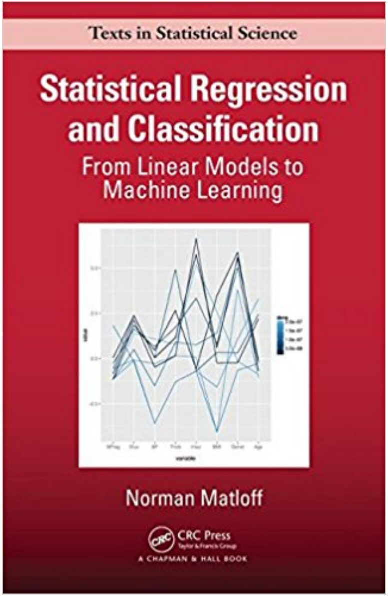
Parallel Computation in R: What We Want, and How We (Might) Get It

Norm Matloff
University of California at Davis

Keynote Address
useR! 2017
Brussels, 6 July, 2017

Shameless Promotion

Shameless Promotion



Out July 28!

(A longheld plan — decades — now finally got around to it.)

Disclaimer

Disclaimer

- "Everyone has an opinion."

Disclaimer

- "Everyone has an opinion."
- I'll present mine.

Norm Matloff
University of California at Davis

Disclaimer

- “Everyone has an opinion.”
- I’ll present mine.
- I will essentially propose general design patterns,

Disclaimer

- "Everyone has an opinion."
- I'll present mine.
- I will essentially propose general design patterns, illustrated with our own package **partools** but meant to be general.

Disclaimer

- "Everyone has an opinion."
- I'll present mine.
- I will essentially propose general design patterns, illustrated with our own package **partools** but meant to be general.
- Dissent is encouraged. :-)

The Drivers and Their Result

The Drivers and Their Result

- Parallel hardware for the masses:

The Drivers and Their Result

- Parallel hardware for the masses:
  - 4 cores standard, 16 not too expensive

The Drivers and Their Result

- Parallel hardware for the masses:
    - 4 cores standard, 16 not too expensive
    - GPUs

The Drivers and Their Result

- Parallel hardware for the masses:
  - 4 cores standard, 16 not too expensive
  - GPUs
  - Intel Xeon Phi, $\approx$ 60 cores (!), coprocessor, as low as a few hundred dollars

The Drivers and Their Result

- Parallel hardware for the masses:
  - 4 cores standard, 16 not too expensive
  - GPUs
  - Intel Xeon Phi, $\approx$ 60 cores (!), coprocessor, as low as a few hundred dollars
- Big Data

The Drivers and Their Result

- Parallel hardware for the masses:
  - 4 cores standard, 16 not too expensive
  - GPUs
  - Intel Xeon Phi, $\approx$ 60 cores (!), coprocessor, as low as a few hundred dollars
- Big Data
  - Whatever that is.

The Drivers and Their Result

- Parallel hardware for the masses:
  - 4 cores standard, 16 not too expensive
  - GPUs
  - Intel Xeon Phi, $\approx$ 60 cores (!), coprocessor, as low as a few hundred dollars
- Big Data
  - Whatever that is.

Result: Users believe,

The Drivers and Their Result

- Parallel hardware for the masses:
  - 4 cores standard, 16 not too expensive
  - GPUs
  - Intel Xeon Phi, $\approx$ 60 cores (!), coprocessor, as low as a few hundred dollars
- Big Data
  - Whatever that is.

Result: Users believe,

> *"I've got the hardware and I've got the data need — so I should be all set to do parallel computation in R on the data."*

Not So Simple

Not So Simple

- Non-“embarrassingly parallel” algorithms.

Not So Simple

- Non-"embarrassingly parallel" algorithms.
- Overhead issues:

Not So Simple

- Non-“embarrassingly parallel” algorithms.
- Overhead issues:
  - Contention for memory/network.

Not So Simple

- Non-“embarrassingly parallel” algorithms.
- Overhead issues:
  - Contention for memory/network.
  - Bandwidth limits — CPU/memory, CPU/network, CPU/GPU.

