

# Package ‘snowFT’

February 15, 2012

**Title** Fault Tolerant Simple Network of Workstations

**Version** 1.2-0

**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.

**Maintainer** Hana Sevcikova <hanas@uw.edu>

**License** GPL (>= 2)

**Depends** R (>= 2.11), snow (>= 0.3-3)

**Suggests** rpvm, rmpi, rlecuyer, rsprng

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

**Repository** CRAN

**Date/Publication** 2010-06-25 05:46:36

## R topics documented:

snowFT-package	2
snowFT-cluster	3
snowFT-process	6
snowFT-rand	7
snowFT-repair	8
snowFT-startstop	9

<b>Index</b>	<b>11</b>
--------------	-----------

---

snowFT-package

*Fault Tolerant Simple Network of Workstations*

---

## 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. It supports the PVM, MPI as well as the socket communication layers.

## Details

Package: snowFT  
Version: 1.2-0  
License: GPL  
Depends: R (>= 2.11), snow (>= 0.3-3)  
Suggests: rpvm, rmpi, rlecuyer, rsprng

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.
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**).
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.
5. No administration overhead: 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 through one single function - `performParallel`.

Note that not all features are available for all communication layers. The fault tolerance is currently available only for PVM. Dynamic resizing is not available for MPI.

**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)))

## End(Not run)
```

---

snowFT-cluster

*Cluster-Level Functions with Fault Tolerance Features*

---

**Description**

Functions that extend the collection of cluster-level functions of the **snow** package while providing fault tolerance, reproducibility and additional management features. The heart of the package is the function `performParallel`.

**Usage**

```
performParallel(count, x, fun, initfun = 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, 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 can be used for example for loading required libraries.
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 <a href="#">makeClusterFT</a> ). Possible values are 'PVM', 'MPI' and 'SOCK'.
cluster.args	List of arguments passed to the function <a href="#">makeClusterFT</a> . 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 <a href="#">makeCluster</a> ). 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 <a href="#">clusterSetupRNG.FT</a> . If gentype="None", no RNG action is taken.
seed, prngkind, para	Seed, kind and parameters for the RNG (see <a href="#">clusterSetupRNG.FT</a> ).
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 is only available for PVM. Furthermore, the dynamic cluster resizing is not available for MPI.

<code>ft_verbose</code>	If TRUE, debugging messages are sent to standard output.
<code>nodes</code>	Indices of cluster nodes.
<code>expr</code>	Expression to evaluate.
<code>...</code>	Additional arguments to pass to function <code>fun</code> .

## Details

`clusterApplyFT` is a fault tolerant version of `clusterApplyLB` of the `snow` package with additional features, such as results reproducibility, computation transparency and dynamic cluster resizing. The master process searches for failed nodes in its waiting time. If failures are detected, the cluster is repaired. All failed computations are restarted (in three additional runs) after the replication loop is finished, and hence the user should not notice any interruptions.

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. Then 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`, `exitfun` in `clusterApplyFT` 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.

`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` 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,
  gentype="RNGstream", seed=rep(1,6), printfun=printfun)

# Setting p<-0 will run rnorm sequentially and should give
# exactly the same results

## End(Not run)
```

---

snowFT-process

*Process control*


---

**Description**

Functions for process control in a cluster.

**Usage**

```
processStatus(node)
findFailedNodes(cl)
```

**Arguments**

cl	Cluster object.
node	Node object.

**Details**

processStatus checks the existence of a node.

findFailedNodes identifies all failed nodes in the cluster.

These functions are available only for the PVM layer.

**Value**

processStatus returns FALSE if the node does not exist, otherwise TRUE.

findFailedNodes returns a matrix with one row per failed node and three columns: 1. node index within the cluster, 2. number of the last replication computed on that node, 3. node id.

**Author(s)**

Hana Sevcikova

---

snowFT-rand

*Random Number Generation*

---

**Description**

Initialize independent random number streams to be used in the cluster. It uses either the L'Ecuyer's random number generator (package rlecuyer required) or the SPRNG generator (package rsprng required).

**Usage**

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

**Arguments**

cl	Cluster object.
type	Type of the RNG. "RNGstream" initializes the L'Ecuyer's RNG. Type "SPRNG" initializes the SPRNG generator.
streamper	Mode of the initialization. Value "node" initializes one random number stream per node. Value "replicate" initializes one stream per replicate.
...	Arguments passed to the underlying function (see details below).
seed	Integer value (SPRNG) or a vector of six integer values (RNGstream) 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 (subject to its argument values) one of the following functions, passing arguments (cl, ...): If the "SPRNG" type is used, then in case of streamper="node" the snow function [clusterSetupSPRNG](#) is called. In case of streamper="replicate", the function only checks the availability of the rsprng package on nodes but no initialization will be done at this point. If the "RNGstream" type is used, then in case of streamper="node" the snow function [clusterSetupRNGstream](#) is called and in case of 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 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:
cl <- makeClusterFT(3)
r<-10
clusterSetupRNG.FT(cl, streamper="replicate",n=r, seed=rep(1,6))
res<-clusterApplyFT(cl,rep(5,10),rnorm)
stopCluster(res[[2]])
print(res[[1]])

## End(Not run)
```

---

snowFT-repair

*Repairing a Cluster*

---

## Description

Functions to add, remove and restart nodes of a snowFT cluster.

