Lab – 1: Analysis

1. Best speedup achieved ≈ 2.15

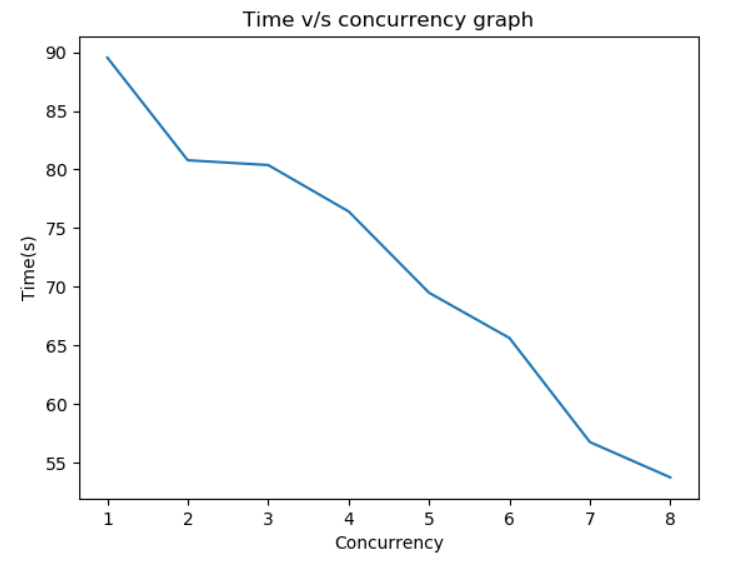
For input of 3003 files

T(s) = 1:51.581 min

T(p) = 0:51.925 min

1. Input size is fixed to the entire directory size i.e. 3003 files
   1. Variation of time with respect to concurrency is stated below :

|  |  |
| --- | --- |
| Concurrency | Time(s) |
| 1 | 89.539 |
| 2 | 80.785 |
| 3 | 80.379 |
| 4 | 76.421 |
| 5 | 69.488 |
| 6 | 65.626 |
| 7 | 56.754 |
| 8 | 53.742 |

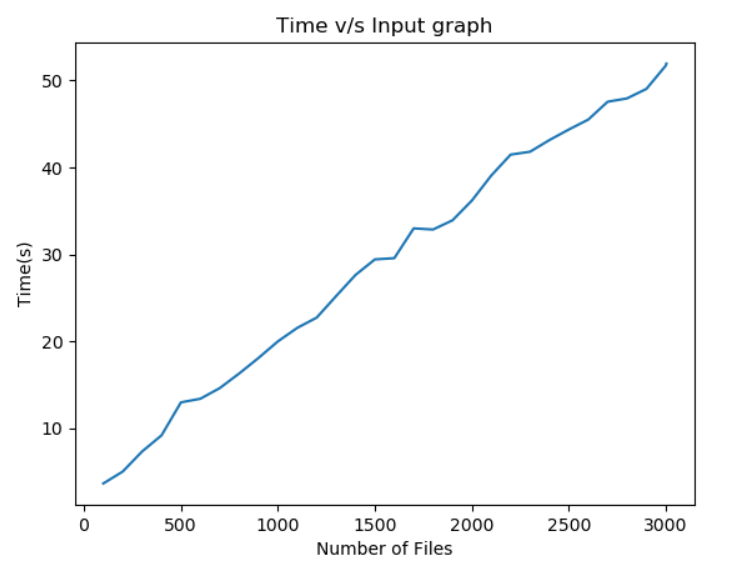


Here we can observe that as we increase the concurrency we can see the required time of execution decreases.

As T(p) is theoretically equal to N/p\*n + reduce time and as p increases N/p decreases

1. Now concurrency is fixed to 8
   1. Variation of time with respect to input size is given below:

|  |  |
| --- | --- |
| Number of Files | Time(s) |
| 100 | 3.646 |
| 200 | 5.014 |
| 300 | 7.325 |
| 400 | 9.177 |
| 500 | 12.970 |
| 600 | 13.385 |
| 700 | 14.606 |
| 800 | 16.284 |
| 900 | 18.078 |
| 1000 | 19.984 |
| 1100 | 21.538 |
| 1200 | 22.716 |
| 1300 | 25.188 |
| 1400 | 27.621 |
| 1500 | 29.412 |
| 1600 | 29.550 |
| 1700 | 32.977 |
| 1800 | 32.849 |
| 1900 | 33.902 |
| 2000 | 36.174 |
| 2100 | 39.053 |
| 2200 | 41.462 |
| 2300 | 41.791 |
| 2400 | 43.136 |
| 2500 | 44.348 |
| 2600 | 45.485 |
| 2700 | 47.541 |
| 2800 | 47.926 |
| 2900 | 49.018 |
| 3000 | 51.701 |
| 3003 | 51.925 |



Here we can observe that as the number of files increase the time also increases in almost a linear fashion

As T(p) is theoretically equal to N/p\*n + reduction time

So as N increases Time also increases

4.

The designed solution is scalable as if we calculate the iso-efficiency it evaluates to :

Ts = N\*(n+a) = O(N\*n)

Here a ≡ Dictionary update operations

Tp = N∗n /p + c

Speedup(s) = Ts/ Tp

Efficiency = Speedup/ p = N∗n/ p∗( N∗n /p +c) = 1/ 1+ c∗p /Ts

So iso efficiency is a linear in p and hence scalable For large amounts of data for practical purposes to overcome synchronization, context-switching and memory overheads. I have implemented a batching approach. Thereby instead of parallelizing each file, a chunk of files are parallelized i.e. a batch of (no. of files / 21) is created and then these batches are parallelized

5.

The designed solution is fault tolerant as late acks is True which means that an acknowledgement is sent only when a task is completed not when a task is received. So if a worker goes down before task completion it doesn’t affects the entire process.