Project 1 README
(For Remote Actors Only)

Group Members:
Shyamala Athaide (UF-ID 68741819)
Neeraj Rao (UF-ID 37737151)

How to launch the program

1. Unzip the project zip files.
2. Use the system names of each of the remote actor machines in the Configuration.scala file (saved in val `Configuration.machine1` to `Configuration.machine10`) and login to the appropriate remote machines.
3. Login to the speaker's machine using the machine name from Configuration.scala (val `Configuration.machineS`). CD to the root of the project directory and compile all the files:

   ```
   scalac -cp out -d out src/project2/*.scala
   ```
4. Run each of the Listener files on their respective machines using

   ```
   scala -cp out project2.Listener<i>
   ```

   where i ranges from 1 to 10

   Now, all the Listeners are registered and waiting for a message from the Speaker
5. Next, run the Speaker class on its machine and provide the appropriate N (the first argument) and K (the second argument) values using the command:

   ```
   scala -cp out project2.Speaker 100000000 20
   ```

   After the calculation is done, the numbers are printed on the speaker's machine.

Logic of the program

We have to calculate the sum of squares of K numbers starting from [1,N]. The most time consuming operation here is calculating the squares of the numbers. Hence, a naïve implementation that simply assigns number sequences to different actors would be very inefficient since the squares of numbers common to overlapping sequences would be calculated multiple times. As an example, consider a sequence length of K = 4. The first four sequences would be
1,2,3,4
2,3,4,5
3,4,5,6
4,5,6,7
We can see that the square of 2 would have to be calculated twice, the square of 3 thrice and the squares of the successive numbers (up to N-2) 4 times.

In order to avoid repetition, we use the fact that, in general, the sum of the squares of K numbers can be represented by

$\left(M-\frac{K}{2}\right)^2+\ldots+(M-2)^2+(M-1)^2+(M)^2+(M+1)^2(M+2)^2+\ldots+\left(M+\frac{K}{2}\right)^2$, where M is the median of the sequence of K numbers. This sum can be simplified to $K*M^2+2*C$, where

$C=\left(1^2+2^2+\ldots+\left(\left\lfloor\frac{K}{2}\right\rfloor\right)^2\right)$ for odd Ks

$C=\left(0.5^2+1.5^2+\ldots+\left(\frac{K}{2}-0.5\right)^2\right)$ for even Ks.

The advantages of this approach are that

a. Each sequence of K numbers is characterized by a *unique* median M and hence, to calculate the sum of each sequence, *only* this one number needs to be squared (i.e., we do not need to square the numbers common to the various sequences repeatedly)
b. C is the *same* for all sequences which means that it (and the squares that make it up) needs to be computed only once!

The constant term C and the range of medians M for a given K are shown in Table 1.

K	1st Median	Intermediate Medians	Last Median	C
Even	$\left(\frac{K}{2}+0.5\right)$	Add 1 to predecessor	$N+\left(\frac{K}{2}-0.5\right)$	$\left(0.5^2+1.5^2+\ldots+\left(\frac{K}{2}-0.5\right)^2\right)$
Odd	$\left\lceil\frac{K}{2}\right\rceil$		$N+\left\lfloor\frac{K}{2}\right\rfloor$	$\left(1^2+2^2+\ldots+\left(\left\lfloor\frac{K}{2}\right\rfloor\right)^2\right)$

Table 1: C and range of M for a given K

Program Architecture

Class Speaker acts as the top-level boss. He is given the N and K values. The actors run on 10 different remote machines. Speaker first calculates C using his CCalculators -- actors working in the layer beneath him, on the same machine. There are W = $\frac{K}{2}$ CCalculators – a number determined as the optimal value for the most efficient CPU time to Real time ratio when the program was completely local. After C has been calculated, the Speaker kills the CCalculators used and then moves on to the calculation of $K*M^2+2*C$.

For this calculation, there are 10 Listener objects (Listener1 to Listener10) beneath the Speaker, each on a different remote machine. The architecture is as shown in Figure 1.

