Reddis & Celery Execution Time

Tuesday, June 10, 2025 12:18 PM

Total Times Graph:

Submission Time: The total time it took to create and submit all DailyLog entries and their associated Celery tasks in the batch. This measures how quickly your system can queue up work.

End-to-End Time: The total time from the start of the test until the last task completed. This represents the complete latency of processing the entire batch.

Throughput Graph

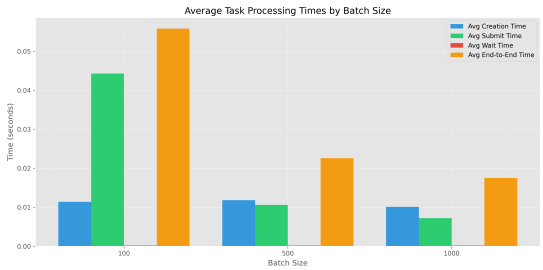
- Submission Throughput (tasks/second): How many tasks can be created and submitted per second. This increases with batch size, showing your system can handle higher loads efficiently.
- End-to-End Throughput (tasks/second): The overall rate at which tasks are completed from start to finish. This also increases with batch size but at a slightly lower rate than submission throughput.

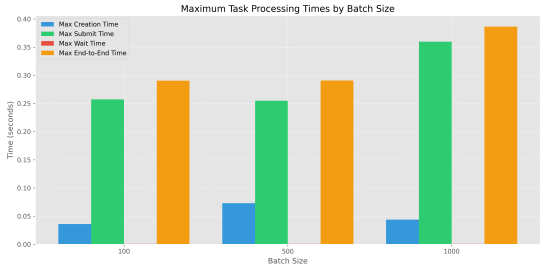
Average Task Times Graph

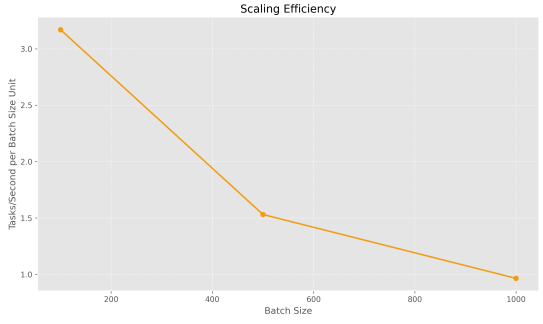
- Avg Creation Time: The average time to create a single DailyLog entry in the database.
- Avg Submit Time: The average time to submit a single Celery task after creating the DailyLog.
- Avg Wait Time: The average time spent waiting for a task to complete after it's been submitted.
- Avg End-to-End Time: The average total time for a single task from creation to completion.

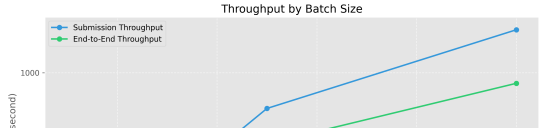
Maximum Task Times Graph

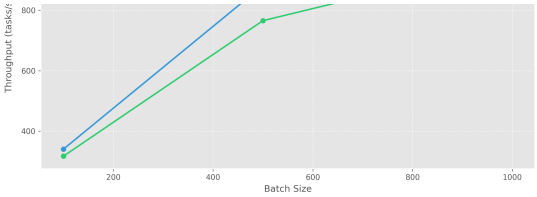
- Max Creation Time: The longest time it took to create any single DailyLog entry.
- Max Submit Time: The longest time it took to submit any single Celery task.
- Max Wait Time: The longest wait time for any task to complete after submission.
- Max End-to-End Time: The longest total time for any single task from creation to completion.

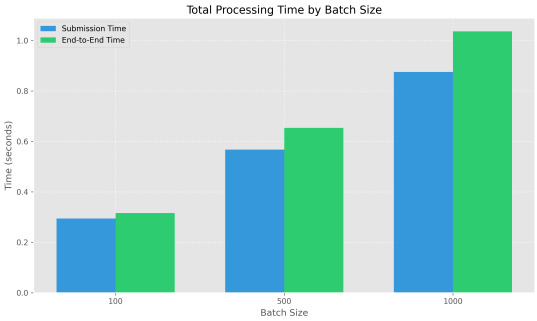












Scaling Efficiency Graph

- This shows the throughput per batch size unit (tasks/second divided by batch size). It indicates how efficiently your system scales as the batch size increases. A flat or increasing line would indicate linear or super-linear scaling, while a decreasing line suggests diminishing returns as batch size grows.

Amortized Overhead:

- With larger batches, the fixed overhead costs (like initializing connections, warming up the worker pool) are spread across more tasks, reducing the average cost per task.

Worker Pool Efficiency:

- Celery's worker pool becomes more efficient when fully utilized. With small batches, workers might not all be busy simultaneously, but with larger batches, all workers can process tasks in parallel without idle time.

Database Connection Pooling:

- When running many tasks, database connections are reused more efficiently, reducing the per-task connection establishment cost.

Resource Caching:

- System resources like file handles, memory caches, and database query plans are better utilized and cached when processing similar tasks in bulk.