A Future for R: Apply Function to Elements in Parallel

Introduction

The purpose of this package is to provide worry-free parallel alternatives to base-R "apply" functions, e.g. `apply()`, `lapply()`, and `vapply()`. The goal is that one should be able to replace any of these in the core with its futurized equivalent and things will just work. For example, instead of doing:

```
library("stats")
x <- 1:10
y <- lapply(x, FUN = quantile, probs = 1:3/4)
```

one can do:

```
library("future.apply")
plan(multiprocess) ## Run in parallel on local computer

library("stats")
x <- 1:10
y <- future_lapply(x, FUN = quantile, probs = 1:3/4)
```

Reproducibility is part of the core design, which means that perfect, parallel random number generation (RNG) is supported regardless of the amount of chunking, type of load balancing, and future backend being used. *To enable parallel RNG, use argument* `future.seed = TRUE`.

Role

Where does the future.apply package fit in the software stack? You can think of it as a sibling to foreach, BiocParallel, plyr, etc. Just as parallel provides `parLapply()`, foreach provides `foreach()`, BiocParallel provides `bplapply()`, and plyr provides `llply()`, future.apply provides `future_lapply()`. Below is a table summarizing this idea:

Package	Functions	Backends

future.apply	Future-versions of common goto `*apply()` functions available in base R (of the 'base' package): `future_apply()`, `future_eapply()`, `future_lapply()`, `future_Map()`, `future_mapply()`, `future_replicate()`, `future_sapply()`, `future_tapply()`, and `future_vapply()`. *The following function is yet not implemented:* `future_rapply()`	All future backends
parallel	`mclapply()`, `mcmapply()`, `clusterMap()`, `parApply()`, `parLapply()`, `parSapply()`, …	Built-in and conditional on operating system
foreach	`foreach()`, `times()`	All future backends via doFuture
BiocParallel	Bioconductor's parallel mappers: `bpaggregate()`, `bpiterate()`, `bplapply()`, and `bpvec()`	All future backends via doFuture (because it supports foreach) or via BiocParallel.FutureParam (direct BiocParallelParam support; prototype)
plyr	`**ply(..., .parallel = TRUE)` functions: `aaply()`, `ddply()`, `dlply()`, `llply()`, …	All future backends via doFuture (because it uses foreach internally)

Note that, except for the built-in parallel package, none of these higher-level APIs implement their own parallel backends, but they rather enhance existing ones. The foreach framework leverages backends such as doParallel, doMC and doFuture, and the future.apply framework leverages the future ecosystem and therefore backends such as built-in parallel, future.callr, and future.batchtools.

By separating `future_lapply()` and friends from the future package, it helps clarifying the purpose of the future package, which is to define and provide the core Future API, which higher-level parallel APIs can build on and for which any futurized parallel backends can be plugged into.

Roadmap

1. Implement `future_*apply()` versions for all common `*apply()` functions that exist in base R. This also involves writing a large set of package tests asserting the correctness and the same behavior as the corresponding `*apply()` functions.

2. Harmonize all `future_*apply()` functions with each other, e.g. the future-specific arguments.

3. Consider additional `future_*apply()` functions and features that fit in this package but don't necessarily have a corresponding function in base R. Examples of this may be "apply" functions that return futures rather than values, mechanisms for benchmarking, and richer control over load balancing.

The API and identity of the future.apply package will be kept close to the `*apply()` functions in base R. In other words, it will *neither* keep growing nor be expanded with new, more powerful apply-like functions beyond those core ones in base R. Such extended functionality should be part of a separate package.