Not So Simple

- Non-"embarrassingly parallel" algorithms.
- Overhead issues:
  - Contention for memory/network.
  - Bandwidth limits — CPU/memory, CPU/network, CPU/GPU.
  - Cache coherency problems (inconsistent caches in multicore systems).

Not So Simple

- Non-"embarrassingly parallel" algorithms.
- Overhead issues:
  - Contention for memory/network.
  - Bandwidth limits — CPU/memory, CPU/network, CPU/GPU.
  - Cache coherency problems (inconsistent caches in multicore systems).
  - Contention for I/O ports.

Not So Simple

- Non-"embarrassingly parallel" algorithms.
- Overhead issues:
  - Contention for memory/network.
  - Bandwidth limits — CPU/memory, CPU/network, CPU/GPU.
  - Cache coherency problems (inconsistent caches in multicore systems).
  - Contention for I/O ports.
  - OS/R limits on number of sockets (network connections).

Not So Simple

- Non-“embarrassingly parallel” algorithms.
- Overhead issues:
    - Contention for memory/network.
    - Bandwidth limits — CPU/memory, CPU/network, CPU/GPU.
    - Cache coherency problems (inconsistent caches in multicore systems).
    - Contention for I/O ports.
    - OS/R limits on number of sockets (network connections).
    - Serialization.

Wish List

Wish List

- Ability to run on various types of hardware — from R.

Wish List

- Ability to run on various types of hardware — from R.
- Ease of use for the non-cognoscenti.

Wish List

- Ability to run on various types of hardware — from R.
- Ease of use for the non-cognoscenti.
- Parameters to tweak for the experts or the daring.

The Non-cognoscenti Can Become the Daring

The Non-cognoscenti Can Become the Daring

Help, I'm in over my head here! – a prominent R developer, entering the parallel comp. world.

The Non-cognoscenti Can Become the Daring

Help, I'm in over my head here! – a prominent R developer, entering the parallel comp. world.

Norm Matloff
University of California at Davis

Non-cognoscenti (cont'd.)

Non-cognoscenti (cont'd.)

- Casual users, even if they are deft programmers, quickly learn that this is no casual operation.

Non-cognoscenti (cont'd.)

- Casual users, even if they are deft programmers, quickly learn that this is no casual operation.
- After getting burned by disappointing performance, some will be emboldened to learn the subtleties.

Non-cognoscenti (cont'd.)

- Casual users, even if they are deft programmers, quickly learn that this is no casual operation.
- After getting burned by disappointing performance, some will be emboldened to learn the subtleties.
- Painless parallel computation is not possible.

Example: Matrix-Vector Multiplication

Example: Matrix-Vector Multiplication

- $D = AX$, with A being $n \times p$ and X being $p \times 1$

Example: Matrix-Vector Multiplication

- $D = AX$, with A being $n \times p$ and X being $p \times 1$
- Naive approach: Parallelize the loop

Example: Matrix-Vector Multiplication

- $D = AX$, with A being $n \times p$ and X being $p \times 1$
- Naive approach: Parallelize the loop

```
for (i in 1:n)
   d[i] <- a[i,] %*% x
```

Example: Matrix-Vector Multiplication

- $D = AX$, with A being $n \times p$ and X being $p \times 1$
- Naive approach: Parallelize the loop

```
for (i in 1:n)
   d[i] <- a[i,] %*% x
```

- Naive use of **foreach** package likely quite slow; scatter-gather overhead a substantial proportion of the overall time.

Example (cont'd.)

Example (cont'd.)

- Solution is obvious:

Example (cont'd.)

- Solution is obvious: For r processes, partition rows A_i into n/r chunks and change the above loop from n iterations to n/r.

Example (cont'd.)

- Solution is obvious: For r processes, partition rows A_i into n/r chunks and change the above loop from n iterations to n/r.

```
for (k in 1:r)
   d[rowblockk] <- a[rowblockk,] %*% x
```

Example (cont'd.)

- Solution is obvious: For r processes, partition rows A_i into n/r chunks and change the above loop from n iterations to n/r.

```
for (k in 1:r)
   d[rowblockk] <- a[rowblockk,] %*% x
```

- But **casual users may miss this**.

Example (cont'd.)

- Solution is obvious: For r processes, partition rows A_i into n/r chunks and change the above loop from n iterations to n/r.

```
for (k in 1:r)
   d[rowblockk] <- a[rowblockk,] %*% x
```

- But **casual users may miss this**. And **automatic parallelization <u>would</u> miss it**.

Use Cases

Use Cases

A few reference examples, somewhat spanning the space:

Use Cases

A few reference examples, somewhat spanning the space:

- Compute-intensive parametric: Quantile regression.

Use Cases

A few reference examples, somewhat spanning the space:

- Compute-intensive parametric: Quantile regression.
- Compute-intensive nonparametric: Nearest-neighbor regression.

Use Cases

A few reference examples, somewhat spanning the space:

- Compute-intensive parametric: Quantile regression.
- Compute-intensive nonparametric: Nearest-neighbor regression.
- Compute-intensive nonparametric: Graph algorithms.

Use Cases

A few reference examples, somewhat spanning the space:

- Compute-intensive parametric: Quantile regression.
- Compute-intensive nonparametric: Nearest-neighbor regression.
- Compute-intensive nonparametric: Graph algorithms.
- Run-of-the-mill aggregation: Group-by-and-find-means op.

Use Cases

A few reference examples, somewhat spanning the space:

- Compute-intensive parametric: Quantile regression.
- Compute-intensive nonparametric: Nearest-neighbor regression.
- Compute-intensive nonparametric: Graph algorithms.
- Run-of-the-mill aggregation: Group-by-and-find-means op.
- Tougher aggregation: Credit card fraud detection.

Software Alchemy (SA)

Software Alchemy (SA)

- My term for method developed by a number of authors (Matloff, 2016).

Software Alchemy (SA)

- My term for method developed by a number of authors (Matloff, 2016).
- Break data into chunks. Apply estimator, say **lm()** to each chunk, then average the results.

Software Alchemy (SA)

- My term for method developed by a number of authors (Matloff, 2016).
- Break data into chunks. Apply estimator, say **lm()** to each chunk, then average the results.
- For parallel comp. with r processes, use r chunks.

Software Alchemy (SA)

- My term for method developed by a number of authors (Matloff, 2016).
- Break data into chunks. Apply estimator, say **lm()** to each chunk, then average the results.
- For parallel comp. with r processes, use r chunks.
- Same statistical accuracy.

Software Alchemy (SA)

- My term for method developed by a number of authors (Matloff, 2016).
- Break data into chunks. Apply estimator, say **lm()** to each chunk, then average the results.
- For parallel comp. with r processes, use r chunks.
- Same statistical accuracy.
- Often produces *superlinear* speedup, i.e. $> r$.

Software Alchemy (SA)

- My term for method developed by a number of authors (Matloff, 2016).
- Break data into chunks. Apply estimator, say **lm()** to each chunk, then average the results.
- For parallel comp. with r processes, use r chunks.
- Same statistical accuracy.
- Often produces *superlinear* speedup, i.e. $> r$.
- Useful in some apps.

Software Alchemy (SA)

- My term for method developed by a number of authors (Matloff, 2016).
- Break data into chunks. Apply estimator, say **lm()** to each chunk, then average the results.
- For parallel comp. with r processes, use r chunks.
- Same statistical accuracy.
- Often produces *superlinear* speedup, i.e. $> r$.
- Useful in some apps.
- Available in **partools** package (NM, C. Fitzgerald), **github.com/matloff**.

Norm Matloff
University of California at Davis

Programming World Views

Programming World Views

- Message passing/distributed comp.: Send data to the R processes; each process works on its data; possibly combine results.

Programming World Views

- Message passing/distributed comp.: Send data to the R processes; each process works on its data; possibly combine results.

  In R, e.g. **parallel** (the part from **snow**), **rMPI**.

Programming World Views

- Message passing/distributed comp.: Send data to the R processes; each process works on its data; possibly combine results.

  In R, e.g. **parallel** (the part from **snow**), **rMPI**.

  In C, e.g.

World Views (cont'd.)

World Views (cont'd.)

- Shared-memory: The processes have access to a common memory, so no data transfer needed.

World Views (cont'd.)

- Shared-memory: The processes have access to a common memory, so no data transfer needed.

  Not (yet) common in R, but do have **Rdsm** (NM), **thread** (R. Bartnik).

World Views (cont'd.)

- Shared-memory: The processes have access to a common memory, so no data transfer needed.

  Not (yet) common in R, but do have **Rdsm** (NM), **thread** (R. Bartnik).

  In C, e.g.

Norm Matloff
University of California at Davis

Premises in This Talk

Premises in This Talk

- There is a lot of hype about parallel computation.

Premises in This Talk

- There is a lot of hype about parallel computation.
- Parallel computation is not for the casual user.

Premises in This Talk

- There is a lot of hype about parallel computation.
- Parallel computation is not for the casual user.
- Efficient **automatic** parallelization —

Premises in This Talk

- There is a lot of hype about parallel computation.
- Parallel computation is not for the casual user.
- Efficient **automatic** parallelization — no user intervention/sophistication needed —

Premises in This Talk

- There is a lot of hype about parallel computation.
- Parallel computation is not for the casual user.
- Efficient **automatic** parallelization — no user intervention/sophistication needed — is generally not possible and should not be expected.

Norm Matloff
University of California at Davis

Premises in This Talk

- There is a lot of hype about parallel computation.
- Parallel computation is not for the casual user.
- Efficient **automatic** parallelization — no user intervention/sophistication needed — is generally not possible and should not be expected. Please stop asking for it. :-)

Premises in This Talk

- There is a lot of hype about parallel computation.
- Parallel computation is not for the casual user.
- Efficient **automatic** parallelization — no user intervention/sophistication needed — is generally not possible and should not be expected. Please stop asking for it. :-)
- As in politics, transparency in software tools is vital. :-)

Premises in This Talk

- There is a lot of hype about parallel computation.
- Parallel computation is not for the casual user.
- Efficient **automatic** parallelization — no user intervention/sophistication needed — is generally not possible and should not be expected. Please stop asking for it. :-)
- As in politics, transparency in software tools is vital. :-) What do those APIs really do?

Premises in This Talk

- There is a lot of hype about parallel computation.
- Parallel computation is not for the casual user.
- Efficient **automatic** parallelization — no user intervention/sophistication needed — is generally not possible and should not be expected. Please stop asking for it. :-)
- As in politics, transparency in software tools is vital. :-) What do those APIs really do?
- UseRs are different from aggregation-oriented (e.g. Spark) users.

Premises in This Talk

- There is a lot of hype about parallel computation.
- Parallel computation is not for the casual user.
- Efficient **automatic** parallelization — no user intervention/sophistication needed — is generally not possible and should not be expected. Please stop asking for it. :-)
- As in politics, transparency in software tools is vital. :-) What do those APIs really do?
- UseRs are different from aggregation-oriented (e.g. Spark) users.
  - Aggregation is only part of what useRs do.

Premises in This Talk

- There is a lot of hype about parallel computation.
- Parallel computation is not for the casual user.
- Efficient **automatic** parallelization — no user intervention/sophistication needed — is generally not possible and should not be expected. Please stop asking for it. :-)
- As in politics, transparency in software tools is vital. :-) What do those APIs really do?
- UseRs are different from aggregation-oriented (e.g. Spark) users.
  - Aggregation is only part of what useRs do.
  - We need iterative estimators, std. errors, linear algebra, etc.

Premises in This Talk

- There is a lot of hype about parallel computation.
- Parallel computation is not for the casual user.
- Efficient **automatic** parallelization — no user intervention/sophistication needed — is generally not possible and should not be expected. Please stop asking for it. :-)
- As in politics, transparency in software tools is vital. :-) What do those APIs really do?
- UseRs are different from aggregation-oriented (e.g. Spark) users.
    - Aggregation is only part of what useRs do.
    - We need iterative estimators, std. errors, linear algebra, etc.
    - Newer methodology, e.g. ML, random graphs etc.

Premises in This Talk

- There is a lot of hype about parallel computation.
- Parallel computation is not for the casual user.
- Efficient **automatic** parallelization — no user intervention/sophistication needed — is generally not possible and should not be expected. Please stop asking for it. :-)
- As in politics, transparency in software tools is vital. :-) What do those APIs really do?
- UseRs are different from aggregation-oriented (e.g. Spark) users.
  - Aggregation is only part of what useRs do.
  - We need iterative estimators, std. errors, linear algebra, etc.
  - Newer methodology, e.g. ML, random graphs etc.
  - UseRs may have become fairly good programmers, but lack systems knowledge.

Premises (cont'd).

Premises (cont'd).

- Use of SA as means of parallelization should be fine for things like linear models, quantile regression, k-nearest neighbor regression etc.

Premises (cont'd).

- Use of SA as means of parallelization should be fine for things like linear models, quantile regression, k-nearest neighbor regression etc.
- Some apps, e.g. graph algorithms, are based on sharing state, so shared-memory world view/hardware may be needed.

Premises (cont'd).

- Use of SA as means of parallelization should be fine for things like linear models, quantile regression, k-nearest neighbor regression etc.
- Some apps, e.g. graph algorithms, are based on sharing state, so shared-memory world view/hardware may be needed.
- But in most of the Use Cases, including the SA ones, distributed world view works well,

Premises (cont'd).

- Use of SA as means of parallelization should be fine for things like linear models, quantile regression, k-nearest neighbor regression etc.
- Some apps, e.g. graph algorithms, are based on sharing state, so shared-memory world view/hardware may be needed.
- But in most of the Use Cases, including the SA ones, distributed world view works well, and may be needed anyway at very large scale.

Premises (cont'd).

- Use of SA as means of parallelization should be fine for things like linear models, quantile regression, k-nearest neighbor regression etc.
- Some apps, e.g. graph algorithms, are based on sharing state, so shared-memory world view/hardware may be needed.
- But in most of the Use Cases, including the SA ones, distributed world view works well, and may be needed anyway at very large scale.
- **Bottom line:** For most Use Cases, use one of the following

Premises (cont'd).

- Use of SA as means of parallelization should be fine for things like linear models, quantile regression, k-nearest neighbor regression etc.
- Some apps, e.g. graph algorithms, are based on sharing state, so shared-memory world view/hardware may be needed.
- But in most of the Use Cases, including the SA ones, distributed world view works well, and may be needed anyway at very large scale.
- **Bottom line:** For most Use Cases, use one of the following
  - SA
  - Distributed computation,

Premises (cont'd).

- Use of SA as means of parallelization should be fine for things like linear models, quantile regression, k-nearest neighbor regression etc.
- Some apps, e.g. graph algorithms, are based on sharing state, so shared-memory world view/hardware may be needed.
- But in most of the Use Cases, including the SA ones, distributed world view works well, and may be needed anyway at very large scale.
- **Bottom line:** For most Use Cases, use one of the following
    - SA
    - Distributed computation, esp. using "Leave it there" concept.

Spark

Spark

One well-publicized distributed approach today is Spark/SparkR.

Spark

One well-publicized distributed approach today is Spark/SparkR.

- MapReduce not well-suited to most of the above Use Cases.

Spark

One well-publicized distributed approach today is Spark/SparkR.

- MapReduce not well-suited to most of the above Use Cases.
- Highly elaborate Spark machinery violates the transparency requirement.

Spark

One well-publicized distributed approach today is Spark/SparkR.

- MapReduce not well-suited to most of the above Use Cases.
- Highly elaborate Spark machinery violates the transparency requirement.
- On the other hand, the distributed file system approach of Hadoop/Spark is good for useRs too.

Example Study: I

Example Study: I

- (Gittens *et al*, 2016). *Matrix Factorizations at Scale: a Comparison of Scientific Data Analytics in Spark and C+MPI Using Three Case Studies*

Example Study: I

- (Gittens *et al*, 2016). *Matrix Factorizations at Scale: a Comparison of Scientific Data Analytics in Spark and C+MPI Using Three Case Studies*
  In spite of careful optimization, performance of Spark ranged from slightly slower to really, really slower. :-)

Example Study: I

- (Gittens *et al*, 2016). *Matrix Factorizations at Scale: a Comparison of Scientific Data Analytics in Spark and C+MPI Using Three Case Studies*
  In spite of careful optimization, performance of Spark ranged from slightly slower to really, really slower. :-) Just not what Spark was designed for.

Example Study: I

- (Gittens *et al*, 2016). *Matrix Factorizations at Scale: a Comparison of Scientific Data Analytics in Spark and C+MPI Using Three Case Studies*
  In spite of careful optimization, performance of Spark ranged from slightly slower to really, really slower. :-) Just not what Spark was designed for.

  My personal side comment: Not clear whether, say, PCA, has much accuracy or usefulness at the truly Big Data scale, including for sparse matrices.

Example Study: II

Example Study: II

Reyes-Ortiz *et al*, *Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf*

Example Study: II

Reyes-Ortiz *et al*, *Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf*

Abstract:

Example Study: II

Reyes-Ortiz *et al*, *Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf*

Abstract:

> *...MPI/OpenMP outperforms Spark by more than one order of magnitude in terms of processing speed and provides more consistent performance. However, Spark shows better data management infrastructure and the possibility of dealing with other aspects such as node failure and data replication*

Example Study: II

Reyes-Ortiz *et al*, *Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf*

Abstract:

> *...MPI/OpenMP outperforms Spark by more than one order of magnitude in terms of processing speed and provides more consistent performance. However, Spark shows better data management infrastructure and the possibility of dealing with other aspects such as node failure and data replication*

I contend that very few useRs, even those who need parallel computation, need to guard against node failure.

The Principle of "Leave It There"

Norm Matloff
University of California at Davis

The Principle of "Leave It There"

Extremely simple idea, but very powerful.

The Principle of "Leave It There"

Extremely simple idea, but very powerful.

- Common setting (e.g. **parallel** package): Scatter/gather.

The Principle of "Leave It There"

Extremely simple idea, but very powerful.

- Common setting (e.g. **parallel** package): Scatter/gather.
  (a) Manager node partitions (scatters) data to worker nodes.

The Principle of “Leave It There”

Extremely simple idea, but very powerful.

- Common setting (e.g. **parallel** package): Scatter/gather.
  (a) Manager node partitions (scatters) data to worker nodes.
  (b) Worker nodes work on their chunks.

The Principle of “Leave It There”

Extremely simple idea, but very powerful.

- Common setting (e.g. **parallel** package): Scatter/gather.
  (a) Manager node partitions (scatters) data to worker nodes.
  (b) Worker nodes work on their chunks.
  (c) Manager collects (gathers) and combines the results.

The Principle of "Leave It There"

Extremely simple idea, but very powerful.

- Common setting (e.g. **parallel** package): Scatter/gather.
  - (a) Manager node partitions (scatters) data to worker nodes.
  - (b) Worker nodes work on their chunks.
  - (c) Manager collects (gathers) and combines the results.
- **But NO, avoid step (c) as much as possible.**

Example of “Leave It There”

Example of "Leave It There"

Say we wish to perform the following on some dataset:

Example of "Leave It There"

Say we wish to perform the following on some dataset:

- Convert categorical variables to dummies.
- Replace NA values by means. (Not great, but just an example.)
- Remove outliers, as def. by $|X - \mu| > 3\sigma$. (Just an example.)
- Run linear regression analysis.

Example of "Leave It There"

Say we wish to perform the following on some dataset:

- Convert categorical variables to dummies.
- Replace NA values by means. (Not great, but just an example.)
- Remove outliers, as def. by $|X - \mu| > 3\sigma$. (Just an example.)
- Run linear regression analysis.

The point is to NOT do the gather op after each of the above steps.

Example of "Leave It There"

Say we wish to perform the following on some dataset:

- Convert categorical variables to dummies.
- Replace NA values by means. (Not great, but just an example.)
- Remove outliers, as def. by $|X - \mu| > 3\sigma$. (Just an example.)
- Run linear regression analysis.

