Distributed Suffix Array Construction

Huibin Shen

Department of Computer Science
University of Helsinki

String Processing Project
April 25, 2012

Outline

Suffix array defination

The **suffix array** of a text T is a lexicographically orderd array of the set $T_{[0\ldots n]}$ of all suffixes of T. More precisely, the suffix array is an array $SA[0\ldots n]$ of integers containing a permutation of set $[0\ldots n]$ such that

$$T_{SA[0]} < T_{SA[1]} < \cdots < T_{SA[n]}.$$

Example: The suffix array of the text $T = banana\$$

i	$SA[i]$	$T_{SA[i]}$
0	6	\$
1	5	a\$
2	3	ana\$
3	1	anana\$
4	0	banana\$
5	4	na\$
6	2	nana\$

Suffix array construction algorithm

Existing suffix array construction algotirhm:

- Prefix Doubling, $O(nlogn)$.
- **DC3**, $O(n)$.
- SAIS, $O(n)$.

All on one computer!

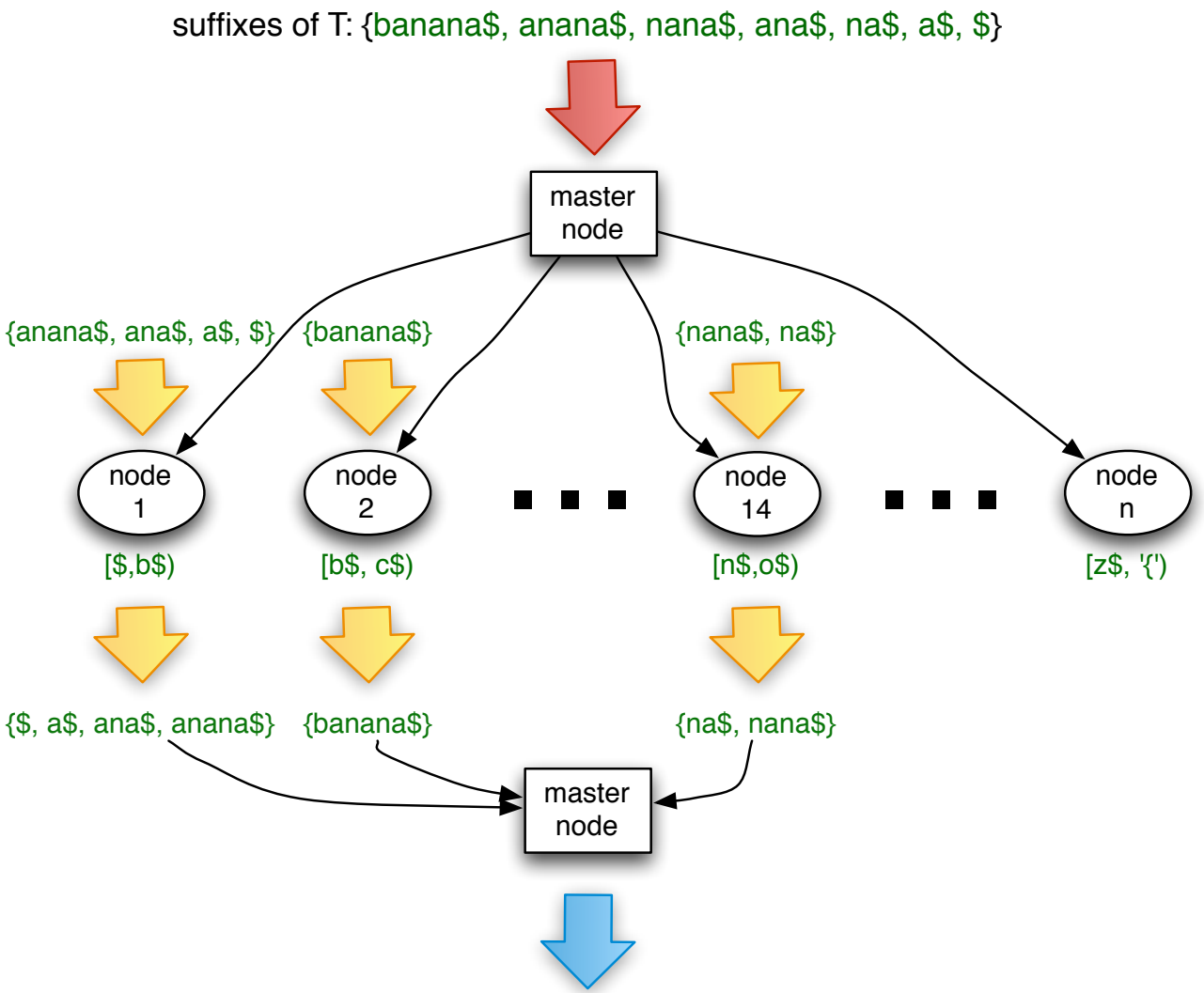
Distributed suffix array construction

Deploy the suffix array construction on clusters!

