

Package ‘biopara’

February 14, 2012

Title Self-contained parallel system for R

Version 1.5

Author Peter Lazar and David Schoenfeld

Description Biopara is a parallel system designed to be used with R.

Maintainer <plazar@amber.mgh.harvard.edu>

License GPL

Repository CRAN

Date/Publication 2007-10-22 18:38:54

R topics documented:

biopara 1

Index 11

biopara *Distributed Parallel System for R*

Description

Biopara is a function designed to allow users of R to distribute execution of coarsely parallel problems over multiple machines for concurrent execution. Biopara is called from within an R session and results are returned to the same session as a list of native R objects that can be directly manipulated without reinterpretation.

Usage

```
biopara(bioparatarget, bioparasource, bioparanruns, bioparafxn)
```

Arguments

bioparatarget	A list containing a string for the hostname of the machine running the master process and a number for the port number of the masters client port
bioparasource	A list containing a string for the hostname of the machine running the client and a number for the local port to receive the return connection from the master. The local port is chosen arbitrarily by user.
bioparanruns	A double precision number or list. If there is only a single item inside bioparafxn, then bioparanruns is the number of times to run this item. If there are multiple items in bioparafxn, bioparanruns is ignored. Bioparanruns is ignored for special commands except "setenv" ,which allows bioparanruns to be substituted with a list.
bioparafxn	A list containing strings to be evaluated on the remote workers machines. A single string must still be encased in a list. If a user specifies a single item, biopara will do bioparanruns number of evaluations of this string. If the list contains numerous strings, bioparanruns is ignored and there will be a single evaluation for each item in the list. There are also special commands that are treated differently and should be place alone in the list. These are: "numservers","reset","last"and "setenv".

Details

Biopara is a function designed to allow users of R to distribute execution of coarsely parallel problems over multiple machines for vast speedup and concurrent execution. Biopara is called from within an R session and results are returned to the same session as a list of native R objects that can be directly manipulated. The topology of the parallel system consists of a single master process to coordinates communications, a series of worker machines designated to do the actual evaluations and a client on the user's machine to send requests and receive results. The biopara function call contains code for running the master process, the worker process and the client process. The arguments' type determines operational mode. Biopara is self-contained and requires no helper utilities outside of R and is thus cross-platform. Biopara operates entirely using R socketconnections so references will be made to tcp/ip port numbers and computer tcp/ip hostnames throughout this document. Doubles refer to the basic default numeric type. Biopara performance is excellent. Using 22 workers and a master, 500 runs of instant return ("1+1") has a turnaround time of 4.7 seconds (less that a hundredth of a second overhead per run).

Client Usage

A user of the client begins the process from a running R session by placing a call to the biopara function (this assumes the master and workers have already been set up). The biopara function call contacts the master process and places a request. The master process then distributes a proper number of tasks to it's worker pool. The workers evaluate their given statement and return the output object of that evaluation as the result. The master constructs an in-order list of these objects and passes this list back to the client over the connection as the result of the biopara call. The order of the list is in the same order as the function call list or the number-of-runs iterator. This output list consists of the exact native R objects that the evaluations returned. The list objects can be used directly in the same R session. Here is the syntax for biopara in client mode:

```
out<-biopara(list("master-name",master-port),list("local-name",local-port),num-runs,list("fxn(args)
or alternately
```

```
out<-biopara(list("master-hostname",master-port),list("local-hostname",local-port),1,
list("fxn(args...)", "fxn(args...)",....., "fxn(args)"))
```

The first syntax is used to evaluate a single command num-runs times. The second syntax is used when one wishes to evaluate a series of dissimilar functions or the same function with dissimilar arguments.

The first biopara argument is a list containing a string and a double. The string argument is the hostname of the machine running the master process. The double is the master processes' client port specified to the master during master creation (the bioparaclientport argument). This list has a third optional string argument that specifies to the master which environment to execute the statements in. This is useful for users wishing to have multiple workspaces present on the cluster.

The second biopara argument is also a list containing a string and a double. The string is the hostname of the machine running the client as the master host would see it. The string "localhost" will not work unless you incidentally have the master on the same machine as well. The double is a port number on the local machine that the master can use when communicating to your client. It can be chosen at will as long as the port is not already in use. You can use the windows/unix command "netstat" to view local ports already in use. The client port specified is automatically created during the call.

