’PARConnector’

December 27, 2013

`PA` *Creates parallel PATasks which can be submitted to PASolve*

Description

`PA` creates a list of PATasks using a syntax similar to mapply. Where mapply applies multi-parameters to a function, PA will create multi-parameter remote executions of a given function.

Usage

```
PA(funcOrFuncName, ..., varies = NULL, input.files = list(),
  output.files = list(), in.dir = getwd(), out.dir = getwd(),
  hostname.selection = NULL, ip.selection = NULL,
  property.selection.name = NULL, property.selection.value = NULL,
  isolate.io.files = FALSE, client = PAClient(), .debug = PADebug())
```

Arguments

`funcOrFuncName`	function handle or function name
`...`	arguments of the funcOrFuncName function which will be vectorized over
`varies`	list of varying parameters which can be a parameter number or a parameter name, if NULL (default) then all parameters are varying
`input.files`	a list of input files which will be transferred from the local machine to the remote executions, see Details for more information
`output.files`	a list of output files which will be transferred from the remote executions to the local machine
`in.dir`	in case input files are used, the directory which will be used as base (default to current working directory)
`out.dir`	in.dir in case ouput files are used, the directory which will be used as base (default to current working directory)
`hostname.selection`	can be used to restrict the remote execution to a given machine, wildcards can be used
`ip.selection`	can be used to restrict the remote execution to a given machine given its IP address

`property.selection.name`	can be used to restrict the remote execution to a given JVM resource where the property is set to the according value
`property.selection.value`	is used in combination with property.selection.name
`isolate.io.files`	should input/output files be isolated in the remote executions, default FALSE. If set to TRUE, when input and output files are copied to USER/GLOBAL space or to the NODE execution, they will be isolated in a folder specific to the current job. It thus guaranties that they will be separated from other jobs execution. On the other hand it will not be possible to reuse the remote files directly in other jobs.
`client`	connection handle to the scheduler, if not provided the handle created by the last call to PAConnect will be used
`.debug`	debug mode

Details

The function can be provided via its name or via a closure object. For builtin functions, it is necessary to provide the name instead of the closure. For user defined function, the function will be analysed and all its depdendencies will be automatically transferred to the remote executions. Dependencies can include other functions or variables defined in the function closure. If the function has a dependency on a package, it's mandatory to manually install and load the package in the remote R executions. PARConnector does not handle automatic package installation. It's of course possible though to do the installation and loading of a package from within the function provided to `PA`

The cardinality (the number of PATask to be created) will be determined by analysing the parameters. If the parameters contains lists or vectors, the biggest length will be the cardinality. Only parameters of the following types logical, integer, character, double, complex, raw, and list will be considered.

It is possible to force unvarying parameters (which will not be taken into account when computing the cardinality), those parameters will be transmitted as they are to the remote evaluations, and will not be scattered. Similarly to mapply, varying lists or vector which are smaller than the cardinality will be extended via looping to match the cardinality. See `mapply` for more information.

When used alone, PA allows to create parallel independant tasks. When used in combination with the two other job conscrution primitives (`PAM` and `PAS`), it allows to create split/merge workflows.

Input/Output Files patterns:
Files path in input.files and output.files list can contain special patterns which are detailed below :

a) Location Patterns*:*
This pattern must be used in the beginning of the path and determines the itinerary of the file from the local computer to the remote compute engine.
The semantic of these patterns varies wether the file is an input file or an output file. It can take the following values :

- "$LOCAL:"

  Input : the LOCAL pattern means that the file path references a file existing on the local machine and will be transferred to the remote node with an intermediate copy in the

USER space. (LOCAL to USER , USER to NODE)
Output : the LOCAL pattern means that the file will be produced by a remote execution and transferred back to the Local machine, with an intermediate copy in the USER space (NODE to USER , USER to LOCAL)

- "$USER:"

  Input : the USER pattern means that the file path references a file existing on the USER space and will be transferred to the remote node (USER to NODE)
  Output : the USER pattern means that the file will be produced by a remote execution and transferred back to the USER space (NODE to USER)

- "$GLOBAL:"

  Input : the USER pattern means that the file path references a file existing on the GLOBAL space and will be transferred to the remote node (GLOBAL to NODE)
  Output : the USER pattern means that the file will be produced by a remote execution and transferred back to the GLOBAL space (NODE to GLOBAL)

- "$LOCAL:$USER:" and "$LOCAL:$GLOBAL:"

  Input : Only valid for input files. It is the explicit version of the $LOCAL: pattern for input files (which is equivalent to $LOCAL:$USER:), this notation allows to choose the GLOBAL space instead of the USER space as intermediate

