StampedLock in Practice

- The Point class shows how to program with StampedLock

```java
class Point {
    ...

    void move(double deltaX,
               double deltaY) {
        long stamp = sl.writeLock();
        try {
            x += deltaX;
            y += deltaY;
        } finally {
            sl.unlockWrite(stamp);
        }
    }

    ...
```
The Point class shows how to program with StampedLock

class Point {
    ...

    void move(double deltaX, double deltaY) {
        long stamp = sl.writeLock();
        try {
            x += deltaX;  
            y += deltaY; 
        } finally {
            sl.unlockWrite(stamp); 
        }
    }
    ...
}
The Point class shows how to program with StampedLock

```java
class Point {
    ...

    void move(double deltaX,
              double deltaY) {
        long stamp = sl.writeLock();
        try {
            x += deltaX;
            y += deltaY;
        } finally {
            sl.unlockWrite(stamp);
        }
    }
    ...
```

Release the write lock
Applying StampedLock in Practice

• The Point class shows how to program with StampedLock

```java
class Point {
    ...

    double distanceFromOrigin() {
        long stamp = sl.tryOptimisticRead();
        double currX = x, currY = y;
        if (!sl.validate(stamp)) {
            stamp = sl.readLock();
            try {
                currX = x; currY = y;
            } finally {
                sl.unlockRead(stamp);
            }
        }
        return Math.sqrt(currX * currX + currY * currY);
    }
    ...

    A read-only method
```
class Point {
    ...
    double distanceFromOrigin() {
        long stamp = sl.tryOptimisticRead();
        double currX = x, currY = y;
        if (!sl.validate(stamp)) {
            stamp = sl.readLock();
            try {
                currX = x; currY = y;
            } finally {
                sl.unlockRead(stamp);
            }
        }
        return Math.sqrt(currX * currX + currY * currY);
    }
    ...
}
The Point class shows how to program with StampedLock

```java
class Point {
    ...
    double distanceFromOrigin() {
        long stamp = sl.tryOptimisticRead();
        double currX = x, currY = y;
        if (!sl.validate(stamp)) {
            stamp = sl.readLock();
            try {
                currX = x; currY = y;
            } finally {
                sl.unlockRead(stamp);
            }
        }
        return Math.sqrt(currX * currX + currY * currY);
    }
    ...
}
```

“Optimistically” read state into local variables

Optimistic read sections typically read fields & hold them in local variables for later use after validation
The Point class shows how to program with StampedLock

class Point {
    ...

    double distanceFromOrigin() {
        long stamp = sl.tryOptimisticRead();
        double currX = x, currY = y;
        if (!sl.validate(stamp)) {
            stamp = sl.readLock();
            try {
                currX = x; currY = y;
            } finally {
                sl.unlockRead(stamp);
            }
        }

        return Math.sqrt(currX * currX + currY * currY);
    }

    ...

Check if another thread acquired a write lock after call to tryOptimisticRead()
The Point class shows how to program with StampedLock

class Point {
    ...
    double distanceFromOrigin() {
        long stamp = sl.tryOptimisticRead();
        double currX = x, currY = y;
        if (!sl.validate(stamp)) {
            stamp = sl.readLock();
            try {
                currX = x; currY = y;
            } finally {
            sl.unlockRead(stamp);
        }
        }
        return Math.sqrt(currX * currX + currY * currY);
    }
    ...
}
Applying StampedLock in Practice

• The Point class shows how to program with StampedLock

class Point {
    ...
    double distanceFromOrigin() {
        long stamp = sl.tryOptimisticRead();
        double currX = x, currY = y;
        if (!sl.validate(stamp)) {
            stamp = sl.readLock();
            try {
                currX = x; currY = y;  // Do “pessimistic” reads
            } finally {
                sl.unlockRead(stamp);
            }
        }
        return Math.sqrt(currX * currX + currY * currY);
    }
    ...
}
Applying StampedLock in Practice

- The Point class shows how to program with StampedLock

```java
class Point {
    // ...
    double distanceFromOrigin() {
        long stamp = sl.tryOptimisticRead();
        double currX = x, currY = y;
        if (!sl.validate(stamp)) {
            stamp = sl.readLock();
            try {
                currX = x; currY = y;
            } finally {
                sl.unlockRead(stamp);
            }
        }
        return Math.sqrt(currX * currX + currY * currY);
    }
    // ...
```
Applying StampedLock in Practice

• The Point class shows how to program with StampedLock

class Point {
    ...
    double distanceFromOrigin() {
        long stamp = sl.tryOptimisticRead();
        double currX = x, currY = y;
        if (!sl.validate(stamp)) {
            stamp = sl.readLock();
            try {
                currX = x; currY = y;
            } finally {
                sl.unlockRead(stamp);}
        }  // No lock to release if validate() succeeded
        return Math.sqrt (currX * currX + currY * currY);
    }
    ...
}
Applying StampedLock in Practice

