Java Readers/Writer Locks

(Part 1)

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
Overview of Readers-Writer Locks
Overview of Readers-Writer Locks

- A readers-writer lock is a synchronizer used to solve readers-writer problems.

See en.wikipedia.org/wiki/Readers-writer_lock
Overview of Readers-Writer Locks

- A readers-writer lock is a synchronizer used to solve readers-writer problems, e.g.
- Allows concurrent read access
Overview of Readers-Writer Locks

- A readers-writer lock is a synchronizer used to solve readers-writer problems, e.g.
  - Allows concurrent read access
  - But exclusive writer access
Overview of Readers-Writer Locks

- Readers/writer locks are used when many concurrent threads access a shared resource.
Overview of Readers-Writer Locks

- Readers/writer locks are used when many concurrent threads access a shared resource, e.g.
- Multiple threads can have read-only access
Overview of Readers-Writer Locks

- Readers/writer locks are used when many concurrent threads access a shared resource, e.g.
  - Multiple threads can have read-only access
  - Only one thread can have write access
Overview of Readers-Writer Locks

- Java supports two types of readers-writer locks

```
Class StampedLock
java.lang.Object
java.util.concurrent.locks.StampedLock

All Implemented Interfaces:
Serializable

public class StampedLock
extends Object
implements Serializable

A capability-based lock with three modes for controlling read/write access. The state of a StampedLock consists of a version and mode. Lock acquisition methods return a stamp that represents and controls access with respect to a lock state; "try" versions of these methods may instead return the special value zero to represent failure to acquire access. Lock release and conversion methods require stamps as arguments, and fail if they do not match the state of the lock. The three
```

```
Class ReentrantReadWriteLock
java.lang.Object
java.util.concurrent.locks.ReentrantReadWriteLock

All Implemented Interfaces:
Serializable, ReadWriteLock

public class ReentrantReadWriteLock
extends Object
implements ReadWriteLock, Serializable
```

Overview of Readers-Writer Locks

- Java’s readers-writer locks may help improve performance
- e.g., when resources are read much more often than written
Overview of Readers-Writer Locks

- Java’s readers-writer locks *may* help improve performance
  - e.g., when resources are *read* much more often than *written*
  - Especially on multi-core & multi-processor platforms

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/ReadWriteLock.html
Overview of Readers-Writer Locks

- Readers-writer locks can be problematic in practice
Overview of Readers-Writer Locks

- Readers-writer locks can be problematic in practice, e.g.
- Can lead to starvation
Overview of Readers-Writer Locks

- Readers-writer locks can be problematic in practice, e.g.
  - Can lead to starvation
  - May be significantly slower than other synchronizers

Overview of Readers-Writer Locks

• Readers-writer locks can be problematic in practice, e.g.
  • Can lead to starvation
  • May be significantly slower than other synchronizers
  • Can be hard to program
Human Known Use of Readers-Writer Locks
A human known use of a readers-writer locking protocol is the process for periodic updates of material on websites that change infrequently.
A human known use of a readers-writer locking protocol is the process for periodic updates of material on websites that change infrequently, e.g.

The Vanderbilt undergraduate course catalog is accessed continuously, but only changes once a year.
A human known use of a readers-writer locking protocol is the process for periodic updates of material on websites that change infrequently, e.g.

The Vanderbilt undergraduate course catalog is accessed continuously, but only changes once a year.

Therefore, multiple readers can access the catalog concurrently.
A human known use of a readers-writer locking protocol is the process for periodic updates of material on websites that change infrequently, e.g.

