Developing parallel programs using snowfall

Jochen Knaus

2010-03-04

Abstract

`snowfall` is an R package for easier parallel programming using clusters. Basically it is build upon the package `snow` [4] using it's network and cluter abilities and therefore offering use of Socket, MPI, PVM and NetWorkSpaces support and can be seen as an "usability wrapper".

`snow` functions can used from within `snowfall` as well.

`snowfall` offers additional support for implicit sequential execution (e.g. for distributing packages using optional parallel support), additional calculation functions, extended error handling, and many functions for more comfortable programming.

Also, `snowfall` can be configured via command line arguments, making the change of cluster settings easier without program change. This can be used to connect to batch- and workloadmanagers.

Finally `snowfall` can be directly connected to the R-specific cluster manager *sfCluster*.

`snowfall` does not add an technical layer of abstraction to `snow`. But beside from the connector to `sfCluster`, it builds an extra layer of usability on the top of `snow`.

It is not thought as an replacement for `snow`, but an addition for inexperienced users or those who seek more comfort using parallel computing and R.

A further introduction to snowfall is published in the R-Journal [2].

For additional documentation, help and examples please visit our website: `http://www.imbi.uni-freiburg.de/parallel`

Contents

1 snowfall

1.1 Getting started

1.1.1 Requirements for sequential execution

Basically, `snowfall` is able to run without any external library. In this case, it is not possible to use parallel execution of commands. All potential calls to parallel functions will be executed sequentially.

Programs written in sequential use with `snowfall` calls can be running in parallel without any code change.

1.1.2 Requirements for parallel execution: Basics

If you just want to use parallel computing on your local PC or laptop you are just fine with basically installation of `snowfall` and `snow`. You can use then a so called socket cluster, for which no additional software needs to be installed.

If you are just wanting to use parallel programming on your local workstation, PC or laptop, you are fine.

1.1.3 Requirements for parallel execution: MPI

You have a running MPI cluster (OpenMPI or any other kind of MPI cluster) available.

Although snowfall is useable with OpenMPI as well, the management software sfCluster can currently only used with LAM/MPI.

1.1.4 Requirements for parallel execution: LAM/MPI

For using sfCluster with snowfall, currently LAM/MPI is needed.

If you are using Debian/Ubuntu Linux, just call
`aptitude install xmpi lam4-dev`[1]

Further you need to install the R-packages `snow` and `Rmpi`.

If your program uses libraries, ensure that these are available on all nodes. If they are not present in R-default path (on given machine), ensure that they are accessible in the same location on all machines (for example `/home/xy/R.libs`).

---

[1]On other Linux distributions there are similar packages with probably different name. It is important that you install the development version of the LAM package, as the `Rmpi` package need these files for installation.

If you want to run programs only on your (multi core) computer without any cluster of many machines, you do not have to setup the cluster yourself, it will be started implicitly in `snowfall`s initialisation.

Using two or more machines for cluster calculations, you need to setup a LAM/MPI cluster and start cluster explicitely.

This is no big thing at all. For example, edit a small textfile like this one:

```
machine1.yourdomain.com cpu=4 sched=yes
machine2.yourdomain.com cpu=2 sched=yes
```

Just enter the machines for your cluster and the amount of CPUs. You start a LAM/MPI cluster using
```
lamboot hostfile
```
where `hostfile` is the little configuration file edited above.

To shutdown just call `lamhalt`.

For further details upon LAM/MPI setup, see [1].

Note: All parallel programs you start are running in this cluster. If your program requests 100 CPUs on your private dual-core machine, you get that amount and 100 R processes are spawn, independent or available ressources (memory, cpus).

For workgroups or larger clusters, management solutions like *sfCluster* are strongly recommended.

1.1.5   Requirements for parallel execution: PVM/NWS

PVM and NetWorkSpaces/Sleight are supported in snowfall as these are useable with snow. But both are less supported by sfCluster (but at least a managed start can be done using sfCluster), so there is no further documentation about their usage here.

1.2   (Short) introduction to parallel programming

The general goal of paralleling your R program is to vectorize the data or calculation loops (probably with wrapper functions), as all calculation functions of `snowfall` are kind of reimplementations of R-list/vector functions.

A good introduction to parallel programming for statistical purposes can be found in [3] and [5].