- The Point class shows how to program with StampedLock

```java
class Point {
    ...
    double distanceFromOrigin() {
        long stamp = sl.tryOptimisticRead();
        double currX = x, currY = y;
        if (!sl.validate(stamp)) {
            stamp = sl.readLock();
            try {
                currX = x; currY = y;
            } finally {
                sl.unlockRead(stamp);
            }
        }
        return Math.sqrt(currX * currX + currY * currY);
    }
    ...
```
The Point class shows how to program with StampedLock

```java
class Point {
    ...  // Conditional write
    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();
                }
            }
        } catch (InterruptedException e) {
            // Handle exception
        }
    }
    ...
}
```
The Point class shows how to program with StampedLock

```java
class Point {
    ...
    void moveIfAtOrigin(double newX, double newY) {
        long stamp = sl.readLock(); // Acquire a read lock
        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();
                }
            }
        } catch (InterruptedException e) {
            // Handle interruption
        }
    }
    ...
```
The Point class shows how to program with StampedLock

class Point {
    ...
    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();
                }
            }
        } catch (InterruptedException e) {
            // Handle interruption
        }
    }
    ...
}
The Point class shows how to program with StampedLock

class Point {
    ...
    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();
                }
            }
        } else {
            sl.unlockRead(stamp);
            stamp = sl.writeLock();
        }
    }
    ...
}
class Point {
    
    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();
                }
            }
        }
        ...
    }
    ...
}
Applying StampedLock in Practice

- The Point class shows how to program with StampedLock

class Point {
    ...  
    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();
                }
            }
        } catch (Exception e) {
            // handle exception
        }
    }
    ...
}
Applying StampedLock in Practice

• The Point class shows how to program with StampedLock

class Point {
  ...
  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; // Exit the loop
        } else {
          sl.unlockRead(stamp);
          stamp = sl.writeLock();
        }
      }
    }
  }
  ...

The Point class shows how to program with StampedLock

```java
class Point {
    ...
    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();
                }
            }
        } catch (ConversionFailedException e) {
            // Conversion failed, release read lock & block until write lock acquired
        }
    }
    ...
}
```

The `x` & `y` field values may change between `unlockRead()` & `writeLock()`!
Applying StampedLock in Practice

- The Point class shows how to program with StampedLock

```java
class Point {
    ...  
    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();
                }
            }
        } catch (InterruptedException e) {
            ...  
        }
    }
    ...
}
```

Must retest loop condition since x & y field values may change between unlockRead() & writeLock()!
class Point {

    ... 

    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();
                }
            }
        } catch (InterruptedException e) {
            // handle exception
        }
    }

    ...
Applying StampedLock in Practice

• The Point class shows how to program with StampedLock

class Point {
    ...
    void moveIfAtOrigin(double newX, double newY) {
        long stamp = sl.readLock();
        try {
            while (x == 0.0 && y == 0.0) {
                ...
                stamp = ws;
                ...
                stamp = sl.writeLock();
            }
        } finally { sl.unlock(stamp); }
    }
    ...
}

Release the appropriate lock
End of Java Readers/Writer Locks (Part 3)
Java Readers/Writer Locks

(Part 4)

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 benefits that readers-writer synchronizers provide to concurrent programs
- Know the key methods in Java 8 StampedLock
- Recognize how to apply Java 8 StampedLock in practice
- Appreciate Java 8 StampedLock usage considerations
StampedLock Usage Considerations
### 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

StampedLock Usage Conventions

- StampedLock often *much* faster than ReentrantReadWriteLock
- However, your mileage may vary!
### StampedLock Usage Conventions

- StampedLock often *much* faster than ReentrantReadWriteLock
- However, your mileage may vary!

<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

Readers-writer lock speedups are only fully realized under certain conditions.
Readers-writer lock speedups are only fully realized under certain conditions, e.g.

- Frequency of reads to writes
  - Ideally, *many* more reads than writes
Readers-writer lock 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 will dominate
Readers-writer lock speedups are only fully realized under certain conditions, e.g.

- Frequency of reads to writes
- Duration of read & write operations
- Contention for the data

- Ideally, *many* concurrent readers
Readers-writer lock 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 cores
  - Ideally, many cores
StampedLock Usage Conventions

- StampedLock is more complicated to use
StampedLock Usage Conventions

- StampedLock is more complicated to use
- Many more methods
StampedLock Usage Conventions

- StampedLock is more complicated to use
  - 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](http://www.techevents.online/using-java-8-lambdas-stampedlock-manage-thread-safety)
StampedLock Usage Conventions

• StampedLock is more complicated to use
  • Many more methods
  • More intricate semantics & usage patterns
• Invariants are tricky with optimistic read locks

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

while (true) {
    mSL.writeLock();
    mX++; mY++;
    mSL.writeUnlock();
}
```

```java
Thread T1

while (true) {
    mSL.writeLock();
    mX++; mY++; mX++; mY++;
    mSL.writeUnlock();
}
```

```java
Thread T2

do {
    stamp = mSL.tryOptimisticRead();
    z = 1 / (mX - mY);
} while (mSL.validate(stamp));
```
StampedLock Usage Conventions

- StampedLock is more complicated to use
  - 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 Booom {  
    int mX = 0;  
    int mY = 1;  
    StampedLock mSL =  
        new StampedLock();  
}  

while (true) {  
    mSL.writeLock();  
    mX++; mY++;  
    mSL.writeUnlock();  
}  

Thread T₁

while (true) {  
    mSL.writeLock();  
    mX++; mY++;  
    mSL.writeUnlock();  
}  

Thread T₂

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

The values of mX & mY may be reordered & thus mX may not always == mY - 1

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

- StampedLock is more complicated to use
  - 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();
        ...
    }
}
```
• Despite its complexity, StampedLock is the best choice for readers-writer locks in Java 8+!

See www.javaspecialists.eu/archive/Issue215.html
End of Java Readers/Writer Locks (Part 4)