- "$USER:$LOCAL:" and "$GLOBAL:$LOCAL:"

  Output : Only valid for output files. It is the explicit version of the $LOCAL: pattern for output files(which is equivalent to $USER:$LOCAL:), this notation allows to choose the GLOBAL space instead of the USER space as intermediate

b) Parameter Patterns*:* This pattern can be used anywhere in the file path and will be replaced by parameters of the funcOrFuncName function taken from the ... list. The pattern is of the form can take the following values :

- An integer i
  In that case %expr% refers to the parameter at index i. For each individual PATask created by the PA call, the %expr% pattern will be replaced by the value of the parameter i for this execution. If this value is a scalar value V, the pattern will generate a single input/output file containing the toString coercion of V. If this value is a vector or list, the pattern will generate multiple input/output files with replacements taken from the vector/list.
- A character string S
  In that case %expr% refers to the parameter named S. The semantic is similar to when using an integer parameter reference.
- An integer vector V
  In that case %expr% refers to multiple parameters at index taken from V. It will use the parameter values and generate as many input/output files as elements of V.

The parameter replacement will be done by using the toString coercion on the parameter value, but if the parameter referenced is a PATask (i.e. a result of a PA call), the pattern will be replaced by the same replacements that were done inside this PATask. This is particularly useful when build split-merge workflows, where an initial replacement needs to be transferred to dependant tasks.

Value

a list of PATask objects which can be submitted to the ProActive Scheduler via a `PASolve` call or given as parameter to other `PA`, `PAS` or `PAM` functions

See Also

`PAS` `PAM` `PASolve` `mapply` `PAJobResult` `PAConnect`

Examples

```
## Not run:
 PA("cos", 1:4)     # will produce 4 PATasks : cos(1) , cos(2), cos(3) and cos(4) (parametric sweep with one pa

 PA("sum", 1:4, 1:2)          # will produce 4 PATasks : sum(1,1) , sum(2,2), sum(3,1) and sum(4,2)    (paramet

 PA("c", 1:4, 1:2, varies= list(1) )            # will produce 4 PATasks : c(1,1:2) , sum(2,1:2), sum(3,1:2) a

 PA( function(in,i) file.show(paste0(in,i)),"in", 1:4, input.files="in%2%")     # will produce 4 PATasks which

 PA( function(in,out,i) file.copy(paste0(in,i), paste0(out,i)),"in","out" 1:4, input.files="in%3%", output.f

 To submit tasks simply pass the produced tasks to a PASolve call :

 PASolve(PA("cos", 1:4))

 See examples in  PAS and PAM help sections for split/merge examples


## End(Not run)
```

`PAConnect` *Connects to a ProActive Scheduler*

Description

`PAConnect` connects to a running ProActive Scheduler using its url and login information. The url and login information can be provided inside the PAConnect call or asked interactively.

Usage

```
PAConnect(url, login, pwd, cred = NULL, .print.stack = TRUE)
```

Arguments

`url`	url of ProActive Scheduler
`login`	login of the user
`pwd`	password of the user, if not provided a popup window will ask to type the password
`cred`	(default to NULL) the path to an encrypt credential file which stores the login information (see ProActive Scheduler manual for more details)
`.print.stack`	(default to TRUE) in case there is a connection problem, should the full Java stack trace be printed or simply the error message

Value

a scheduler connection handle, which can be used in other PARConnector functions

See Also

PASolve

Examples

```
## Not run:
 PAConnect("http://localhost:8080/rest/rest","demo","demo") # connects to a local ProActive Scheduler runnin


## End(Not run)
```

PADebug *sets PARConnector Debug mode*

Description

PADebug can be used either to set the Debug mode to on/off or to know the current state of the debug mode.

Usage

```
PADebug(debug = FALSE)
```

Arguments

debug	to set the debug mode to on (TRUE) or off (FALSE)

Details

In Debug mode a lot of verbose information will be printed (detailed content of PATask created, code analysis debugging, remote execution trace)

Value

the current or new state of the debug mode

PAJobState — *Retrieves information about a ProActive job*

Description

`PAJobState` prints a formatted table diplaying the state of a ProActive job.

Usage

```
PAJobState(job.id, client = PAClient())
```

Arguments

`job.id`	id of the proactive job
`client`	connection handle to the scheduler, if not provided the handle created by the last call to `PAConnect` will be used

See Also

`PAState`

PAM — *Creates a single merge PATask which can be used in combination with `PA` and `PAS` to create split/merge workflows*

Description

`PAM` uses the same parameter semantic as `PA` , but instead of creating a set of parallel tasks, it will produce a single task which will aggregate the results of a list of tasks produced by `PA`.

Usage

```
PAM(funcOrFuncName, ..., varies = list(), input.files = list(),
  output.files = list(), in.dir = getwd(), out.dir = getwd(),
  hostname.selection = NULL, ip.selection = NULL,
  property.selection.name = NULL, property.selection.value = NULL,
  isolate.io.files = FALSE, client = PAClient(), .debug = PADebug())
```

Arguments

`funcOrFuncName`	function handle or function name
`...`	arguments of the funcOrFuncName function which will be vectorized over
`varies`	list of varying parameters which can be a parameter number or a parameter name, if NULL (default) then all parameters are varying
`input.files`	a list of input files which will be transferred from the local machine to the remote executions, see Details section in `PA` for more information
`output.files`	a list of output files which will be transferred from the remote executions to the local machine

`in.dir`	in case input files are used, the directory which will be used as base (default to current working directory)
`out.dir`	in.dir in case ouput files are used, the directory which will be used as base (default to current working directory)
`hostname.selection`	can be used to restrict the remote execution to a given machine, wildcards can be used
`ip.selection`	can be used to restrict the remote execution to a given machine given its IP address
`property.selection.name`	can be used to restrict the remote execution to a given JVM resource where the property is set to the according value
`property.selection.value`	is used in combination with property.selection.name
`client`	connection handle to the scheduler, if not provided the handle created by the last call to PAConnect will be used
`.debug`	debug mode

Details

`PAM` has always a cardinality of 1, it is used solely as a multi-parameter aggregation.

Value

a PATask object which can be submitted to the ProActive Scheduler via a `PASolve` call or given as parameter to other `PA`, `PAS` or `PAM` functions

See Also

`PA` `PAS` `PASolve` `mapply` `PAConnect`

Examples

```
## Not run:
 see examples in PAS and PA help sections before reading these examples

 PAM("sum",
   PA(function(x) {x*x},
      PAS("identity", 1:4)))  # will produce 6 PATasks producing the following results :

 t1: 1:4
 t2: 1*1
 t3: 2*2
 t4: 3*3
 t5: 4*4
 t6: sum(1*1,2*2,3*3,4*4)

 Explanation for t6 : the lower part of the statement produces 4 parallel tasks which are given as parameter t
 The results of those tasks are merged via the sum function, similar to sum( res[t2],res[t3],res[t4],res[t4])


## End(Not run)
```

PAPullFile *Transfer a file from a ProActive Data space to the local machine*

Description

`PAPullFile` will transfer a file existing in a shared data space to the local computer. The Scheduler controls two main spaces :

Usage

```
PAPullFile(space, pathname, outputFile, client = PAClient(), .nb.tries = 2,
  .print.stack = TRUE)
```

Arguments

`space`	name of the data space to transfer the file to
`pathname`	location of the file inside the remote data space
`outputFile`	local path of the file where the file will be copied to. The file must be absolute
`client`	connection handle to the scheduler, if not provided the handle created by the last call to `PAConnect` will be used

Details

- The USER Space : a data space reserved for a specific user.
- The GLOBAL Space : a data space accessible to all users.

See Also

`PAPushFile`

Examples

```
## Not run:
 PAPullFile("USER","/in.txt",file.path(getwd(),"in2.txt")) # will transfer file at USER/in.txt to a local fil

## End(Not run)
```

PAPushFile *Transfer a file from the local machine to a ProActive Data space*

Description

`PAPushFile` will copy a local file to a shared data space available to a ProActive Scheduler. The Scheduler controls two main spaces :

Usage

```
PAPushFile(space, path, fileName, inputFile, client = PAClient(),
  .print.stack = TRUE)
```

Arguments

`space`	name of the data space to transfer the file to
`path`	path inside the remote data space where the file will be copied to
`fileName`	name of the file that will be created in the remote data space
`inputFile`	local path of the file
`client`	connection handle to the scheduler, if not provided the handle created by the last call to `PAConnect` will be used

Details

- The USER Space : a data space reserved for a specific user.
- The GLOBAL Space : a data space accessible to all users.

See Also

`PAPullFile`

Examples

```
## Not run:
 PAPushFile("USER","/","in.txt", "in.txt") # will transfer local file in.txt to the USER space

## End(Not run)
```

`PARConnector-package` *Parallel execution of R functions and split/merge workflows using ProActive Scheduler*

Description

The **PARConnector** package allows remote execution of R functions using the ProActive Scheduler.