The third biopara argument, num-runs, is the number of times to perform an evaluation of the function specified. This will be used if there is only a single item in the function list. It is useful for repeating a function/simulation that features randomization and produces different results with each run. If there are multiple calls in the function list, then biopara will make a single run for each item in the list and num-runs will be ignored but should still be present as a filler number. For special service function "setenv" (described below) num-runs can be replace with a list as well.

The fourth biopara argument is a list of strings to evaluate as function calls or statements. Even if there is a single item, it must be placed in a list. During execution, the worker processes will place each of these strings into the statement:

```
output<-eval(parse(text=your-function-string))
```

Please note this occurs on the worker host and not the user's R session (see caveats). If you send a single string, then the master will create num-runs copies of the statement and distribute these as jobs to it's workers. If there are multiple strings, there will be one job for each item in the list with the num-runs argument being ignored. Please note that the function strings will be evaluated on a remote machine so it will be necessary to export your environment using the "setenv" special command described below. System() calls are blocked for security reasons. The strings can be expressions such as "1+1" or function calls such as "sum(a,b)". Function calls that require string arguments will need to have the enclosed string's quotes backslash escaped as to not prematurely end the function call string (see examples' section library() call).

The client has a few special commands available to facilitate system use and maintenance. These should be placed by themselves into the function list and are treated differently. The special command "numservers" returns the number of responding workers. This is useful for determining the maximum number of workers available when deciding how to divide a data set. The special command "hosts" returns a list of socket descriptions to the worker ring. This is useful for seeing which computers are part of the worker ring. Special command "last" returns a history of cluster usage. The special command "reset" clears a user's environment across the cluster. The most important special function is "setenv". Setenv is used to export your data, variables and functions to the cluster. Calling setenv in base mode will pull in the entire client machine's R desktop environment (as given by ls() in your current session) and it will transmit this environment to every worker in the

ring. Without a `setenv`, the user will be limited to the base R functions and explicit numerical or string data in function calls. To export user defined functions, simply source them or load them to memory. Occasionally, a user will require finer control over their environment on the cluster. A `setenv` can be called and the `num-runs` field replaced with a list of strings containing the variable and function names to be exported to the cluster. The `setenv` will then only send these specific objects. A user's environment can be queried by running a `biopara` call to function `ls()` with a single run. Since all workers have the same environment, any response will contain your correct environment for the cluster. The environment will be assigned a default tag corresponding to the user name of the user running R on the client. If a user requires multiple environments containing different variables and functions, the `biopara` first argument list can take a third optional string argument. This string will be used in place of the default tag and allow a single user to refer to multiple environments. A user can `"setenv"` with the tag to populate the environment with functions and variables. Any subsequent calls must bear this tag argument to take advantage of this new environment. A `"reset"` will only clear one environment at a time (the tag's environment or the default environment for no tag).

After a worker has evaluated an expression string, the worker will return whatever object the expression would ordinarily produce as its output. The master will collect these output objects and place them in a list. The list will be in the order of the `numruns` iterator or the exact order of the function list in a multi-item function list. This result list is transmitted back to the client as a list of objects and the socketconnections between client and master are closed. This list is returned as the output of the `biopara` call. The list does not have to be reinterpreted as the returned objects are R native so they can be manipulated immediately.

Caveats to Using the System

Any references to create and read files and traverse paths should be relative to the current working directory `./`. The requested function calls will be executed on a remote machine using a different R instance from the worker's working directory (specified in the `bioparaconfig` option list). Any references to files on the client machine directory structure will fail. The most graceful way of handling output to files (and file reading) is to have all of the data being used loaded to memory on the client's R instance, calling `biopara "setenv"` to export structures in memory and then calling functions that take in-memory arguments. The R session hosting the client can then receive the output list from a `biopara` call, process the data within the list and write a single output file. This has been successfully tested with rather large data sets.

One could also manually place all input files into the working directories of all workers. Uniformity is very important. Please take into consideration that the user has no direct control over which worker receives which run instance so pathing is an issue unless there is a cluster environment with identical pathing on all workers.

