解決分散式資料編排管理工具 Apache Airflow 中 Webserver 在讀取大型日誌時發生 OOM 的問題

Resolve the OOM issue in the Webserver when reading large logs in the distributed data orchestration tool, Apache Airflow

指導教授：莊坤達
專題成員：劉哲佑
開發工具：Python, Apache Airflow, Docker, Memray
測試環境：Mac Air, M2 chip, 16GB RAM, in Docker

一、簡介：

Why (Current State)

Apache Airflow 有一個看 Tasks 執行結果的 UI 介面
這個 Feature 會需要根據 (timestamp, line number) 去 Sort + Aggregate 所有 Log Records
因為本身是分散式系統，所以有 5 個 Log Sources 需要 Sort

原本的實作是用 Python Bulit-in 的 sort 去實作
當遇到要讀比較大的 Log Records 時 Web Server 會 OOM(Out Of Memory)

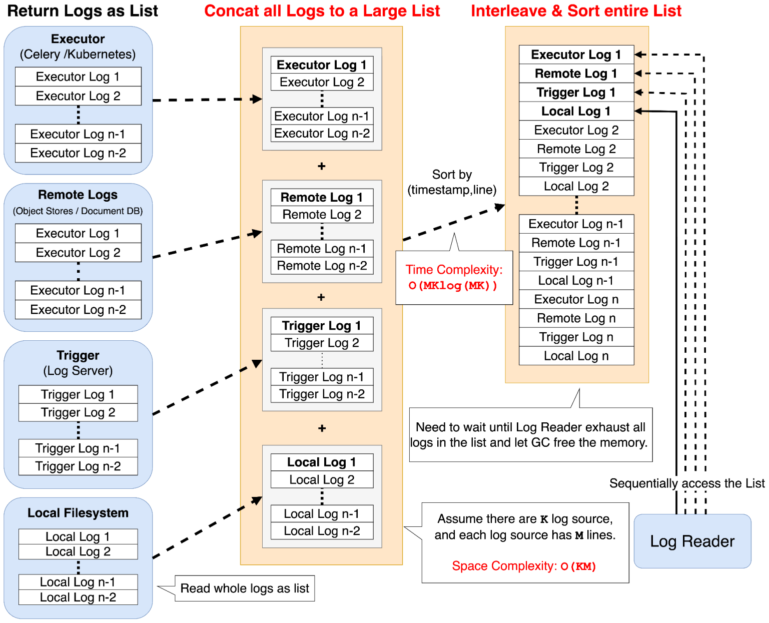
Challenges

- Memory bottlenecks
- Streamable interface for whole read path
- Compatible interfaces for 10+ providers
- Memory-efficient Serialization

How

- Read in chunks
- Streaming Interface
- K-Way Merge with Heap instead of Sorting whole logs
- Compatible interfaces for providers

Before Refactor



Return Logs as List

Concat all Logs to a Large List

Interleave & Sort entire List

Executor (Celery /Kubernetes): Executor Log 1, Executor Log 2, ⋮, Executor Log n-1, Executor Log n-2

Remote Logs (Object Stores / Document DB): Executor Log 1, Executor Log 2, ⋮, Executor Log n-1, Executor Log n-2

Trigger (Log Server): Trigger Log 1, Trigger Log 2, ⋮, Trigger Log n-1, Trigger Log n-2

Local Filesystem: Local Log 1, Local Log 2, ⋮, Local Log n-1, Local Log n-2

Read whole logs as list

Large list: **Executor Log 1**, Executor Log 2, ⋮, Executor Log n-1, Executor Log n-2 + **Remote Log 1**, Remote Log 2, ⋮, Remote Log n-1, Remote Log n-2 + **Trigger Log 1**, Trigger Log 2, ⋮, Trigger Log n-1, Trigger Log n-2 + **Local Log 1**, Local Log 2, ⋮, Local Log n-1, Local Log n-2

Assume there are **K** log source, and each log source has **M** lines.

Space Complexity: `O(KM)`

Sort by (timestamp,line)

Time Complexity: `O(MKlog(MK))`

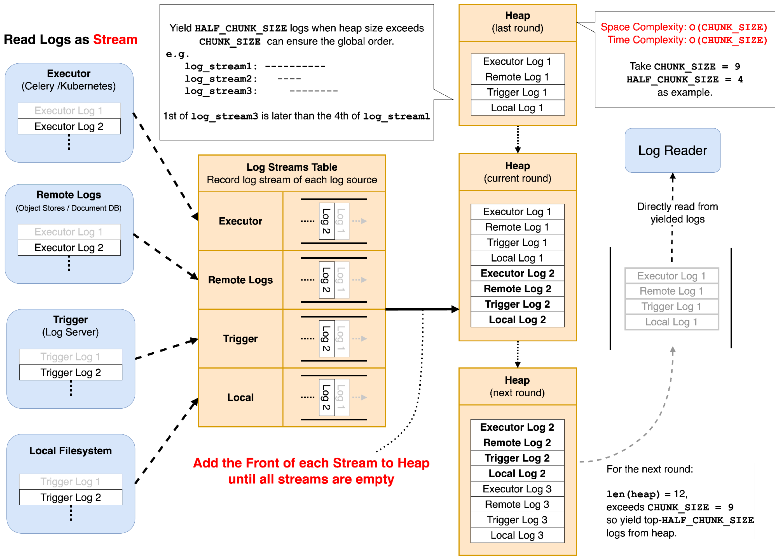
Sorted list: **Executor Log 1**, **Remote Log 1**, **Trigger Log 1**, **Local Log 1**, Executor Log 2, Remote Log 2, Trigger Log 2, Local Log 2, ⋮, Executor Log n-1, Remote Log n-1, Trigger Log n-1, Local Log n-1, Executor Log n, Remote Log n, Trigger Log n, Local Log n

Need to wait until Log Reader exhaust all logs in the list and let GC free the memory.

Sequentially access the List

Log Reader

After Refactor



Read Logs as Stream

Executor (Celery /Kubernetes): Executor Log 1, Executor Log 2, ⋮

Remote Logs (Object Stores / Document DB): Executor Log 1, Executor Log 2, ⋮

Trigger (Log Server): Trigger Log 1, Trigger Log 2, ⋮

Local Filesystem: Trigger Log 1, Trigger Log 2, ⋮

Yield `HALF_CHUNK_SIZE` logs when heap size exceeds `CHUNK_SIZE` can ensure the global order.

```
e.g.
   log_stream1: ----------
   log_stream2:    ----
   log_stream3:      --------
```

1st of `log_stream3` is later than the 4th of `log_stream1`

Log Streams Table
Record log stream of each log source

Executor	····· Log 2 Log 1 ···▸
Remote Logs	····· Log 2 Log 1 ···▸
Trigger	····· Log 2 Log 1 ···▸
Local	····· Log 2 Log 1 ···▸

Add the Front of each Stream to Heap until all streams are empty

Heap (last round): Executor Log 1, Remote Log 1, Trigger Log 1, Local Log 1

Space Complexity: `O(CHUNK_SIZE)`
Time Complexity: `O(CHUNK_SIZE)`

Take `CHUNK_SIZE = 9`
`HALF_CHUNK_SIZE = 4`
as example.

Heap (current round): Executor Log 1, Remote Log 1, Trigger Log 1, Local Log 1, **Executor Log 2**, **Remote Log 2**, **Trigger Log 2**, **Local Log 2**

Heap (next round): **Executor Log 2**, **Remote Log 2**, **Trigger Log 2**, **Local Log 2**, Executor Log 3, Remote Log 3, Trigger Log 3, Local Log 3

Log Reader

Directly read from yielded logs

Executor Log 1, Remote Log 1, Trigger Log 1, Local Log 1

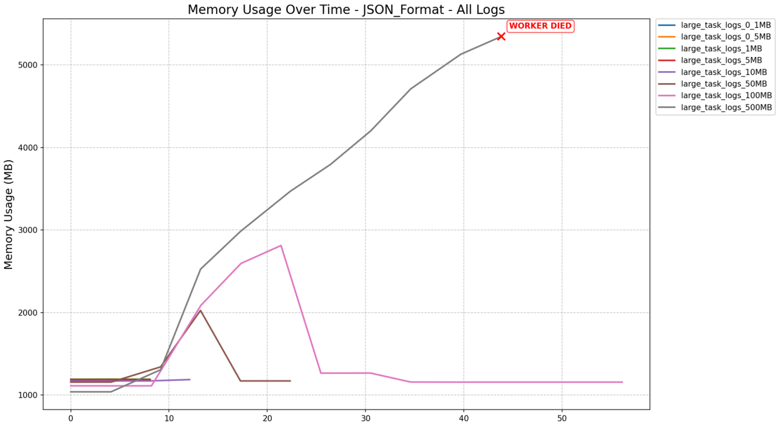
For the next round:

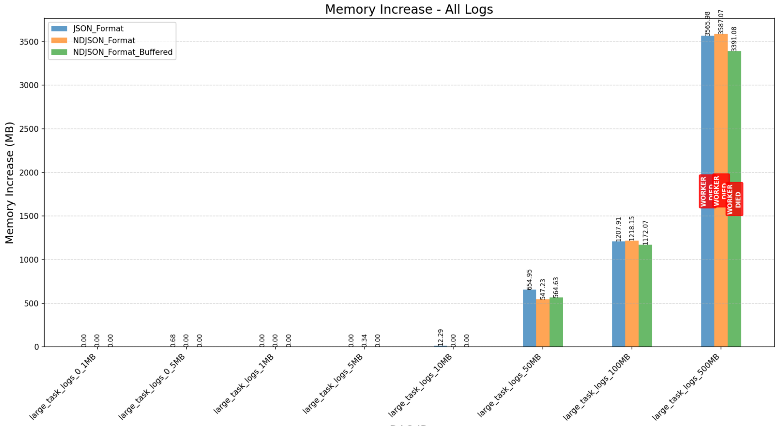
`len(heap)` = 12,
exceeds `CHUNK_SIZE = 9`
so yield top-`HALF_CHUNK_SIZE`
logs from heap.

二、測試結果:

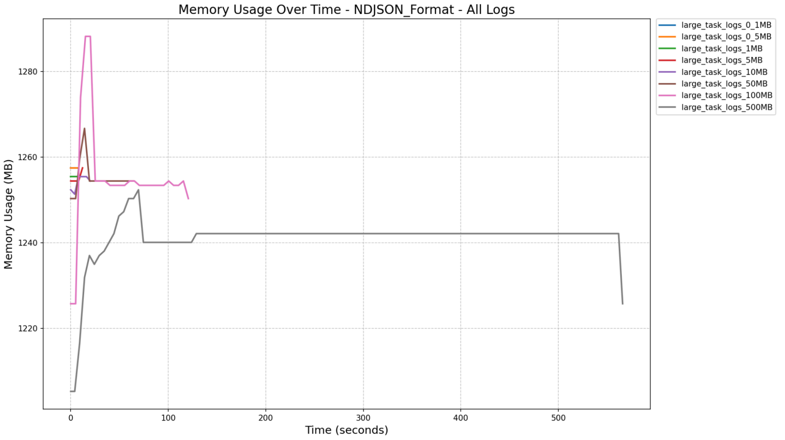
- **90% reduction** in memory usage **with similar processing times**
- Memory usage dropped by nearly **100x** (`3587MB → 33MB` for `500MB` logs).

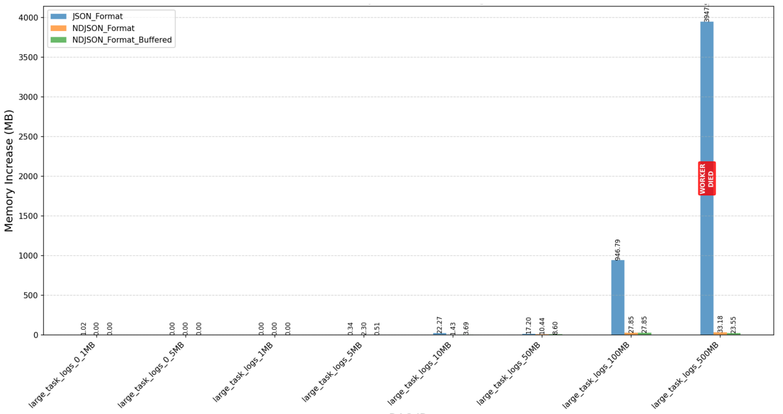
Before Refactor





After Refactor


Memory Usage Over Time - NDJSON_Format - All Logs
Memory Usage (MB)
Time (seconds)
large_task_logs_0_1MB
large_task_logs_0_5MB
large_task_logs_1MB
large_task_logs_5MB
large_task_logs_10MB
large_task_logs_50MB
large_task_logs_100MB
large_task_logs_500MB


Memory Increase (MB)
JSON_Format
NDJSON_Format
NDJSON_Format_Buffered
WORKER DIED
large_task_logs_0_1MB
large_task_logs_0_5MB
large_task_logs_1MB
large_task_logs_5MB
large_task_logs_10MB
large_task_logs_50MB
large_task_logs_100MB
large_task_logs_500MB