The package features :

- simple parametric sweep remote execution of R functions with a syntax similar to `mapply`
- automatic transfer of user-defined functions and their dependencies
- automatic transfer of input/output files
- primitive for waiting results
- general purpose primitives to check the current state of ProActive Scheduler
- ability to create complex split/merge workflows in a compact,user-friendly syntax

Author(s)

The ProActive Team

Examples

```
## Not run:

A typical PARConnector session :

> library(PARConnector)

> PAConnect("http://localhost:8080/rest/rest","demo","demo")

Connected to Scheduler at  http://localhost:8080/rest/rest
[1] "Java-Object{org.ow2.proactive.scheduler.rest.SchedulerClient@39f46204}"

> res = PASolve("cos",1:4)

Job submitted (id : 405)
 with tasks : t1, t2, t3, t4

> res

PARJob1 (id: 405)  (status: Running)
t1 : Pending
t2 : Running at 192.168.1.187 (local-LocalNodes-0) (0
t3 : Running at 192.168.1.187 (local-LocalNodes-2) (0
t4 : Pending

> PAWaitFor(res)

$t1
[1] 0.5403023

$t2
[1] -0.4161468

$t3
[1] -0.9899925

$t4
[1] -0.6536436

> res = PASolve(PAM("sum",
+                PA(function(x) {x*x},
+                  PAS("identity", 1:4))))

Job submitted (id : 406)
 with tasks : t1, t2, t3, t4, t5, t6

> res

PARJob2 (id: 406)  (status: Running)
t1 : Running at 192.168.1.187 (local-LocalNodes-0) (0
t2 : Pending
t3 : Pending
t4 : Pending
t5 : Pending
t6 : Pending
```

```
> PAWaitFor(res)

$t1
[1] 1 2 3 4

$t2
[1] 1

$t3
[1] 4

$t4
[1] 9

$t5
[1] 16

$t6
[1] 30

## End(Not run)
```

PAS *Creates a single split PATask which can be used in combination with PA and PAM to create split/merge workflows*

Description

PAS uses the same parameter semantic as PA , but instead of creating a set of parallel tasks, it will produce a single task whose result (expected to be a list of vector) will be scattered across dependent tasks.

Usage

```
PAS(funcOrFuncName, ..., varies = NULL, input.files = list(),
  output.files = list(), in.dir = getwd(), out.dir = getwd(),
  hostname.selection = NULL, ip.selection = NULL,
  property.selection.name = NULL, property.selection.value = NULL,
  isolate.io.files = FALSE, client = PAClient(), .debug = PADebug())
```

Arguments

funcOrFuncName	function handle or function name
...	arguments of the funcOrFuncName function which will be vectorized over
varies	list of varying parameters which can be a parameter number or a parameter name, if NULL (default) then all parameters are varying
input.files	a list of input files which will be transferred from the local machine to the remote executions, see Details section in PA for more information
output.files	a list of output files which will be transferred from the remote executions to the local machine
in.dir	in case input files are used, the directory which will be used as base (default to current working directory)

`out.dir`	in.dir in case ouput files are used, the directory which will be used as base (default to current working directory)
`hostname.selection`	can be used to restrict the remote execution to a given machine, wildcards can be used
`ip.selection`	can be used to restrict the remote execution to a given machine given its IP address
`property.selection.name`	can be used to restrict the remote execution to a given JVM resource where the property is set to the according value
`property.selection.value`	is used in combination with property.selection.name
`client`	connection handle to the scheduler, if not provided the handle created by the last call to PAConnect will be used
`.debug`	debug mode

Details

The cardinality will be, like for `PA`, determined by analysing the parameters and finding the longest list/vector among them. But the cardinality will be used in a different way, as it will be used when the result of the `PAS` call is given to a `PA` call to build a workflow. In that case, the cardinality of the `PAS` result will be used to produce as many `PA` tasks.

Value

a PATask object which can be submitted to the ProActive Scheduler via a `PASolve` call or given as parameter to other `PA`, `PAS` or `PAM` functions

See Also

`PA` `PAM` `PASolve` `mapply` `PAJobResult` `PAConnect`

Examples

```
## Not run:
 PAS("identity", 1:4) # will produce a split task of cardinality 4 that will output the vector 1:4

 PA(function(x){x*x},  PAS("identity", 1:4)) # will produce 5 PATasks producing the following results :
 t1: 1:4
 t2: 1*1
 t3: 2*2
 t4: 3*3
 t5: 4*4

 Explanation : This is because the tasks created by the PAS call, given as parameter to the PA call will produc
 as the PAS task produce the vector 1:4, the first PA task will receive the parameter 1, the second PA task wil

 (PAS(function(out,ind){for (i in ind) {file.create(paste0(out,i))}}, "out", 1:4, output.files="out%2%") # w


## End(Not run)
```

PASolve *Create and submit a ProActive R Job*

Description

PASolve take in parameter a list of PATasks produced by PA PAS or PAM calls and submits a new job to ProActive Scheduler.

