Pomiary porównawcze

Jan Gawroński

18.11.2025

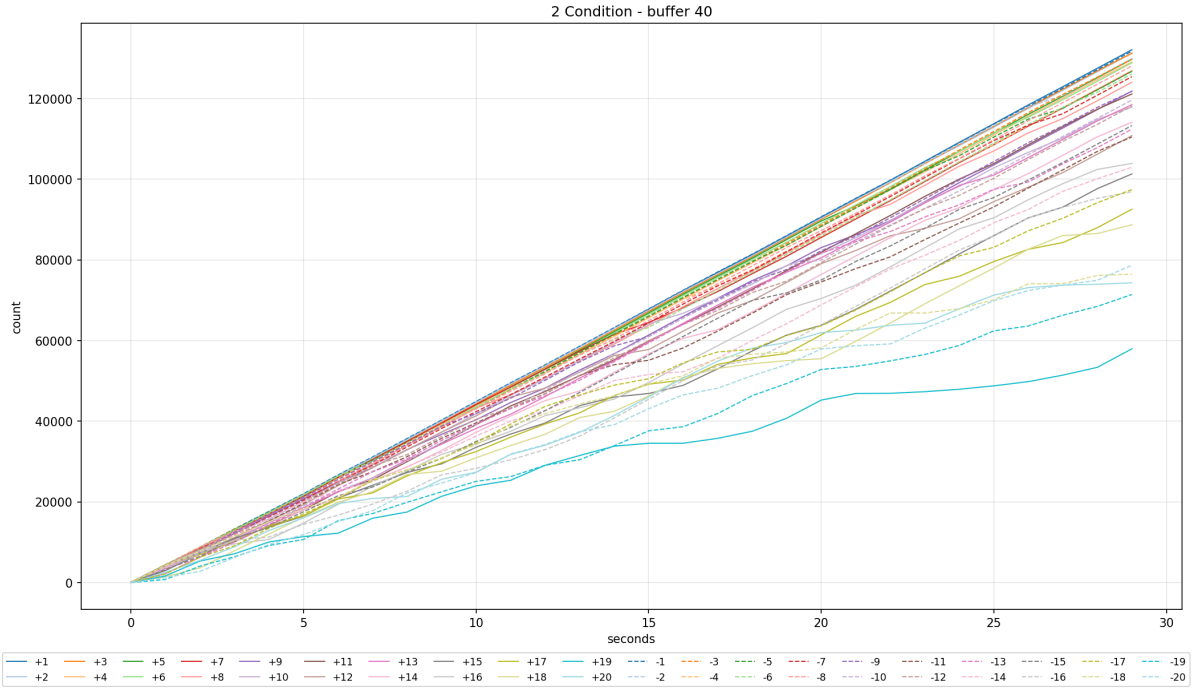
1 2 warunki

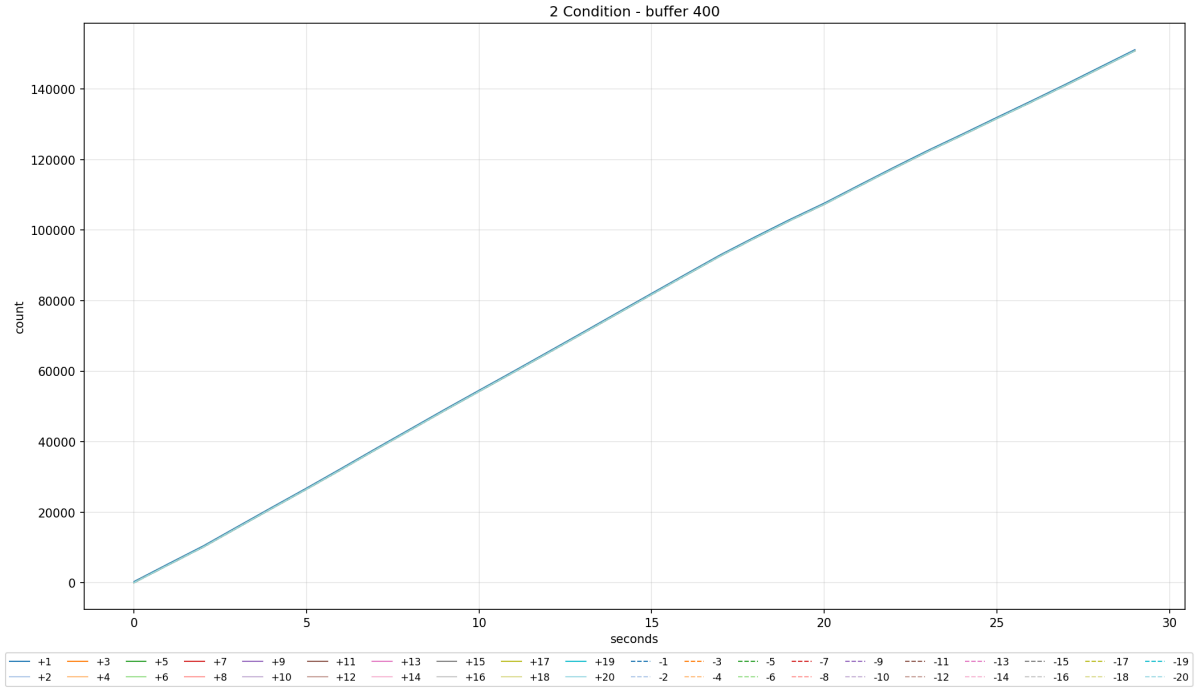
1.1 Kod

```
public class RandomProduceConsume {
    private int buffer = 0;
    private final int bufferMax;
    private final ReentrantLock lock = new ReentrantLock(true);
    private final Condition notFull = lock.newCondition();
    private final Condition notEmpty = lock.newCondition();

    public RandomProduceConsume(int bufferMax) {
        this.bufferMax = bufferMax;
    }

    public void produce(int amount) {
        lock.lock();
        try {
            while (buffer + amount >= bufferMax)
                notFull.await();
            buffer += amount;
            notEmpty.signal();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }
    }

    public void consume(int amount) {
        lock.lock();
        try {
            while (buffer - amount < 0)
                notEmpty.await();
            buffer -= amount;
```

```
                notFull.signal();
            } catch (InterruptedException e) {
                e.printStackTrace();
            } finally {
                lock.unlock();
            }
        }
    }
```