A shared medium (NFS drive) is also very useful as long as the relative paths to the shared medium are the same for every worker. This latter setup is our setup of choice for a uniform ROCKS cluster. Please note that execution is carried out by the user running the `biopara` worker on the worker's host. This user needs to have permissions to all of the affected directories and files (this is an administration issue but still of user concern if one wishes to refer to their shared home directory).

It is best to run the `setenv` in the same environment where the data to be used was defined in the first place. Nesting a `biopara "setenv"` call inside another function will have the `setenv` perform its `ls()` inside your function and not your desktop. If large amounts of data is being created at a certain function depth, it is best to place a call to `biopara setenv` at that level before calling another function that is to perform a `biopara` computation run using that data.

The library and data function calls do not transport data over setenv. It will be necessary to call `library()` and `data()` inside a `biopara` function call. Libraries can also be loaded clusterwide (if necessary) by making a reference to the `library()` command inside one of your user defined functions to be run on the workers. All workers must actually have the libraries installed. Libraries can also be loaded manually before a functioncall in a roundabout manner by creating a run for function `library(yourlibrary)` (make sure your enclosed quotes are backslash escaped) and set the number of runs to be greater than the number of workers (obtained using the "numservers" special request). Finally, instead of calling the dependant function alone, the library call can be embedded inside the actual functioncall as such:

```
biopara(list(master,port),list(client,cport),nruns,list("library(lib);depedantfunc()"))
```

`Biopara` still sees only a single string and will treat the string as single call. The multiple semicolon seperated statements will be handled by `eval` on the workers.

There is a known bug with certain functions and structures that return function calls as part of their output. Since `biopara` uses `dump` to create textual representations of objects to send through connections, these calls appear inside the structure. When these are fed into `eval(parse(text=dump-outout-text))` to recreate the output objects, R will try to execute the call. This leads to either execution of your call on the master or re-evaluation of your command on the client, neither of which is desirable. An specific example is the R "boot" function, which returns it's originating function call at index 8 as generated from the statement "`call<-match.call()`". This particular example cause the client to re-evaluate your statement when the results are returned. A simple workaround would be to return the call as "`as.character(your-call)`". Alternately, you could inline reassign the result:

```
biopara(list(master,port),list(client,cport),nruns,list("out<-boot(args);out[[8]]<-as.character(out
```

This would inline-replace the problem item with a string representation before returning the results. Since the above is still technically a single string (containing semicolon seperated commands, perfectly valid), `biopara` would see it as a single command and perform `nruns` runs.

The master currently supports a single parallel computation request at a time due to the architecture. The master turns away connections to further requests unit it is ready to accept and process them. A user should therefore fine tune his application to take advantage of the maximum number of workers using the special command "numservers". This way there are no idle workers attached to a busy master.

Setup and Administration

Before `biopara` can be used, the master and it's worker ring must be established. The `biopara` function determines it's current role (client, master or worker) based on the typing of the inputs. The client mode used typing `list-list-double-list` or alternately `list-list-list-list` for the `setenv` special command. `Biopara` master mode launches if it detects a `string-double-double-list` argument pattern. Here are the arguments for the `biopara` master:

```
biopara(mastername,port-for-workers,port-for-clients,list(configs))
```

The `mastername` is the hostname of the machine running the master as seen from external hosts. The port for workers is the port used by the master to talk with it's workers. The port for clients is the port used by the clients to send requests to the master. The `list(configs)` is a list of lists of the form:

```
list(list("hostname-of-worker1",outbound-port,"worker-working-dir",inbound-port,"launch command"),list("hostname..."),list("hostname..."))
```

The `hostname-of-worker` is a string for the hostname of the computer carrying the worker as seen from the master host. The `outbound port` is a double for the port number used by the worker to

talk to the master. The worker working dir is a string with a path on the worker host where the biopara worker process will leave temporary files. The inbound port is a double for the port used by the master to talk to the worker. The launch command is a string that is executed by the master to automatically launch the worker on the remote host when the master starts up. Usually this is an ssh/telnet/rsh background session to run a script on the remote machine. This script starts R and launches biopara in worker mode. An example is included in the biopara("help") mode that combines ssh, echo and piping. If no such command is possible/desired, simply enter an empty "" string. The master will still attempt to connect to the described worker but will not make an attempt to spawn it. Such workers need to be manually launched before the master is started. It will be necessary to start all of the desired workers without launch commands before the master since the master only reads its configuration and contacts workers at startup. Workers may leave a running master (by simply closing the R session on the worker host) but workers may not join an already running master.