Usage

```
PASolve(..., client = PAClient(), .debug = PADebug(),
  jobName = str_c("PARJob", .peekNewSolveId()),
  jobDescription = "ProActive R Job", priority = "normal",
  cancelOnError = TRUE)
```

Arguments

...	list of PATasks produced by PA PAS or PAM calls
client	connection handle to the scheduler, if not provided the handle created by the last call to PAConnect will be used
jobName	name of the ProActive job to be created
jobDescription	description of this job
priority	priority of this job
cancelOnError	sets the cancelling mode mechanism whenever an error occur in one tasks, does it cancel the whole job ? Default to TRUE

Details

a PAJobResult object will be returned. The object will bear the current state of the job, which can be displayed simply by showing or printing the object. Special functions PAWaitFor and PAWaitAny can be used to wait for the results.

Value

a PAJobResult object which acts as a placeholder for receiving actual results

See Also

PA PAS PAM PAJobResult PAConnect

Examples

```
## Not run:

 > res = PASolve("cos",1:4)   # shortcut for PASolve(PA("cos",1:4)), submits a parallel job of 4 tasks
 Job submitted (id : 405)
 with tasks : t1, t2, t3, t4
 > res                             # display the current state
 PARJob1 (id: 405)  (status: Running)
 t1 : Pending
 t2 : Running at 192.168.1.187 (local-LocalNodes-0) (0%)
```

```
t3 : Running at 192.168.1.187 (local-LocalNodes-2) (0%)
t4 : Pending
> PAWaitFor(res)                 # wait for the results and return them in a list
$t1
[1] 0.5403023

$t2
[1] -0.4161468

$t3
[1] -0.9899925

$t4
[1] -0.6536436



> res = PASolve(PAM("sum",
                PA(function(x) {x*x},
                PAS("identity", 1:4))))        # submits a split/merge job of six tasks

> res
PARJob2 (id: 406)  (status: Running)
t1 : Running at 192.168.1.187 (local-LocalNodes-0) (0%)
t2 : Pending
t3 : Pending
t4 : Pending
t5 : Pending
t6 : Pending

> PAWaitFor(res)        # wait for the results and return them in a list
$t1
[1] 1 2 3 4

$t2
[1] 1

$t3
[1] 4

$t4
[1] 9

$t5
[1] 16

$t6
[1] 30

## End(Not run)
```

PAState — *Display the list of jobs in the scheduler wether pending, running or finished*

Description

`PAState` display the current state of the scheduler with a list of jobs

Usage

```
PAState(client = PAClient())
```

Arguments

`client`	connection handle to the scheduler, if not provided the handle created by the last call to PAConnect will be used

See Also

`PAConnect`

`PAWaitAny` *Waits for the first available result among a list of results controlled by a PAJobResult object*

Description

PAWaitAny is used on a PAJobResult object to block the R interpreter until the first result is available. The R result object will be then returned as a factor, named by the task name. If the PAWaitAny is called a second time, then the second result will be waited and returned. After all results are consumed, a call to PAWaitAny will return NA.

Usage

```
PAWaitAny(paresult = PALastResult(), ...)
```

Arguments

`paresult`	a PAJobResult object
`timeout`	a long value specifying an optional timeout in milisecond
`callback`	a single parameter function which can be called when results are received. It can be useful to udapte graphical user interfaces for examples. Default to NULL.

Value

A result

See Also

`PASolve` and `PAWaitFor`

`PAWaitFor` *Waits for all results controlled by a PAJobResult object*

Description

PAWaitFor is used on a PAJobResult object to block the R interpreter until all results are available. The R result objects will be then returned inside a list. It is possible to wait for a subset instead of the whole list by using subscript indexing : PAWaitFor(res[1:3]) will wait for only the results at index 1,2,3.

Usage

```
PAWaitFor(paresult = PALastResult(), ...)
```

Arguments

`paresult`	a PAJobResult object
`timeout`	a long value specifying an optional timeout in milisecond
`callback`	a single parameter function which can be called when results are received. It can be useful to udapte graphical user interfaces for examples. Default to NULL.

Value

A list of results

See Also

`PASolve` and `PAWaitAny`

Index