Notes for the Caching Computations Lecture

Shuai Wang

Tuesday, May 05, 2015

cacher package

There are two packages. One is cacheSweave package which interacts with Sweave, and I don't talk about it here. Another is cacher package, which is a stand alone package. It assumes that you have some code in an R file. It reads your code, evaluates it, and stores the results in key-value database. The code expressions are all given SHA-1 hash values so if there are any changes they can be tracked. The code can be reevaluated if necessary. The idea is that you put everything, the code and data, in a package that you could just give to someone. I call this the cacher package. So the other people can get the analysis or clone the analysis and look at subsets of code or inspect specific data objects. One assumption that I made is that other people do not necessarily have the resources that you have, so they may not want to run the entire Markov Chain Monte Carlo simulation that you did to get the posterior distribution or the histogram you've got at the end. They might just want to look at specific end products of that, or when they see the end products, they may do need to look at the whole MCMC. So the idea is that, you kind of peel the onion a little bit rather than just go straight to the core. Hence the model is basically that you have some code source file, there's code that goes in, there's data that goes in, and out comes results.

Using cacher as an author

1. For author, cacher parses the R source file, creates the necessary cache directories and subdirectories.
2. cacher cycles through each expression in the source file:

- if an expression has never been evaluated, it evaluates it and store any resulting R objects in the cache database,
- if a cached result exists, it lazy-loads the results from the cache database and move to the next expression,
- if an expression does not create any R objects (i.e., there is nothing to cache), it adds the expression to the list of expressions where evaluation needs to be forced,
- it writes out metadata for this expression to the metadata file.

The cachepackage() function creates a cacher package storing source file, cached data objects, and metadata. Package file is zipped and can be distributed. (Things outputted to the console and figures plotted are not cached.)

Using cacher as a reader

Readers can unzip the file and immediately investigate its contents via cacher package.

```
library(cacher)
clonecache(id="092dcc7dda4b93e42f23e038a60e1d44dbec7b3f") #clone the cacher package by
specifying the SHA-1 hash of the cacher package
clonecache(id="092d") #same as above, usually the first four digits are enough to identify
the SHA-1 hash
showfiles() #you can see the file cached:  "top20.R"
```

```
sourcefile("top20.R") #go through the analysis
code() #look at the code
graphcode() #make a graph of the code:  graph all the data objects to show how they come
together (a rough graph of how the analysis worked)
objectcode("cities") #show you the specific lines of codes that generate the object
"cities"
loadcache() #you can look at specific data objects after running this function, this
function loads pointers to specific data objects, so any time you access an object, it
will be transferred from wherever it's coming from
cities #the object "cities" is transferred from the cache database file, its signature is
also showed (each object has a signature), then its content is showed.  Once it is loaded
into your system, you don't have to do it every single time, just the first time.
```

Some remarks

- What happens when cloning an analysis: local directories created, source code files and metadata are downloaded, data objects are not downloaded by default (not downloaded initially), references to data objects are loaded and corresponding data can be lazy-loaded on demand.
- runcode() function executes code in the source file in a specific way: expressions resulting in an object being created are not run by default, and the resulting object is lazy-loaded into the workspace; expression not resulting in objects are evaluated.
- checkcode() function evaluates all expressions from scratch (no lazy-loading), results of evaluation are checked against stored results (checking their signatures) to see if the results are the same as what the author calculated (setting RNG seeds is critical for this to work).
- The integrity of data objects can be verified with the checkobjects() function to check (checking signatures) for possible corruption of data (i.e. in transit).

cacher Summary

- The cacher package can be used by authors to create cache packages from data analyses for distribution
- Readers can use the cacher package to inspect others' data analyses by examining cached computations
- cacher is mindful of readers' resources and efficiently loads only those data objects that are needed