Western Norway University of Applied Sciences

ALICE

Implementation and Performance Analysis of the ALICE grid middleware JAliEn's Job Optimizer

Haakon Andre Reme-Ness

On behalf of the ALICE Collaboration

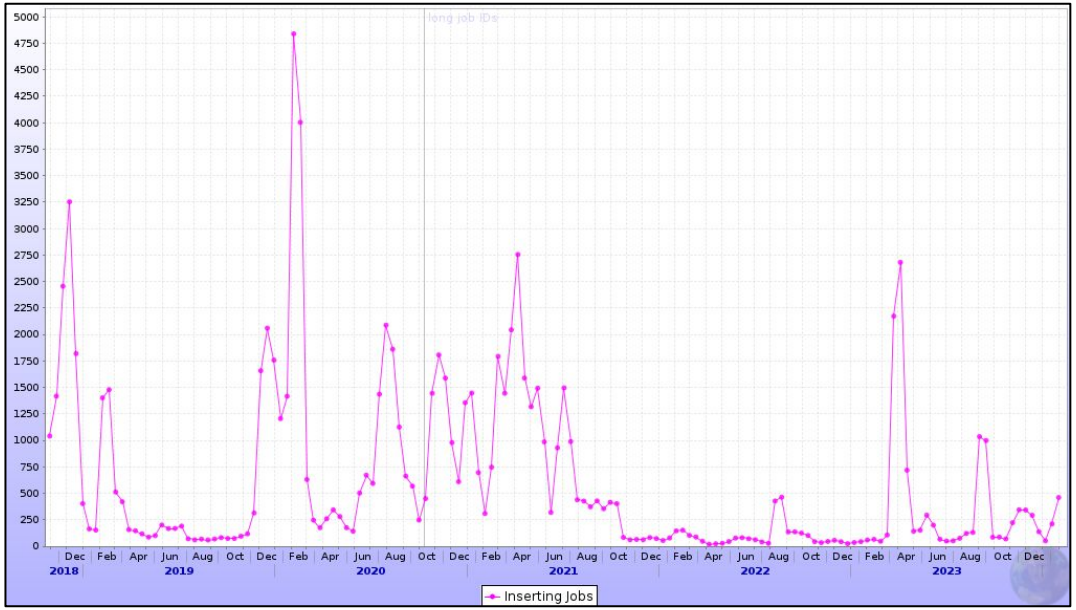
CHEP 2024, 21 - 25 October

Introduction

- Job Optimizer is a service responsible for splitting a larger job into smaller subjobs for the ALICE grid middleware JAliEn
- Job submission frequency varies
  - Often a large number of jobs are submitted at once
  - Which induces a high load on the Job Optimizer and creates a queue of jobs to be split
  - This in turn delays the processing of the workload on the Grid
- Improving the service efficiency:
  - Make the service horizontally scalable
    - Reduce the time between inserting a new workload in the queue and its subjobs starting on the grid nodes
  - Improve the interactions with the DB
    - Reduce table locking
    - Shorten connection time
  - If possible, improve job splitting to better make use of grid resources

Example of Job Optimizer overload

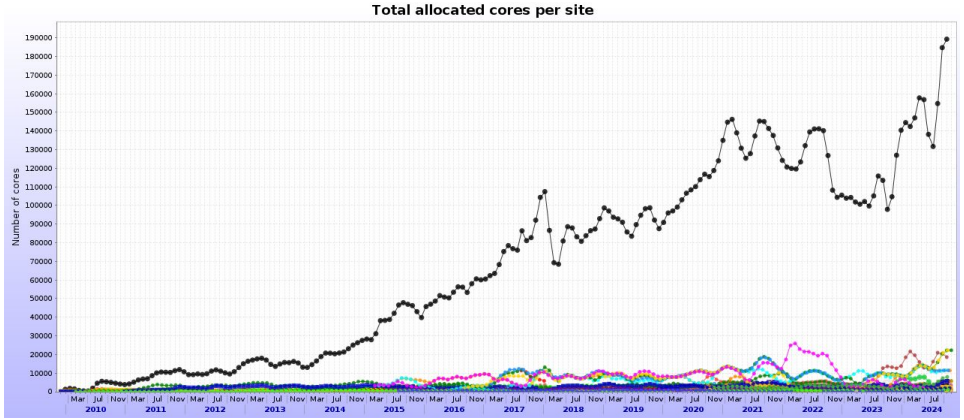
- Single Job Optimizer:
  - Unable to handle bursty job submission
  - Not scalable for higher traffic
- Large backlog of workloads queued for processing



“Inserting” is the state indicating it is ready to be picked up by Job Optimizer

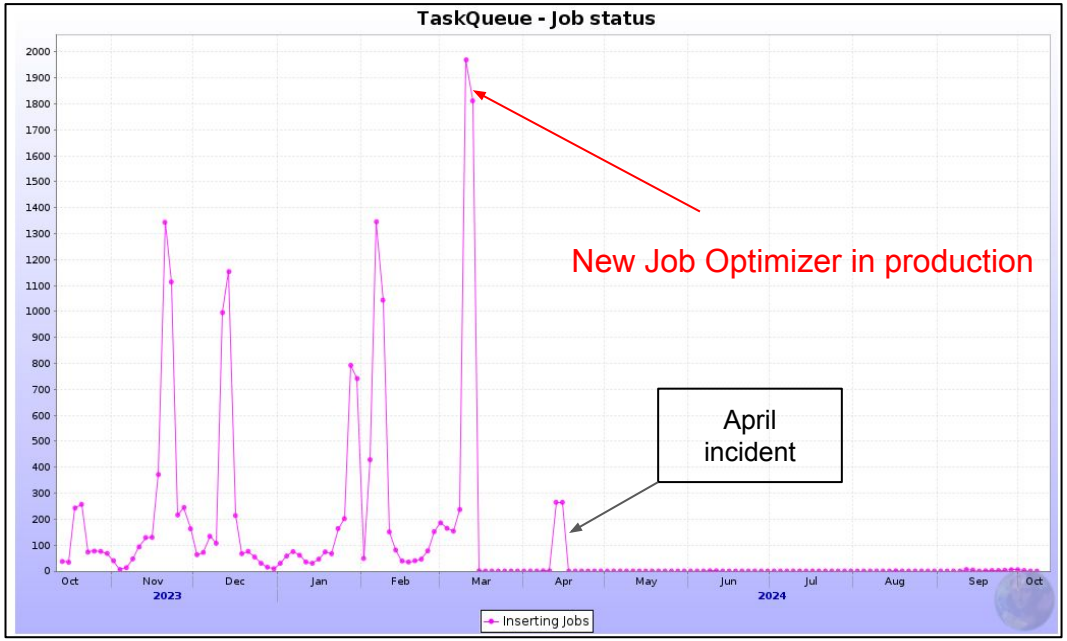
Horizontal scaling of the Job Optimizer

- New Job Optimizer capabilities
  - Multiple instances and threads
  - