COL733: Cloud Computing

Lab Analysis

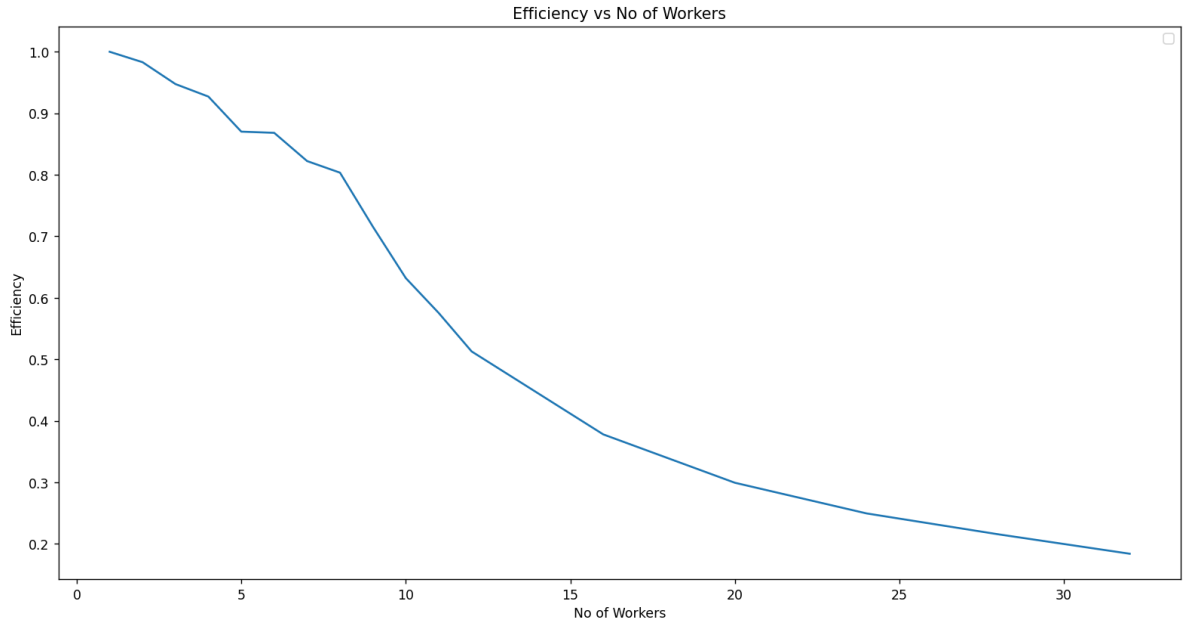
Shivam Singh (2020CS10383)

Lab 1 Analysis: Batch Processing

250 files of size approx 2.7 MB containing tweets were used for testing on baadalvm with configuration of 8 core CPU.

- **Speedup**
  Speedup (S) = T_s / T_p
  Here, T_s = 22.852 s and T_p = 3.697 s (p = 8)
  => S = 22.852 / 3.697
  => S = 6.18
  Max Speed up was observed with 8 threads, which made the tweets processing program ~6 times faster than that on single core.

- **Efficiency vs No. of Workers**
  Efficiency (E) = S/p = T_s / T_p*p
  Here, S = 6.18 and p = 8
  => E = 6.18/8
  => E = 0.77
  Efficiency corresponding to Max Speed up is 0.77.
  However, the trend of efficiency on increasing no. of workers (processor) is depicted by following plot:



As we can clearly see the efficiency keeps on decreasing as we keep on increasing the number of processors. The write to redis is the bottleneck here, as it is serial and synchronization is required for access of redis. Due to this there is decrease in efficiency. Apart from that when the number of core is less than the number of thread, it that case there is slow down due to context switches.

- **Isoefficiency and Scalablity**

  The isoefficiency, as we know, is the rate at which the problem size needs to grow, for the efficiency to remain constant on adding one processor. The isoefficiency here is O(p) as we can see the fraction in the denominator and it suggests that the input size (n) needs to grow in linear proportion to number of processors(p). Thus our implementation of the word count program is scalable.

