COL733
Lab Assignment – 1
Batch Processing

ANALYSIS :-

Time taken by serial program (Ts) = 43.033 s
Speedup = Ts/Tp
Efficiency = Speedup/p

1. Speedup and Efficiency on varying the no. of processors and the No. of files to be distributed to a processor in 1 task (Chunk size) :

Chunk Size = 100

No. of processors (p)	Time taken (Tp)	Speedup(Ts/Tp)	Efficiency(Speedup/p)
1	46.657 s	0.92232677	0.92232677
2	26.477 s	1.62529743	0.81264872
3	19.907 s	2.16170191	0.7205673
4	17.031 s	2.52674535	0.63168634
5	15.017 s	2.86561897	0.57312379
6	13.729 s	3.1344599	0.52240998
7	13.161 s	3.26973634	0.46710519
8	12.570 s	3.42346858	0.42793357

Chunk size = 150

No. of processors (p)	Time taken (Tp)	Speedup(Ts/Tp)	Efficiency(Speedup/p)
1	47.205 s	0.91161953	0.91161953
2	26.626 s	1.61620221	0.80810111
3	20.348 s	2.11485158	0.70495053
4	16.817 s	2.55889873	0.63972468
5	14.888 s	2.89044868	0.57808974
6	14.103 s	3.0513366	0.5085561
7	13.059 s	3.29527529	0.47075361
8	12.817 s	3.35749395	0.41968674

Chunk size = 200

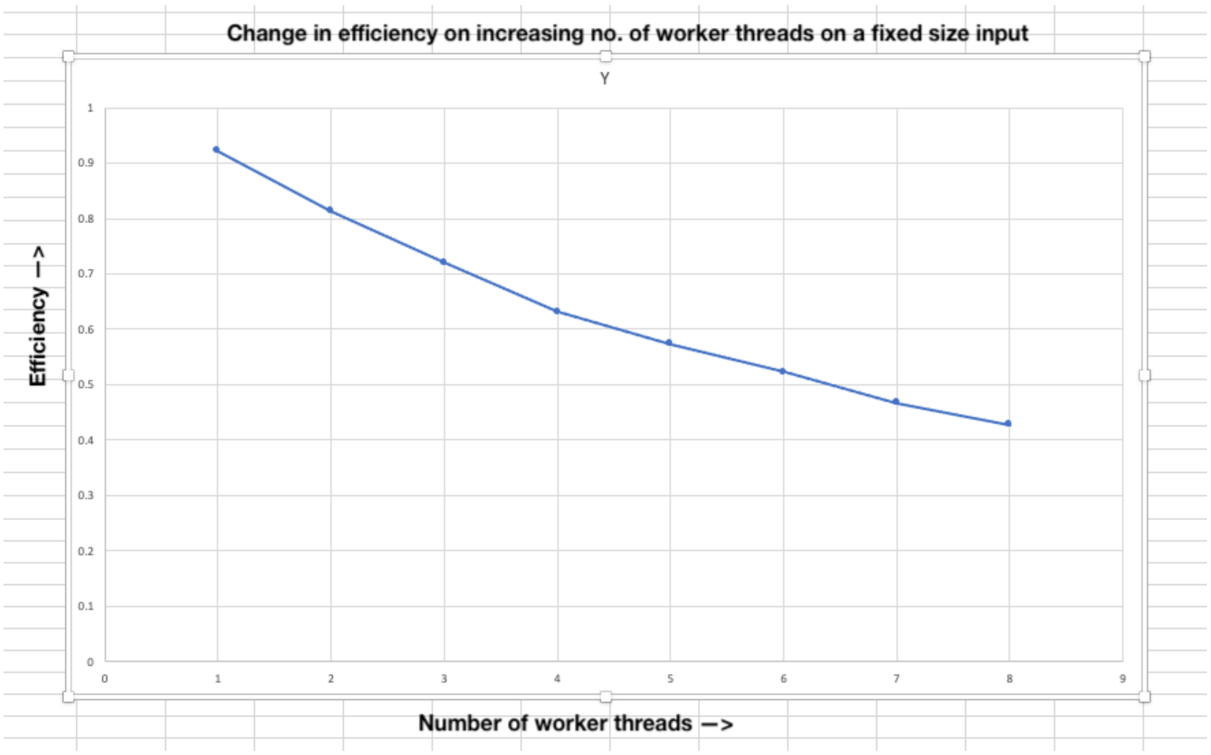
No. of processors (p)	Time taken	Speedup(Ts/Tp)	Efficiency(Speedup/p)
1	45.641 s	0.94285839	0.94285839
2	26.887 s	1.60051326	0.80025663
3	19.465 s	2.21078859	0.73692953
4	16.607 s	2.5912567	0.64781418
5	14.344 s	3.00006972	0.60001394
6	14.688 s	2.92980664	0.48830111
7	12.776 s	3.36826863	0.48118123
8	12.674 s	3.39537636	0.42442205

2. Varying input size by keeping the number of work threds constant (=8)

Input size	Time parallel	Time serial	Speedup	Efficiency
500	3.233s	7.523s	2.32694092	0.29086762
1000	5.120s	14.196s	2.77265625	0.34658203
1500	6.650s	21.803s	3.27864662	0.40983083
2000	8.438s	29.734s	3.52382081	0.4404776
2500	9.555s	37.148s	3.88780743	0.48597593
3000	12.726s	43.922s	3.45135942	0.43141993

- **What is the best speedup achieved over serial.py?**
  We can see from these tables that the best speedup which we can achieve is **3.42346858**, which we get when the concurrency is 8 and when the chuck size which is basically the number of files given to one processor in one task is 100.

