Package 'snowFT'

July 23, 2025

Title Fault Tolerant Simple Network of Workstations

Version 1.6-1

Date 2023-09-20

Author Hana Sevcikova, A. J. Rossini

Description Extension of the snow package supporting fault tolerant and reproducible applications, as well as supporting easy-to-use parallel programming - only one function is needed. Dynamic cluster size is also available.

Maintainer Hana Sevcikova <hanas@uw.edu>

License GPL (>= 2)

Imports parallel, snow (>= 0.4)

Depends R (>= 3.0-0), rlecuyer

Suggests Rmpi

URL http://www.stat.washington.edu/hana/parallel/snowFT-doc.pdf

NeedsCompilation no

Repository CRAN

Date/Publication 2023-09-21 06:50:02 UTC

Contents

snowFT-package *Fault Tolerant Simple Network of Workstations*

Description

Extension of the snow package supporting fault tolerant and reproducible applications, dynamic cluster resizing, as well as supporting easy-to-use parallel programming - only one function is needed. It supports the MPI and the socket communication layers.

Details

Package:	snowFT
Version:	1.6-0
License:	GPL

The main function of this package, `performParallel`, handles all tasks that are necessary for evaluating a user-defined function in parallel. These include creating a cluster, initializing nodes, handling a random number generator, processing the given function on the cluster and cleaning up. In the very basic settings (i.e. when using with the socket layer), no additional software is necessary. The package can be used on a single multi-processor/core machine, homogeneous cluster, or a heterogeneous group of computers.

The package supports creating and handling a **snow** cluster that is:

1. Fault tolerant: The master checks repeatedly for failures in its waiting time and initiates a failure recovery if needed. (This feature has been implemented for the PVM layer. Unfortunately, the PVM layer had to be switched off due to the **rpvm** package not being currently maintained.)
2. Load balanced AND produces reproducible results: one stream of random numbers associated with one replicate (instead of one stream per node as handled by **snow** and **parallel**).
3. Computationally transparent: Currently processed replicates and failed replicates stored into files. Allows defining a function that is called after each given number of replicates.
4. Dynamically resizeable: The cluster size is stored in a file which is read by the master repeatedly. In case of a modification the cluster is updated. (Not available for MPI.)
5. Administration overhead minimized: All administration is managed by the master in its waiting time. (Note that there is a time-overhead for creating and destroying the cluster, as well as the RNG initialization. Thus, simple operations, such as the example below, will not gain from running in parallel.)
6. Allows running processes sequentially with the same random numbers as it would in parallel. Thus, results can be compared between the two modes.
7. Easy to use: All features, including creating the cluster, RNG initialization and clean-up, are available via one single function - `performParallel`.

Author(s)

Hana Sevcikova, A. J. Rossini

Maintainer: Hana Sevcikova <hanas@uw.edu>

References

http://www.stat.washington.edu/hana/parallel/snowFT-doc.pdf

See Also

performParallel, clusterCall

Examples

```
## Not run:
# generates 500 times 1000 normally distributed random numbers on 5 nodes
# (all localhost)
res <- performParallel(5, rep(1000, 500), fun = rnorm, cltype = "SOCK")
print(mean(unlist(res)))

# View cluster usage
# number of physical nodes
P <- parallel::detectCores(logical = FALSE)
t <- snow::snow.time(performParallel(P, rep(1e6, 50),
        fun = function(x) median(rnorm(x)), cltype = "SOCK"))
plot(t)

## End(Not run)
```

---

snowFT-cluster *Cluster-Level Functions*

Description

Functions that extend the collection of cluster-level functions of the **parallel/snow** package while providing additional features, including reproducibility and dynamic cluster resizing. The heart of the package is the function performParallel.

Usage

```
performParallel(count, x, fun, initfun = NULL, initexpr = NULL,
                export = NULL, exitfun = NULL,
                printfun = NULL, printargs = NULL,
                printrepl = max(length(x)/10,1),
                cltype = getClusterOption("type"),
                cluster.args = NULL,
                gentype = "RNGstream", seed = sample(1:9999999,6),
```

```
                 prngkind = "default", para = 0,
                 mngtfiles = c(".clustersize",".proc",".proc_fail"),
                 ft_verbose = FALSE, ...)

clusterApplyFT(cl, x, fun, initfun = NULL, initexpr = NULL,
               export = NULL, exitfun = NULL,
               printfun = NULL, printargs = NULL,
               printrepl = max(length(x)/10,1), gentype = "None",
               seed = rep(123456,6), prngkind = "default", para = 0,
               mngtfiles = c(".clustersize",".proc",".proc_fail"),
               ft_verbose = FALSE, ...)

clusterCallpart(cl, nodes, fun, ...)

clusterEvalQpart(cl, nodes, expr)

printClusterInfo(cl)
```

Arguments

`count`	Number of cluster nodes. If `count=0`, the process runs sequentially.
`cl`	Cluster object.
`x`	Vector of values to be passed to function `fun`. Its length determines how many times `fun` is to be called. `x[i]` is passed to `fun` (as its first argument) in the i-th call.
`fun`	Function or character string naming a function.
`initfun`	Function or character string naming a function with no arguments that is to be called on each node prior to the computation. It is passed to workers using `clusterCall`. It can be used for example for loading required libraries or sourcing data files.
`initexpr`	Expression evaluated on workers at the time of node initialization. It corresponds to what would be passed to `clusterEvalQ` before the computation. `initfun` and `initexpr` can be used for the same purpose, but `initexpr` does not need to have a form of a function.
`export`	Character vector naming objects to be exported to workers.
`exitfun`	Function or character string naming a function with no arguments that is to be called on each node after the computation is completed.
`printfun, printargs, printrepl`	`printfun` is a function or character string naming a function that is to be called on the master node after each `printrepl` completed replicates, and thus it can be used for accessing intermediate results. Arguments passed to `printfun` are: a list (of length `|x|`) of results (including the non-finished ones), the number of finished results, and `printargs`.
`cltype`	Character string that specifies cluster type (see `makeClusterFT`). Possible values are 'MPI' and 'SOCK' ('PVM' is currently not available).

`cluster.args` List of arguments passed to the function `makeClusterFT`. For the 'SOCK' layer, the most useful argument in this list is `names` which can contain a vector of host names, or a list containing specification for each host (see Example in `makeCluster`). Due to the dynamic resizing feature, the length of this vector (or list) does not need to match the size of the cluster - it is used as a pool from which hosts are taken as they are needed. Another useful argument is `outfile`, specifying name of a file to which slave node output is to be directed.

`gentype` Character string that specifies the type of the random number generator (RNG). Possible values: "RNGstream" (L'Ecuyer's RNG), "SPRNG", or "None", see `clusterSetupRNG.FT`. If `gentype="None"`, no RNG action is taken.

`seed, prngkind, para`
Seed, kind and parameters for the RNG (see `clusterSetupRNG.FT`). Seed can be an integer or a vector of six integers.

`mngtfiles` A character vector of length 3 containing names of management files: `mngtfiles[1]` for managing the cluster size, `mngtfiles[2]` for monitoring replicates as they are processed, `mngtfiles[3]` for monitoring failed replicates. If any of these files equals an empty string, the corresponding management actions (i.e. dynamic cluster resizing, outputting processed replicates, and cluster repair in case of failures) are not performed. If the files already exist, their content is overwritten. Note that the cluster repair action was only available for PVM which is switched off. Furthermore, the dynamic cluster resizing is not available for MPI.

`ft_verbose` If TRUE, debugging messages are sent to standard output.

`nodes` Indices of cluster nodes.

`expr` Expression to evaluate.

`...` Additional arguments to pass to function `fun`.

Details

`clusterApplyFT` is a version of `clusterApplyLB` of the **parallel/snow** package with additional features, such as results reproducibility, computation transparency and dynamic cluster resizing. The master process does the management in its waiting time.

The file `mngtfiles[1]` (which defaults to '.clustersize') is initially written by the master prior to the computation and it contains a single integer value corresponding to the number of cluster nodes. The value can be arbitrarily changed by the user (but should remain in the same format). The master reads the file in its waiting time. If the value in this file is larger than the current cluster size, new nodes are created and the computation is expanded on them. If on the other hand the value is smaller, nodes are successively discarded after they finish their current computation. The arguments `initfun, initexpr, export` and `exitfun` in the `clusterApplyFT` function are only used, if there are changes in the cluster, i.e. if new nodes are added or if nodes are removed from cluster.

The RNG uses the scheme 'one stream per replicate', in contrary to 'one stream per node' used by `clusterApplyLB`. Therefore with each replicate, the RNG is reset to the corresponding stream (identified by the replicate number). Thus, the final results are reproducible regardless of how many nodes were used.

`performParallel` is a wrapper function for `clusterApplyFT` and we recommend using this function rather than using `clusterApplyFT` directly. It creates a cluster of `count` nodes; on all nodes it

calls `initfun`, evaluates `initexpr` and `export`, and initializes the RNG. Then it calls `clusterApplyFT`. After the computation is finished, it calls `exitfun` on all nodes and stops the cluster. If `count=0`, function `fun` is invoked sequentially with the same settings (including random numbers) as it would in parallel. This mode can be used for debugging purposes.

`clusterCallpart` calls a function `fun` with identical arguments `...` on nodes specified by indices `nodes` in the cluster `cl` and returns a list of the results.

`clusterEvalQpart` evaluates a literal expression on nodes specified by indices `nodes`.

`printClusterInfo` prints out some basic information about the cluster.

Value

`clusterApplyFT` returns a list of two elements. The first one is a list (of length `|x|`) of results, the second one is the (possibly updated) cluster object.

`performParallel` returns a list of results.

