Time Measurements part 1

Tomas Marx-Raacz von Hidvég

StudentId code: tendn09
Linnaeus University, 1dv507
tendn09.student@lnu.se

Abstract

This report is intended to investigate time measurement of two types of concatenations of strings that is used in the programming language of java. Our experiment uses the StringBuilder class and the standard "+" concatenation. We aim to investigate the difference between the two and also see to what extent these can be used when run for a full second.

Our experiment shows that the StringBuilder is far superior to the standard concatenation. With single character concatenations reaching almost as high as 94,5 million concatenations under 1 second. This is compared to the standard concatenation of strings reaching just over 115000.

Introduction

Among the first things a student learn in Java is concatenation of strings using the "+" operator. Later one gets introduced to Classes that do these types of things like StringBuffer and StringBuilder.

In this experiment we will compare standard concatenation using a "+" operator and the StringBuilder. First we aim to concatenate a string with just one character, in this case the letter "t".
Second test is going to be run using a string 80 characters long including white spaces " ".

The Experiment

The test setup for these runs is a heavily modified 5 year old Asus G751jy:
https://www.asus.com/us/ROG-Republic-Of-Gamers/ROG-G751JY/specifications/
It is equipped with Intel I7 4710HQ mobile 4 core @2,5ghz, 32GB of memory and equipped with SATA SSDs. It is running on Windows 10 Pro and we used Java 14 and changed the Java virtual machine with the following parameters to increase the heap size as well as the stack size:

-Xmx4096, -Xms16384, -Xss16m

We also closed all non system critical applications to reduce interference from the system as well as making sure to the best of our knowledge to make the application for running the test does not interfere with the actual test. In that respect we isolated the actual run of the test from the rest of the code and made sure to use garbagecollector before each run. The following code ran the test:

```java
public int measureTime() {

        StringBuilder builtString = new StringBuilder();

        this.runtime.gc();

        long before = System.currentTimeMillis();

        for (int i = 0; i < this.repetitions; i++){

            builtString.append(this.itemToAdd);}

        long after = System.currentTimeMillis();

        return (int) (after - before);}
```

And:

```java
public int measureTime() {

        this.runtime.gc();

        String builtString = "";
        long before = System.currentTimeMillis();
        for (int i = 0; i < this.repetitions; i++){
            builtString += this.itemToAdd;
        }
        long after = System.currentTimeMillis();
        return (int) (after - before);}
```

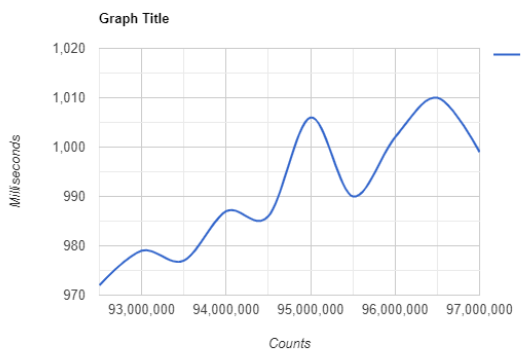
The above mentioned code is what the author considered to be the bare minimum to run per test.
It consists of a garbagecollection, an initiation of the `String` or the `Stringbuilder`, and the actual test using `System.currentTimeMillis()`.Each of these tests received their own class with various helper methods for random automating the test suites.

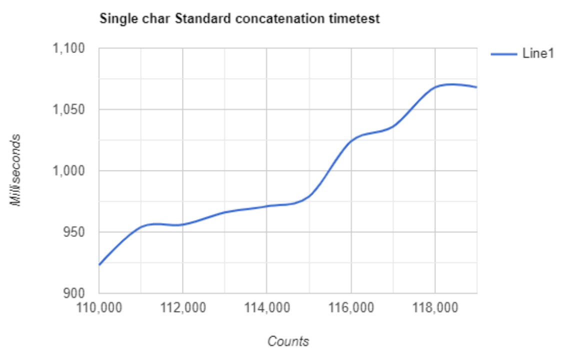
In all four test suites the experiment started of with a try-and-error search for a number of concatenations close to, but not over 1000 milliseconds. After this an interval was chosen from said initial test phase. These depended largely on the process of reaching the said number of concatenations. Each of these tests were run in 10 repetitions with an average calculated.
The numbers of concatenations found were:

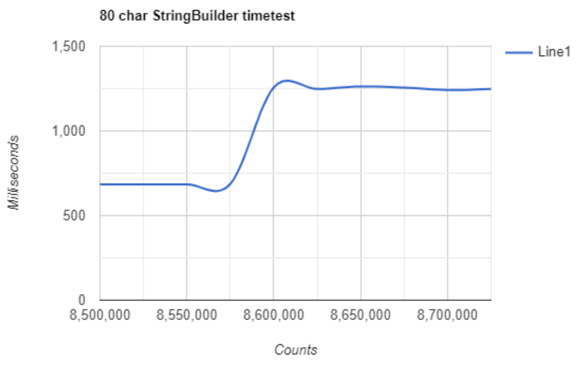
	StringBuilder single	StringBuilder large	String concat single	String concat large
Found repetitions	94500000	8600000	114000	12400
Start of interval	92500000	8500000	110000	12200
Increase per step	500000	25000	1000	50

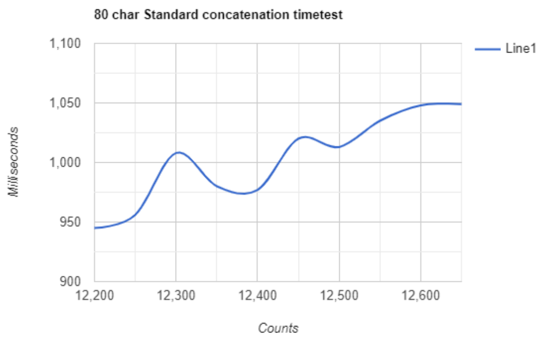
We can already see a pattern emerging, but further tests and results was required to try to spot the closest we could get to a second and what it yielded as a result.
The tests were as mentioned before done in 10 repetition of which an average was calculated.
Prior to the tests a warmup session of 50 repetitions of the "found repetitions" value was run after which 10 steps of 10 tests were conducted, starting at the interval start found in the table above and each consecutive with the increase counted in. Each average was then saved and plotted into a curve.

To begin with these fluctuations are very interesting, but we can clearly see an increasing curve in both. The initially found value in the try-and-error type investigation seems to be a good measure for the StringBuilder but the found value for the strings was slightly lower than what the real test provided. 94500000 for the StringBuilder and 115500 for the String concatentation.









On the large tests we found a very weird, and reoccurring discrepancy with the StringBuilder.
Just after 8600000 concatenations it suddenly increases violently with almost half a second.
We found this during our initial trials as well. This test was conducted numerous times and the conclusion is that something in StringBuilder changes when an object becomes too large. As for the standard String concatenation that if you draw a straight line from top to bottom reading the number is pretty close to the one we found initially at 12450.

Conclusion

The tests speak for themselves even with the surprising discrepancy in the large string StringBuilder test. The StringBuilder class is far superior when it comes to concatenation largely because it actually concatenates and adds to itself rather than what the String does when it creates a new instance of itself containing the added character/characters.

Possible Improvements

This was a first attempt into researching time measurement, and without doubt it did not provide perfect results. Whether this was due to old hardware that showed a high amount of hardware fluctuation or problems within the code can be discussed further. In some of the tests the intervals were a bit wide and hence made it a bit hard to pinpoint the closest one but in others, like the single String concatenation we can with some security pinpoint the exact location of the closest point.
It would be very interesting to see if someone has managed to replicate the surprising discrepancy in the StringBuilder large string test as well. In this specific case I find it hard to believe that it is code or hardware related.