Database - AS4

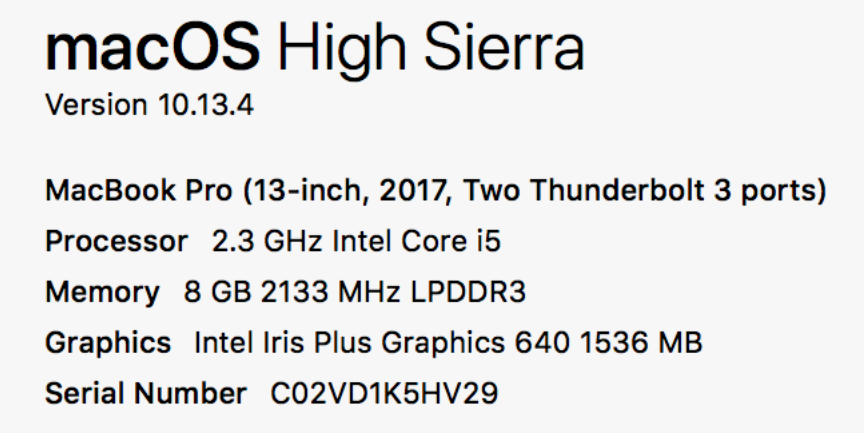
104062203 陳涵宇
104062232 廖子毅
104062234 林士軒

- How do we alter the source code to boost the performance ?

We modify five classes, which are “Buffer.java”, “BufferMgr.java”, “BufferPoolMgr.java”, “BlockID.java”, “FileMgr.java”. we will mention what we altered in a row as following descriptions.

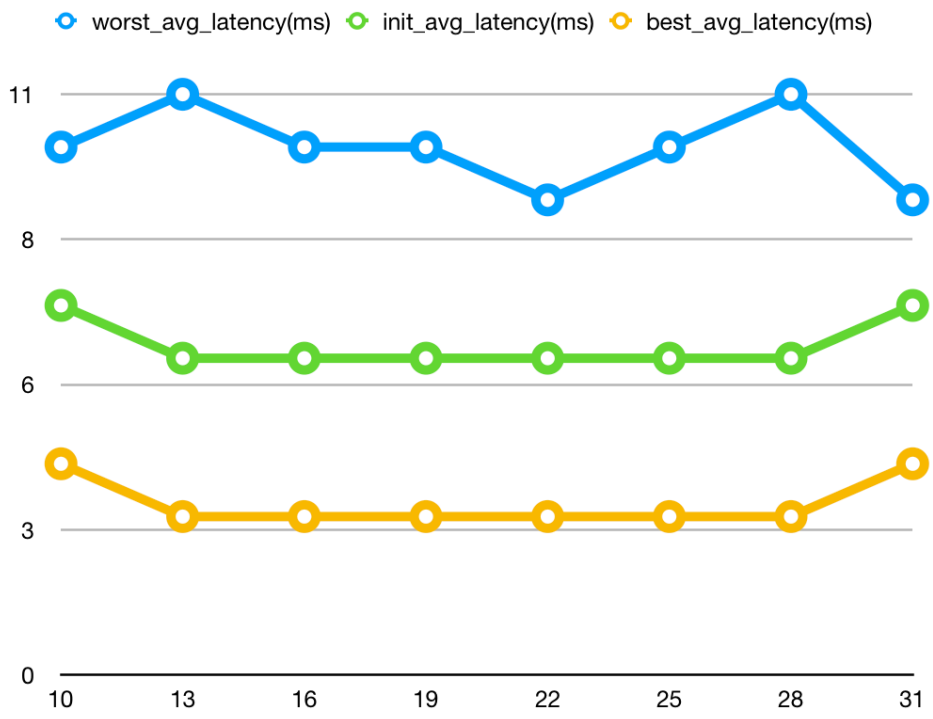
  - Buffer
    - We modify some sync functions in this class because we wanna to reduce the critical section size. That is, we reduce the latency time because we won’t waste lots of time to waiting resource even if we did not need it.
  - BufferMgr
    - We also use above methods to get the better performance. Hence, we modify sync function and reduce the size of critical section, too. It means that we find out some resources which need to be protected and just limit critical section on it.
  - BufferPoolMgr
    - Actually, we try many methods in this class. However, we adopt the previous strategy in the end. We had tried two method which were in the java.util.concurrent.atomic and locks, “AtomicInteger” and “Lock” to improve the performance, but we failed. We think although we reduce the size of the critical section, the increment time due to waiting shared resource decrease our performance. Thus, we gave up these two methods. Actually, its performance were worse than the initial one.
  - BlockID
    - In this class, we add two boolean vars to avoid executing hashCode and toString function again and again.
  - FileMgr
    - We use hashMap and ReentrantLock and add a new hashMap<String, ReentrantLock> (String for Filename) so that we can execute multiple files in the same time. However, our method won’t provide a file which can do different method synchronously.