Lab 2 Analysis: Fault Tolerance

We can affirm that our system maintains consistent word counts even if a worker crashes, assuming redis is fault tolerant. The assertion is supported by following points:

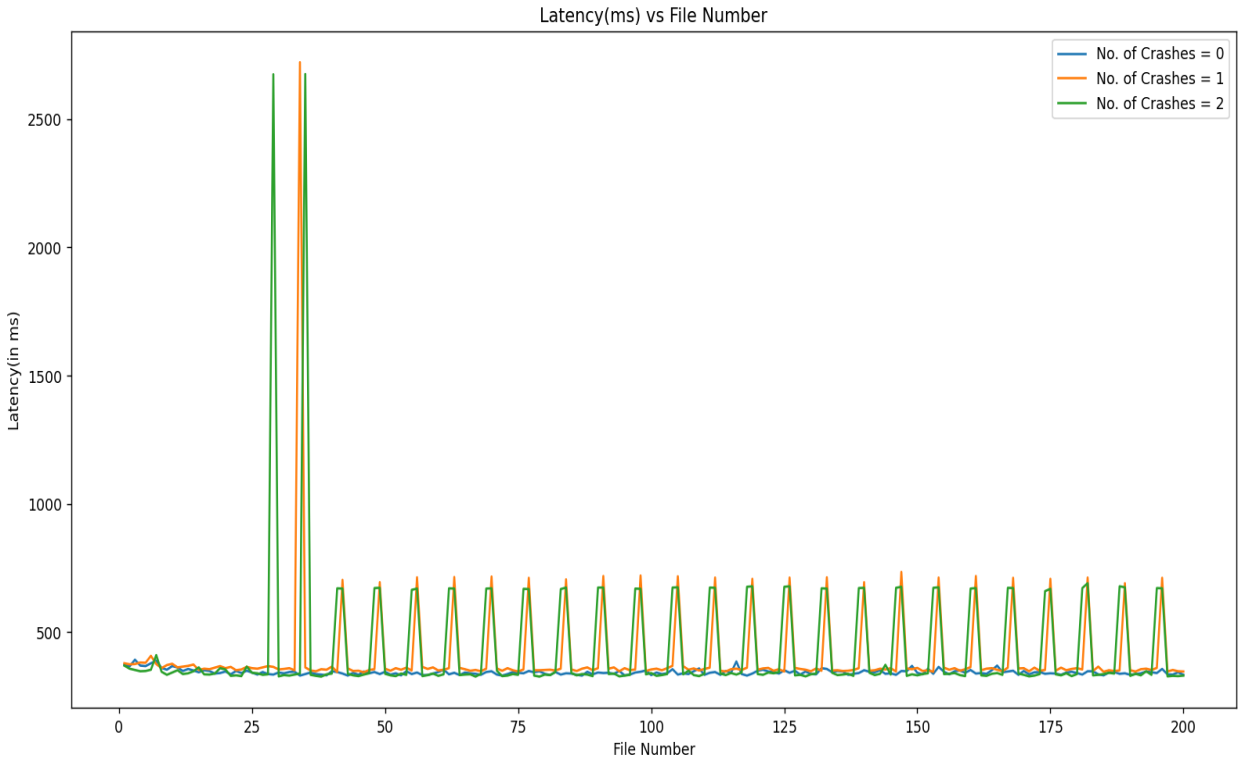
- **Acknowledgement:** If a worker read a file from redis stream and but it crashes without updating the word count, the file remains unacknowledged and using "**xautoclaim**" it can be taken by other worker later for processing.
- Worker crashes during the word count update process are also managed, since all word count data is stored in redis and updates in redis server occur atomically.
- **"Add counts"** function guarantees that the entire process of incrementing the word count and acknowledging the file in redis is atomic.

Lab 3: Stream Processing

- **Load Balancing Method Implemented**
  Each worker examines whether there are files that have remained unprocessed (i.e unacknowledged) for more than 2.4 seconds. This can be done by setting the min idle time in **"xautoclaim"**. If such files exist, the worker will claim them and initiate processing. This mechanism effectively addresses the issue of straggler workers and balances the load between all of them so that all the files are processed in decent time even if some worker fails or slows down.

- **Effects of Crashing of Workers**

  The measurements for this were taken by averaging over several runs and 200 input files each of 2.7 MB size and 7 total workers with batch size m = 8.



When all workers are functioning smoothly the graph depicts nearly a consistent starting line with minimal fluctuations. However, when a single worker experiences a crash, the graph shows a significant spike in latency, clearly indicating the impact of the crash. Likewise, in case of two worker crashes, the graph exhibits two prominent latency peaks.

- **Effect of slowdown of workers**

  When we limit cpu for workers we see that when the limit is very tight i.e. about 30 we get latency peaks but as we loosen the limit to 50 they decrease and till 70 it approaches normal behavior.