- Each node of the cluster becomes a bucket for a subset of all the suffiexes
- Each node sorts the subset of suffixes independently.
- Merge the result of each node.

Diagram is shown on next slide.

Distributed suffix array construction diagram



Implementation detail

- Besides the distributed suffix array construction algorithm, a linear time suffix array construction algotirhm **DC3** is also implemented for comparison.
- All codes are written in python.
- Important pieces of code of distributed suffix array construction are shown here, while codes for DC3 are ignored.

Codes for distributing the tasks

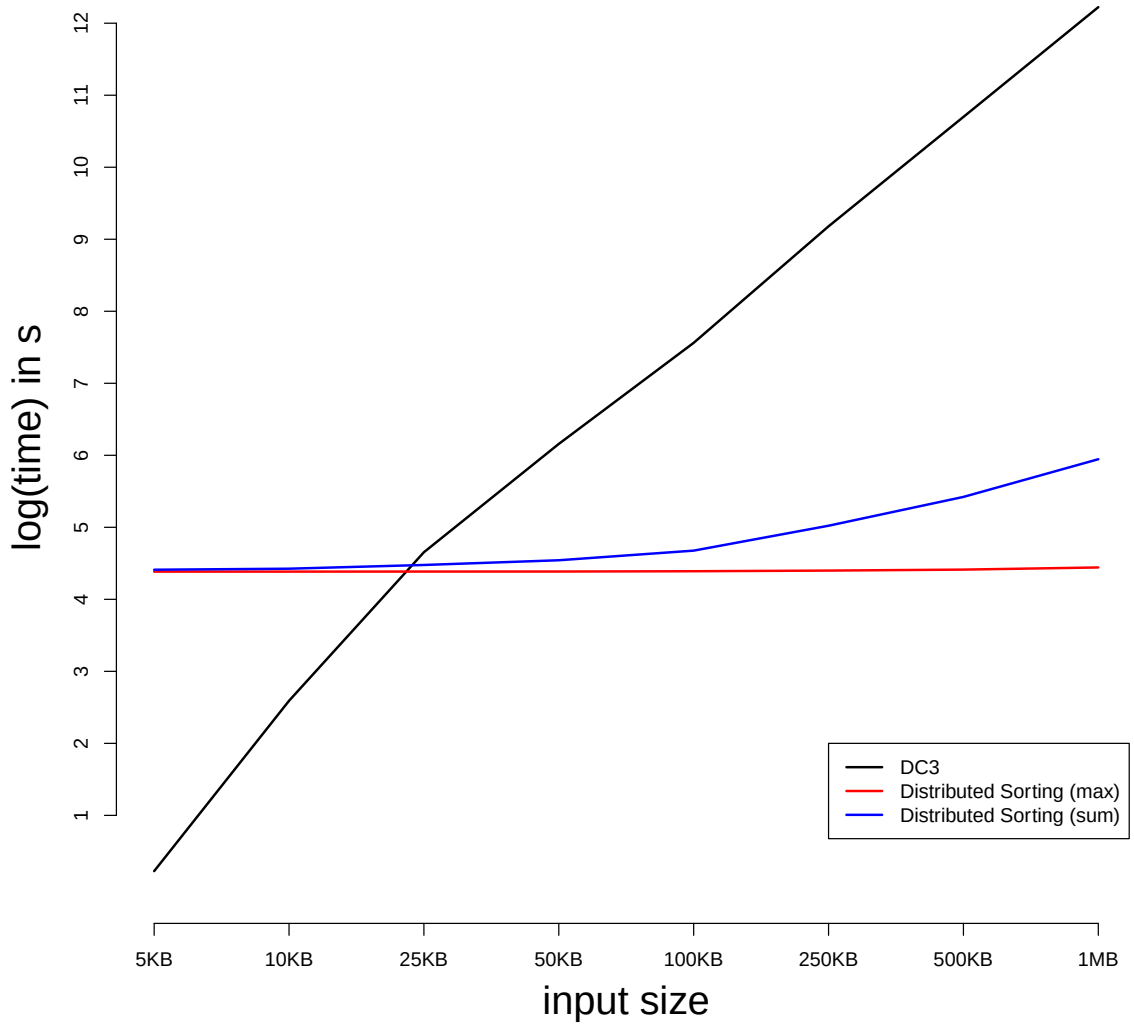
```
# deploy one task on one node
def deploy(node,left,right):
    os.system("ssh %s 'cd /home/fs/hzshen/string_proj/; python
on_one_node.py %s %s %s'" % (node,node,left,right))

#deploy tasks on nodes
for i in range(n_nodes):
    node,load = nodes[i]
    # pid is the index of the left pivot for one bucket
    t.append(Process(target=deploy,args=(node,p_id,p_id+1)))
    p_id += 1
for p in t:
    p.start()
for thread in t:
    p.join()
```

