Rapport between Static Load Balancing and Dynamic Load Balancing to calculate the number of prime number.

This experience was realized by QUIEF Hippolyte *(50171350) using a computer with a processor Intel i7 2.7GHz and 8Go of RAM.*

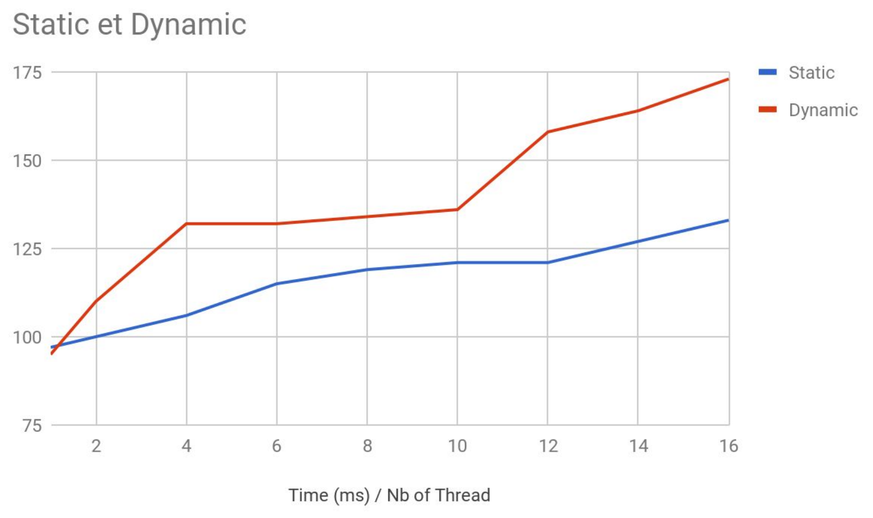
This experience have the goal to demonstrate the difference between a Static Load Balancing approach and a Dynamic Load Balancing approach. A short and small program in JAVA was created, this program computes the total of prime numbers between 1 and 200000, which is 17984.

To realize this program in a dynamic approach, we launch *x* threads who will compute with a common variable. This variable is *synchronize*, meaning is lock, when another thread try to access it. This must be done to prevent to thread accessing the same data and read/write in the same time. In this program, the common variable is the current number a thread is computing. The goal of this approach is to give to every thread an equivalent amount of work.

The static approach will launch *x* thread and give each of them an interval of data. The thread doesn't share its data to anyone. He will compute it in it's own way. The goal of this approach is to give to every thread an equivalent amount of data.

Time (ms) / Nb of Thread	**1**	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**
Static	97	100	106	115	119	121	121	127	133
Dynamic	95	110	132	132	134	136	158	164	173