The point is to NOT do the gather op after each of the above steps. Leave the data there (in distributed form).

Example of "Leave It There"

Say we wish to perform the following on some dataset:

- Convert categorical variables to dummies.
- Replace NA values by means. (Not great, but just an example.)
- Remove outliers, as def. by $|X - \mu| > 3\sigma$. (Just an example.)
- Run linear regression analysis.

The point is to NOT do the gather op after each of the above steps. Leave the data there (in distributed form).

Note too: The last step can be done in parallel too, with SA.

Norm Matloff
University of California at Davis

Comparing Just a Few Packages

Comparing Just a Few Packages

A few packages that facilitate the above approach:

Comparing Just a Few Packages

A few packages that facilitate the above approach:

pkg	flexibility	high-level ops
partools	high	few
ddR	medium	medium
multidplyr	low	more

Going One Step Further: Distributed Files

Going One Step Further: Distributed Files

- Since will do "Leave it there" over many ops,

Going One Step Further: Distributed Files

- Since will do "Leave it there" over many ops,
- might as well distribute a persistent version of the data, i.e. have **distributed files.**

Going One Step Further: Distributed Files

- Since will do "Leave it there" over many ops,
- might as well distribute a persistent version of the data, i.e. have **distributed files.**
- Like Hadoop/Spark, but without the complex machinery.

Going One Step Further: Distributed Files

- Since will do "Leave it there" over many ops,
- might as well distribute a persistent version of the data, i.e. have **distributed files.**
- Like Hadoop/Spark, but without the complex machinery.
- Our **partools** package includes various functions for managing distributed files

Distributed Files in partools

Distributed Files in partools

- File **x** spread across **x.001**, **x.002** etc.
- **filesplit():** Make distributed file from monolithic one.
- **fileread():** If node **i** does **fileread(x,d)**, then **x.i** will be read into the variable **d**.
- **filesave():** Saves distributed data to distributed file.
- Etc.

Partools Example of "Leave It There"

Partools Example of “Leave It There”

- Say have distributed file **xy**, physically stored in files **xy.001**, **xy.002** etc.

Partools Example of “Leave It There”

- Say have distributed file **xy**, physically stored in files **xy.001**, **xy.002** etc.
- Say we have written functions (not shown) **NAtoMean** and **deleteOuts**, to handle missing values and remove outliers, as mentioned before. The functions have been given to the workers

"Leave It There" Example (cont'd.)

"Leave It There" Example (cont'd.)

```
# do NA removal at each worker,
# on the worker's chunk of xy
clusterEvalQ(cls,xy <- apply(xy,2,NAtoMean))
# do the outlier removal at each worker,
# on the worker's chunk of xy
clusterEvalQ(cls,xy <- apply(xy,2,deleteOuts))

# use Software Alchemy to perform linear regression,
# returning just the coefficients in this case
calm(cls,'y ~ .,data=xy')$tht
```

What Is Happening

Norm Matloff
University of California at Davis

What Is Happening

E.g.

```
clusterEvalQ(cls, xy <- apply(xy,2,NAtoMean))
```

What Is Happening

E.g.

```
clusterEvalQ(cls,xy ← apply(xy,2,NAtoMean))
```

We are saying, At each worker node, do

```
xy ← apply(xy,2,NAtoMean))
```

which means, each node does the **apply** op *on its portion of* **xy**.

Norm Matloff
University of California at Davis

"Leave It There" Example (cont'd.)

“Leave It There” Example (cont’d.)

The key point:

"Leave It There" Example (cont'd.)

The key point:

For typical data analysis, hopefully we have:

"Leave It There" Example (cont'd.)

The key point:

For typical data analysis, hopefully we have:

- *Data file stored in distributed fashion.*

"Leave It There" Example (cont'd.)

The key point:

For typical data analysis, hopefully we have:

- *Data file stored in distributed fashion.*
- *Lots of "leave it there" ops:*

"Leave It There" Example (cont'd.)

The key point:

For typical data analysis, hopefully we have:

- *Data file stored in distributed fashion.*
- *Lots of "leave it there" ops:*
  - *Parallel.*

"Leave It There" Example (cont'd.)

The key point:

For typical data analysis, hopefully we have:

- *Data file stored in distributed fashion.*
- *Lots of "leave it there" ops:*
    - *Parallel.*
    - *No network delay.*

"Leave It There" Example (cont'd.)

The key point:

For typical data analysis, hopefully we have:

- *Data file stored in distributed fashion.*
- *Lots of "leave it there" ops:*
    - *Parallel.*
    - *No network delay.*
    - *No serialization overhead.*

"Leave It There" Example (cont'd.)

The key point:

For typical data analysis, hopefully we have:

- *Data file stored in distributed fashion.*
- *Lots of "leave it there" ops:*
  - *Parallel.*
  - *No network delay.*
  - *No serialization overhead.*
- *Have occasional "collect" ops, hopefully small in size, e.g. from an aggregation such as* **colMeans**.

"Leave It There" Example (cont'd.)

The key point:

For typical data analysis, hopefully we have:

- *Data file stored in distributed fashion.*
- *Lots of "leave it there" ops:*
    - *Parallel.*
    - *No network delay.*
    - *No serialization overhead.*
- *Have occasional "collect" ops, hopefully small in size, e.g. from an aggregation such as* **colMeans***.*
- *If change data or create new data, save in distributed file form too!*

"Leave It There" Example (cont'd.)

The key point:

For typical data analysis, hopefully we have:

- *Data file stored in distributed fashion.*
- *Lots of "leave it there" ops:*
    - *Parallel.*
    - *No network delay.*
    - *No serialization overhead.*
- *Have occasional "collect" ops, hopefully small in size, e.g. from an aggregation such as* **colMeans***.*
- *If change data or create new data, save in distributed file form too! Use* **partools::filesave***.*

Norm Matloff
University of California at Davis

Heavy Use of SA

Heavy Use of SA

- Have SA forms of

Heavy Use of SA

- Have SA forms of
  - **lm()/glm()**

Heavy Use of SA

- Have SA forms of
  - **lm()/glm()**
  - k-NN

Heavy Use of SA

- Have SA forms of
    - **lm()/glm()**
    - k-NN
    - random forests

Heavy Use of SA

- Have SA forms of
  - **lm()/glm()**
  - k-NN
  - random forests
  - PCA

Heavy Use of SA

- Have SA forms of
  - **lm()/glm()**
  - k-NN
  - random forests
  - PCA
  - **quantile()**

Heavy Use of SA

- Have SA forms of
  - **lm()/glm()**
  - k-NN
  - random forests
  - PCA
  - **quantile()**
- Very easy to make your own SA functions.

Various Collection Ops

Various Collection Ops

E.g. **addlists()**.

Various Collection Ops

E.g. **addlists()**.
Say have distributed list, 2 compoments. From one, manager node receives

```
list(a=3,b=8)
```

and from the other

```
list(a=5,b=1,c=12)
```

The functions "adds" them, producing (non-distributed)

```
list(a=8,b=9,c=12)
```

Norm Matloff
University of California at Davis

Conclusions

Conclusions

No “silver bullet.”

Conclusions

No “silver bullet.” But the following should go a long way toward your need for parallel computation.

Conclusions

No "silver bullet." But the following should go a long way toward your need for parallel computation.

- SA for the computational stuff.

Conclusions

No “silver bullet.” But the following should go a long way toward your need for parallel computation.

- SA for the computational stuff.
- For aggregation, “leave it there” and distributed files.

Conclusions

No "silver bullet." But the following should go a long way toward your need for parallel computation.

- SA for the computational stuff.
- For aggregation, "leave it there" and distributed files.
- Could do in other packages, not just **partools**.

Conclusions

No “silver bullet.” But the following should go a long way toward your need for parallel computation.

- SA for the computational stuff.
- For aggregation, “leave it there” and distributed files.
- Could do in other packages, not just **partools**.

Ready for the dissent. :-)