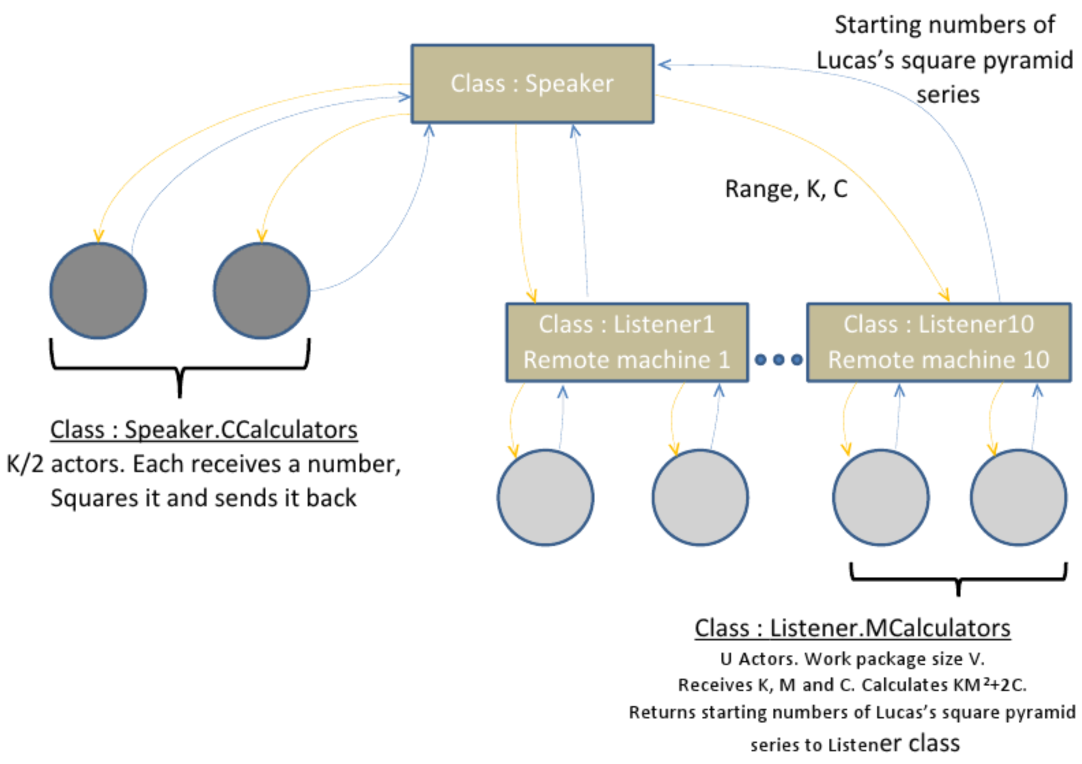
Starting numbers of Lucas's square pyramid series
Class : Speaker
Range, K, C
Class : Listener1
Remote machine 1
Class : Listener10
Remote machine 10
Class : Speaker.CCalculators
K/2 actors. Each receives a number, Squares it and sends it back
Class : Listener.MCalculators
U Actors. Work package size V.
Receives K, M and C. Calculates KM²+2C.
Returns starting numbers of Lucas's square pyramid series to Listener class

Figure 1: Program Architecture

The Speaker evenly divides N to send to 10 remote machines. This amount (N/10) is called the Range and is given to all the remote actors. Each remote actor also gets a value startAt, which is the first number from which he has to start calculating the medians. For instance, for N=20, and 2 remote actors, both actors get Range = 10. Listener1 gets startAt = 1 and Listener2 gets startAt = 11.

Each Listener object has its own MCalculator objects. It will pass these values in groups of work unit size V to the MCalculators below it. Thus, each MCalculator handles V numbers starting from startAt. The Listener also passes C and K so that the MCalculator can calculate $K * M^2 + 2 * C$. If this sum is a perfect square, each MCalculator will return the starting number of that sequence to the Listener. The Listener adds every number sent back in this way to a List. When all the MCalculators have finished reporting in, the list is sent back to the Speaker on the remote machine and the Listener cleans up, killing all the MCalculators and then exiting itself.

All the Listener objects carry out this computation in parallel on 10 different machines. Note that the Speaker assumes that the Listener objects are set up and running. If a Listener fails to respond because it is unreachable from the Speaker or there is a network failure, the Speaker will not know about it.

Output of scala project1.scala 1000000 20

No sequences were found that satisfied the conditions specified for these values of N and K.

Observations

This code could easily handle values of N=100000000 and K=20 using a Max JVM Heap Size of 1GB and an Initial Heap Size of 512 MB. Detailed timing performance and analysis are available below.

Observation 1:
Scalability of program: we ran the program with different values of N on different number of remote machines to see how the performance was affected. We used 4, 6, 8, and 10 remote machines. Results are shown for a large input size of N = 100000000 in Figure 2. As number of remote machines is increased, the performance of the program increases almost exponentially, as shown in the graph below. This clearly shows that the program can be scaled up efficiently for a large input size. Also, since all the additional machines will be using the same source code, the system can be scaled up in practically no time! We only need to add more Listeners for the added remote machines.