1.3   Introduction to usage of snowfall

Basically, usage of `snowfall` always works with the following scheme:

1. Initialization using `sfInit()`. Set up the cluster (if needed) and the internal functions. `sfInit` must be called before using any function of the `snowfall` package.[2]

2. Export needed variables/objects to all slaves.

3. Do some parallel calculations using `snowfall` calculation functions. Repeat as many times as needed.

4. End parallel execution using `sfStop()`.

The initialisation differs if you use `snowfall` alone or with the management tool *sfCluster*. In this chapter we only cover a standalone usage of `snowfall`. For usage with *sfCluster*, see chapter 2.

If you are firm on using the R package `snow`, starting with or porting your program to `snowfall` is easy.

The complete initialisation is done with a single call to `sfInit()`. The main arguments are `parallel`, `cpus` and `type`, giving the running mode (parallel execution or sequential execution), the amount of CPUs if executing in parallel mode and the type of the underlying cluster. If running in sequential mode, `cpus` is ignored (and set to one). Without a given `type` a socket cluster is started, which does not need any further software installed and therefore most likely runs anywhere immidiately. This is the desired choice for executing on a laptop or single multicore machine, too. Please note, that on Windows an installed Personal Firewall may alert the network access, please allow this.

Sequential mode can be useful for developing the program, probably on a single core laptop without installed cluster or running Windows operating system. Also sequential mode is needed to deploy a package using `snowfall` safely, where you cannot assume a user have an useable cluster installed.

Other arguments for `sfCluster` are `restore`, `socketHosts`, `slaveOutfile` and `nostart`. See package help for description.

If the initialisation fails, probably because of missing base libraries `Rmpi` and `snow`, `snowfall` falls back to sequential mode with a warning message.

In sequential and parallel execution, all functions are useable in both modes in the same way and returning the same results.

```
sfInit( parallel=FALSE )

sfLapply( 1:10, exp )
```

---

[2]The only exception is the function `sfSetMaxCPUs()`, which raises or limits the configured maximum CPU count.

```
sfStop()

sfInit( parallel=TRUE, cpus=5 )

## Now, index 1 is calculated on CPU1, 2 on CPU2 and so on.
## Index 6 is again on CPU1.
## So the whole call is done in two steps on the 5 CPUs.
sfLapply( 1:10, exp )

sfStop()
```

Please note: Most of the `snowfall` functions are stopping the program on failure by default (by calling `stop()`). This is much safer for unexperienced users. If you want own failure handling, install your own handler `options(error = ...)` to prevent snowfall from stopping in general. Also most of the functions feature an argument `stopOnError` which set to `FALSE` prevents the functions from stopping. Do not forget to handle potential errors in your program if using this feature.

The given behavior is not only better for unexperienced users, any other behavior would be very nasty on package deployment.

1.4 Writing parallel programs with snowfall

1.4.1 General notes and simple example

If you detected parts of your program which can be parallelised (loops etc) it is in most cases a fast step to give them a parallel run.

First, rewrite them using Rs list operators (lapply, apply) instead of loops (if they are not yet calculated by list operators).

Then write a wrapper function to be called by the list operators and manage a single parallel step. Note there are no local variables, only the data from the list index will be given as argument.

If you need more than one variable argument, you need to make the required variables global (assign to global environment) and export them to all slaves. `snowfall` provides some functions to make this process easier (take a look at the package help).

```
sfInit( parallel=TRUE, cpus=4 )

b <- c( 3.4, 5.7, 10.8, 8, 7 )

## Export a and b in their current state to all slaves.
```

```
sfExport( ''b'' )

parWrapper <- function( datastep, add1, add2 ) {
  cat( ''Data: '', datastep, ''ADD1:'', add1, ''ADD2:'', add2, ''\n'' )

  ## Only possible as ''b'' is exported!
  cat( ''b:'', b[datastep] )

  ## Do something

  return( datastep )
}

## Calls parWrapper with each value of a and additional
## arguments 2 and 3.
result <- sfLapply( 1:5, parWrapper, 2, 3 )

sfStop()
```

1.4.2 Basic load balancing using `sfClusterApplyLB`

All parallel wrappers around the R-list operators are executed in blocks: On one step the first n indices are calculated, then the next n indices, where n is the number of CPUs in the cluster.

This behavior is quite ok in a homogenous cluster, where all or mostly all machines are built with equal hardware and therefore offer the same speed. In heterogenous infrastructures, speed is depending on the slowest machine in the cluster, as the faster machines have to wait for it to finish its calculation.

