Module 6: Critical Thinking

Brady Chin

Colorado State University Global

CSC507-2: Foundations of Operating Systems

Dr. Joseph Issa

February 23rd, 2025

Critical Thinking 6

In the case of processing large files, with millions of rows, execution times are typically a serious constraint when the operation includes reading, processing, and writing large volumes of data. The bigger the files are, the more indispensable is effective handling of data. Under such circumstances, the construction of techniques for optimizing file processing—by making use of parallelism, memory usage reduction, and disk I/O restriction—acquires the highest priority. This can be achieved by breaking up the file into smaller chunks and processing them simultaneously, among other strategies. In the following discussion, we'll explore various methods for speeding up file processing and identify the most effective approaches for handling files that are significantly larger than the typical workload.

Efficiency and Processing Time

The most efficient method of dealing with significantly larger files is to scale by reducing parallelism, memory accesses, and disk usage (Databricks, 2025, February 4). Running the process as is would be the most sluggish because it would be single-threaded with plenty of memory and disk access. Dividing the file into smaller chunks with more of them and processing them in parallel is more efficient, but the number of partitions to be used is system-dependent on the number of CPU cores. For an eight-core machine, splitting the file into twenty or ten chunks would be optimal for CPU usage without too much contention. Too few partitions, such as two, may not be able to use available processing time, but too many, such as twenty, may be susceptible to overhead due to too much context switching and disk contention.

For further efficiency, it's important to read the file in pieces rather than opening the whole file into memory (Saturn Cloud, 2023, Decemeber 7). Reading via a memory-mapped file (mmap) or buffered reading (csv.reader or pandas.read_csv(chunksize=…)) can prevent excessive RAM usage. Further, instead of writing to multiple temp files and then consolidating them, a queue-based system can be more effective in processing.

By multiprocessing with an optimum number of partitions, the processing time can be reduced. For example, a several-hour single-threaded process can be cut down to a fraction of it if properly parallelized. But after a point, disk I/O becomes the bottleneck, and optimization cannot help regardless of the amount. A balance of CPU parallelism, optimum reading of files, and minimum disk access is the secret to the optimum processing time (MoldStud Research Team, 2025 February 13).

Conclusion

For large files, efficiency is essential to reduce processing time. Partitioning the file into segments and parallel processing are measures that can significantly speed up operations, especially if the system is able to run several processes simultaneously. However, the secret to maximum performance is achieving the right balance between parallelism and resources like disk I/O and memory. By employing proper reading techniques and minimizing unnecessary disk writes, speed of data processing tasks can be greatly improved. Lastly, whether or not an ideal solution can be found remains dependent on hardware specifics and what the task type is, but careful attention to such techniques will yield large benefits in terms of performance.

References

Databricks (2025, February 4) *Best Practices for Performance Efficiency.* Apache Software Foundation. https://docs.databricks.com/aws/en/lakehouse-architecture/performance-efficiency/best-practices

MoldStud Research Team, (2025 February 13) *SQL Tuning Strategies for Optimal CPU Memory IO Balance.* MoldStud. https://moldstud.com/articles/p-sql-tuning-strategies-for-optimal-cpu-memory-io-balance

Saturn Cloud (2023, December 7) *How to Efficiently Read Large CSV Files in Python Pandas.* https://saturncloud.io/blog/how-to-efficiently-read-large-csv-files-in-python-pandas/