Towards performance tuning mindset

Presenter Viorel Chelaru

Senior Java Developer @ METRO Systems

About me

- Viorel Chelaru
  - Mathematics & Computer Science @ UGAL
  - Master in Artificial Intelligence @ PUB
  - Java developer for 15 years
  - Challenging projects keep me alive

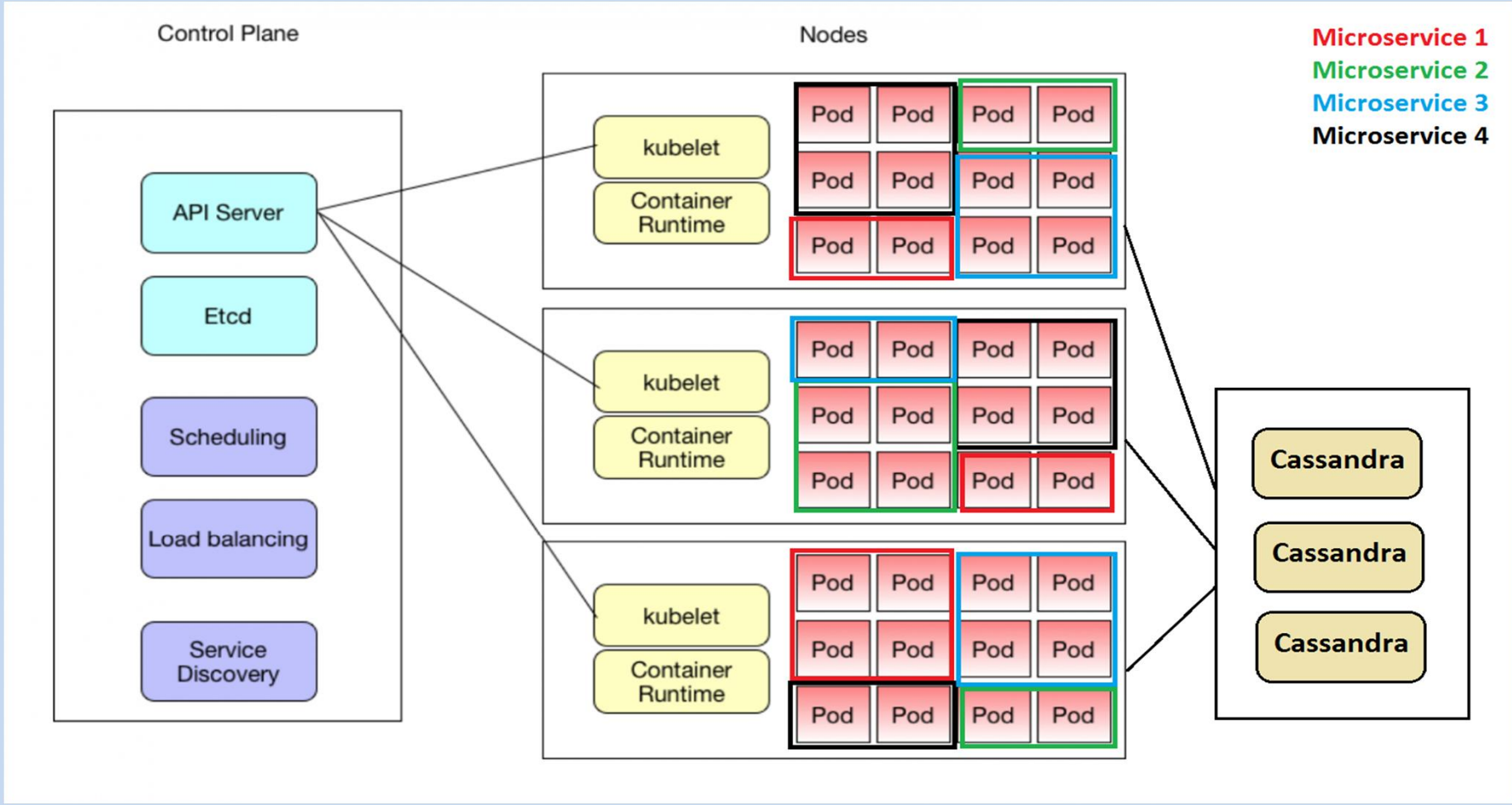
Do we really need fast Java code ?

- I never heard a client say:
  - I am happy the 100 lines CSV upload takes 30 minutes.
- Black Friday : websites slow, until they go down
- Cashier desk in store: card payment takes 10 minutes
-

Do we really need fast Java code ?

- I never heard a client say:
  - I am happy the 100 lines CSV upload takes 30 minutes.
- Black Friday : websites slow, until they go down
- Cashier desk in store: card payment takes 10 minutes
- Software performance affects our lives

ODIN project @ METRO



- Task processing system
- REST uService in SpringBoot
- Deployed in Docker on Kubernetes

- 14 uServices
- ~200 pods on 15 nodes
- 2-80 pods per uService

Use case : Big file upload

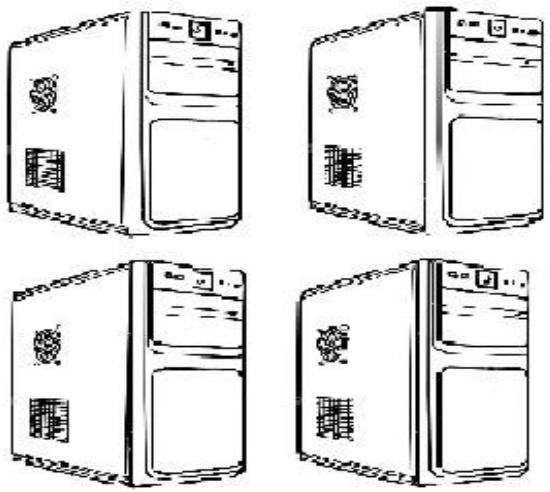
I need to upload
1,000,000 lines CSVs
.....every day.....

```
1 Article,Variant,Bundle,Store,Size,Price,Manufacture date
2 Item_1,1,5305,12,7,0.57,2018-10-01 00:00:00,323
3 Item_2,1,9744,1,6,0.99,2018-10-01 00:00:00,415
4 Item_3,1,6213,5,4,1.41,2018-10-01 00:00:00,517
5 Item_4,1,6126,6,4,1.16,2018-10-01 00:00:00,612
6 Item_5,1,8887,17,4,0.56,2018-10-01 00:00:00,483
7 Item_6,1,8009,7,9,0.90,2018-10-01 00:00:00,516
8 Item_7,1,1847,6,1,1.26,2018-10-01 00:00:00,218
9 Item_8,1,6544,5,10,0.80,2018-10-01 00:00:00,195
10 Item_9,1,9687,17,6,0.77,2018-10-01 00:00:00,131
```

ODIN

Read all CSV lines
Insert them in Cassandra
Process each line

Notes:
Serve many simultaneous requests
Upload file as fast as possible
K8s redistributes pods on nodes



It’s easy to implement a file upload

```
Path path = Paths.get(DataUtil.DATA_FILEPATH);

try (Cluster cluster = DbUtil.getCluster(null);
        Session session = cluster.connect("od_inbound_dev");
        BufferedReader br = Files.newBufferedReader(path, Charset.forName("UTF-8"));) {

    PreparedStatement psInsert = session.prepare(CQL_INSERT);
    UUID fileId = UUIDs.timeBased();

    //skip header
    br.readLine();

    // read the file line by line, and insert into Cassandra
    String line = null;
    while ((line = br.readLine()) != null) {
        if (line.trim().isEmpty()) {
            continue;
        }

        String[] columns = line.split(",");
        String article = columns[0];
        int variant = Integer.parseInt(columns[1]);
        int bundle = Integer.parseInt(columns[2]);
        int store = Integer.parseInt(columns[3]);
        int size = Integer.parseInt(columns[4]);
        float price = Float.parseFloat(columns[5]);
        Date manufactureDate = sdf.parse(columns[6]);
        int valabilityDays = Integer.parseInt(columns[7]);

        session.execute(psInsert.bind(fileId, article, variant, bundle, store, size, price, manufactureDate, valabilityDays));
    }

} catch (Exception e) {
    e.printStackTrace();
    throw e;
}
```

read CSV file
iterate all lines

parse data line

bind data in PreparedStatement
execute PreparedStatement

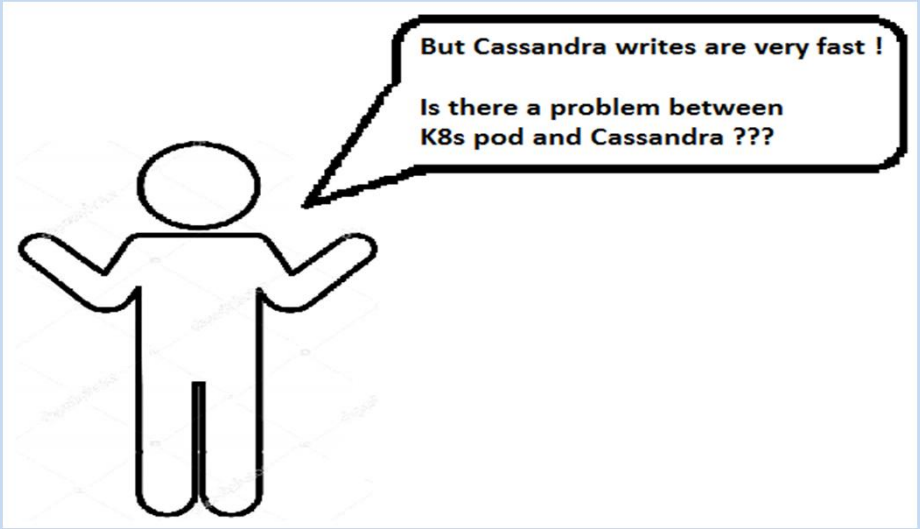
It's easy ??? to implement a file upload

```
Path path = Paths.get(DataUtil.DATA_FILEPATH);

try (Cluster cluster = DbUtil.getCluster(null);
        Session session = cluster.connect("od_inbound_dev");
        BufferedReader br = Files.newBufferedReader(path, Charset.forName("UTF-8"));) {

    PreparedStatement psInsert = session.prepare(CQL_INSERT);
    UUID fileId = UUIDs.timeBased();

    //skip header
    br.readLine();

    // read the file line by line, and insert into Cassandra
    String line = null;
    while ((line = br.readLine()) != null) {
        if (line.trim().isEmpty()) {
            continue;
        }

        String[] columns = line.split(",");
        String article = columns[0];
        int variant = Integer.parseInt(columns[1]);
        int bundle = Integer.parseInt(columns[2]);
        int store = Integer.parseInt(columns[3]);
        int size = Integer.parseInt(columns[4]);
        float price = Float.parseFloat(columns[5]);
        Date manufactureDate = sdf.parse(columns[6]);
        int valabilityDays = Integer.parseInt(columns[7]);

        session.execute(psInsert.bind(fileId, article, variant, bundle, store, size, price, manufactureDate, valabilityDays));
    }

} catch (Exception e) {
    e.printStackTrace();
    throw e;
}
```

- Runs in ~22 hours
- 30Mb memory footprint

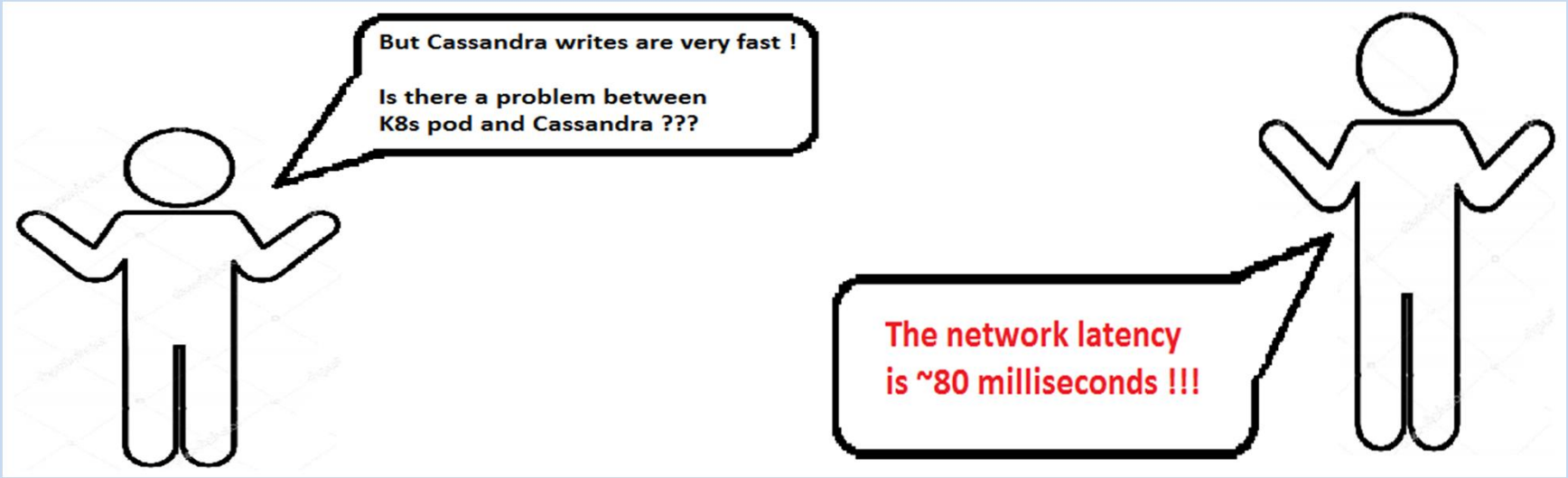
Profiling to find the cause

- 60.9% - 2,793 ms - 1 inv. net.imworld.performance_tunning.simple.FileImportSimple.main
  - 60.9% - 2,793 ms - 1 inv. net.imworld.performance_tunning.simple.FileImportSimple.fileImportSimple
    - 48.5% - 2,223 ms - 5,625 inv. com.datastax.driver.core.Session.execute
    - 3.3% - 149 ms - 5,625 inv. com.datastax.driver.core.PreparedStatement.bind
    - 3.1% - 141 ms - 5,625 inv. java.text.SimpleDateFormat.parse
    - 1.1% - 52,669 µs - 5,625 inv. java.lang.String.split
    - 0.8% - 34,755 µs - 5,625 inv. java.io.BufferedReader.readLine
    - 0.4% - 20,434 µs - 5,625 inv. java.lang.Float.parseFloat
    - 0.3% - 15,257 µs - 28,125 inv. java.lang.Integer.parseInt
    - 0.3% - 12,167 µs - 28,125 inv. java.lang.Integer.valueOf
    - 0.1% - 6,314 µs - 5,625 inv. java.lang.Float.valueOf
    - 0.1% - 6,068 µs - 5,625 inv. java.lang.String.isEmpty
    - 0.1% - 5,935 µs - 5,625 inv. java.lang.String.trim
- 39.1% - 1,793 ms - 3 inv. io.netty.util.concurrent.DefaultThreadFactory$DefaultRunnableDecorator.run



Profiling to find the cause

- 60.9% - 2,793 ms - 1 inv. net.imworld.performance_tunning.simple.FileImportSimple.main
  - 60.9% - 2,793 ms - 1 inv. net.imworld.performance_tunning.simple.FileImportSimple.fileImportSimple
    - 48.5% - 2,223 ms - 5,625 inv. com.datastax.driver.core.Session.execute
    - 3.3% - 149 ms - 5,625 inv. com.datastax.driver.core.PreparedStatement.bind
    - 3.1% - 141 ms - 5,625 inv. java.text.SimpleDateFormat.parse
    - 1.1% - 52,669 µs - 5,625 inv. java.lang.String.split
    - 0.8% - 34,755 µs - 5,625 inv. java.io.BufferedReader.readLine
    - 0.4% - 20,434 µs - 5,625 inv. java.lang.Float.parseFloat
    - 0.3% - 15,257 µs - 28,125 inv. java.lang.Integer.parseInt
    - 0.3% - 12,167 µs - 28,125 inv. java.lang.Integer.valueOf
    - 0.1% - 6,314 µs - 5,625 inv. java.lang.Float.valueOf
    - 0.1% - 6,068 µs - 5,625 inv. java.lang.String.isEmpty
    - 0.1% - 5,935 µs - 5,625 inv. java.lang.String.trim
- 39.1% - 1,793 ms - 3 inv. io.netty.util.concurrent.DefaultThreadFactory$DefaultRunnableDecorator.run



Multi-threading is faster !

```java
// read the file line by line, and insert into Cassandra
String line = null;
int count = 0;
List<String> linesBuffer = new LinkedList<>();
while ((line = br.readLine()) != null) {
    if (line.trim().isEmpty()) {
        continue;
    }

    count++;
    linesBuffer.add(line);

    if (count % linesPerThread == 0) {
        futures.add(processLinesBuffer(linesBuffer, fileId));
        linesBuffer.clear();
    }
}

if (!linesBuffer.isEmpty()) {
    futures.add(processLinesBuffer(linesBuffer, fileId));
}

//wait for all tasks to finish
for (Future<?> future : futures) {
    future.get();
}
```

```java
private Future<?> processLinesBuffer(List<String> linesBuffer, UUID fileId) {
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    List<String> threadLinesBuffer = new LinkedList<>();
    threadLinesBuffer.addAll(linesBuffer);
    Runnable r = new Runnable() {

        @Override
        public void run() {

            for (String line : threadLinesBuffer) {
                String[] columns = line.split(",");
                String article = columns[0];
                int variant = Integer.parseInt(columns[1]);
                int bundle = Integer.parseInt(columns[2]);
                int store = Integer.parseInt(columns[3]);
                int size = Integer.parseInt(columns[4]);
                float price = Float.parseFloat(columns[5]);
                Date manufactureDate;
                try {
                    manufactureDate = sdf.parse(columns[6]);
                } catch (ParseException e) {
                    throw new RuntimeException(e);
                }
                int valabilityDays = Integer.parseInt(columns[7]);

                session.execute(psInsert.bind(fileId, article, variant,
                    bundle, store, size, price, manufactureDate, valabilityDays));
            }
        }
    };
    return taskExecutor.submit(r);
}
```

Multi-threading is faster !

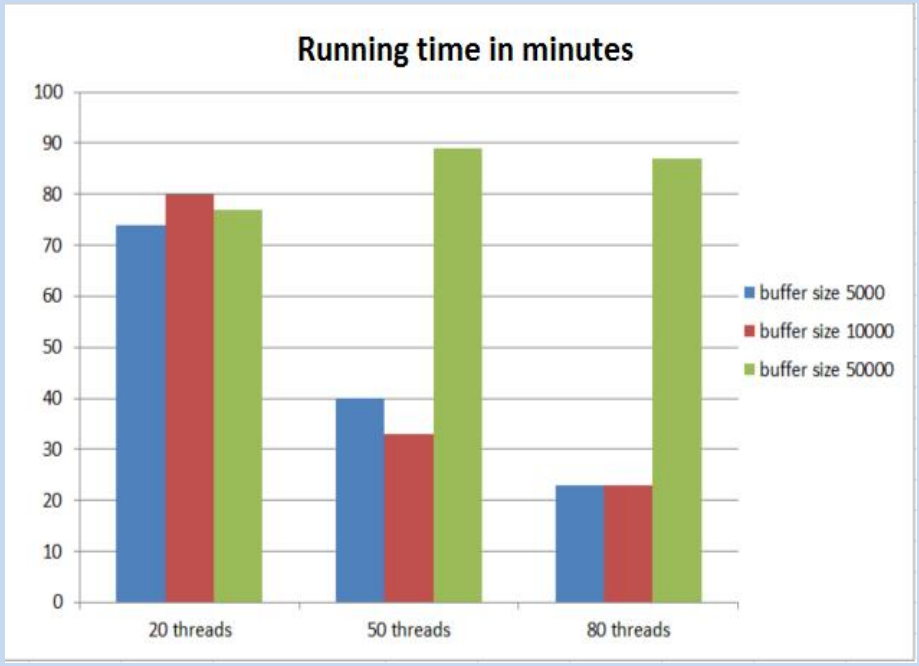
```java
// read the file line by line, and insert into Cassandra
String line = null;
int count = 0;
List<String> linesBuffer = new LinkedList<>();
while ((line = br.readLine()) != null) {
    if (line.trim().isEmpty()) {
        continue;
    }

    count++;
    linesBuffer.add(line);

    if (count % linesPerThread == 0) {
        futures.add(processLinesBuffer(linesBuffer, fileId));
        linesBuffer.clear();
    }
}

if (!linesBuffer.isEmpty()) {
    futures.add(processLinesBuffer(linesBuffer, fileId));
}

//wait for all tasks to finish
for (Future<?> future : futures) {
    future.get();
}
```

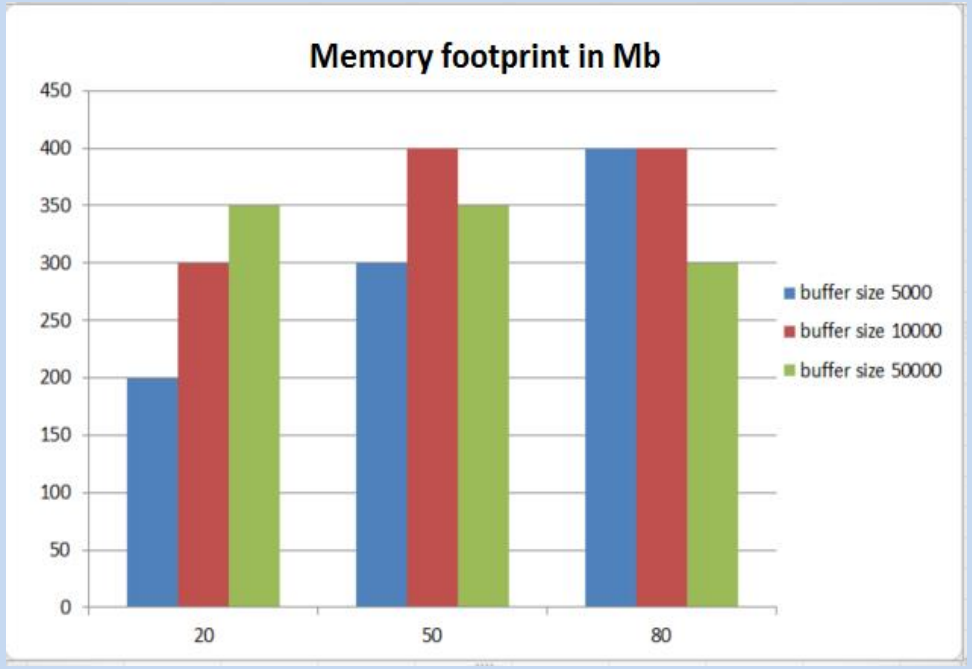
```java
private Future<?> processLinesBuffer(List<String> linesBuffer, UUID fileId) {
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    List<String> threadLinesBuffer = new LinkedList<>();
    threadLinesBuffer.addAll(linesBuffer);
    Runnable r = new Runnable() {

        @Override
        public void run() {

            for (String line : threadLinesBuffer) {
                String[] columns = line.split(",");
                String article = columns[0];
                int variant = Integer.parseInt(columns[1]);
                int bundle = Integer.parseInt(columns[2]);
                int store = Integer.parseInt(columns[3]);
                int size = Integer.parseInt(columns[4]);
                float price = Float.parseFloat(columns[5]);
                Date manufactureDate;
                try {
                    manufactureDate = sdf.parse(columns[6]);
                } catch (ParseException e) {
                    throw new RuntimeException(e);
                }
                int valabilityDays = Integer.parseInt(columns[7]);

                session.execute(psInsert.bind(fileId, article, variant,
                    bundle, store, size, price, manufactureDate, valabilityDays));
            }
        }
    };
    return taskExecutor.submit(r);
}
```

2 parameters to adjust

- #lines per thread
- #threads in the pool

Multi-threading results





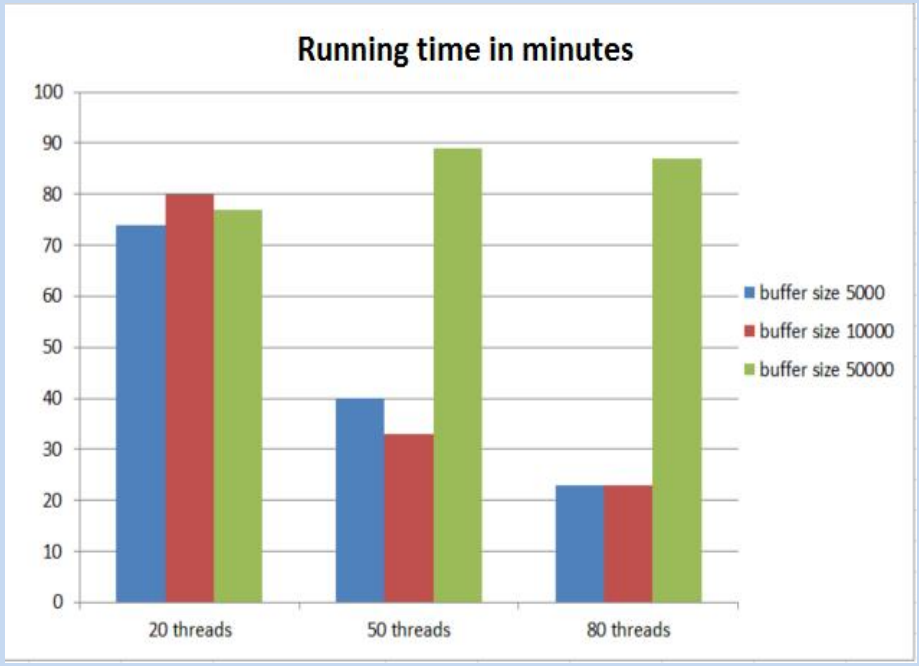
Multi-threading solution:

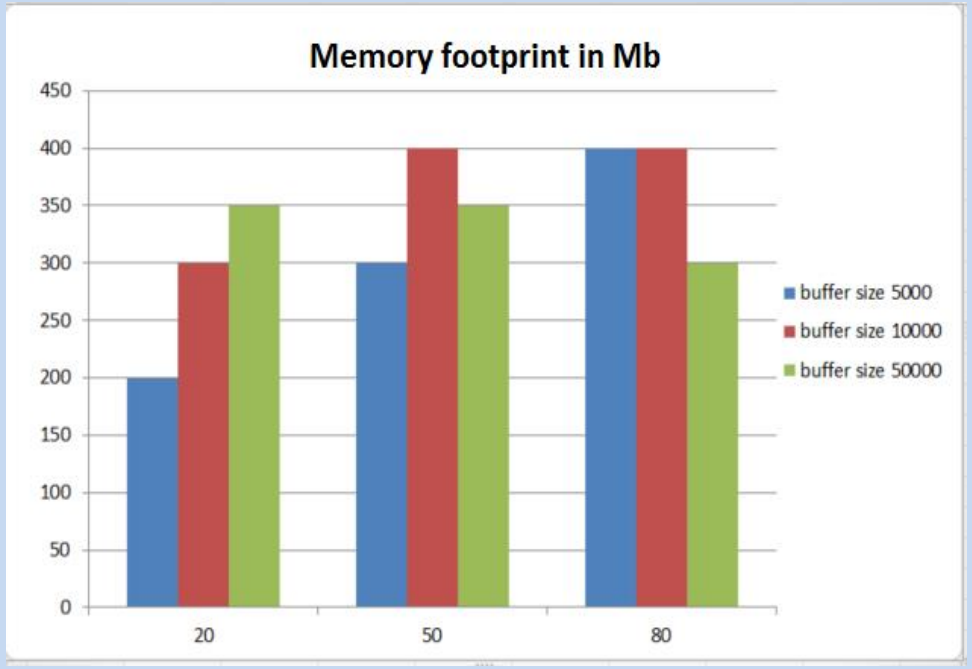
- read 1,000,000 lines with a pool of 80 threads, each thread inserting 10,000 lines
- Running time 23 mins
- Memory footprint 400 Mb

Using Cassandra SimpleStatement (CQL concat), not binding to a PreparedStatement

- Running time 18 mins

Multi-threading results





Multi-threading solution:
- read 1,000,000 lines with a pool of 80 threads, each thread inserting 10,000 lines
- Running time 23 mins
- Memory footprint 400 Mb

Using Cassandra SimpleStatement (CQL concat), not binding to a PreparedStatement
- Running time 18 mins

Still NOT HAPPY ! 18 minutes is too much !
1 pod has 450Mb required memory !

Inserting with Cassandra batches

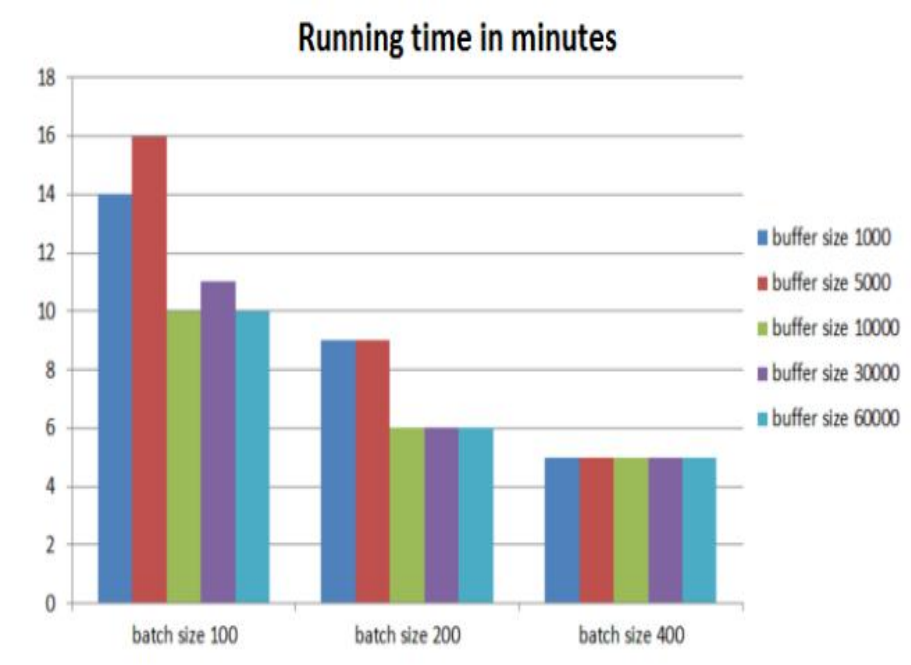
```
// read the file line by line, and insert into Cassandra
String line = null;
List<String> linesBuffer = new ArrayList<>();
while ((line = br.readLine()) != null) {
    if (line.trim().isEmpty()) {
        continue;
    }

    linesBuffer.add(line);

    if (linesBuffer.size() == linesBufferSize) {
        insertLinesBuffer(fileId, linesBuffer, batchSize);
        linesBuffer.clear();
    }
}

if (!linesBuffer.isEmpty()) {
    insertLinesBuffer(fileId, linesBuffer, batchSize);
}
```

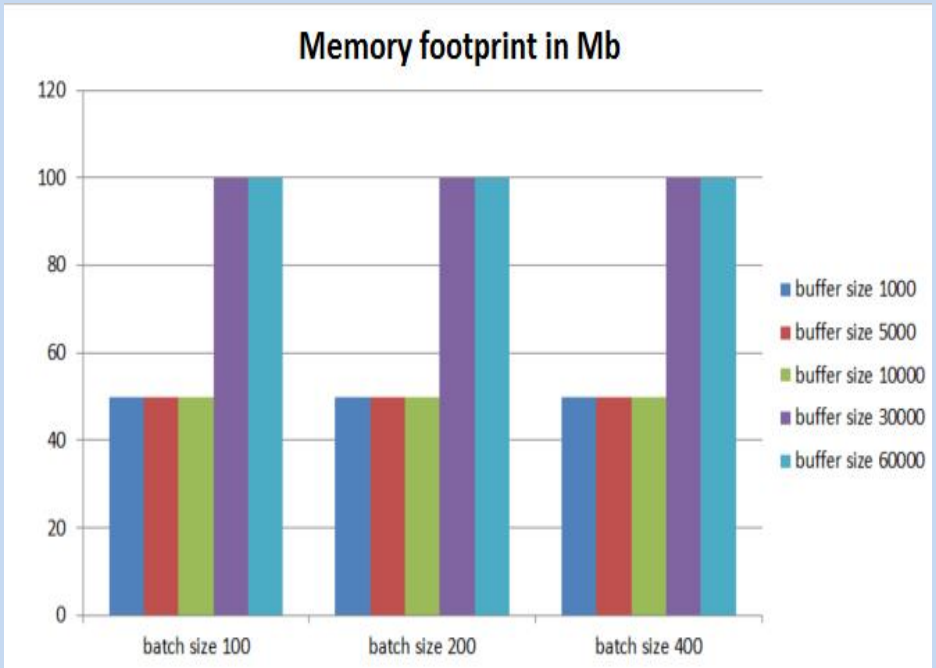
```
private void insertLinesBuffer(UUID fileId, List<String> linesBuffer, int batchSize)
        throws ParseException {
    //logger.info("insert buffer of "+linesBuffer.size()+" lines");

    BatchStatement batch = new BatchStatement();
    for (String line : linesBuffer) {
        String[] columns = line.split(",");
        String article = columns[0];

        int variant = Integer.parseInt(columns[1]);
        int bundle = Integer.parseInt(columns[2]);
        int store = Integer.parseInt(columns[3]);
        int size = Integer.parseInt(columns[4]);
        float price = Float.parseFloat(columns[5]);
        Date manufactureDate = sdf.parse(columns[6]);
        int valabilityDays = Integer.parseInt(columns[7]);

        batch.add(psInsert.bind(fileId, article, variant, bundle, store, size,
                price, manufactureDate, valabilityDays));

        if (batch.size() == batchSize) {
            session.execute(batch);
            batch.clear();
        }
    }
    if (batch.size() > 0) {
        session.execute(batch);
        batch.clear();
    }
}
```

Benefit:

send multiple statements in 1 network call

Inserting with Cassandra batches





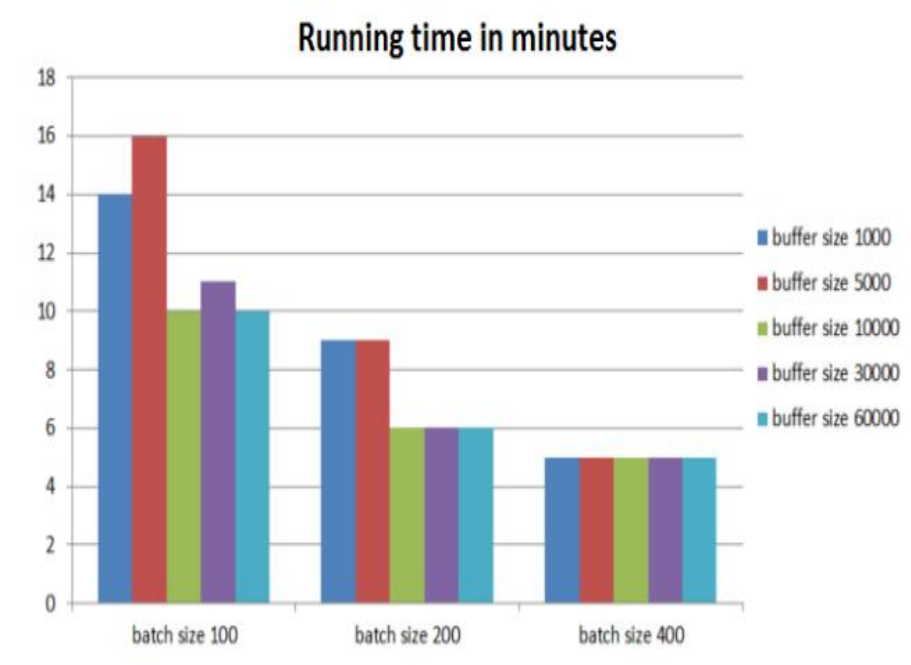
Cassandra Batches solution:

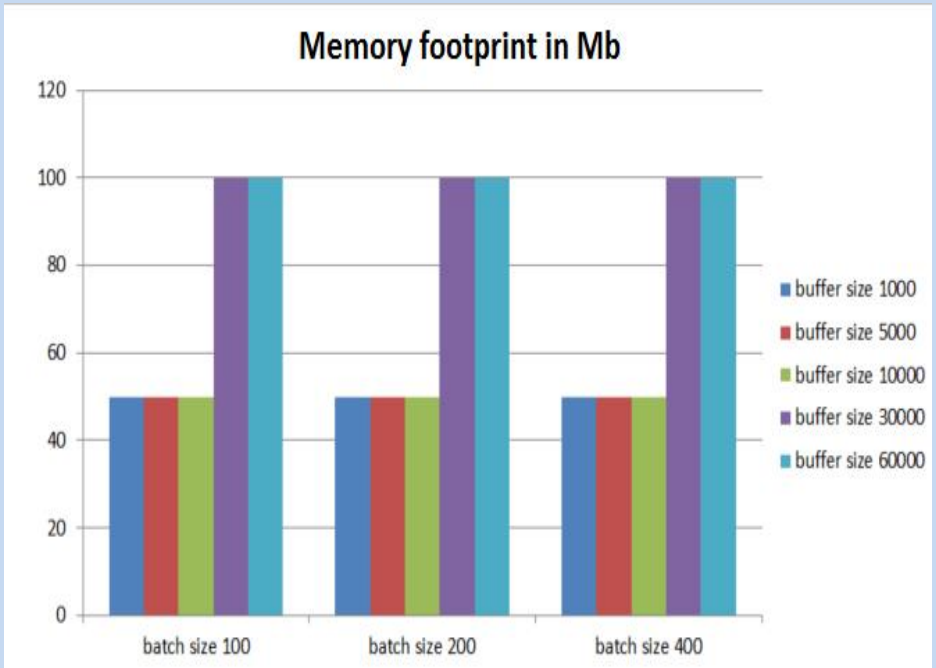
- read 1,000,000 lines in buffers of 1000 lines, insert buffers in baches of 400 inserts
- Running time 5 mins
- Memory footprint 50 Mb

Using Cassandra SimpleStatement (CQL concat), not binding to a PreparedStatement

- Running time 11 mins

Inserting with Cassandra batches





Cassandra Batches solution:
- read 1,000,000 lines in buffers of 1000 lines, insert buffers in batches of 400 inserts
- Running time 5 mins
- Memory footprint 50 Mb

Using Cassandra SimpleStatement (CQL concat), not binding to a PreparedStatement
- Running time 11 mins

We're getting somewhere !
5 minutes is pretty good, but can I do better ?

Does Garbage Collection count ?

Cassandra Batches solution:

- read 1,000,000 lines in buffers of 1000 lines
- insert buffers in batches of 400 inserts
- Use G1 garbage collector
- Running time 4 mins
- Memory footprint 50 – 80 Mb

Does Garbage Collection count ?

Cassandra Batches solution:

- read 1,000,000 lines in buffers of 1000 lines
- insert buffers in batches of 400 inserts
- Use G1 garbage collector
- Running time 4 mins
- Memory footprint 50 – 80 Mb

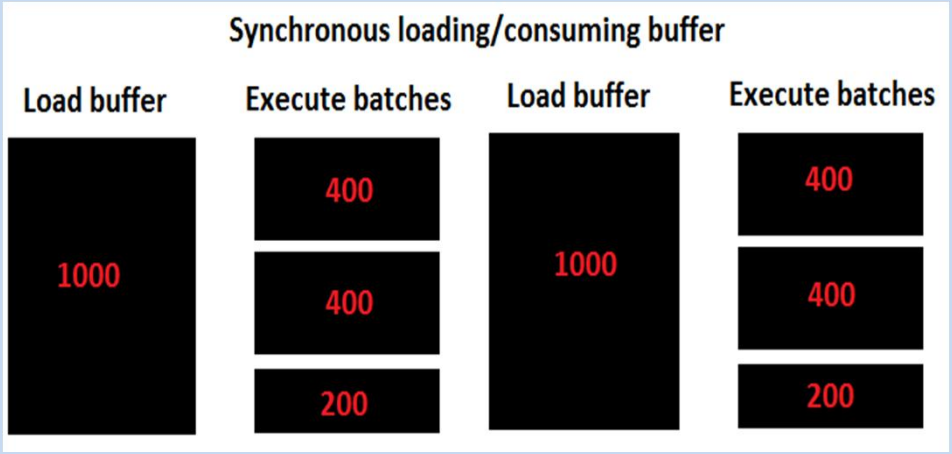
Wowww, I just squized 1 MINUTE !
4 minutes is niceee….
I can see the end of the tunnel !