If your parallel algorithm is using different time for different problems, load balancing will reduce overall time in homogenous clusters greatly.

snow and so **snowfall** feature a simple load balanced method to avoid waiting times in such environments. If calling `sfClusterApplyLB` the faster machines get further indices to calculate without waiting for the slowest to finish its step. `sfClusterApplyLB` is called like `lapply`.

If your local infrastructure is such an heterogenous structure, this function is the way to go. It can also be handy in homogenous clusters where other users spawn processes, too, so sometimes load differs temporarily.

A visualisation of basic load balacing can be found in [3].

```
sfInit( parallel=TRUE, cpus=2 )

calcPar <- function( x ) {
```

```
  x1 <- matrix( 0, x, x )
  x2 <- matrix( 0, x, x )

  for( var in 1:nrow( x1 ) ) x1[var,] = runif( ncol( x1 ) )
  for( var in 1:nrow( x2 ) ) x2[var,] = runif( ncol( x1 ) )

  b <- sum( diag( ( x1 %*% x2 ) %*% x1 ) )
  return( b )
}

result <- sfClusterApplyLB( 50:100, calcPar )

sfStop()
```

1.4.3 Intermediate result saving and restoring using `sfClusterApplySR`

Another helpful function for long running clusters is `sfClusterApplySR`, which saves intermediate results after processing n-indices (where n is the amount of CPUs). If it is likely you have to interrupt your program (probably because of server maintenance) you can start using `sfClusterApplySR` and restart your program without the results produced up to the shutdown time.

Please note: Only complete n-blocks are saved, as the function `sfLapply` is used internally.[3]

The result files are saved in the temporary folder `/.sfCluster/RESTORE/x`, where x is a string with a given name and the name of the input R-file.

`sfClusterApplySR` is called like `sfClusterApplyLB` and therefore like `lapply`.

If using the function `sfClusterApplySR` result are always saved in the intermediate result file. But, if cluster stopped and results could be restored, restore itself is only done if explicitly stated. This aims to prevent false results if a program was interrupted by intend and restarted with different internal parameters (where with automatical restore probably results from previous runs would be inserted). So handle with care if you want to restore!

If you only use one call to `sfClusterApplySR` in your program, the parameter `name` does not need to be changed, it only is important if you use more than one call to `sfClusterApplySR`.

```
sfInit( parallel=TRUE, cpus=2 )

# Saves under Name default
```

---

[3]This function is an addition to `snow` and therefore could not be integrated in the load balanced version.

```
resultA <- sfClusterApplySR( somelist, somefunc )

# Must be another name.
resultB <- sfClusterApplySR( someotherlist, someotherfunc, name="CALC_TWO" )

sfStop()
```

If cluster stops probably during run of `someotherfunc` and restarted with restore-Option, the complete result of `resultA` is loaded and therefore no calculation on `somefunc` is done. `resultB` is restored with all the data available at shutdown and calculation begins with the first undefined result.

Note on restoring errors: If restoration of data fails (probably because list size is different in saving and current run), `sfClusterApplySR` stops. For securely reason it does not delete the RESTORE-files itself, but prompt the user the complete path to delete manually and explicitly.

1.5   Fault tolerance

Differing from `snowFT`, the fault tolerance extension for `snow`, `snowfall` does not feature fault tolerance (see [6]).

This is due to the lack of an MPI implementation of `snowFT`.

1.6   Controlling snowfall using the command line

snowfall can be widely controlled via command line arguments.

This is useful for fast changing of cluster parameters (e.g. changing the host names in a Socket cluster) on a raw installation and it serves as connection to sfCluster. Of course it can be used as connection to any other workload- or batch managing software, too.

On the commandline there are the following parameters:

parallel	Switch to parallel execution. Default is sequential execution
cpus=X	Amount of CPUs wanted. Without --parallel, a value $X > 1$ switch to parallel execution.
type=X	Type of cluster. Allowed values are SOCK, MPI, PVM and NWS.
session=X	Session number. snowfall logfiles contain number, but only needed with sfCluster.
restoreSR	Enables restoring of previously saved results from `sfClusterApplySR` calls.
hosts=X	List of hosts for Socket (SOCK) or NetWorkSpaces (NWS) clusters. Entries are comma seperated. Any entry may contain colon seperated value for the amount of processors on this machine. Example: `--hosts=machine1:4,machine2,123.123.12.13:2` (this spawns 4 workers on machine1, one on machine2 and two on 123.123.12.13).
tmpdir=X	Specify temporary directory for logfiles and R-output.

