Word-Document-Frequency-PRAM-Algorithm analysis

Assumption:

1. Just inside the root directory no documents are present, only directories are there.
2. Stopwords are excluded in Document Frequency Counting.

To solve this question, we first stored all the file names present in the given directory in a vector. For this, the program creates min(2*numCores, number of directories just inside root directory) threads for getting the file names. A workpool model is used here, i.e. a parallel for loop is run on the list of directories inside root directories, each thread takes one directory and traverses it recursively to retrieve names of files inside that directory and when it returns, it checks if some other directory is left to be explored and does traversal on that directory and so on. List of file names obtained by each thread is merged into a single list of file names.

For creating the word document frequency index, number of threads spawned = 2* number of cores (to exploit HyperThreading). Each thread will create its own word document index which will be merged to create a global index later. For creating the local index of a thread, a workpool model is used i.e. a parallel for loop is run on the list of files and each thread processes 1 file at a time.

After creating local indices, merging to a single global index is done using the reduction template. At each level, a thread merges 2 local indices into 1. So from bottom level to top level the number of threads are halved at each level.

For getting the k max frequent words, a min heap of size k is created. The function iterates over each word in the global index and inserts the word in the min heap if the frequency of this word is greater than the frequency of word at the root of heap. Finally the heap will contain the k max frequent words.

Cost Analysis

Let b = average branching factor
d = average depth
w = number of words per file
t = number of threads
Total files = b^d

Sequential Cost Calculation:

Time taken to traverse root directory = $1 + b + b^2 + \dots b^d = O(b^{d+1})$
Time taken to create index = $O(b^d*w)$
Time taken to get k most frequent words(using min heap) = $O(b^d*w*\log k)$
Sequential cost = sequential time complexity = $O(b^d*w*\log k + b^{d+1})$

Parallel Cost Calculation:

Time taken to traverse root directory(by one thread)= $(b/t)*(1 + b + b^2 + \dots b^{d-1}) = O(b^{d+1}/t)$
Time taken to create local index = $O((b^d/t)*w)$
Size of one hash map = s = $O((b^d/t)*w)$

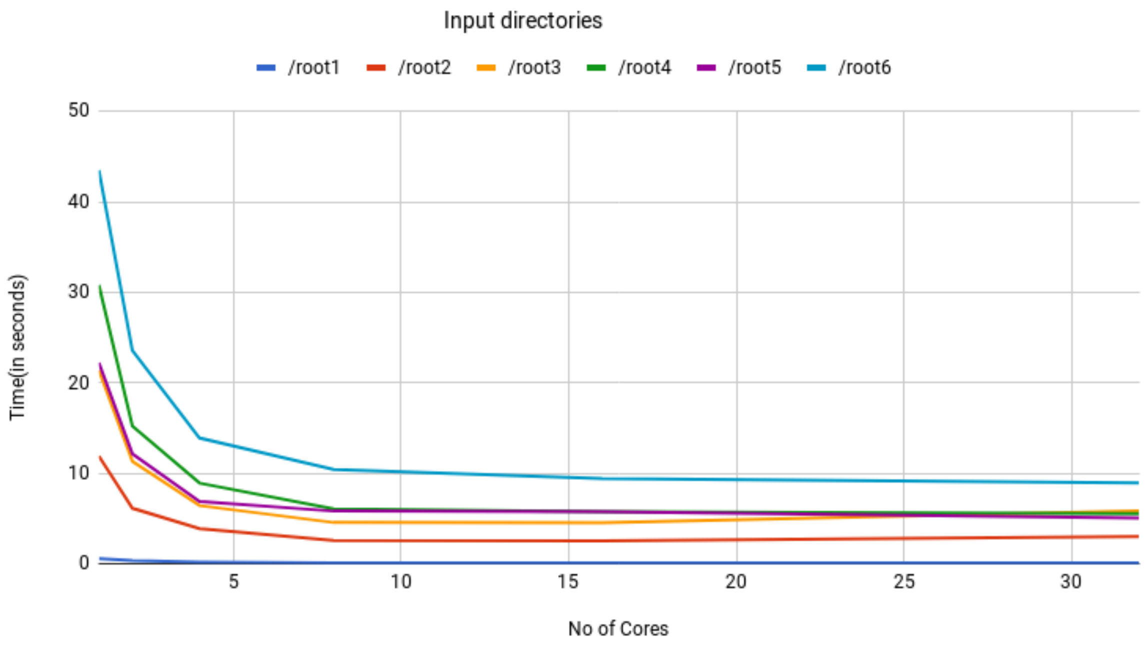
Time taken to merge local indices to create global index (using reduction template)= s + 2s + 4s + …. + (t/2)*s = s[t - 1] = O(ts) = O(b^d*w)
Time taken to get k most frequent words(using min heap) = O(b^d*w*logk)
Parallel time complexity = O(b^{d+1}/t + b^d*w*logk)
Parallel cost = O((t/2)*(b^{d+1}/t + b^d*w*logk))
Results of Question 2 are shown below:

No of Cores	**1**	**2**	**4**	**8**	**16**	**32**
/root1	0.610472	0.391024	0.246248	0.164279	0.147378	0.144487
/root2	11.937185	6.188866	3.928289	2.618586	2.582294	3.062235
/root3	21.434965	11.341022	6.478076	4.607337	4.565631	5.905183
/root4	30.838567	15.248915	8.962478	6.1123	5.785661	5.581537
/root5	22.266326	12.203375	6.937046	5.869809	5.794286	5.106382
/root6	43.513076	23.58714	13.942761	10.477017	9.483234	8.995281

Parameters of Input Directories:

Input directory	**Max Depth**	**Average Depth**	**Average Branching Factor**	**Number of Files**
/root1	2	2	66	3984
/root2	2	2	494	43912
/root3	8	5	448	89998
/root4	5	3	493	134584
/root5	2	2	1001	81668
/root6	2	2	1015	163336

Question 2


Input directories
/root1
/root2
/root3
/root4
/root5
/root6
50
40
30
20
10
0
Time(in seconds)
5
10
15
20
25
30
No of Cores