1. Result of the 2 approach with 1, 2, 4, 6, 8, 10, 12, 14 and 16 threads

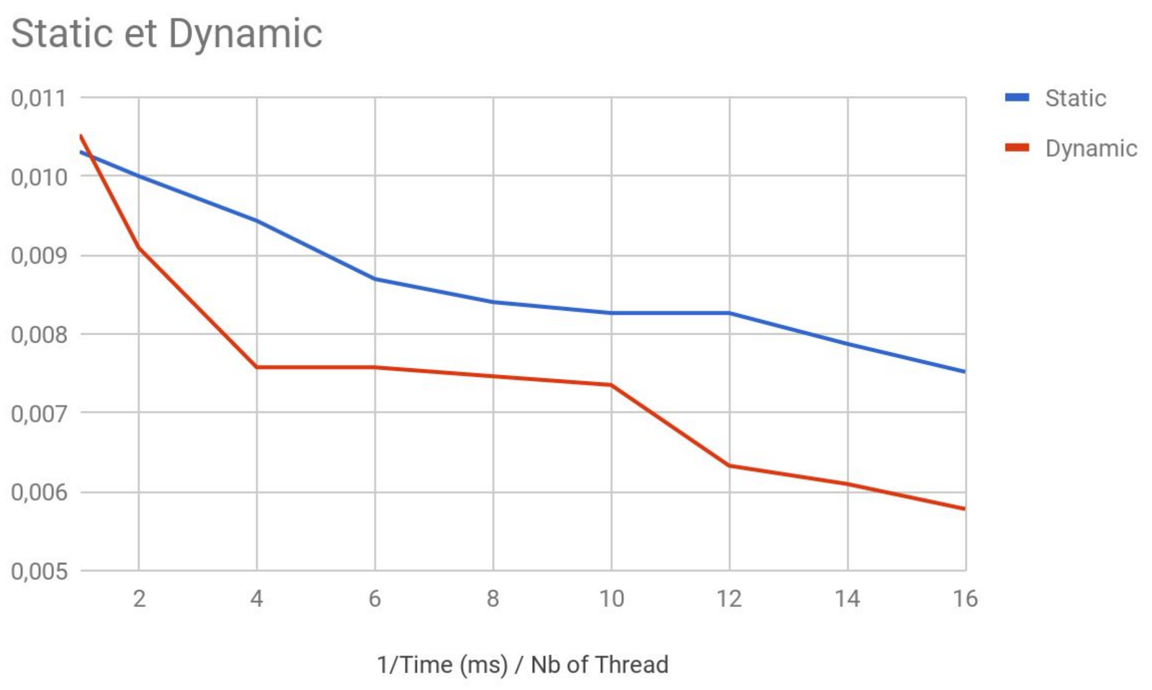


2. The graph created with the data of the array 1

These array and graph show a big difference between the 2 approaches. Both are them are influence by the number of thread, more we add some, more it takes time for this exemple.

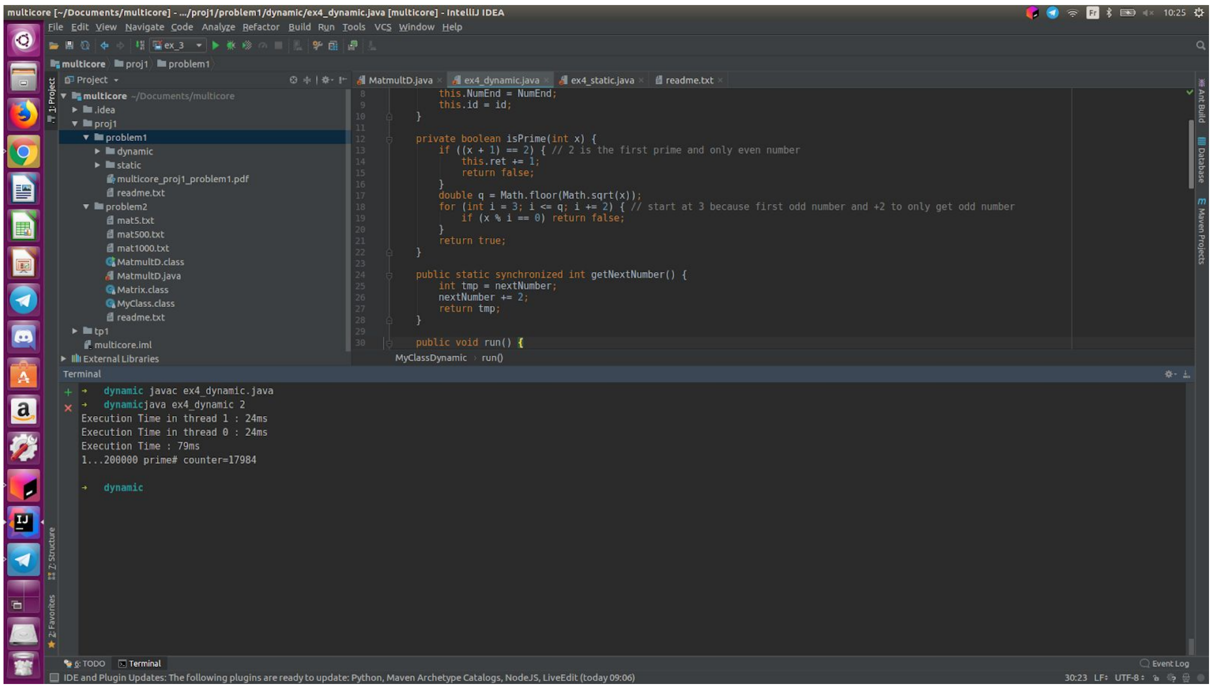
1/Time (ms) / Nb of Thread	**1**	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**
Static	0,010	0,010	0,009	0,009	0,008	0,008	0,008	0,008	0,008
Dynamic	0,011	0,009	0,008	0,008	0,007	0,007	0,006	0,006	0,006