For using these arguments, just add these after an `-args` on the commandline (which forces R not to treat these arguments as R ones).

```
R -no-save -args -parallel -cpus=2 < program.R
```

Starts R and forces snowfall to start in parallel mode with 2 CPUs (in this case: using a Socket-cluster, as this is the default).

Note: arguments on the command line have lower priority as settings from the `sfInit` call. That means that the above example only works if initialisation is done via `sfInit()`, but not with `sfInit( parallel=FALSE )`, as then sequential execution is forced.

Further examples should explan the feature:

- `R -no-save -args -parallel -type=MPI -cpus=4 < program.R` (start using 4 workers in an existing MPI cluster. If no MPI cluster exists, a plain one is started on your local machine only. Beware of this, as you have to shutdown this cluster afterwards manually.).

- `R -no-save -args -parallel -type=SOCK -hosts=localhost:3,singlema,othmach:4 < program.R` (Starts a socket cluster with two machines and 7 CPUs: 3 on `localhost`, 4 on `othmach` and one worker on `singlema`).

1.7 Traps, Internals

`snowfall` limits the amount of CPUs by default (to 40). If you need more CPUs, call `sfSetMaxCPUs()` *before* calling `sfInit()`. Beware of requesting more CPUs as you have ressources: there are as many R processes spawned as CPUs wanted. They are distributed across your cluster like in the given scheme of the LAM

host configuration. You can easily kill all machines in your cluster by requesting huge amounts of CPUs or running very memory consuming functions across the cluster. To avoid such common problems use *sfCluster*.

For some functions of `snowfall` it is needed to create global variables on the master. All these variables start with prefix ".`sf`", please do not delete them. The internal control structure of `snowfall` is saved in the variable .`sfOptions`, which should be accessed through the wrapper functions as the structure may change in the future.

2 Using *sfCluster* with `snowfall`

2.1 About *sfCluster*

sfCluster is a small management tool, helping to run parallel R-programs using `snowfall`. Mainly, it exculpates the user from setting up a LAM/MPI cluster on his own. Further, it allows multiple clusters per user and therefore executes any parallel R program in a single cluster. These clusters are built according to the current load and usage of your cluster (this means: only machines are taken with free ressources).

Also, execution is observed and if problems arise, the cluster is shut down.

sfCluster can be used with R-interactive shell or batch mode and also feature a special batch mode with visual logfile and process-displaying.

For further details about installation, administration and configuration of *sfCluster*, please visit `http://www.imbi.uni-freiburg.de/parallel` or run `sfCluster --help` if you installed it yet.

2.2 Starting R using *sfCluster*

An *sfCluster* execution is following these steps:

1. Test memory usage of program if not explicitly given. This is done via a default temporary (10 minutes) sequential run to determinate the maximum usage of RAM on a slave. This is important for allocating ressources on slaves.

2. Detect free ressources in cluster universe.[4] Take machines with free ressources matching users request.

3. Start LAM/MPI cluster with previous built setting.

---

[4]Which are all potentially useable machines.

4. Run R with parameters for `snowfall` control.

5. LOOP: Observe execution (check processes, memory usage, and machine state). In monitoring mode: Display state of cluster and logfiles on screen.

6. On interruption or regular end: shutdown cluster.

2.3   Using *sfCluster*

The most common parameters of *sfCluster* are `--cpus`, with which you request a certain amount of CPUs among the cluster (default is 2 in parallel and 1 in sequential mode). There is a builtin limit for the amount of CPUs, which is changeable using the *sfCluster* configuration.

There are four execution modes:

-b	Batchmode (Default)	Run silent on terminal.
-i	Interactive R-shell	Ability to use interactive R-shell with cluster.
-m	Monitoring mode	Visual processmonitor and logfile viewer.
-s	Sequential execution (no cluster usage)	Run without cluster on single CPU.

To avoid the (time consuming) memory test, you can specify a maximum amount of memory usable per slave via option `--mem`. The behavior on excessing this memory usage is configurable (default: cluster stop).

The memory usage limit is very important for not getting your machines into swapping (means: shortage of physical RAM), which would hurt performance badly.

So, simple calls to *sfCluster* could be

```
## Run a given R program with 8 cpus and max. 500MB (0.5 gigabytes) in monitoring mode
sfCluster -m --cpus=8 --mem=0.5G myRprogram.R

## Run nonstopping cluster with real quiet output.
nohup sfCluster -b --cpus=8 --mem=500M myRprogram.R --quiet

## Start R interactive shell with 4 cores. With 300MB memory (MB is default unit)
## No R-file is given for interactive mode.
sfCluster -i --cpus=4 --mem=300
```

For all possible options and further examples for *sfCluster* usage, see `sfCluster --help`.

