5. Map-Reduce

February 16, 2014

In [1]:
```
addprocs(5);
```

In [2]:
```
nprocs()
```

Out [2]:
```
6
```

In [3]:
```
@everywhere function par(I)
  d=(length(I[1]),length(I[2]))
  m=fill(myid(),d)
  println(I)
  println(d)
  return m
end
```

In [4]:
```
m=DArray(par,(8,8),[2:5])
```

```
        From worker 2:  (1:4,1:4)
        From worker 2:  (4,4)
        From worker 3:  (5:8,1:4)
        From worker 3:  (4,4)
        From worker 4:  (1:4,5:8)
        From worker 4:  (4,4)
        From worker 5:  (5:8,5:8)
        From worker 5:  (4,4)
8x8 DArray{Int64,2,Array{Int64,2}}:
 2  2  2  2  4  4  4  4
 2  2  2  2  4  4  4  4
 2  2  2  2  4  4  4  4
 2  2  2  2  4  4  4  4
 3  3  3  3  5  5  5  5
 3  3  3  3  5  5  5  5
 3  3  3  3  5  5  5  5
 3  3  3  3  5  5  5  5
```

In [5]:
```
m.indexes
```

Out [5]:
```
2x2 Array{(Range1{Int64},Range1{Int64}),2}:
 (1:4,1:4)  (1:4,5:8)
 (5:8,1:4)  (5:8,5:8)
```

In [6]:
```
m.chunks
```

Out [6]:
```
2x2 Array{RemoteRef,2}:
 RemoteRef(2,1,15)  RemoteRef(4,1,17)
 RemoteRef(3,1,16)  RemoteRef(5,1,18)
```

In [7]:
```
fetch(@spawnat 2 localpart(m))
```

Out [7]:
```
4x4 Array{Int64,2}:
 2  2  2  2
 2  2  2  2
```

```
 2  2  2  2
 2  2  2  2
```

In [8]:
```
procs(m)
```

Out [8]:
```
4-element Array{Int64,1}:
 2
 3
 4
 5
```

In [9]:
```
map(fetch,{@spawnat p sum(localpart(m)) for p in procs(m)})
```

Out [9]:
```
4-element Array{Any,1}:
 32
 48
 64
 80
```

In [10]:
```
reduce(+,map(fetch,{@spawnat p sum(localpart(m)) for p in procs(m)}))
```

Out [10]:
```
224
```

In [11]:
```
# "Map" function.
# Takes a string. Returns a HashTable with the number of times each word
# appears in that string.
@everywhere function wordcount(text)
    words=split(text,r"\s+|'\n'|'\t'|:|;|,|!|\"|\.|,",false)
    counts=Dict()
    for w in words
        counts[w]=get(counts,w,0)+1
    end
    return counts
end
```

In [12]:
```
# "Reduce" function.
# Takes a collection of HashTables in the format returned by wordcount()
# Returns a HashTable in which words that appear in multiple inputs
# have their totals added together.
@everywhere function wcreduce(wcs)
  counts=Dict()
  for words_dict in wcs
    for (k,v) in words_dict
            counts[k] = get(counts,k,0)+v
        end
    end
    return counts
end
```

In []:
```
## Splits input string into nprocs() equal-sized chunks (last one rounds up),
## and @spawns wordcount() for each chunk to run in parallel. Then fetch()s
## results and performs wcreduce().
## Limitations: splitting the string and reduction step are single-threaded.
#@everywhere function parallel_wordcount(text)
#    lines=split(text,r"'\n'|-|/|@|<|>",false)
#    np=nprocs()
#    unitsize=ceil(length(lines)/np)
#    wcounts={}
#    rrefs={}
#    # spawn procs
#    for i=1:np
#        first=unitsize*(i-1)+1
#        last=unitsize*i
#        if last>length(lines)
#            last=length(lines)
#        end
#        subtext=join(lines[int(first):int(last)],"\n")
#        push!(rrefs, @spawn wordcount( subtext ) )
```

```
#    end
#    # fetch results
#    while length(rrefs)>0
#        push!(wcounts,fetch(pop!(rrefs)))
#    end
#    # reduce
#    count=wcreduce(wcounts)
#    return count
#end
```

In [13]:
```
# Splits input string into nprocs() equal-sized chunks (last one rounds up),
# and @spawns wordcount() for each chunk to run in parallel. Then fetch()s
# results and performs wcreduce().
# Limitations: splitting the string and reduction step are single-threaded.
@everywhere function parallel_wordcount(text)
    lines=split(text,r"'\n'|-|/|@|<|>",false)
    np=nprocs()
    unitsize=ceil(length(lines)/np)
    wcounts={}
    rrefs={}
    # spawn procs
    wcounts=@parallel (hcat) for i=1:np
        first=unitsize*(i-1)+1
        last=unitsize*i
        if last>length(lines)
            last=length(lines)
        end
        subtext=join(lines[int(first):int(last)],"\n")
        wordcount( subtext )
    end
    # reduce
    count=wcreduce([wcounts])
    return count
end
```

In []:
```
## Takes the name of a result file, and a list of input file names.
## Combines the contents of all files, then performs a parallel_wordcount
## on the resulting string. Writes the results to result_file.
## Limitation: Performs all file IO single-threaded.
#function wordcount_files(input_file_names)
#    text=""
#    for f = input_file_names
#        fh=open(f)
#        text=join( {text,readall(fh)}, "\n" )
#        close(fh)
#    end
#    wc=parallel_wordcount(text)
#    for (k,v) = wc
#        println(k,"=",v)
#    end
#end
```

In [14]:
```
# Takes the name of a result file, and a list of input file names.
# Combines the contents of all files, then performs a parallel_wordcount
# on the resulting string. Writes the results to result_file.
# Limitation: Performs all file IO single-threaded.
@everywhere function wordcount_files(input_file_names)
  alltext=@parallel (hcat) for f in input_file_names
        fh=open(f)
        text=readall(fh)
        close(fh)
        text
    end
    if length(input_file_names)>1
        alltext=join(alltext," ") #to string
    end
    wc=parallel_wordcount(alltext)
```

```
    for (k,v) = wc
        println(k,"=",v)
    end
end
```

In [15]:
```
args=["input1.txt","input2.txt","input2.txt"]#"input3.txt"]
wordcount_files(args)
```

```
hello=9
worlds=1
world=8
```

In [16]:
```
a=rand(1000,1000)
```

Out [16]:
```
1000x1000 Array{Float64,2}:
 0.0517374  0.0152483   0.80411     ...  0.479001    0.653943   0.0367196
 0.146748   0.848664    0.409353        0.31859     0.521018   0.734305
 0.715162   0.259179    0.397266        0.491788    0.613991   0.180984
 0.958573   0.494933    0.661546        0.136645    0.396342   0.580141
 0.565306   0.121129    0.696058        0.970205    0.29401    0.469865
 0.214514   0.669887    0.515675    ...  0.417429    0.65146    0.330153
 0.15334    0.719821    0.992338        0.444458    0.10136    0.387632
 0.883156   0.115288    0.810726        0.381167    0.929734   0.765495
 0.300044   0.991191    0.160662        0.855636    0.414701   0.235031
 0.901976   0.87941     0.77229         0.00710375  0.169607   0.978175
 0.799794   0.376283    0.118597    ...  0.297163    0.699328   0.707928
 0.295569   0.891411    0.914866        0.713082    0.428782   0.362351
 0.470957   0.511056    0.697885        0.418426    0.936151   0.440171
 ⋮                                   ⋱
 0.280302   0.00580387  0.288502        0.0155195   0.0120698  0.829593
 0.171318   0.102613    0.149078        0.313723    0.768054   0.0986667
 0.116159   0.813688    0.229467    ...  0.422101    0.546511   0.159127
 0.178151   0.525928    0.815876        0.351391    0.992611   0.794671
 0.285969   0.0739864   0.553199        0.79889     0.567411   0.827591
 0.425504   0.668291    0.862984        0.667938    0.359662   0.929267
 0.947711   0.279987    0.741968        0.55772     0.443613   0.11408
 0.941318   0.0612145   0.893251    ...  0.0408433   0.454859   0.0619818
 0.934473   0.822634    0.951973        0.811734    0.386578   0.480984
 0.89429    0.63726     0.171954        0.498869    0.510155   0.454814
 0.614473   0.912342    0.911658        0.210643    0.31863    0.605135
 0.378741   0.341306    0.0657988       0.623196    0.164191   0.78803
```

In [17]:
```
d=distribute(a)
```

Out [17]:
```
1000x1000 DArray{Float64,2,Array{Float64,2}}:
 0.0517374  0.0152483   0.80411     ...  0.479001    0.653943   0.0367196
 0.146748   0.848664    0.409353        0.31859     0.521018   0.734305
 0.715162   0.259179    0.397266        0.491788    0.613991   0.180984
 0.958573   0.494933    0.661546        0.136645    0.396342   0.580141
 0.565306   0.121129    0.696058        0.970205    0.29401    0.469865
 0.214514   0.669887    0.515675    ...  0.417429    0.65146    0.330153
 0.15334    0.719821    0.992338        0.444458    0.10136    0.387632
 0.883156   0.115288    0.810726        0.381167    0.929734   0.765495
 0.300044   0.991191    0.160662        0.855636    0.414701   0.235031
 0.901976   0.87941     0.77229         0.00710375  0.169607   0.978175
 0.799794   0.376283    0.118597    ...  0.297163    0.699328   0.707928
 0.295569   0.891411    0.914866        0.713082    0.428782   0.362351
 0.470957   0.511056    0.697885        0.418426    0.936151   0.440171
```

```
⋮                                    ⋱
0.280302   0.00580387  0.288502      0.0155195   0.0120698  0.829593
0.171318   0.102613    0.149078      0.313723    0.768054   0.0986667
0.116159   0.813688    0.229467   ...  0.422101    0.546511   0.159127
0.178151   0.525928    0.815876      0.351391    0.992611   0.794671
0.285969   0.0739864   0.553199      0.79889     0.567411   0.827591
0.425504   0.668291    0.862984      0.667938    0.359662   0.929267
0.947711   0.279987    0.741968      0.55772     0.443613   0.11408
0.941318   0.0612145   0.893251   ...  0.0408433   0.454859   0.0619818
0.934473   0.822634    0.951973      0.811734    0.386578   0.480984
0.89429    0.63726     0.171954      0.498869    0.510155   0.454814
0.614473   0.912342    0.911658      0.210643    0.31863    0.605135
0.378741   0.341306    0.0657988     0.623196    0.164191   0.78803
```

In [19]:
```
@time reduce(+,map(fetch,[@spawnat p sum(localpart(d)) for p in procs(d)]))
```

```
elapsed time: 0.006800079 seconds (295516 bytes allocated)
500086.18435936
```

Out [19]:

In [21]:
```
@time reduce(+,d)
```

```
elapsed time: 0.043511909 seconds (309876 bytes allocated)
500086.18435936
```

Out [21]: