RFORK: A SOLUTION TO A MULTITHREAD R WITH FORK/JOIN FRAMEWORK.

HENGRUI LUO

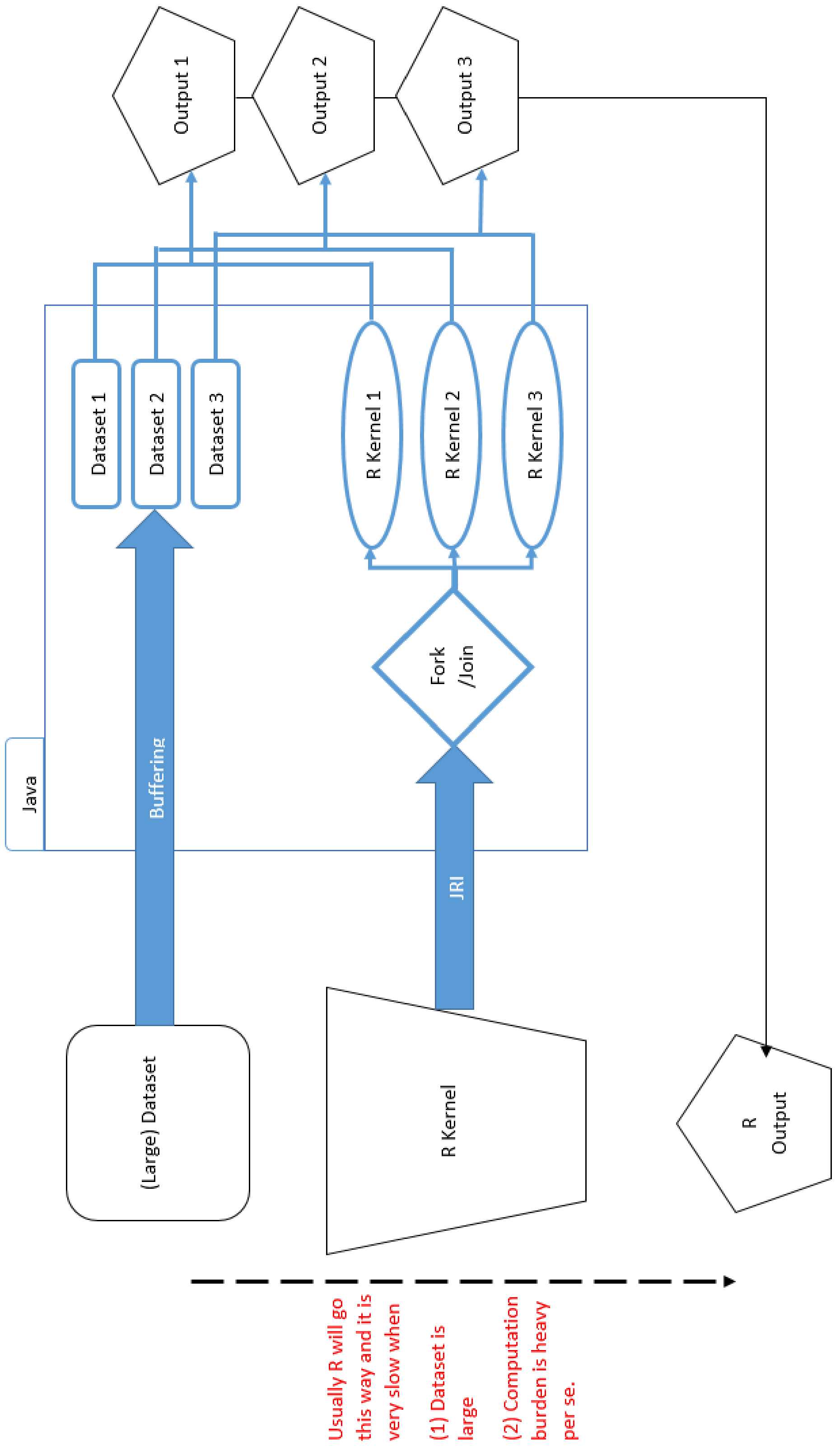
This is a framework instead of program, you should take your own risk, Rfork does not come with any warranty. I am still adding more features to it and hopefully it will become more flexible in the second update.

It is well-known that `R` (and its ancestor S) remains a singlethreaded program for eternity. This framework allows you to deploy `R` computation in a multithread fashion using Java-R interface (JRI) and the fork/join framework provided by $Java^{TM}$ 8 (and later versions).

To reader's concern, this framework will not speed up the script under arbitrary situation, actually it may slow down the whole process in terms of total computation time due to communication cost. However it has advantage that it allows `R` to handle large data and make use of multi-core resources at the same time if available. For example if you do not have sufficient memory to load a large dataset into `R` or analyze it using only one core on a cluster, then Rfork is your solution. For example, Rfork has been successfully applied to one of the author's earlier project `GWASReader`(`https://github.com/hrluo/GWASReader`) which allows analyzing of GWAS data (2~3GB) using a set of statistical tests relatively fast on personal computer using `R`. The author believes that such a framework will be of interest to statisticians, data analysts and other `R` users.

This framework not only allows you to fork/join a single `R` script but also incorporates a buffer mechanism that allows you to deal with large data files by buffering. The logic is that we can create more than one `REngine` objects in Java and fork/join the tasks to multiple `R` kernels in $Java^{TM}$. This shall be considered as a solution of paralleling computation in `R` unless we have a new way of distributing tasks within a single `R` kernel. Its performance is much better than existing parallel computation package provided in `R` like parallel. Moreover, it can deal with massive datasets like genetic datasets.

Following diagram shows how Rfork works to relieve the burden when multi-core computation resource is available. When there is only one core available, this framework will only cost a constant term of time cost (communication inbetween R and Java).


Output 1
Output 2
Output 3
Dataset 1
Dataset 2
Dataset 3
R Kernel 1
R Kernel 2
R Kernel 3
Fork /Join
Java
Buffering
JRI
(Large) Dataset
R Kernel
R Output
Usually R will go this way and it is very slow when
(1) Dataset is large
(2) Computation burden is heavy per se.

The method of using this framework is as following:

- First decide how you will like to split the (large) data file. This framework provides a convenient buffering mechanism, which allows you to read a raw file by line. By default an `Rbuffer` object will read one line at one time and treat it as the smallest unit of analysis. You can alter it by changing the buffering setting.
- After you decide how to split the file, you may want to decide whether you want an output for each chunk after `R` analyzed the result.
  - If you want an output file for each chunk analyzed, then use `Rbuffer.bufferWrite` function which takes an output file path argument that stores the output from R.
  - If you do not want an output file for each chunk analyzed, then use `Rbuffer.bufferRead` function which simply execute some analysis in `R` and then the result could be return or print to device.
- Then you can alter the `myrec` object, which is of `Record` class(each `Record` object has one unique `R` kernel, and each `Rbuffer` object has one unique `Record` object, and each `RTask` object has one unique `Rbuffer` object. Such a hierarchy allows only one kernel per task) , inside each loop of buffering. In this aspect we use JRI to realize following functionalities.
  - `Reval(command)` allows you to execute a command just like you do in `R` prompt.
  - `Rfunc(script location)` allows you to execute an `R` script file using absolute location, by default it will execute the file `Rfunc.R` under the same folder.
  - `Rsend(script location,parameters)` allows you to execute an `R` script file using absolute location, also passing parameters into the script if wanted.
  - `Rassign(parameter name,parameter value)` allows you to assign values to a variable in current `R` kernel.
- Finally you may want to compile and execute Rexecutor.java to see the result. However, there are two types of recursion tasks in this file, for further details of these two methods, we refer the reader to Oracle's document here(`https://docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html`) .

R-Java is an intermediate solution to the multithread goal, and shall not be regarded as an ultimate one. Moreover, when there is a cluster of machines, the fork/join framework will not require too much additional setup and hence provides an intrinsic flexibility. This framework beats RPython incorporation because Python cannot incorporate with R as smoothly as Java and GIL(Global Interpreter Lock) prevents multithreads if Python-C extension is not implemented. And R-C (or R-Cpp) interaction will attain the optimal computational speed if implemented. What prevents us from doing R-Cpp framework is the over-complicated memory allocation in C and its highly application dependent setup for cluster. R-Java if the compromise inbetween these two alternatives.

E-mail address: `luo.619@osu.edu`