2.4 The snowfall-side of *sfCluster*

If you start an R program using `snowfall` with *sfCluster*, the latter waits until `sfInit()` is called and then starts the observation of the execution.

The default behavior if using *sfCluster* is just to call `sfInit()` without any argument. Use arguments only if you want to explicitly overwrite given settings by *sfCluster*.

2.5 Proposed development cycle

The following development cycle is of course a proposal. You can skip or replace any step depending on your own needs.

1. Develop program in sequential mode (start using option `-s`).

2. Test in parallel mode using interactive mode to detect directly problems on parallelisation (start using option `-i`).

3. Try larger test runs using monitoring mode, observing the cluster and probably side effects during parallel execution (start using option `-m`). Problems arise on single nodes will be visible (like non correct working libraries).

4. Do real runs using silent batch mode (start using options `-b --quiet`). Probably you want to run these runs in the background of your Unix shell using `nohup`.

2.6 Future sfCluster

These additions are planned for the future:

- Port to OpenMPI

- Faster SSH connections for observing

- Extended scheduler for system ressources

3 History of snowfall changes

You can also call: RShowDoc("NEWS", package="snowfall")

- 1.83 (API changes: minor additions)

– sfIsRunning: new function giving a logical is sfInit() was called or not. Needed, as all other snowfall functions implicitely call sfInit() if it was not called.

- 1.82

    – Internal refactorings.

- 1.81

    – Change in sfInit() MPI startup so sfCluster can run with snow > 0.3 now.
    – sfExport now also works in sequential mode (writing to global environment). This prevented sequential execution in some cases.

- 1.80 (API changes: minor additions)

    – snowfall passes packages checks of R 2.10.1 without warning or error. Internal state is now only saved in the namespace itself (thanks to Uwe Ligges for the tipp).
    – sfExport can now also export objects in a specific namespace (argument 'namespace')
    – sfExport: behavior in error case manageable (stopOnError)
    – sfExport: smaller bugfixes.
    – sfRemoveAll can now also remove hidden names (argument 'hidden')
    – sfRemoveAll is more robust now (some minor bugfixes, more checks)
    – sfRemoveAll bugfix for multiple removals (thanks to Greggory Jefferis)
    – Bugfix on exception list on sfExportAll
    – Refactorings in sfTest()
    – snowfall now has a NEWS doc ;)
    – No warning on Mac OS because of default Mac-R command line arg 'gui' (thanks to Michael Siegel).

- 1.71 (API changes: none)

    – Exporting of objects using `sfExport` is speed up (round 30
    – Fixed a bug on Windows in `sfSource`

- 1.70 (API changes: minor additions, BEHAVIOR CHANGES: logging)

    – Behavior change: new default: no logging of slave/worker output.
    – API change: new argument `slaveOutfile` on `sfInit()`.

- API change: new argument `restore` on `sfInit()`.
- API change: new argument `master` on `sfCat`.
- Windows startup fixed.
- NWS startup fixed.
- sfSapply is working as intended.
- Changing CPU amount during runtime (with multiple sfInit() calls with different settings in a single program) is now possible using socket and NWS clusters.
- Dozens of small glitches inside snowfall fixed (also messages are made more precisly).
- Package vignette slightly extended.

References

[1] Greg Burns, Raja Daoud, and James Vaigl. LAM: An Open Cluster Environment for MPI. Technical report, 1994. `http://www.lam-mpi.org/download/files/lam-papers.tar.gz`.

[2] Jochen Knaus, Christine Porzelius, Harald Binder, and Guido Schwarzer. Easier parallel computing in R with snowfall and sfCluster. *The R Journal*, 1:54–59, 2009.

[3] A.J. Rossini, Luke Tierney, and Na Li. Simple parallel statistical computing in R. *Journal of Computational and Graphical Statistics*, 16(2):399–420, 2007.

[4] A.J. Rossini, Luke Tierney, and Na Li. Snow : A parallel computing framework for the R system. *International Journal of Parallel Programming*, 2008. Online Publication: `http://www.springerlink.com/content/3v37mg0k63053567` (2008-12-23).

[5] Hana Ševčíková. Statistical simulations on parallel computers. *Journal of Computational and Graphical Statistics*, 13(4):886–906, 2004.

[6] Hana Ševčíková and A.J. Rossini. Pragmatic parallel computing. submitted to journal of statistical software. 2004.