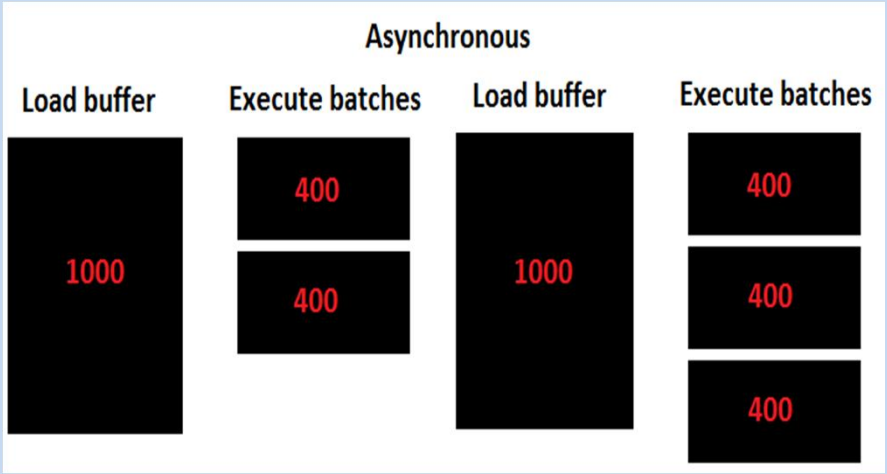
3000 batches ? Should be 2500 batches !

Until now
1,000,000 lines read in buffers of 1000 lines
Insert each buffer in 400 inserts batches
1000 lines
1 buffer => 3 batches (400,400,200)
1,000,000 lines
1000 buffers * 3 batches / buffer = 3000 batches

But
1,000,000 lines / 400 inserts = 2500 batches !!!!

Does it worth optimizing ?
500 batches * 80ms latency / batch = 40 seconds





Final solution

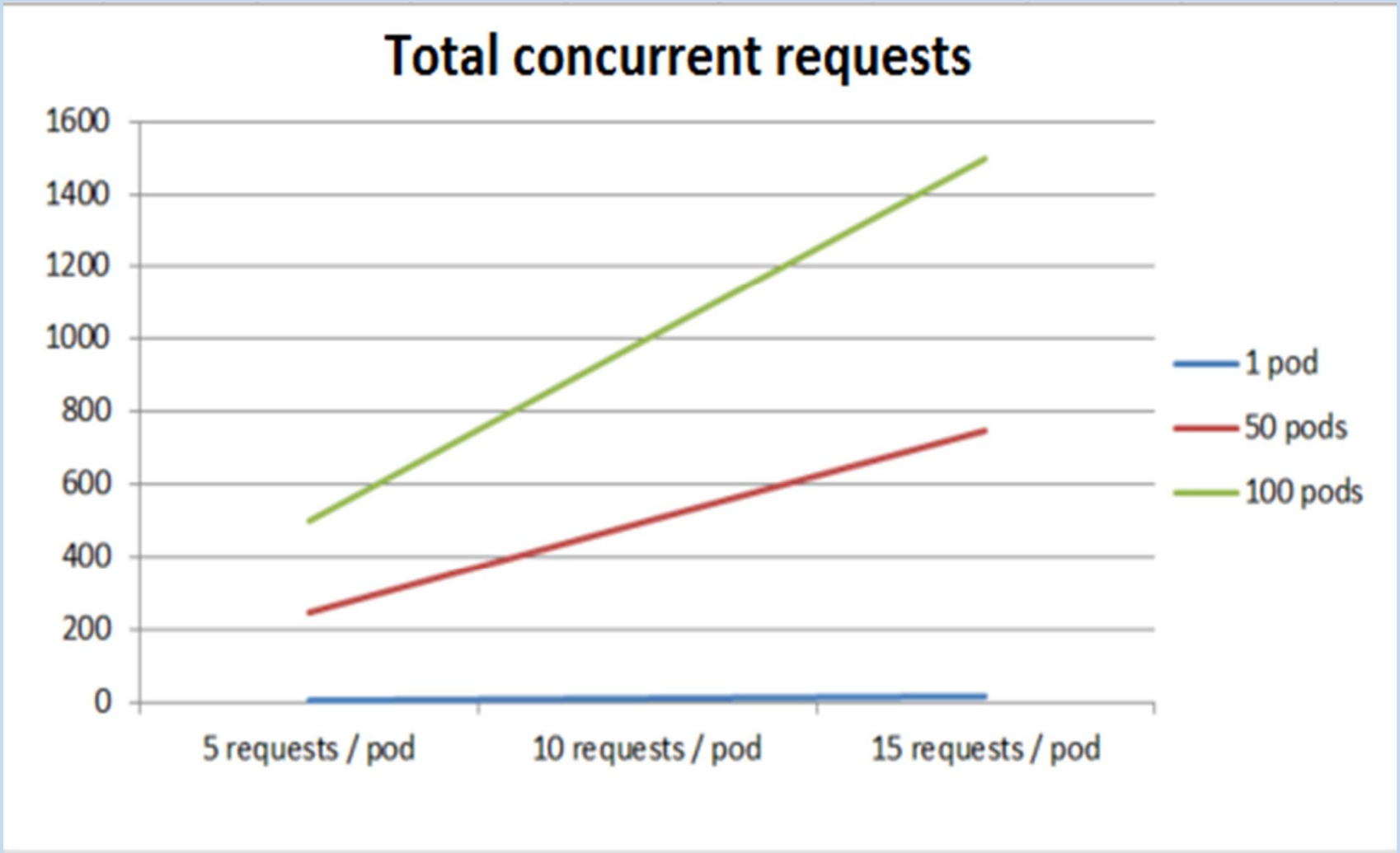
- Use Cassandra batches (single partition)
- read 1,000,000 lines in buffers of 1000 lines
- insert buffers in batches of 400 inserts
- Decouple loading buffer and executing insert batches
- Use G1 garbage collector

Running time 3 mins
Memory footprint 50 – 80 Mb

From 22 hours to 3 minutes ??? What a journey !
We improve 440 times !!

I can go like this “forever” **J**

Performance & Scalability



I don’t have to be a genius to fine tune performance

I am restless, ambitious , skeptical, always unsatisfied !

I always keep performance in mind when I code.

I HAVE time for performance tuning, if I prioritize my work.

I use my analytical thinking, not my typing skills.

THINK !

IT AIN’T ILLEGAL …. YET

Eddie Griffin, The comedian

Thank You !

- Questions ?

- Code is on GitHub, look me up