Distributed Data Structures and Chunk Based Computing

Indrajit Roy (Principal Researcher, HP)

Workshop on Distributed Computing in R, HP Labs, 2015

Challenges in distributed computing

- Goal: Need a simple API but with good performance

- Solutions:
  - Custom code: Very good performance, but suitable for only ninja programmers
  - R Packages with parallelism constructs: Data movement challenges
  - Interface to other systems (Hadoop, Spark): Need to learn external system's API, installation, not native to R

- Issues to tackle:
- Handle large datasets
- Minimize data movement

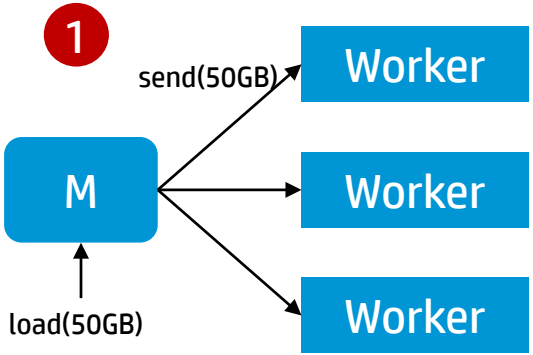
Example: Cost of data movement

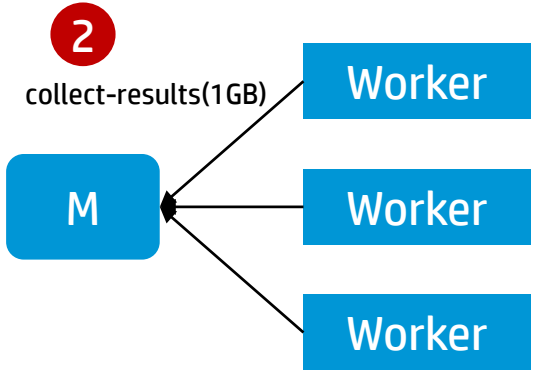
- Many applications reuse data
  - Multi-analysis on same data: load once, run many operations
  - Iterative algorithms: most machine learning + graph algorithms

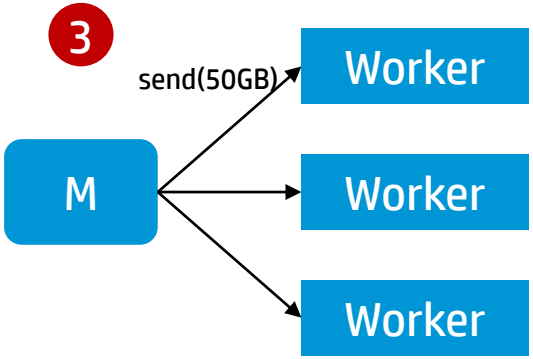
- No persistent reference to distributed data -> High data movement costs

- Common workflow:
- Load data at master->send to workers -> collect at master -> send to worker->repeat

Example: Cost of data movement







Redundant data movement!
No permanent data reference

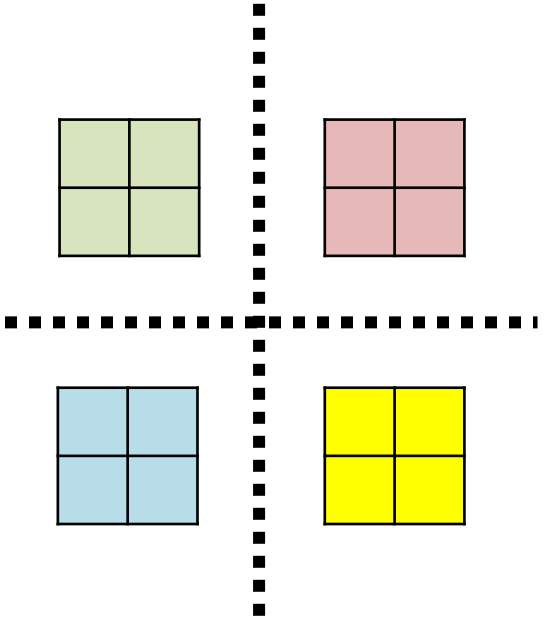
Our approach

Open source package “distributedR”

Enhancement #1: Distributed data structures

- Extend existing R data-structures: darray, dframe, dlist
- Relies on user defined partitioning

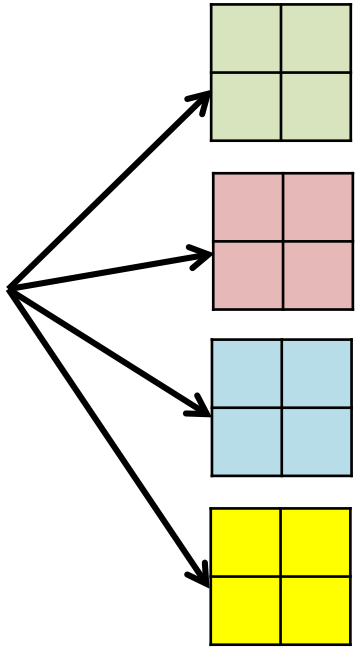
darray



Enhancement #2: Distributed loop

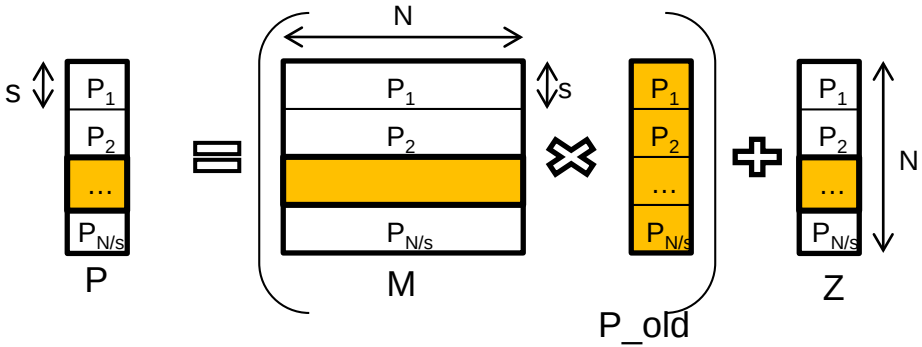
- Express computations over partitions
- Execute across the cluster

foreach $f(x)$



Other option- dapply() ?
Should allow functions on any pair of partitions?

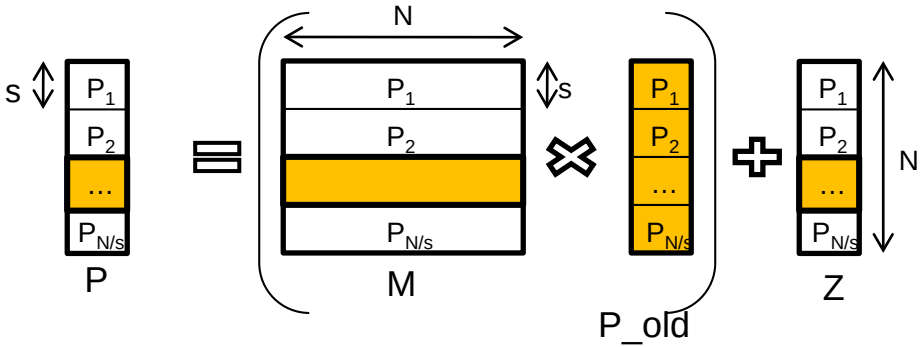
Example: Writing distributed PageRank




```
M ← darray(dim=c(N,N),blocks=c(s,N))
P ← darray(dim=c(N,1),blocks=c(s,1))
while(..){
  foreach(i,1:len,
    function(p=splits(P,i),m=splits(M,i)
             x=splits(P_old),z=splits(Z,i)) {
             p ← (m*x)+z
                 update(p)
     })
  P_old ← P
}
```

Create Distributed Array

Example: Writing distributed PageRank




```
M ← darray(dim=c(N,N),blocks=c(s,N))
P ← darray(dim=c(N,1),blocks=c(s,1))
while(..){
  foreach(i,1:len,
    function(p=splits(P,i),m=splits(M,i)
             x=splits(P_old),z=splits(Z,i)) {
             p ← (m*x)+z
                 update(p)
    })
  P_old ← P
}
```

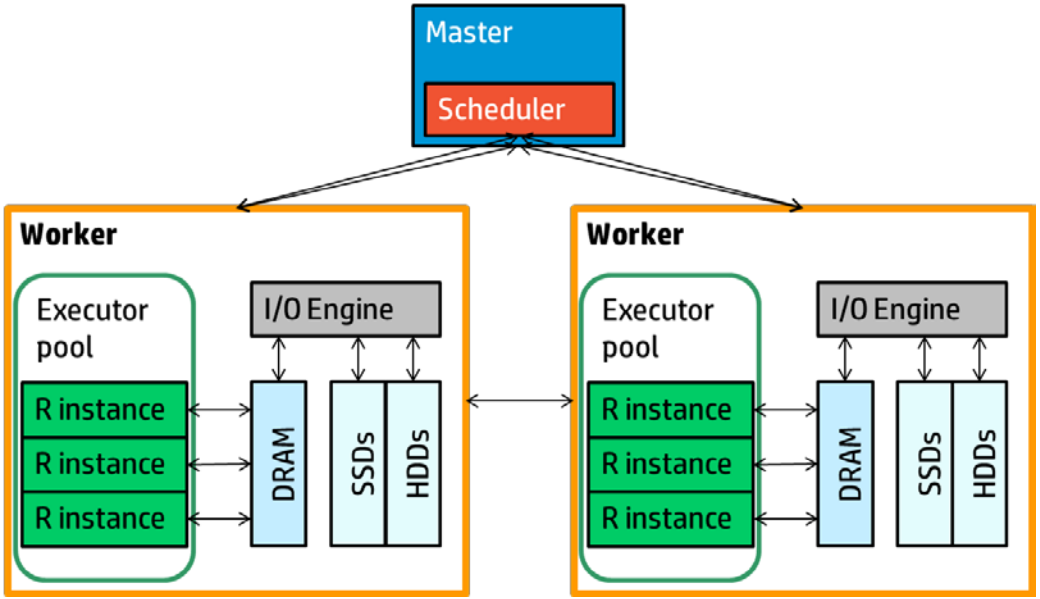
Execute function in a cluster

Pass array partitions

Behind the scenes

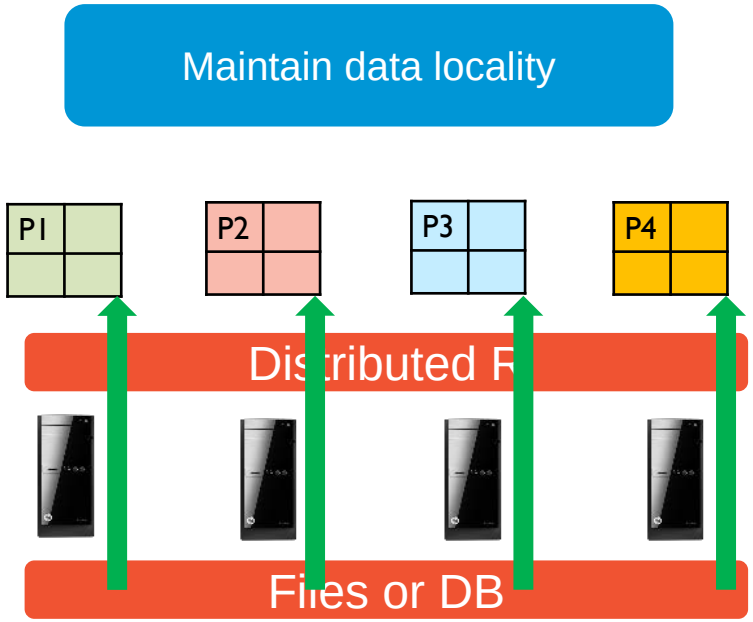
Distributed R architecture

- Programmers never think about physical location of data
- Master and scheduler: performs task and data scheduling
- Worker: executes tasks



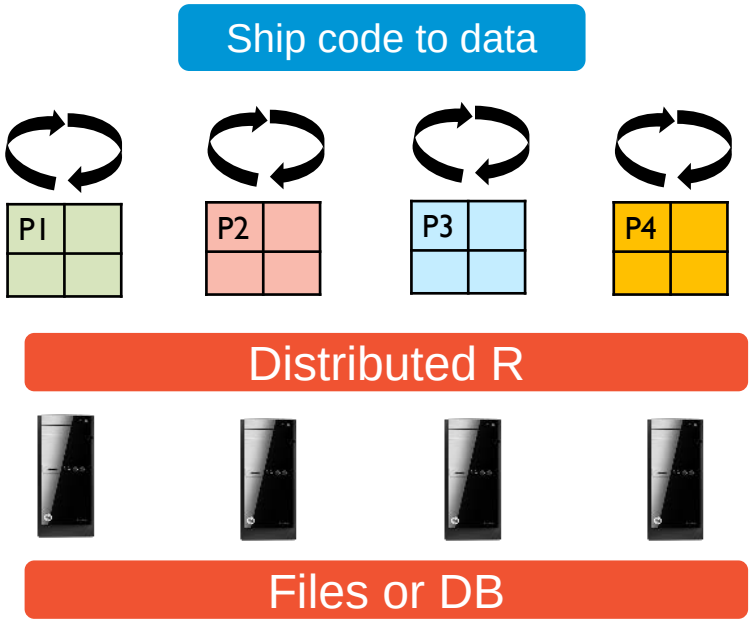
Distributed R architecture

```
1.Load_data()
```



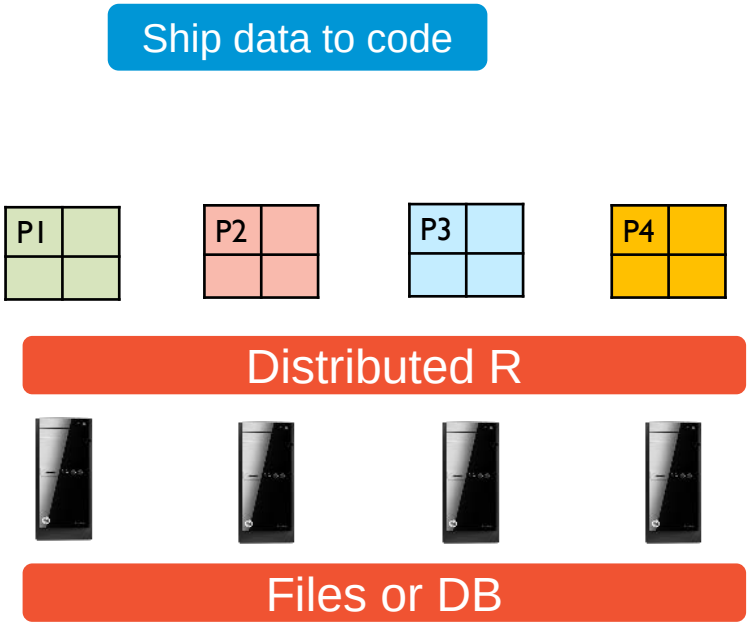
Distributed R architecture

```
1.Load_data()

2.run-function(..)
```



Distributed R architecture

```
1.Load_data()

2.run-function(..)
```



Experience with applications

Examples of what we have written

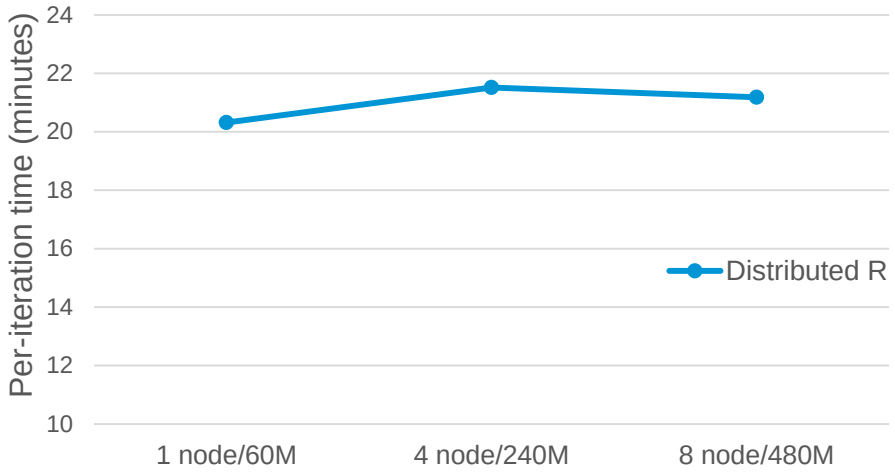
- Embarrassingly parallel applications (add a field to each row)
- Simple statistics (mean, sum, colSums, ..)
- Regression (glm!), clustering algorithms
- Graph algorithms: PageRank
- Tree based algorithms: gradient boosting, randomforest ...

Scalability and performance are good

- Regression on billions of rows, analysis on 10B edge graphs, etc.
- Dataset size: 500+GB

K-means clustering

Rows=60M to 480M, Cols=100
Centers = 1,000
Node = 12 core, 96GB RAM server
Max. dataset size: 360 GB



Some observations

Summary: Keep it simple!

- Focus on data structures. Extend existing R data-structures.
- Add a parallelism construct that works with data-structures (chunk based apply(), foreach()?)
- May need to handle cases when chunk sizes are not known beforehand
- May need built in support for shuffle operation (e.g., groupBy)
- Extend scheduler to disks?

Thank you