The master process will use the current working directory of R as its working directory. The user running the master will require read/write access to this directory as the master will create dumpData.R files in this directory. It is recommended to run biopara in its own directory as opposed to a directory full of user files for this reason. It is also recommended that the biopara master be run by a user secured from unnecessary permissions. The biopara workers will be doing arbitrary eval statements on the worker nodes and various file related calls on the master host (though they are banned from system() calls. Security is discussed below). Workers and masters should not be launched with the same working directory since the dumpfiles may collide. It is also recommended that the working directories for both the master and workers be on a local drive and not a shared resource (like an NFS or network drive) since large datasets and lots of workers may generate a non-trivial network load and negatively impact performance.

The final step to running the master is to place a file all.allow in the master's working directory. Without this, the master will refuse all client connections with a permissions error. The master will look for either the all.allow file or a client-hostname.allow file in its working directory on connection from a client. This is a very basic form of trusted hosts described further below.

If automatic worker creation is not available or practical, the user must manually establish all of the workers before the master is launched. A worker can technically be run on any machine that supports R and is capable of seeing the master host via tcp/ip. To start a worker manually, simply go to the desired host for the worker, launch R, and type

```
biopara(worker-port, "worker-working directory", "master-hostname", master-port-for-workers)
```

The worker looks for the typing pattern double-string-string-double. The worker port is a double for the port used by the worker to communicate with the master. The worker working directory is a string for the directory used by the worker to store tempfiles. This should not be the same directory as the master's working directory or the working directory of a client. The master hostname is a string for the hostname of the master host as seen from the worker. The master port for workers is a double for the bioparamasterport defined for the master. Once launched, the worker will block on a socket and wait indefinitely for the master to contact it. The workers last for the duration of the master's session but there is currently no way to signal them to shut down if the master dies unexpectedly so they need to be shut down manually as well from the R session's interrupt.

Behind the scenes

Each of the workers maintains an identical environment file for each biopara user. A fresh worker that receives a "setenv" call will be transmitted the biopara user's R desktop environment (or the listed variables specified in the bioparanruns field of the setenv command). This received environment is saved to a local file on the worker using the default username tag (or user specified tag) and

placed in the working directory on the worker. This environment file is deleted and the environment unloaded if a "reset" command is received from that user. If the worker has an environment loaded and detects that a command being sent is from a different user, the worker will unload it's current environment and load the environment file for the new user. This way, the worker does not have to reload it's environment until users are changed. This is a performance gain for large datasets or jobs with large numbers of runs. The master maintains only a record of it's workers and does not keep any environment variables in it's memory. As a method of data persistence, even if a worker is shut down and restarted, the environments will persist since they are file based. Technically, a worker can even be moved to a different machine altogether while preserving it's environment as long as the working directory is copied over.

As mentioned before, the Biopara system uses R socketConnections for all communications. The client is fed a pair of host/port combinations. This information is used to set up a 2 way hand-shaking with the server. The client uses the first argument list to connect to the master. The client then sends the second argument list to the master, which the master uses to connect back to your client. This 2 way connection is used because of port blocking. A process will block indefinitely on a "serversocket receive" and use near zero resources doing so. This makes the system idle very gracefully and wait for results very gracefully. The client-connect either immediately fails or immediately succeeds. Transmission has similar characteristics that favor client sockets instead, blocking on sends.

The data transmitted is generated by using the dump function on various objects to turn them into text files. This text is read back into memory as and streamed across the sockets. This is much faster than file passing. Each stream has three parts. The first is a string bearing the variable or function name. The second part is a number bearing the size of the data stream. The third part is a stream containing the text representation of the data itself. The streams are read in and the data portion is assigned to the variable name. It is possible to send arbitrary structures this way. This has been successfully tested with over 100 meg data structures.