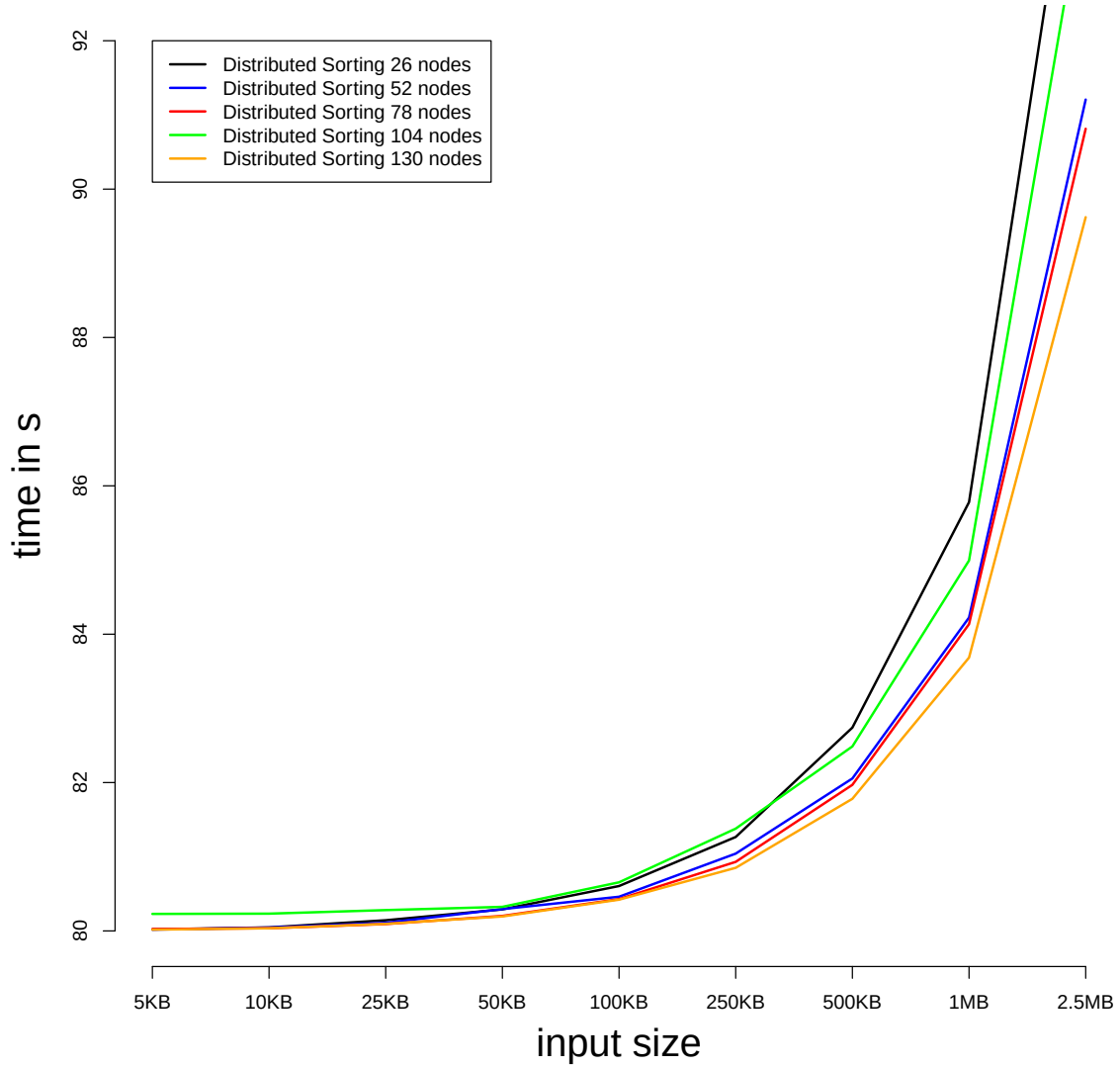
Codes for sorting on one bucket

```
# T is the orginal text, pivots is the list of pivots
# p1, p2 are the indices of left, right pivots
# pivots[p1] is the left pivot, pivots[p2] is the right pivot
# iteration indicates the recurrsion level
# tuples are like [('b','a',1),('a','n',2),...]
# output is the suffix array
def sortSuffixes(T,pivots,p1,p2,iteration,tuples,output):
    tuples = sorted(tuples,key = lambda x: str(x[0])+str(x[1]))
    # partition filters out the tuples that are not belong to this
buckets. Divide tulples by the first two chars into small buckets
until one
    buckets = partition(tuples,p1,p2)
    iteration += 1
    for i in xrange(len(buckets)):
        if len(buckets[i]) == 1:
            output.append(buckets[i][0][2])
        else:
            tuples2 = [] # go one level deeper
            for j in xrange(len(buckets[i])):
                tuples2.append((i,TT[buckets[i][j][2]
+iteration-1],buckets[i][j][2]))
            sortSuffixes(TT,pivots,p1,p2,iteration,tuples2,output)
```

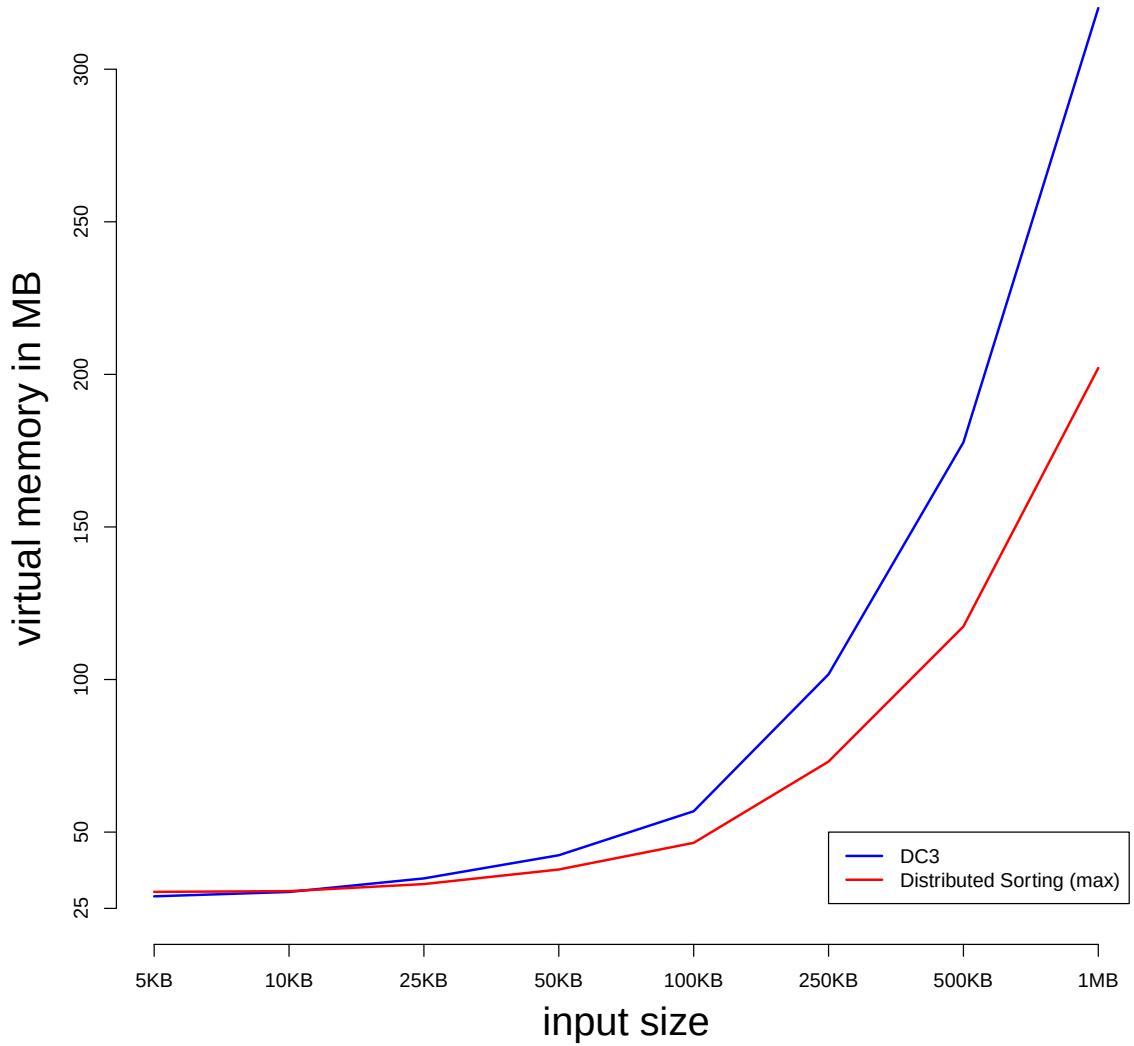
Time comparison between DC3 and distributed sorting with 104 nodes



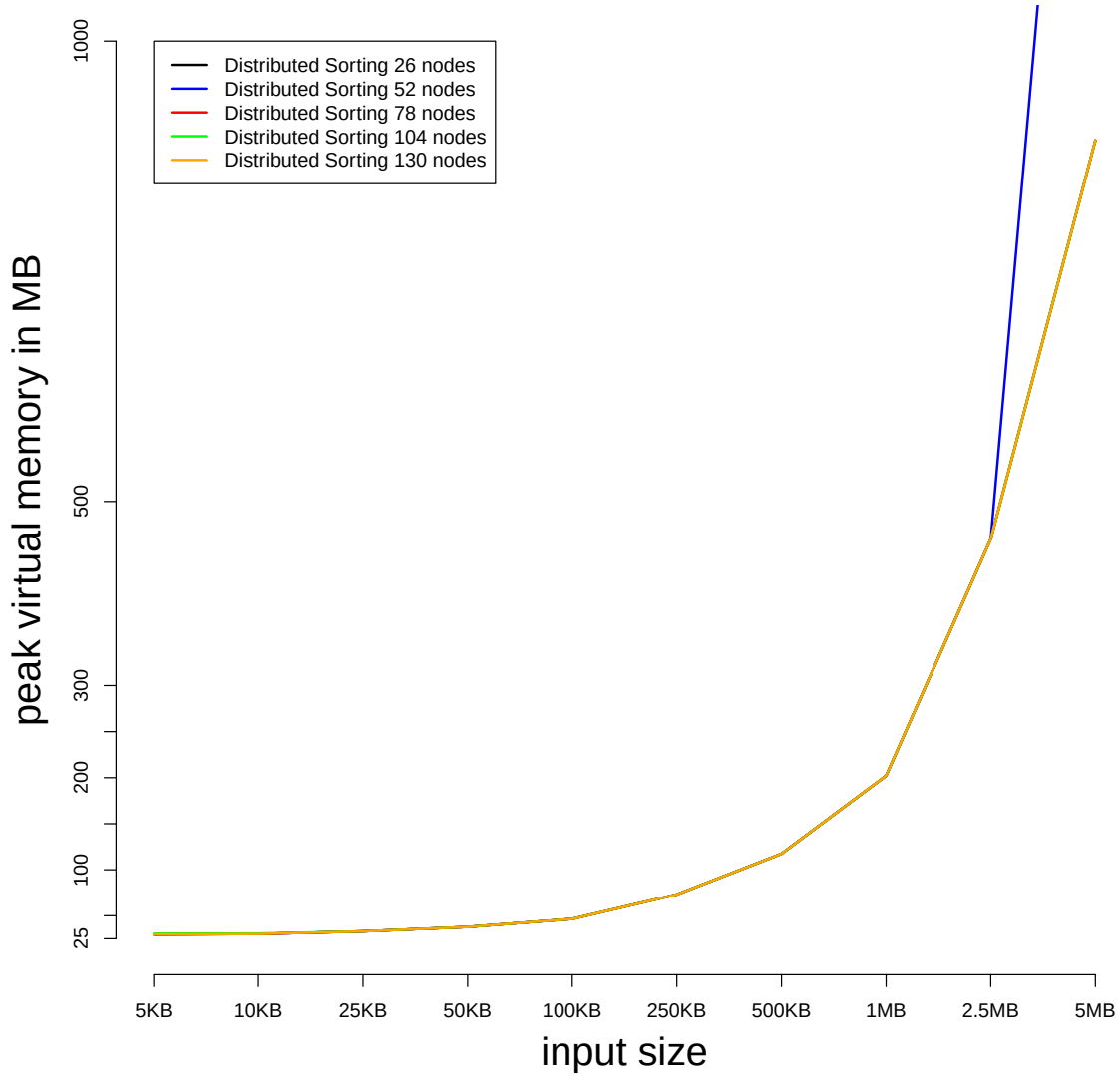
Time comparison in distributed sorting with different nodes



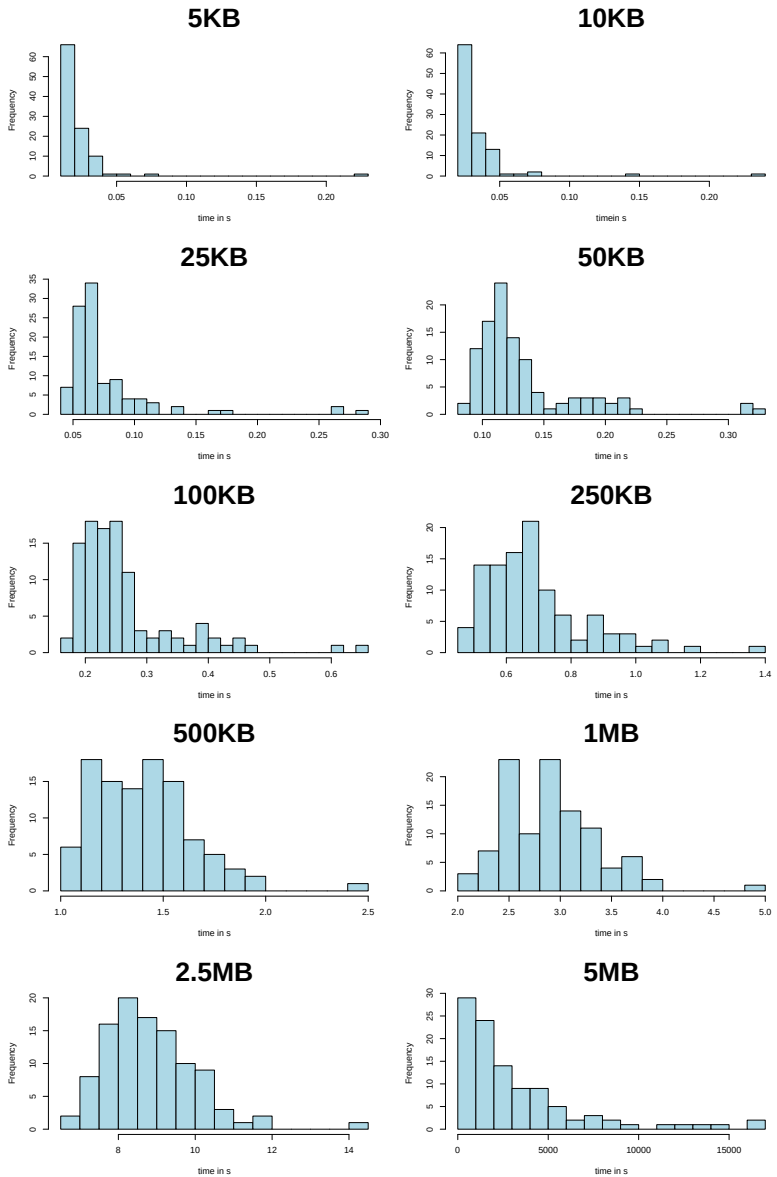
Space comparison between DC3 and distributed sorting with 104 nodes



Space comparison in distributed sorting with different nodes



Running time distribution of distributed sorting over 104 nodes



Lesson and future

- Do **not** use python when implementing string processing algorithm!
- Dynamically construct pivots based on string distribution so that every nodes (buckets) receive as equal amount of task as possible.
- Questions?