1.2 Wykresy





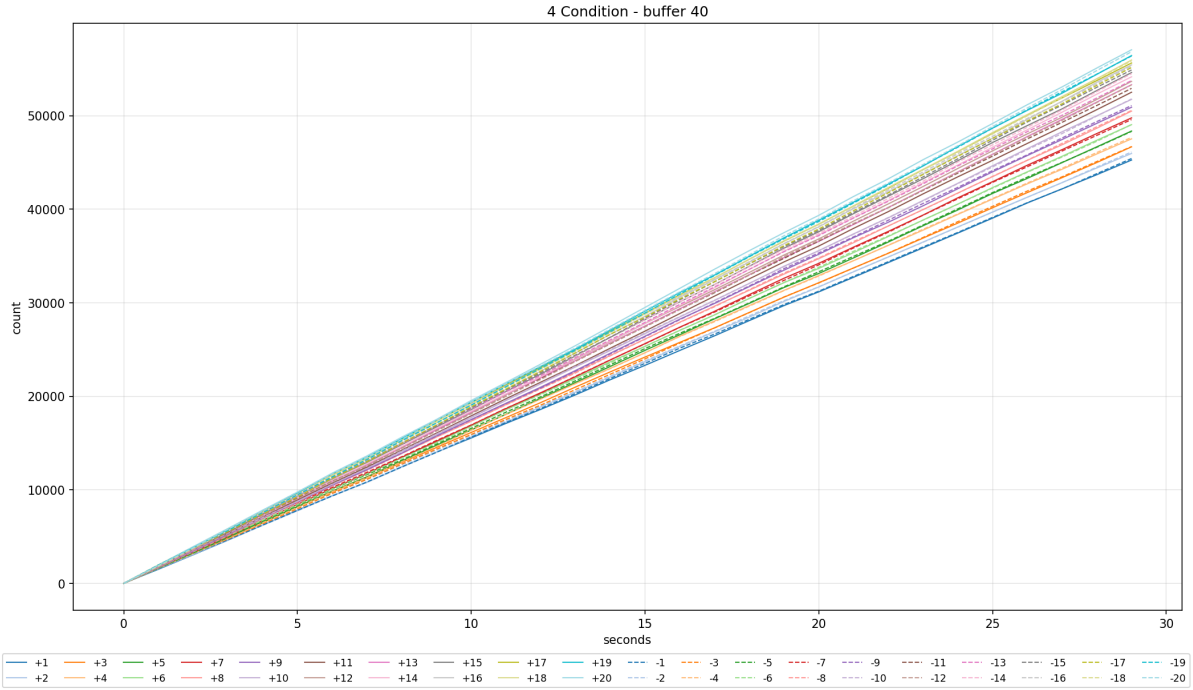
2 4 warunki

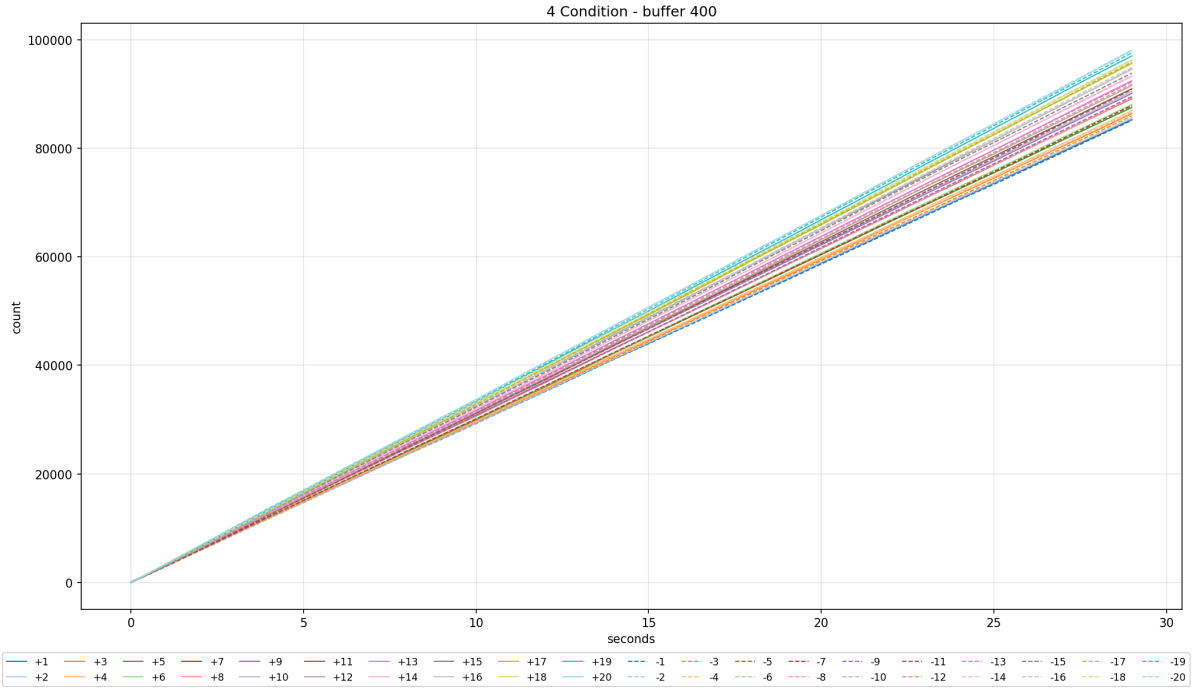
2.1 Kod

```
ublic class CorrectRandomProduceConsume {
    private int buffer = 0;
    private final int bufferMax;
    private final ReentrantLock lock = new ReentrantLock(true);
    private final Condition firstProd = lock.newCondition();
    private final Condition restProd = lock.newCondition();
    private final Condition firstCons = lock.newCondition();
    private final Condition restCons = lock.newCondition();

    private boolean isFirstProducerWaiting = false;
    private boolean isFirstConsumerWaiting = false;

    public CorrectRandomProduceConsume(int bufferMax) {
        this.bufferMax = bufferMax;
    }

    public void produce(int amount) {
        lock.lock();
        try {
            while (isFirstProducerWaiting)
                restProd.await();
            isFirstProducerWaiting = true;
            while (buffer + amount >= bufferMax)
```

```
                firstProd.await();
            buffer += amount;
            isFirstProducerWaiting = false;
            restProd.signal();
            firstCons.signal();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }
    }

    public void consume(int amount) {
        lock.lock();
        try {
            while (isFirstConsumerWaiting)
                restCons.await();
            isFirstConsumerWaiting = true;
            while (buffer - amount < 0)
                firstCons.await();
            buffer -= amount;
            isFirstConsumerWaiting = false;
            restCons.signal();
            firstProd.signal();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }
    }
}
```

2.2 Wykresy



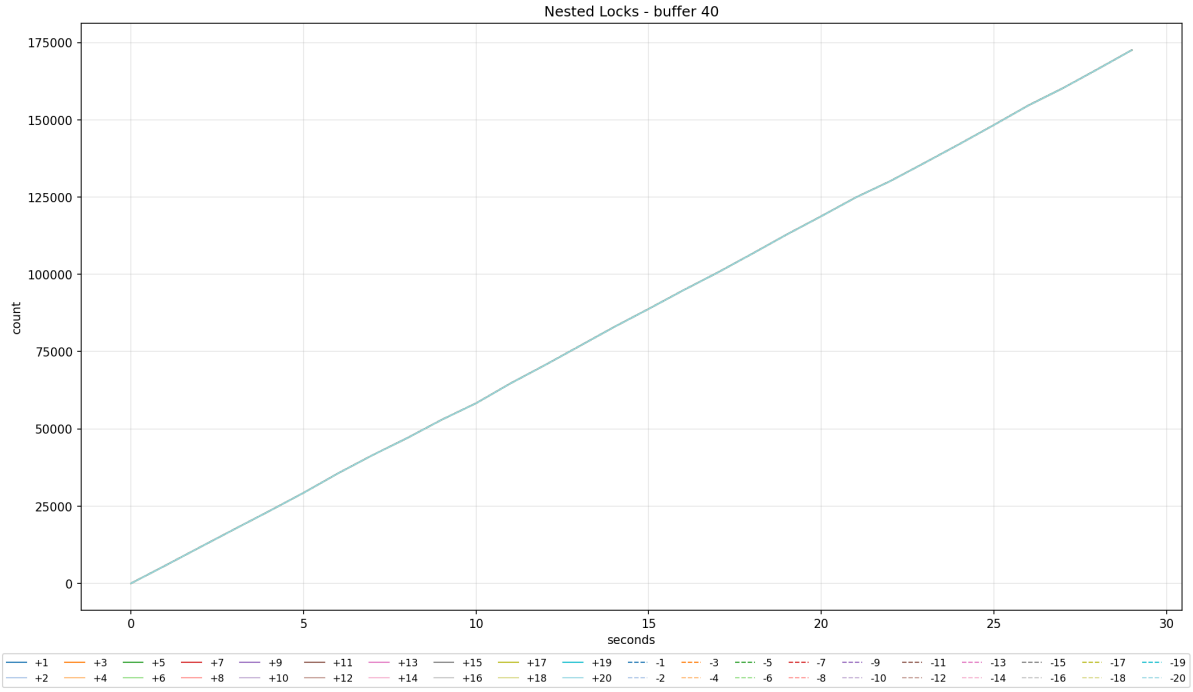


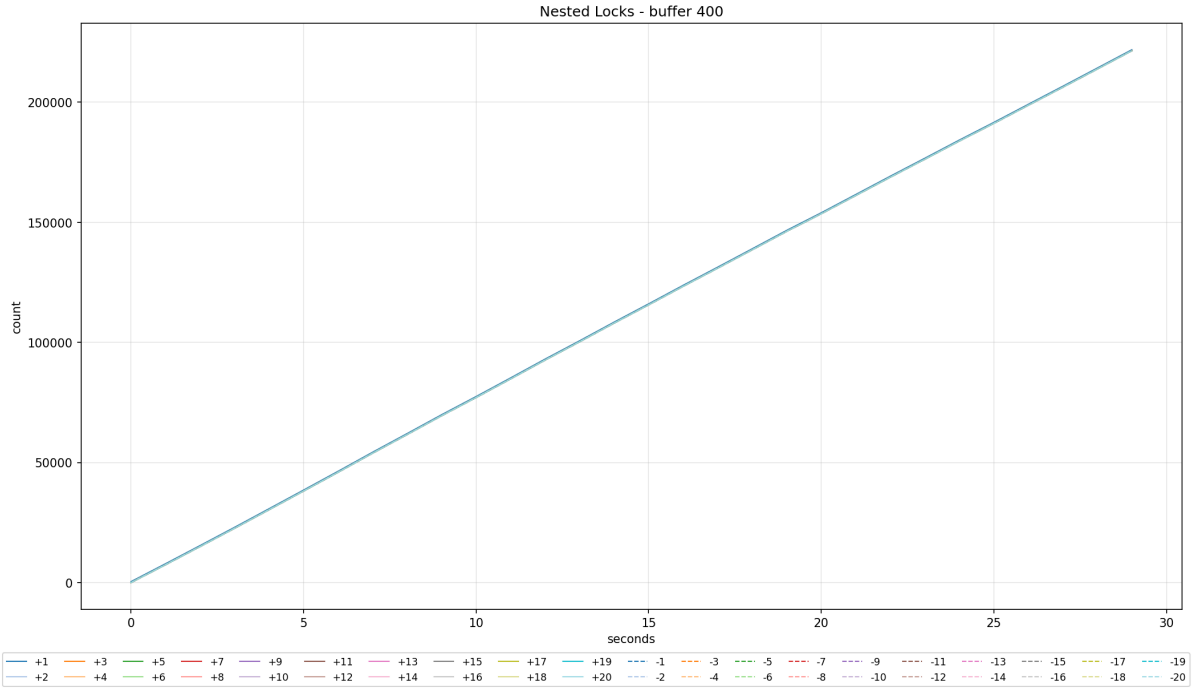
3 Zagnieżdżone blokady

3.1 Kod

```
public class NestedLocks {
    private int buffer = 0;
    private final int bufferMax;
    private final ReentrantLock prodLock = new ReentrantLock(true);
    private final ReentrantLock consLock = new ReentrantLock(true);
    private final ReentrantLock lock = new ReentrantLock();
    private final Condition prod = lock.newCondition();
    private final Condition cons = lock.newCondition();


    public NestedLocks(int bufferMax) {
        this.bufferMax = bufferMax;
    }

    public void produce(int amount) {
        prodLock.lock();
        lock.lock();
        try {
            while (buffer + amount >= bufferMax)
                prod.await();
            buffer += amount;
            cons.signal();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
            prodLock.unlock();
        }
    }

    public void consume(int amount) {
        consLock.lock();
        lock.lock();
        try {
            while (buffer - amount < 0)
                cons.await();
            buffer -= amount;
            prod.signal();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
            consLock.unlock();
        }
    }
}
```

3.2 Wykresy





4 Wnioski

Zwiększony bufor zwiększa przepustowość zgodnie z oczekiwaniami.
Najszybszym rozwiązaniem okazały się zagnieżdżone blokady, wynika to z tego, że blokady

to szybszy mechanizm niż warunki.