Parallel computing with snow

**SNOW** (Tierney et al): Simple Network of Workstations
Based on master-slave model: master distributes \( N \) replicates across \( p \) slaves.

**Four basic steps** (in user’s responsibility):
1. create a cluster of given size
2. initialize random number generator (rlecuyer, rsprng) \( \rightarrow \) one stream per node
3. one call for repeated evaluation of a user-defined function, using `clusterApply` or `clusterApplyLB`
4. destroy cluster

---

**Example**

Execute function `fun` 10 times on 5 nodes.

\[
\begin{align*}
\text{clusterApply} & (cl, \text{rep}(2,5), \text{fun}) \\
\text{clusterApplyLB} & (cl, \text{rep}(1,10), \text{fun})
\end{align*}
\]

- results reproducible, runtime depends on the slowest node
- faster but results are not reproducible

---

**snowFT: Fault Tolerant SNOW**

- eliminates drawbacks of SNOW and adds new features:
  - Fault tolerance
  - Load balancing AND reproducibility
  - Computation transparency
  - Dynamic cluster resizing
  - Ease of use
  - Support for sequential runs

---

**snowFT Features**

1. **Fault tolerance**
   - Master checks repeatedly for failure in its waiting time
   - Failure recovery:
     - replace failed nodes by new created ones
     - repeat computation for failed replicates
     - if after repeating still some failed slaves, notification \& compute results only on the non-failed replicates

2. **Load balancing AND reproducibility**
   - one stream associated with one replicate
     - \( R \) streams in total
     - each stream uniquely identified by replication number
     - each slave knows about initial states of all streams

3. **Computation transparency**
   - Allows one to define a function that is called after each given number of replicates.
   - Stores failed replicates and currently processed replicates into files.

4. **Dynamic cluster resizing**
   - The number of nodes is kept in a file. In its waiting time, the master reads it and updates the cluster.

5. **Ease of use**
   - Only one function needed for all features!
   - `performParallel(count, ...)`
     - creates a cluster of size `count`
     - [initializes each node with a given function]
     - initializes RNG (rlecuyer or rsprng)
     - performs the actual computation on the cluster
     - [performs aftermath operations on each node]
     - stops the cluster and returns result

6. **Reproducing parallel results sequentially**
   - Use `performParallel(count=0, ...)` to process simulation sequentially.
   - Results are reproducible for any value of `count`.

---

**snowFT Example**

```r
my.function <- function(m, sigma) {
  Sys.sleep(3) # Here as a black box. Replace by any computation.
  return(rnorm(1, m, sigma))
}
N <- 1000
cluster.size <- 8
res <- performParallel(cluster.size, 1:N, my.function, sigma=1)
```

**Conclusions**

Parallel programming in R is EASY and EFFICIENT!

Download `snowFT` from CRAN: [http://www.r-project.org](http://www.r-project.org)