- The Vanderbilt undergraduate course catalog is accessed continuously, but only changes once a year.
- Therefore, multiple readers can access the catalog concurrently.
- But only one writer can access the catalog when it’s updated annually.
End of Java Readers/Writer Locks (Part 1)
Java Readers/Writer Locks

(Part 2)

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
Overview of StampedLock
Overview of StampedLock

- Provides a readers-writer implementation in Java 8

**Class StampedLock**

```java
java.lang.Object
java.util.concurrent.locks.StampedLock
```

**All Implemented Interfaces:**

Serializable

```java
public class StampedLock
extends Object
implements Serializable
```

A capability-based lock with three modes for controlling read/write access. The state of a StampedLock consists of a version and mode. Lock acquisition methods return a stamp that represents and controls access with respect to a lock state; "try" versions of these methods may instead return the special value zero to represent failure to acquire access. Lock release and conversion methods require stamps as arguments, and fail if they do not match the state of the lock. The three modes are:

- **Writing.** Method `writeLock()` possibly blocks waiting for exclusive access, returning a stamp that can be used in method `unlockWrite(long)` to release the lock. Untimed and timed versions of `tryWriteLock` are also provided. When the lock is held in write mode, no read locks may be obtained, and all optimistic read validations will fail.

- **Reading.** Method `readLock()` possibly blocks waiting for non-exclusive access, returning a stamp that can be used in method `unlockRead(long)` to release the lock. Untimed and timed versions of `tryReadLock` are also provided.

- **Optimistic Reading.** Method `tryOptimisticRead()` returns a non-zero stamp only if the lock is not currently held in write mode. Method `validate(long)` returns true if the lock has not been acquired in write mode since obtaining a given stamp. This mode can be thought of as an extremely weak version of a read-lock, that can be broken by a writer at any time. The use of optimistic mode for short read-only code segments often

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/StampedLock.html](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/StampedLock.html)
Overview of StampedLock

- Provides a readers-writer implementation in Java 8
- Much more efficient & scalable than ReentrantReadWriteLock

Class StampedLock

```
java.lang.Object
java.util.concurrent.locksStampedLock
```

All Implemented Interfaces:
Serializable

```
public class StampedLock
extends Object
implements Serializable
```

A capability-based lock with three modes for controlling read/write access. The state of a StampedLock consists of a version and mode. Lock acquisition methods return a stamp that represents and controls access with respect to a lock state; "try" versions of these methods may instead return the special value zero to represent failure to acquire access. Lock release and conversion methods require stamps as arguments, and fail if they do not match the state of the lock. The three modes are:

- **Writing.** Method `writeLock()` possibly blocks waiting for exclusive access, returning a stamp that can be used in method `unlockWrite(long)` to release the lock. Untimed and timed versions of `tryWriteLock()` are also provided. When the lock is held in write mode, no read locks may be obtained, and all optimistic read validations will fail.

- **Reading.** Method `readLock()` possibly blocks waiting for non-exclusive access, returning a stamp that can be used in method `unlockRead(long)` to release the lock. Untimed and timed versions of `tryReadLock()` are also provided.

- **Optimistic Reading.** Method `tryOptimisticRead()` returns a non-zero stamp only if the lock is not currently held in write mode. Method `validate(long)` returns true if the lock has not been acquired in write mode since obtaining a given stamp. This mode can be thought of as an extremely weak version of a read-lock that can be broken by a writer at any time. The use of optimistic mode for short read-only code segments often
• Provides a readers-writer implementation in Java 8

```java
public class StampedLock implements java.io.Serializable {
```
Overview of StampedLock

• Provides a readers-writer implementation in Java 8

```java
public class StampedLock implements java.io.Serializable {

  Does not implement ReadWriteLock interface, does not use AbstractQueuedSynchronizer, & does not apply Bridge pattern
```


Overview of StampedLock

- Provides three locking modes

```java
public class StampedLock implements java.io.Serializable {
 ...
```

These modes go above & beyond what's supported in ReentrantReadWriteLock
Overview of StampedLock

- Provides three locking modes
- Writing

```java
public class StampedLock
implements java.io.Serializable {
...
public long writeLock() { ... }
public long tryWriteLock() { ... }
public long tryWriteLock
(long time,
 TimeUnit unit) { ... }
...
```