3. Result of the performance by the 2 approach



4. Graph of the performance by the 2 approach

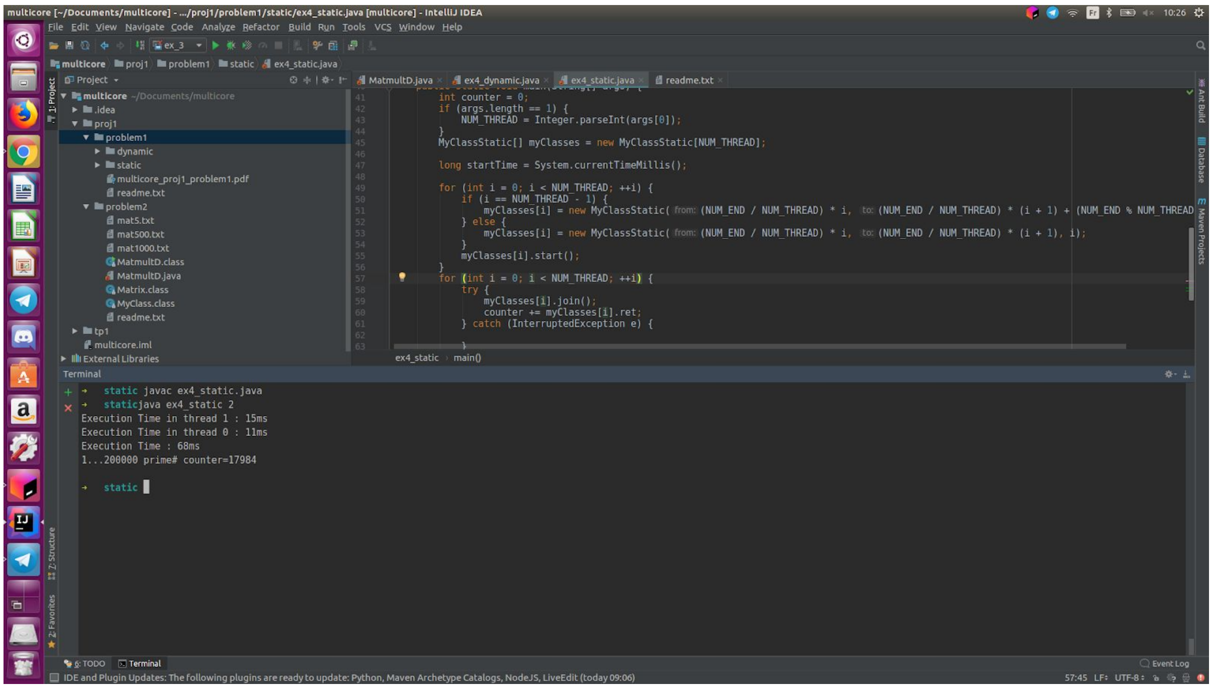
The reason which might be under the result of the dynamic approach is the *synchronization* of the data. Because lot of thread want to access to the data in the same time, it might be because of the computation is too fast and a queue to access the data is created. In the other hand, the time for the static approach depend on the most difficult part of data a thread must compute. Here we can say that the computation is too fast and there is no need to add thread to this program with so few data to compute.



```java
        this.NumEnd = NumEnd;
        this.id = id;
    }

    private boolean isPrime(int x) {
        if ((x + 1) == 2) { // 2 is the first prime and only even number
            this.ret += 1;
            return false;
        }
        double q = Math.floor(Math.sqrt(x));
        for (int i = 3; i <= q; i += 2) { // start at 3 because first odd number and +2 to only get odd number
            if (x % i == 0) return false;
        }
        return true;
    }

    public static synchronized int getNextNumber() {
        int tmp = nextNumber;
        nextNumber += 2;
        return tmp;
    }

    public void run() {
```

```
→  dynamic javac ex4_dynamic.java
→  dynamicjava ex4_dynamic 2
Execution Time in thread 1 : 24ms
Execution Time in thread 0 : 24ms
Execution Time : 79ms
1...200000 prime# counter=17984

→  dynamic
```

5. Execution of Dynamic Load Approach program with 2 thread



```java
        int counter = 0;
        if (args.length == 1) {
            NUM_THREAD = Integer.parseInt(args[0]);
        }
        MyClassStatic[] myClasses = new MyClassStatic[NUM_THREAD];

        long startTime = System.currentTimeMillis();

        for (int i = 0; i < NUM_THREAD; ++i) {
            if (i == NUM_THREAD - 1) {
                myClasses[i] = new MyClassStatic( from: (NUM_END / NUM_THREAD) * i,  to: (NUM_END / NUM_THREAD) * (i + 1) + (NUM_END % NUM_THREAD
            } else {
                myClasses[i] = new MyClassStatic( from: (NUM_END / NUM_THREAD) * i,  to: (NUM_END / NUM_THREAD) * (i + 1), i);
            }
            myClasses[i].start();
        }
        for (int i = 0; i < NUM_THREAD; ++i) {
            try {
                myClasses[i].join();
                counter += myClasses[i].ret;
            } catch (InterruptedException e) {

            }
```

```
→  static javac ex4_static.java
→  staticjava ex4_static 2
Execution Time in thread 1 : 15ms
Execution Time in thread 0 : 11ms
Execution Time : 68ms
1...200000 prime# counter=17984

→  static
```