- **Given a fixed input size, measure how the efficiency of the word-count application varies with an increase in worker threads allocated to the application. Justify.**



The above graph clearly shows that the efficiency decreases with the increase in the number of working threads.

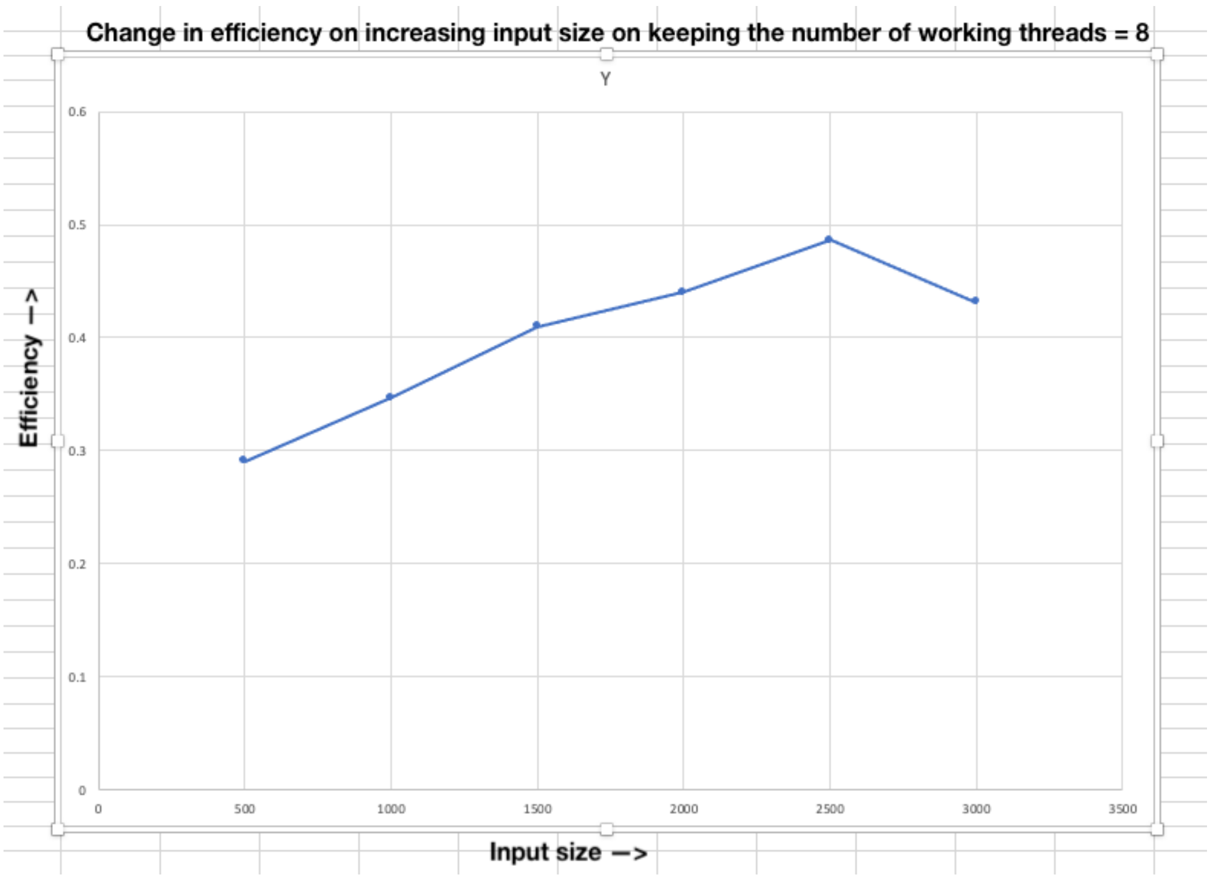
The factors contributing it are :-

1. Increase in the overhead : As number of threads increase the overhead in

establishing a communication between them increases which decreases their efficiency.

2. Increase in the idle time : The parallelly running programs output dictionary which have to be merged in the Master program by a single thread, so at that time, all the other threads except one are idle and hence the if we sum up the idle time of the threads in the program, it increases with increase in the number of threads and hence the efficiency decreases.

- **Given a fixed worker thread (= 8) allocated to the application, measure how the efficiency of the word-count application varies with input size. Justify.**



The above graph shows that initially the efficiency increases with the increase in the number of threads but as the input size increases more and more the efficiency starts decreasing.

This increase is because
-> When the input size is less, then some of the threads doesn't get work to do or less work to do as compared to the others and are sitting idle for some time (Eg. When input size = 500, chunk size = 100, only 5 threads will get work in the starting)
So, due to the uneven division of work because the work is so less, the idle time of threads increases and the efficiency decrease.
But as we increase the amount of work , the efficiecny starts to increase upto a certain point.

But after that it decreases
-> Because of the reasons mentioned in the question above, that the idle time increases as the merging of all dictionaries takes place in the Master program, where only 1 thread is employed during the merging process.

- **The designed solution is scalable. Justify.**

  It is scalable as if the threads are increased the time is reduced, i.e., the Speedup increases, hence if we provide more resources than it is able to utilise them.
  And if we have enough resources than increasing the input size also increases the speedup, which shows that it is able to process larger inputs given to it with a good speedup.
  Hence, the program is scalable.

- **The designed solution is fault-tolerant. Justify.**

  The designed solution is fault tolerant as,

  1. The acknowledgement of the completion of the task is received after its actual completion as acks_late is set to True in our program. So if a task is assigned to a processor and the processor is dead in betwenn then Celery can figure this out as acknowledgement won't be received in such a case and thus it will assign it to some other processor.

  2. The task of counting the words is independent of each other, even if is is done twice the results won't be different.

Submitted By :-
- Nikita Bhamu
- 2018CS50413