## Usage

```
addtoCluster(cl, spec, ..., options = defaultClusterOptions)
removefromCluster(cl, nodes, ft_verbose = FALSE)
repairCluster(cl, nodes, ..., options = defaultClusterOptions)
```

## Arguments

<code>spec</code>	Cluster specification. Using PVM, it is the number of additional nodes to create.
<code>cl</code>	Cluster object.
<code>nodes</code>	Indices of nodes.
<code>options</code>	Cluster options object.
<code>ft_verbose</code>	If TRUE, debugging messages are sent to standard output.
<code>...</code>	Cluster option specifications.

**Details**

`addtoCluster` adds new nodes to cluster `cl`. For "PVM" cluster the `spec` argument should be an integer specifying the number of nodes to create.

`removefromCluster` removes nodes given by indices `nodes` from cluster `cl`.

`repairCluster` replaces nodes (given by indices `nodes`) by new created nodes and loads snowFT on the new nodes.

Cluster options used in `addtoCluster` and `repairCluster` should be the same as used for creating the cluster `cl` (see [makeClusterFT](#)).

**Value**

All three functions return the updated cluster object.

**Author(s)**

Hana Sevcikova

**Examples**

```
## Not run:
# create a cluster of size five
cl <- makeClusterFT(5)
clusterApply(cl, 1:5, get("+"), 3)
# add three nodes
cl <- addtoCluster(cl,3)
printClusterInfo(cl)
# remove nodes 3 and 7
cl <- removefromCluster(cl, c(3, 7), ft_verbose=TRUE)
# check for failures
fail<-findFailedNodes(cl)
cl<-repairCluster(cl,fail[,1])
# stop cluster
stopCluster(cl)

## End(Not run)
```

---

snowFT-startstop

*Starting snowFT Cluster*


---

**Description**

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

**Usage**

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

## Arguments

spec	Cluster size.
type	Character string that specifies cluster type. "PVM", "MPI" and "SOCK" are supported.
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 <a href="#">makeCluster</a> ). 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 <code>outfile</code> , specifying name of a file to which slave node output is to be directed.

## 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.

The cluster should be stopped by [stopCluster](#) of the **snow** package.

## See Also

`snow-startstop` functions of the **snow** package.

## Examples

```
## Not run:
c1 <- makeClusterFT(5)
res <- clusterApplyFT(c1, 1:10, get("+"), 3)
stopCluster(res[[2]])
print(res[[1]])

## End(Not run)
```

# Index

- \*Topic **package**
  - snowFT-package, [2](#)
- \*Topic **programming**
  - snowFT-cluster, [3](#)
  - snowFT-package, [2](#)
  - snowFT-process, [6](#)
  - snowFT-rand, [7](#)
  - snowFT-repair, [8](#)
  - snowFT-startstop, [9](#)

[addtoCluster \(snowFT-repair\)](#), [8](#)

[clusterApplyFT](#), [8](#)  
[clusterApplyFT \(snowFT-cluster\)](#), [3](#)  
[clusterCall](#), [3](#)  
[clusterCallpart \(snowFT-cluster\)](#), [3](#)  
[clusterEvalQpart \(snowFT-cluster\)](#), [3](#)  
[clusterSetupRNG.FT](#), [4](#)  
[clusterSetupRNG.FT \(snowFT-rand\)](#), [7](#)  
[clusterSetupRNGstream](#), [7](#)  
[clusterSetupRNGstreamRepli](#)  
    ([snowFT-rand](#)), [7](#)  
[clusterSetupSPRNG](#), [7](#)

[findFailedNodes \(snowFT-process\)](#), [6](#)

[makeCluster](#), [4](#), [10](#)  
[makeClusterFT](#), [4](#), [9](#)  
[makeClusterFT \(snowFT-startstop\)](#), [9](#)  
[makeSOCKclusterFT \(snowFT-startstop\)](#), [9](#)

[performParallel](#), [2](#), [3](#), [8](#)  
[performParallel \(snowFT-cluster\)](#), [3](#)  
[printClusterInfo \(snowFT-cluster\)](#), [3](#)  
[processStatus \(snowFT-process\)](#), [6](#)

[removefromCluster \(snowFT-repair\)](#), [8](#)  
[repairCluster \(snowFT-repair\)](#), [8](#)

[snowFT \(snowFT-package\)](#), [2](#)  
[snowFT-cluster](#), [3](#)

[snowFT-package](#), [2](#)  
[snowFT-process](#), [6](#)  
[snowFT-rand](#), [7](#)  
[snowFT-repair](#), [8](#)  
[snowFT-startstop](#), [9](#)  
[stopCluster](#), [10](#)