

Package ‘snowFT’

June 2, 2009

Title Fault Tolerant Simple Network of Workstations

Version 1.0-1

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. It is written for the PVM communication layer.

Maintainer Hana Sevcikova <hanas@u.washington.edu>

License GPL (>= 2)

Depends R (>= 2.3), snow (>= 0.3-3), rpvm (>= 1.0.2)

Suggests rlecuyer, rsprng

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

Repository CRAN

Date/Publication 2009-06-02 15:42:55

R topics documented:

snowFT-package	2
snowFT-cluster	3
snowFT-process	5
snowFT-rand	6
snowFT-repair	7
snowFT-startstop	8

Index	10
--------------	-----------

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 is written for the PVM communication layer.

Details

Package: snowFT
 Version: 1.0-1
 License: GPL
 Depends: R (>= 2.3), snow (>= 0.3-3), rpvm (>= 1.0.2)

Index:

addtoCluster	Repairing a Cluster
clusterSetupRNG.FT	Random Number Generation
makeClusterFT	Starting snowFT Clusters
performParallel	Cluster-Level Functions with Fault Tolerant Features
processStatus	Process control

The package supports a snow cluster that is:

1. Fault tolerant: The master checks repeatedly for failure 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. Easy to use: All features, including creating the cluster, RNG initialization and clean-up, are available through one single function - `performParallel`.

Author(s)

Hana Sevcikova, A. J. Rossini

Maintainer: Hana Sevcikova <hanas@u.washington.edu>

References

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

See Also

[snow-package](#), [performParallel](#)

Examples

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

snowFT-cluster

Cluster-Level Functions with Fault Tolerant Features

Description

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

Usage

```
clusterApplyFT(cl, x, fun, initfun, exitfun, printfun, printargs, printrepl,
              gentype, seed, prngkind, para, mngtfiles, ft_verbose, ...)

performParallel(count, x, fun, initfun, exitfun, printfun, printargs,
              printrepl, cltype, gentype, seed, prngkind, para, mngtfiles,
              ft_verbose, ...)

clusterCallpart(cl, nodes, fun, ...)
clusterEvalQpart(cl, nodes, expr)
```

Arguments

<code>cl</code>	Cluster object.
<code>count</code>	Number of cluster nodes.
<code>fun</code>	Function or character string naming a function.
<code>x</code>	Array whose length determines how many times <code>fun</code> is to be called. <code>x[i]</code> is passed to <code>fun</code> (as its first argument) at <i>i</i> th call.
<code>initfun</code>	Function or character string naming a function with no arguments that is to be called on each node prior to the computation. Default: <code>NULL</code> .

<code>exitfun</code>	Function or character string naming a function with no arguments that is to be called on each node after the computation is completed. Default: <code>NULL</code> .
<code>printfun, printargs, printrepl</code>	<code>printfun</code> is a function or character string naming a function that is to be called on the master node after each <code>printrepl</code> completed replicates, and thus it can be used for accessing intermediate results. Arguments passed to <code>printfun</code> are a list (of length <code> x </code>) of results (including the non-finished ones), the number of finished results, and <code>printargs</code> . Defaults: <code>printfun=printargs=NULL, printrepl=max(length(x)/10,1)</code> .
<code>cltype</code>	Character string that specifies cluster type (see <code>makeClusterFT</code>). Default: <code>getClusterOption("type")</code> .
<code>gentype</code>	Character string that specifies type of the used RNG. Possible values: "RNGstream" (default for <code>performParallel</code>) - L'Ecuyer's RNG, "SPRNG", or "None" (default for <code>clusterApplyFT</code>). See <code>clusterSetupRNG.FT</code> . If <code>gentype="None"</code> , no RNG action is taken.
<code>seed, prngkind, para</code>	Seed, kind and parameters for the RNG (see <code>clusterSetupRNG.FT</code>). Defaults: <code>seed=rep(123456,6), prngkind="default", para=0</code> .
<code>mngtfiles</code>	A character vector of length 3 containing names of management files: <code>mngtfiles[1]</code> for managing the cluster size, <code>mngtfiles[2]</code> for storing the replicates being currently computed, <code>mngtfiles[3]</code> for storing the failed replicates. If any of these files equals an empty string, the corresponding management actions are not performed. If the files already exist, their content is overwritten. Default: <code>c(".clustersize", ".proc", ".proc_fail")</code> .
<code>ft_verbose</code>	If <code>TRUE</code> , debugging messages are sent to standard output.
<code>expr</code>	Expression to evaluate.
<code>nodes</code>	Indices of cluster nodes.
<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]` 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.

`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`.

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)
## 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.

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="RNGstream" (default) initializes the L'Ecuyer's RNG. type="SPRNG" initializes the SPRNG generator.
streamper	streamper="node" initializes one random number stream per node. streamper="replicate" (default) 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 `performParalell`, 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)
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>...</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:
cl <- makeClusterFT(5)
clusterApply(cl, 1:5, get("+"), 3)
cl <- addtoCluster(cl,3)
fail<-findFailedNodes(cl)
cl<-repairCluster(cl,fail[,1])
## End(Not run)
```

snowFT-startstop *Starting snowFT Clusters*

Description

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

Usage

```
makeClusterFT(spec, type = getClusterOption("type"), ...)
```

Arguments

<code>spec</code>	Cluster specification.
<code>type</code>	Character string that specifies cluster type. Only <code>type="PVM"</code> supported.
<code>...</code>	cluster option specifications

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. Only the cluster type "PVM" is supported in this version. The `spec` argument should be an integer specifying the number of slave nodes to create. See `setDefaultClusterOptions()` of the snow package for setting cluster options.

The cluster should be stopped by `stopCluster` of the snow package.

See Also

`snow-startstop` functions of the snow package.

Examples

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

## End(Not run)
```

Index

*Topic **package**

snowFT-package, 1

*Topic **programming**

snowFT-cluster, 3

snowFT-package, 1

snowFT-process, 5

snowFT-rand, 6

snowFT-repair, 7

snowFT-startstop, 8

snowFT (*snowFT-package*), 1

snowFT-cluster, 3

snowFT-package, 1

snowFT-process, 5

snowFT-rand, 6

snowFT-repair, 7

snowFT-startstop, 8

addtoCluster (*snowFT-repair*), 7

clusterApplyFT, 7

clusterApplyFT (*snowFT-cluster*), 3

clusterCallpart (*snowFT-cluster*),
3

clusterEvalQpart
(*snowFT-cluster*), 3

clusterSetupRNG.FT, 4

clusterSetupRNG.FT (*snowFT-rand*),
6

clusterSetupRNGstreamRepli
(*snowFT-rand*), 6

findFailedNodes (*snowFT-process*),
5

makeClusterFT, 4, 7

makeClusterFT (*snowFT-startstop*),
8

performParallel, 2

performParallel (*snowFT-cluster*),
3

processStatus (*snowFT-process*), 5

removefromCluster
(*snowFT-repair*), 7

repairCluster (*snowFT-repair*), 7

snow-package, 2