6. Execution of Static Load Approach program with 2 thread

Code of the program

Dynamic

```
class MyClassDynamic extends Thread {
    public static int nextNumber = 1; // start at 1 because of every prime number add odd (except 2)
    private int NumEnd;
    private int id;
    public int ret = 0;

    MyClassDynamic(int NumEnd, int id) {
        this.NumEnd = NumEnd;
        this.id = id;
    }

    private boolean isPrime(int x) {
        if ((x + 1) == 2) { // 2 is the first prime and only even number
            this.ret += 1;
            return false;
        }
        double q = Math.floor(Math.sqrt(x));
        for (int i = 3; i <= q; i += 2) { // start at 3 because first odd number and +2 to only get odd number
            if (x % i == 0) return false;
        }
        return true;
    }

    public static synchronized int getNextNumber() {
        int tmp = nextNumber;
        nextNumber += 2;
        return tmp;
    }

    public void run() {
        int currentNb = this.getNextNumber();

        long startTime = System.currentTimeMillis();
        while (currentNb < this.NumEnd) {
            if (this.isPrime(currentNb) == true) this.ret += 1;
            currentNb = this.getNextNumber();
        }
        long endTime = System.currentTimeMillis();
        long timeDiff = endTime - startTime;

        System.out.println("Execution Time in thread " + this.id + " : " + timeDiff + "ms");
    }
}

class ex4_dynamic {
    private static final int NUM_END = 200000;
    private static int NUM_THREAD = 4;

    public static void main(String[] args) {
        int counter = 0;
        if (args.length == 1) {
            NUM_THREAD = Integer.parseInt(args[0]);
        }
        MyClassDynamic[] myClasses = new MyClassDynamic[NUM_THREAD];

        long startTime = System.currentTimeMillis();
        for (int i = 0; i < NUM_THREAD; ++i) {
            myClasses[i] = new MyClassDynamic(NUM_END, i);
            myClasses[i].start();
        }
        for (int i = 0; i < NUM_THREAD; ++i) {
            try {
                myClasses[i].join();
                counter += myClasses[i].ret;
            } catch (InterruptedException e) {

            }
        }

        long endTime = System.currentTimeMillis();
        long timeDiff = endTime - startTime;

        System.out.println("Execution Time : " + timeDiff + "ms");
        System.out.println("1..." + NUM_END + " prime# counter=" + counter + "\n");
    }
}
```

Static

```
class MyClassStatic extends Thread {
    private int from;
    private int to;
    private int id;
    public int ret = 0;

    MyClassStatic(int from, int to, int id) {
        this.from = from;
        this.to = to;
        this.id = id;
    }

    private boolean isPrime(int x) {
        if (x % 2 == 0) return false; // If even number return false
        double q = Math.floor(Math.sqrt(x));

        for (int i = 3; i <= q; i += 2) { // start at 3 because first odd number and +2 to only get
odd number
            if ((x % i == 0) && (i != x)) return false;
        }
        return true;
    }

    public void run() {
        long startTime = System.currentTimeMillis();

        for (int i = this.from; i < this.to; i++) {
            if (this.isPrime(i) == true) this.ret += 1;
        }
        long endTime = System.currentTimeMillis();
        long timeDiff = endTime - startTime;

        System.out.println("Execution Time in thread " + this.id + " : " + timeDiff + "ms");
    }
}

class ex4_static {
    private static final int NUM_END = 200000;
    private static int NUM_THREAD = 4;

    public static void main(String[] args) {
        int counter = 0;
        if (args.length == 1) {
            NUM_THREAD = Integer.parseInt(args[0]);
        }
        MyClassStatic[] myClasses = new MyClassStatic[NUM_THREAD];

        long startTime = System.currentTimeMillis();

        for (int i = 0; i < NUM_THREAD; ++i) {
            if (i == NUM_THREAD - 1) {
                myClasses[i] = new MyClassStatic((NUM_END / NUM_THREAD) * i, (NUM_END /
NUM_THREAD) * (i + 1) + (NUM_END % NUM_THREAD), i);
            } else {
                myClasses[i] = new MyClassStatic((NUM_END / NUM_THREAD) * i, (NUM_END /
NUM_THREAD) * (i + 1), i);
            }
            myClasses[i].start();
        }
        for (int i = 0; i < NUM_THREAD; ++i) {
            try {
                myClasses[i].join();
                counter += myClasses[i].ret;
            } catch (InterruptedException e) {

            }
        }

        long endTime = System.currentTimeMillis();
        long timeDiff = endTime - startTime;

        System.out.println("Execution Time : " + timeDiff + "ms");
        System.out.println("1..." + NUM_END + " prime# counter=" + counter + "\n");
    }
}
```