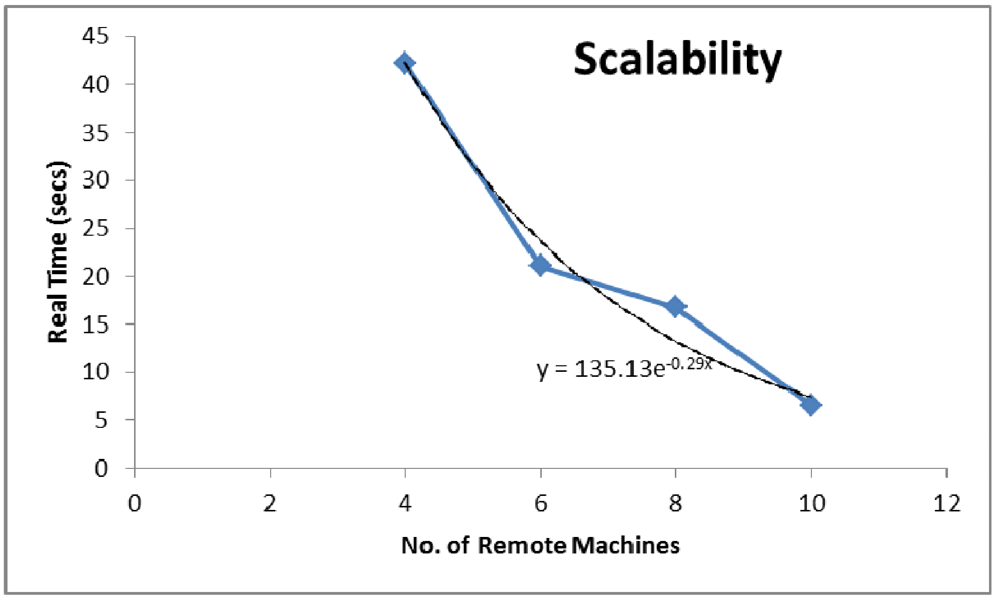

Figure 2: Program Scalability Performance

Observation 2:
Variation in runtimes for N = 10^8 when run multiple times on the same 10 machines. The only explanation we could find for this is that the machines had different CPU loads. We noticed that higher run times resulted when other users were also running JVMs on the machine (presumably for this project).

Run 1	Run 2
6.585 s	12.765 s

Observation 3:
Using Ints versus Lists to pass numbers between MCalculator <-> Listener and Listener <-> Speaker: we wanted to investigate how using different data structures for actor messages would affect program performance. We assumed that using Ints would result in faster run times because the VM wouldn't have to roll and unroll it as for a List. This was indeed true for the smaller values of N. However, this change didn't improve performance for N = 10^8. Our understanding is that for large Ns, the sheer

number of messages being passed around (which is much larger when Ints are used than when Lists are being used because every Int is passed back individually), especially between remote actors rather than MCalculator <-> Listener which are on the same machine, counters the performance gains of not having to unroll List.

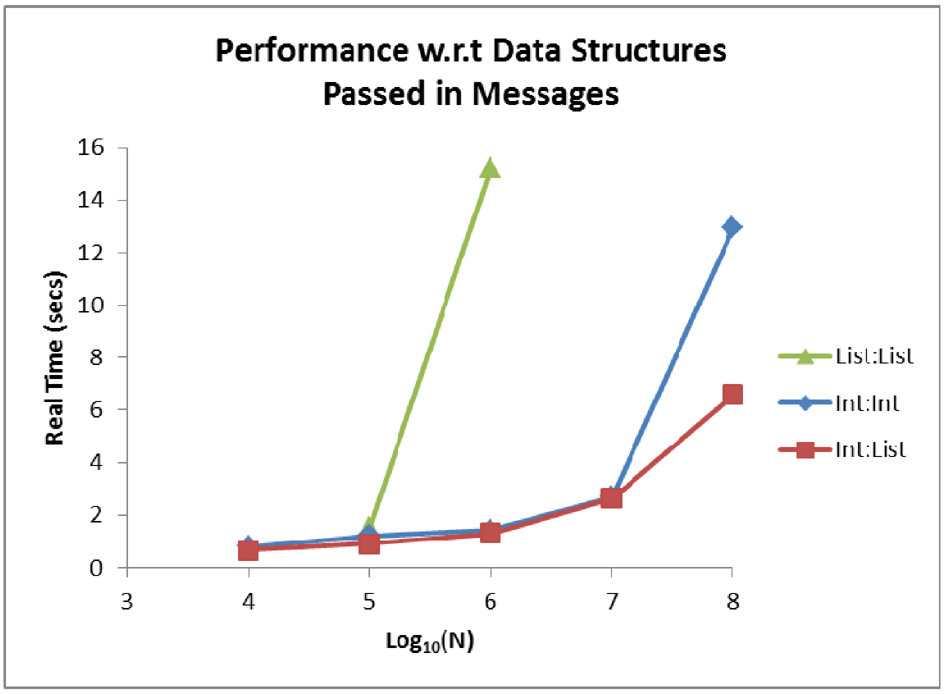

Figure 3: Effect of data structures used as messages

In Figure 3, the first data structure mentioned is the one passed from MCalculator to the Listener, and the second data structure is passed from the Listener to the Speaker. Thus, Int:List is when Ints are passed from MCalculator to the Listener and Lists are passed from the Listeners to the Speaker.

Observation 4:
When we used a case object to pass back messages from a remote Listener to the Speaker, we found that even though both actors knew about the case object individually (and compiled and ran without any problems on their own machines), the Speaker couldn't recognize the case object when it was sent back from a remote actor (the exact error was a ClassNotFound exception on the message receiving side i.e. the Speaker side). We are still investigating the cause of this issue.