- Experiments
  - Env.

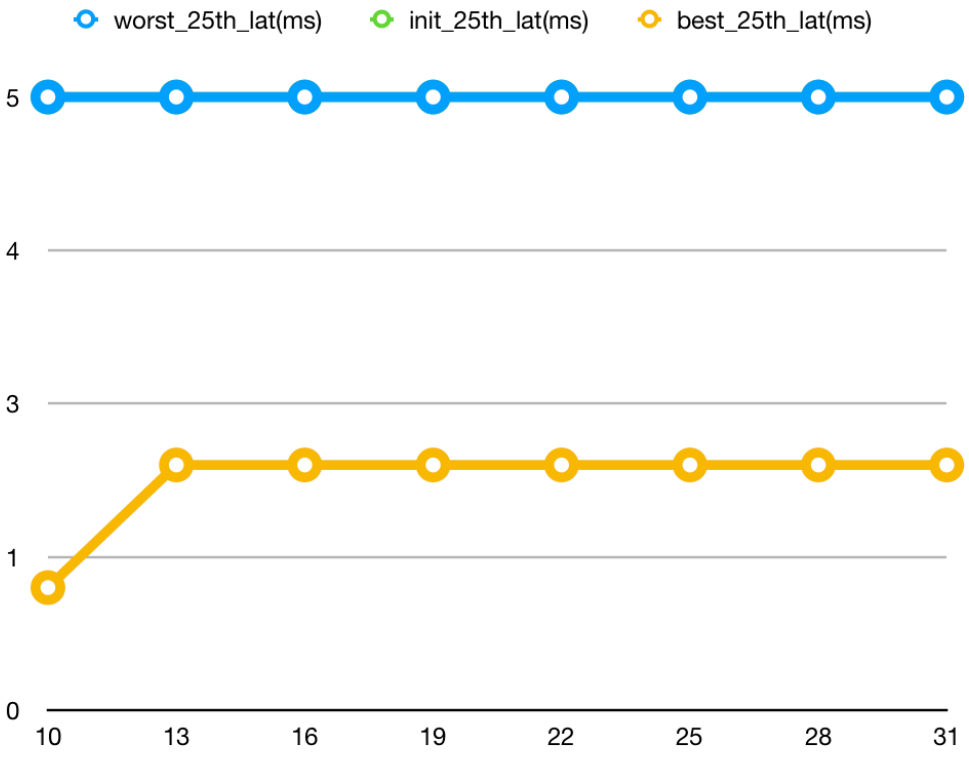


- Perfomance Comparison

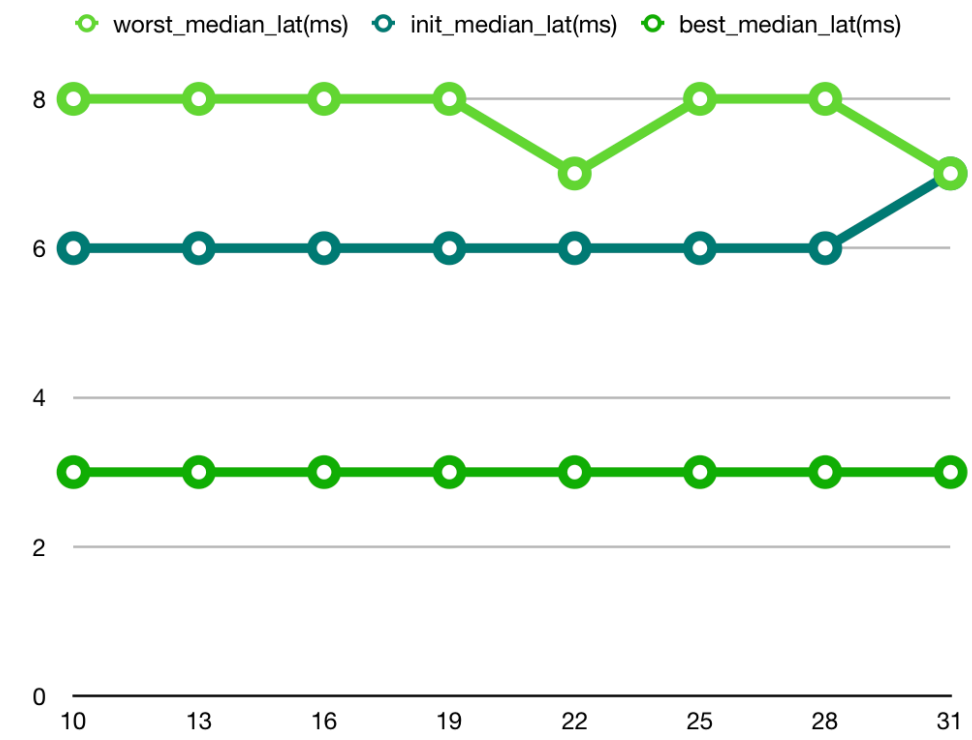
-AVG Latency(ms)



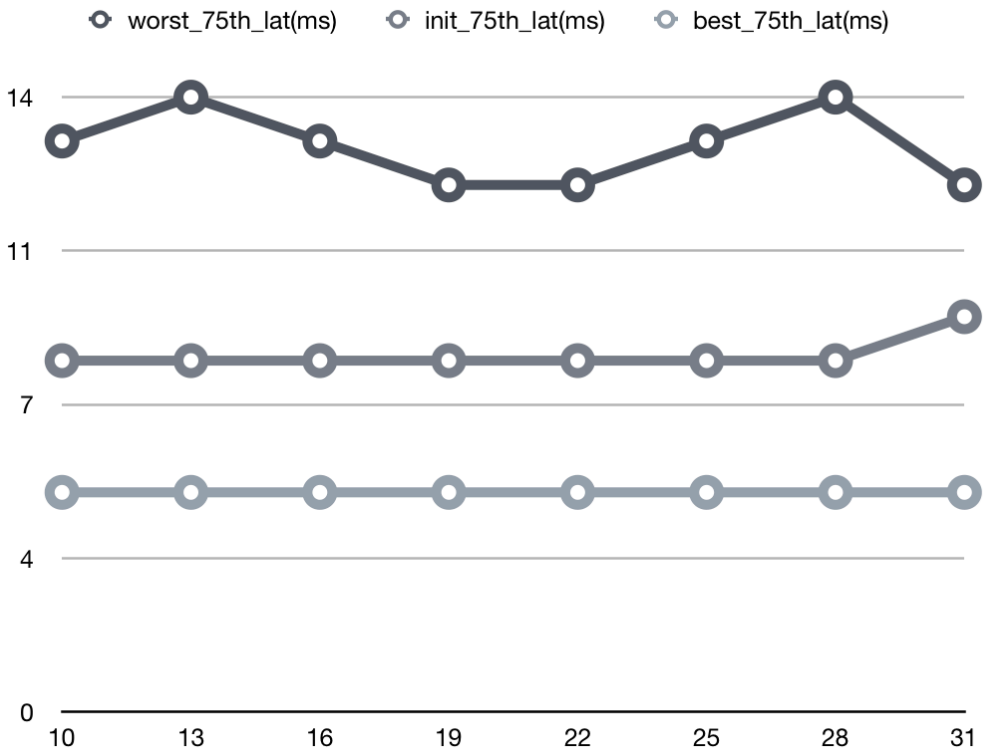
- 25th_lat



- median_lat



- 75th_lat



- Discuss and conclude why your optimization works

We implement this assignment by two methods, reducing the size of critical sections and the duplicated work because the initial source code will waste lots of time to wait for the same resources they don’t need or redo something which won’t change again and again. In the first method, we redefine the size of critical section and find out what resource are really need to be protected. In the other one, we use a flag to determine whether the var changed. By doing so, our code can outperform twice than the initial one.