Writing mode is “pessimistic” since it assumes contention may occur, so no other thread can acquire the lock while it’s held, i.e., a write lock is “exclusive”
Overview of StampedLock

• Provides three locking modes

• Writing

```java
public class StampedLock
    implements java.io.Serializable {
    ...
    public long writeLock() { ... }

    public long tryWriteLock() { ... }

    public long tryWriteLock
        (long time,
         TimeUnit unit) { ... }

    ...
}
```

All methods return a “stamp” value, which is a long that contains a version & a mode
Overview of StampedLock

- Provides three locking modes
- Writing
  - Acquires lock exclusively, blocking until available

```java
public class StampedLock implements java.io.Serializable {
    ...
    public long writeLock() { ... }  
    public long tryWriteLock() { ... }
    public long tryWriteLock
    (long time,
     TimeUnit unit) {...}
    ...
```
Overview of StampedLock

• Provides three locking modes
  • Writing
    • Acquires lock exclusively, blocking until available
    • Acquires lock exclusively if it’s immediately available

```java
class StampedLock implements java.io.Serializable {
  ...
  public long writeLock() { ... }
  public long tryWriteLock() { ... }
  public long tryWriteLock(long time, TimeUnit unit) { ... }
  ...
```
Overview of StampedLock

• Provides three locking modes
  • Writing
    • Acquires lock exclusively, blocking until available
    • Acquires lock exclusively if it’s immediately available
    • Acquires lock exclusively if available within given time

public class StampedLock
  implements java.io.Serializable {
    ...
    public long writeLock() { ... }
    public long tryWriteLock() { ... }
    public long tryWriteLock
      (long time,
       TimeUnit unit) {...
    ...
  ...
Overview of StampedLock

- Provides three locking modes
  - Writing
  - Reading

```java
public class StampedLock
    implements java.io.Serializable {
    ...
    public long readLock() { ... }
    public long tryReadLock() { ... }
    public long tryReadLock(
        long time,
        TimeUnit unit) { ... }
    ...
}
```

Reading mode is “pessimistic” since it assumes contention may occur, though other thread can acquire the lock for reading, i.e., a read lock is “shared”
Overview of StampedLock

- Provides three locking modes
  - Writing
  - Reading

```java
public class StampedLock implements java.io.Serializable {
    ...
    public long readLock() { ... }
    public long tryReadLock() { ... }
    public long tryReadLock(long time,
                              TimeUnit unit) { ... }
    ...
```
Overview of StampedLock

• Provides three locking modes
  • Writing
  • Reading
  • Acquires lock non-exclusively, blocking until available

```java
public class StampedLock
    implements java.io.Serializable {
    ...
    public long readLock() { ... }
    public long tryReadLock() { ... }
    public long tryReadLock(
        long time,
        TimeUnit unit) { ... }
    ...
```
Overview of StampedLock

- Provides three locking modes
  - Writing
  - Reading
    - Acquires lock non-exclusively, blocking until available
    - Acquires lock non-exclusively if immediately available

```java
public class StampedLock
    implements java.io.Serializable {
    ...
    public long readLock() { ... }
    public long tryReadLock() { ... }
    public long tryReadLock(
        long time,
        TimeUnit unit) {...}
    ...
```
**Overview of StampedLock**

- Provides three locking modes
  - Writing
  - Reading
    - Acquires lock non-exclusively, blocking until available
    - Acquires lock non-exclusively if immediately available
    - Acquires lock non-exclusively if it is available within given time

```java
public class StampedLock implements java.io.Serializable {
    ...
    public long readLock() { ... }
    public long tryReadLock() { ... }
    public long tryReadLock(long time, TimeUnit unit) { ... }
    ...
```
Overview of StampedLock

- Provides three locking modes
  - Writing
  - Reading
  - Optimistic reading

public class StampedLock
    implements java.io.Serializable {
    ...
    public long tryOptimisticRead()
    {
        ...
    }
    public boolean validate
        (long stamp) { ...
    }
    ...

This reading mode is “optimistic” since it assumes contention will not occur, so other threads can obtain the lock optimistically, i.e., the lock is “probabilistic”
Overview of StampedLock

- Provides three locking modes
  - Writing
  - Reading
  - Optimistic reading

```java
public class StampedLock implements java.io.Serializable {
    ...
    public long tryOptimisticRead() {
        ...
    }
    ...
    public boolean validate (long stamp) {
        ...
    }
    ...
}
```

Optimistic read sections typically read fields & hold them in local variables for later use after validation.
Overview of StampedLock

• Provides three locking modes
  • Writing
  • Reading
  • Optimistic reading
• Returns an “observation stamp” for later validation or 0 if the lock is currently held exclusively

```
public class StampedLock
  implements java.io.Serializable {
    ...
    public long tryOptimisticRead() {
      ... }
    public boolean validate (long stamp) {
      ... }
    ...

Internally, tryOptimisticRead() does a volatile read on a field inside of the StampedLock, which ensures fresh values of fields are observed
```
Overview of StampedLock

- Provides three locking modes
  - Writing
  - Reading
  - Optimistic reading
    - Returns an “observation stamp” for later validation or 0 if the lock is currently held exclusively
    - Returns true if lock hasn’t been acquired exclusively since stamp was issued

```java
class StampedLock implements java.io.Serializable {
    public long tryOptimisticRead() {
        ...
    }
    public boolean validate(long stamp) {
        ...
    }
    ...
}
```

If validate() succeeds (i.e., returns true) synchronization overhead is very low & there’s no need to unlock the “lock”
Overview of StampedLock

- Conditionally perform conversions across lock modes

```java
public class StampedLock implements java.io.Serializable {
    ...
    public long tryToConvertToWriteLock (long stamp) {
        ...
    }

    public long tryToConvertToReadLock (long stamp) {
        ...
    }

    public long tryToConvertToOptimisticRead (long stamp) {
        ...
    }
    ...
```
Overview of StampedLock

- Conditionally convert to a write lock

```java
public class StampedLock implements java.io.Serializable {
    ...

    public long tryToConvertToWriteLock (long stamp) { ... }

    public long tryToConvertToReadLock (long stamp) { ... }

    public long tryToConvertToOptimisticRead (long stamp) { ... }

    ...
}
```
Overview of StampedLock

- Conditionally perform conversions across lock modes
- If lock state matches stamp, performs one following action

```java
public class StampedLock
    implements java.io.Serializable {
    ...
    public long
        tryToConvertToWriteLock
        (long stamp) { ... }

    public long
        tryToConvertToReadLock
        (long stamp) { ... }

    public long
        tryToConvertToOptimisticRead
        (long stamp) { ... }

    ...
```
• Conditionally perform conversions across lock modes
• If lock state matches stamp, performs one following action
• If stamp represents holding a write lock, return it

```java
public class StampedLock
    implements java.io.Serializable {
    ...
    public long
        tryToConvertToWriteLock
        (long stamp) { ... }

    public long
        tryToConvertToReadLock
        (long stamp) { ... }

    public long
        tryToConvertToOptimisticRead
        (long stamp) { ... }
    ...
```
• Conditionally perform conversions across lock modes
  • If lock state matches stamp, performs one following action
    • If stamp represents holding a write lock, return it
    • If stamp represents holding a read lock—and a write lock is available—atomically release read lock & return write stamp

public class StampedLock
  implements java.io.Serializable { ... 
  public long tryToConvertToWriteLock (long stamp) { ... } 
  public long tryToConvertToReadLock (long stamp) { ... } 
  public long tryToConvertToOptimisticRead (long stamp) { ... } 
...
Overview of StampedLock