Features

Biopara has fault tolerance and load balancing built in. When a task is sent to a worker, a timestamp is taken and stored along with the command information sent to the worker. When the very first task in a command returns, the total elapsed time is taken. This is used as a litmus test for the best possible completion time for that particular command. This best time is multiplied by a lenience factor of 3 to create a threshold of execution. For all subsequent tasks, if the execution time exceeds this figure, that host is considered down and is locked out and no further tasks are sent to it. The task information for that timestamp is then retrieved and placed back in the to-do queue for later reprocessing. This way, a large job will return even if some hosts are lost along the way. To simulate load balancing, a worker is immediately sent another task (if one exists to be sent) when that worker returns data. In this, the master inherently load balances as it favors faster workers. Locked out workers are locked out for the duration of the master's life so as not to interfere with future runs. In addition, in case the client is disconnected from the master during a run, the master will leave a temp file in it's working directory containing a variable "parrans". The temp file will bear the first few letters of the username tag attached to that particular run.

Security

Since we will be evaluating arbitrary strings from a publically accessible port, there must be some security precautions present. We have 2 simple schemes present to facilitate nominal security. The first is a form of "trusted hosts". When the master receives a client connection, it checks for the presence of a hostname.allow file in it's current working directory. If no such file is present, the master drops the connection. The admin can create these files manually for direct control or place an

all.allow file in the working directory to disable this feature. Another simple security feature is the blocking of all system() calls to the workers. This limits unwanted intruders to creating/deleting files in biopara's access sphere. The biopara user should therefore have very limited permissions, mostly for creating/deleting files in it's working directory. These simple security features are adequate for intranet use of biopara coupled with operation of the server only during computation. A more complete security scheme is to firewall the master's client port and make firewall exceptions to individual computers. This makes the system usable only by trusted hosts.

Value

When used in client mode, biopara returns a list with each element being an in-order solution to each of it's input bioparafxn arguments. It will return whatever structure your evaluated command string would normally return. It is possible to return arbitrary structures. For bioparanruns operation with a single bioparafxn item, the list will be in the order of the bioparanruns iterator.

Author(s)

Peter Lazar <plazar@amber.mgh.harvard.edu> and David Schoenfeld <dschoenfeld@partners.org>

Examples

```
#These examples assume a master called my.server.edu running on port 39000 and a client
#1.2.3.4 using return port 40000. The client port is chosen arbitrarily from free ports by the user
#The request is for 5 runs of myfxn(a,b). This will place the output of myfxn(a,b) into
#variable "f" as a list in the order of the run iterator. Please note that one must
#have already defined a and b in a setenv (described below) in order for the cluster
#to be aware of the values of a and b.
## Not run: f<-biopara(list("my.server.edu",39000),list("1.2.3.4",40000),5,list("myfxn(a,b)"));
#This is a request to produce 3 runs of all different functions. Nruns is ignored and the return
#list is in the order of the request list.
## Not run: f<-biopara(list("my.server.edu",39000),list("1.2.3.4",40000),5,list("myfxn(a)","1+1","sum(4)"))
#The following will fail unless the client and master are on the same node. The reference of
#localhost is evaluated on the master computer and the return connection will not be established.
## Not run: f<-biopara(list("my.server.edu",39000),list("localhost",40000),5,list("myfxn(a)"))
#How to export your environment to the cluster of workers without fine tuning
#This will collect your username from your client machine and attach the environment to this label
## Not run: f<- biopara(list("my.server.edu",39000),list("1.2.3.4",40000),0,list("setenv"))
#How to export your environment using fine tuning to export the list mylist and the function myfxn
## Not run: f<- biopara(list("my.server.edu",39000),list("1.2.3.4",40000),list("mylist","myfxn"),list("setenv"))
#This specifies to the system to attach your environment to tag "myusername" instead
#This has no other system effect as everything is being evaluated under the user executing biopara
#Please note that you must use this same tag in order to manipulate this environment in the future
#Also, any runs must also bear this tag to take advantage of this environment
## Not run: f<-biopara(list("my.server.edu",39000,"myusername"),list("1.2.3.4",40000),5,list("myfxn(a,b)"))
#This is a request to list your environment on the cluster.
## Not run: f<-biopara(list("my.server.edu",39000),list("1.2.3.4",40000),1,list("ls()"))
#Here is a special command "reset" that requests the cluster to clear one's environment
## Not run: f<-biopara(list("my.server.edu",39000),list("1.2.3.4",40000),1,list("reset"))
#This is a special command "last" that lists cluster usage and timestamps of user access
## Not run: f<-biopara(list("my.server.edu",39000),list("1.2.3.4",40000),1,list("last"))
#Here is a special command "numservers" that queries the cluster for the number of active workers.
#This is useful when determining how to break up a large computation for maximum parallelism
```

```

## Not run: f<-biopara(list("my.server.edu",39000),list("1.2.3.4",40000),1,list("numservers"))
#Here is a special command "hosts" that returns a list of connections from the master.
#This is useful for determining if certain hosts are part of the worker ring
## Not run: f<-biopara(list("my.server.edu",39000),list("1.2.3.4",40000),1,list("hosts"))

#Master process configuration

#This is a special command to display the master's configuration instructions
biopara("help")
#This configures a master on my.server.edu using port 36000 to listen for worker connections
#and port 39000 for client connections.
#It is configured to connect to a worker on localhost listening on port 38000 and sending on 37000
#This assumes the worker has already been started with those parameters prior to master launch.
#The start command is blank so the master will make no effort to establish any workers.
#The master will, however, attempt to contact the workers inside the config list.
bioparamastername <- "my.server.edu";
bioparamasterport <- 36000;
bioparaclientport <- 39000;
bioparaconfig <- list(list("localhost",37000,"c:/",38000,""));
## Not run: biopara(bioparamastername,bioparamasterport,bioparaclientport,bioparaconfig)
#Here is a bioparaconfig for a pair of workers.
#The launch commands are ssh single-command background sessions to run a script.
#Included in the "help" command is an example using ssh, echo and piping that allows
#one to launch R and pipe it a commandline with arguments.
bioparaconfig<-list(
list("worker1.edu",42000,"/tmp",43000,"ssh -f worker1.edu ~myuser/runwkr.sh"),
list("worker2.edu",37000,"/temp",38000,"ssh -f worker2.edu /usr/local/R/runwkr.sh"));

#Worker process configuration

#Please note this is only necessary if the master command string cannot be used to launch
#the worker processes automatically.
#This launches a worker listening on port 38000 and transmitting on port 37000.
#This worker is configured for master on "localhost" and uses /tmp as it's working directory
## Not run: biopara(37000,"/tmp","localhost",38000)
#To launch a worker to listen to my.server.edu using ports 38000 and 37000 for communications
## Not run: biopara(37000,"/tmp","my.server.edu",38000)

#Single host test for system

#You will need 3 instances of R for this test. It uses the / directory as temp this is not
#recommended but should be sufficient for simple tests.
#Start by launching the worker
## Not run: biopara(37000,"/","localhost",38000)
#Then start a master pointing at this one worker
## Not run: biopara("localhost",36000,39000,list(list("localhost",37000,"/",38000,"")))
#Issue a simple run
## Not run: out<-biopara(list("localhost",39000),list("localhost",40000),5,list("sum(1,1)"))
#A more complex run

```

```
## Not run: out<-biopara(list("localhost",39000),list("localhost",40000),5,list("sum(1,1)","1+1","library(\"base\")"))
#A setenv for a single var
myvar<-3
## Not run: out<-biopara(list("localhost",39000),list("localhost",40000),1,list("setenv"))
#A setenv for a function only exporting the function
myfunc <- function(a,b){a+b}
## Not run: out<-biopara(list("localhost",39000),list("localhost",40000),list("myfunc"),list("setenv"))
#Using the new variables and function
## Not run: out<-biopara(list("localhost",39000),list("localhost",40000),3,list("myfunc(myvar,myvar)"))
#To see your variables. You will see biopara holding variables as well. Do not redefine these.
## Not run: out<-biopara(list("localhost",39000),list("localhost",40000),1,list("ls()"))
#Clear your variables
## Not run: out<-biopara(list("localhost",39000),list("localhost",40000),1,list("reset"))
#See your worker process. It will be the entry before the 36000 and 39000 entries
## Not run: out<-biopara(list("localhost",39000),list("localhost",40000),1,list("hosts"))
```

Index

- *Topic **data**
 - biopara, 1
- *Topic **manip**
 - biopara, 1
- *Topic **misc**
 - biopara, 1
- *Topic **optimize**
 - biopara, 1
- *Topic **utilities**
 - biopara, 1

biopara, 1