Author(s)

Hana Sevcikova

Examples

```
## Not run:
# generates n normally distributed random numbers in r replicates
# on p nodes and prints their mean after each r/10 replicate.

printfun <- function(res, n, args = NULL) {
  res <- unlist(res)
  res <- res[!is.null(res)]
  print(paste("mean after:", n, "replicates:", mean(res),
           "(from", length(res), "RNs)"))
  }

r <- 1000; n <- 100; p <- 5
res <- performParallel(p, rep(n,r), fun = rnorm, seed = 1,
                printfun = printfun)

# Setting p <- 0 will run the rnorm call above sequentially and
# should give exactly the same results
res.seq <- performParallel(0, rep(n,r), fun = rnorm, seed = 1,
                printfun = printfun)
identical(res, res.seq)

# Example with worker initialization
mean <- 20
sd <- 10
myfun <- function(r) rdnorm(r, mean = mean, sd = sd)

res <- unlist(performParallel(p, rep(1000, 100), fun = myfun, seed = 123,
          initexpr = library(extraDistr), export = c("mean", "sd")))
hist(res)
```

```
# See example in ?snowFT for plotting cluster usage.

## End(Not run)
```

---

snowFT-rand *Random Number Generation*

Description

Initialize independent random number streams to be used in the cluster. It uses the L'Ecuyer's random number generator implemented in the **rlecuyer** package.

Usage

```
clusterSetupRNG.FT (cl, type = "RNGstream", streamper="replicate", ...)
clusterSetupRNGstreamRepli (cl, seed=rep(12345,6), n, ...)
```

Arguments

`cl`	Cluster object.
`type`	Type of the RNG. Currently, only `"RNGstream"` is supported. It initializes the L'Ecuyer's RNG.
`streamper`	Mode of the inititalization. Value `"node"` initializes one random number stream per node. Value `"replicate"` initializes one stream per replicate.
`...`	Arguments passed to the underlying function (see details bellow).
`seed`	A single integer or a vector of six integer values used as seed for the RNG.
`n`	Number of streams to be created. It should correspond to the number of replicates in the computation.

Details

`clusterSetupRNG.FT` calls one of the following functions, while passing arguments `(cl, ...)`: For `streamper="node"`, the snow function `clusterSetupRNGstream` is called; For `streamper="replicate"`, the function `clusterSetupRNGstreamRepli` is called. In the latter case, the argument `n` has to be given that corresponds to the total number of streams created for the computation. This mode is used by `clusterApplyFT`. Note that when using the function `performParallel`, the user does not need to initialize the RNG separately, since it is accomplished within the function.

`clusterSetupRNGstreamRepli` loads the **rlecuyer** package and on each node it creates `n` streams. The streams are named by their ordinal number.

Examples

```
## Not run:
# Generate 50 independent (normally distributed) random numbers
# on 3 nodes using 10 RNG streams
cl <- makeClusterFT(3)
r <- 10
# reproducible results
for (i in 1:3) {
    clusterSetupRNG.FT(cl, streamper = "replicate", n = r, seed = 123)
    cat("\n")
    print(unlist(clusterApplyFT(cl, rep(5,r), rnorm, gentype = "RNGstream")[[1]]))

}

# non-reproducible results (method used in snow)
for (i in 1:3) {
    clusterSetupRNG.FT(cl, streamper = "node", seed = 123)
    cat("\n")
    print(unlist(clusterApplyFT(cl, rep(5,r), rnorm, gentype = "RNGstream")[[1]]))
}
stopClusterFT(cl)

## End(Not run)
```

---

snowFT-startstop *Starting snowFT Cluster*

Description

Functions to start and stop a snowFT cluster and to set default cluster options.

Usage

```
makeClusterFT(spec, type = getClusterOption("type"),
    names = NULL, ft_verbose = FALSE, ...)

stopClusterFT(cl)
```

Arguments

spec	Cluster size.
type	Character string that specifies cluster type. "MPI" and "SOCK" are supported ("PVM" is currently not available).
names	Used only for the 'SOCK' layer. It should be a vector of host names, or a list containing specification for each host (see Example in makeCluster). Due to the dynamic resizing feature, the length of this vector (or list) does not need to match the size of the cluster spec - it is used as a pool from which hosts are taken as they are needed. If names is NULL, each node is started on 'localhost'.

ft_verbose	If TRUE, debugging messages are sent to standard output.
...	Cluster option specifications. A useful option is outfile, specifying name of a file to which slave node output is to be directed.
cl	Cluster object.

Details

makeClusterFT starts a cluster of the specified or default type, loads the **snowFT** library on each node and returns a reference to the cluster. See makeCluster for more details.

stopClusterFT stops the cluster.

See Also

snow-startstop functions of the snow package.

Examples

```
## Not run:
cl <- makeClusterFT(5, ft_verbose = TRUE)
res <- clusterApplyFT(cl, 1:10, get("+"), y = 3)
stopClusterFT(res[[2]])
print(unlist(res[[1]]))

## End(Not run)
```

Index