- Conditionally perform conversions across lock modes
  - If lock state matches stamp, performs one following action
    - If stamp represents holding a write lock, return it
    - If stamp represents holding a read lock—and a write lock is available—atomically release read lock & return write stamp
    - If stamp represents a read that’s optimistic, return write stamp if available

```java
public class StampedLock implements java.io.Serializable {
    ...
    public long tryToConvertToWriteLock (long stamp) { ... }
    ...
    public long tryToConvertToReadLock (long stamp) { ... }
    ...
    public long tryToConvertToOptimisticRead (long stamp) { ... }
    ...
```
• Conditionally perform conversions across lock modes
  • If lock state matches stamp, performs one following action
    • If stamp represents holding a write lock, return it
    • If stamp represents holding a read lock—and a write lock is available—atomically release read lock & return write stamp
    • If stamp represents a read that’s optimistic, return write stamp if available
  • Else return zero

```java
public class StampedLock
    implements java.io.Serializable {
    ...
    public long
        tryToConvertToWriteLock
            (long stamp) { ... }

    public long
        tryToConvertToReadLock
            (long stamp) { ... }

    public long
        tryToConvertToOptimisticRead
            (long stamp) { ... }
    ...
```
Overview of StampedLock

- Conditionally perform conversions across lock modes

```java
public class StampedLock implements java.io.Serializable {
    ...
    public long
    tryToConvertToWriteLock
    (long stamp) { ... }

    public long
    tryToConvertToReadLock
    (long stamp) { ... }

    public long
    tryToConvertToOptimisticRead
    (long stamp) { ... }
    ...
```
Overview of StampedLock

- Conditionally perform conversions across lock modes
- If lock state matches stamp, performs one following action

```java
public class StampedLock
    implements java.io.Serializable {
    ...
    public long tryToConvertToWriteLock
        (long stamp) { ... }

    public long tryToConvertToReadLock
        (long stamp) { ... }

    public long tryToConvertToOptimisticRead
        (long stamp) { ... }
    ...
```
Overview of StampedLock

- Conditionally perform conversions across lock modes
- If lock state matches stamp, performs one following action
- If stamp represents holding a write lock atomically release it & obtain read lock

```java
class StampedLock implements java.io.Serializable {
    ...
    public long tryToConvertToWriteLock (long stamp) {
        ...
    }
    public long tryToConvertToReadLock (long stamp) {
        ...
    }
    public long tryToConvertToOptimisticRead (long stamp) {
        ...
    }
    ...
}
```
Overview of StampedLock

• Conditionally perform conversions across lock modes
  • If lock state matches stamp, performs one following action
    • If stamp represents holding a write lock atomically release it & obtain read lock
    • If stamp represents holding a read lock, return it

```java
public class StampedLock implements java.io.Serializable {
    ...
    public long tryToConvertToWriteLock(long stamp) { ... }

    public long tryToConvertToReadLock(long stamp) { ... }

    public long tryToConvertToOptimisticRead(long stamp) { ... }
    ...
```
Overview of StampedLock

- Conditionally perform conversions across lock modes
  - If lock state matches stamp, performs one following action
    - If stamp represents holding a write lock atomically release it & obtain read lock
    - If stamp represents holding a read lock, return it
    - If stamp represents holding an optimistic read, return read stamp only if available

```java
class StampedLock implements java.io.Serializable {
    public long tryToConvertToWriteLock(long stamp) { ... }
    public long tryToConvertToReadLock(long stamp) { ... }
    public long tryToConvertToOptimisticRead(long stamp) { ... }
    ...
```
Overview of StampedLock

- Conditionally perform conversions across lock modes
  - If lock state matches stamp, performs one following action
    - If stamp represents holding a write lock atomically release it & obtain read lock
    - If stamp represents holding a read lock, return it
    - If stamp represents holding an optimistic read, return read stamp only if available
  - Else return zero

```java
public class StampedLock implements java.io.Serializable {
    ...
    public long tryToConvertToWriteLock (long stamp) { ... }

    public long tryToConvertToReadLock (long stamp) { ... }

    public long tryToConvertToOptimisticRead (long stamp) { ... }
    ...
}
```
Overview of StampedLock

• Conditionally perform conversions across lock modes

public class StampedLock
  implements java.io.Serializable {

  public long
    tryToConvertToWriteLock
    (long stamp) { ... }

  public long
    tryToConvertToReadLock
    (long stamp) { ... }

  public long
    tryToConvertToOptimisticRead
    (long stamp) { ... }

  ...


• Conditionally perform conversions across lock modes
• If lock state matches stamp, performs one following action

public class StampedLock
    implements java.io.Serializable {
        ...
        public long
            tryToConvertToWriteLock
            (long stamp) { ... }

        public long
            tryToConvertToReadLock
            (long stamp) { ... }

        public long
            tryToConvertToOptimisticRead
            (long stamp) { ... }
        ...
    }
Overview of StampedLock

- Conditionally perform conversions across lock modes
- If lock state matches stamp, performs one following action
- If stamp represents holding a lock release it & return an observation stamp

```java
public class StampedLock
    implements java.io.Serializable {
    ...

    public long
        tryToConvertToWriteLock
        (long stamp) { ... }

    public long
        tryToConvertToReadLock
        (long stamp) { ... }

    public long
        tryToConvertToOptimisticRead
        (long stamp) { ... }

    ...
```
• Conditionally perform conversions across lock modes
  • If lock state matches stamp, performs one following action
    • If stamp represents holding a lock release it & return an observation stamp
  • If stamp represents holding an optimistic read, return it if it’s validated

```java
public class StampedLock implements java.io.Serializable {
    ...
    public long tryToConvertToWriteLock (long stamp) { ... }

    public long tryToConvertToReadLock (long stamp) { ... }

    public long tryToConvertToOptimisticRead (long stamp) { ... }

    ...
}
```
Overview of StampedLock

- Conditionally perform conversions across lock modes
  - If lock state matches stamp, performs one following action
    - If stamp represents holding a lock release it & return an observation stamp
    - If stamp represents holding an optimistic read, return it if it’s validated
  - Else return zero

```java
public class StampedLock
    implements java.io.Serializable {
    ...
    public long
        tryToConvertToWriteLock
        (long stamp) { ... }

    public long
        tryToConvertToReadLock
        (long stamp) { ... }

    public long
        tryToConvertToOptimisticRead
        (long stamp) { ... }

    ...
```
Overview of StampedLock

• Provides several ways to release the lock

```java
public class StampedLock
    implements java.io.Serializable {
    ...

    public void unlockWrite
        (long stamp) { ... }

    public void unlockRead
        (long stamp) { ... }

    public void unlock
        (long stamp) { ... }

    ...
```
Overview of StampedLock

- Provides several ways to release the lock
- Releases exclusive lock if state matches given stamp

```java
public class StampedLock
    implements java.io.Serializable {
    ... 
    public void unlockWrite
        (long stamp) { ... }

    public void unlockRead
        (long stamp) { ... }

    public void unlock
        (long stamp) { ... }

    ... 
```
Overview of StampedLock

- Provides several ways to release the lock
  - Releases exclusive lock if state matches given stamp
  - Releases non-exclusive lock if state matches given stamp

```java
public class StampedLock
    implements java.io.Serializable {
    ...
    public void unlockWrite
        (long stamp) { ... }

    public void unlockRead
        (long stamp) { ... }

    public void unlock
        (long stamp) { ... }

    ...
```
Overview of StampedLock

- Provides several ways to release the lock
  - Releases exclusive lock if state matches given stamp
  - Releases non-exclusive lock if state matches given stamp
  - Releases lock if lock state matches given stamp

public class StampedLock
    implements java.io.Serializable {

    public void unlockWrite
        (long stamp) { ... }

    public void unlockRead
        (long stamp) { ... }

    public void unlock
        (long stamp) { ... }

    ...

unlock() is less efficient than unlockWrite() & unlockRead()
End of Java Readers/Writer Locks (Part 2)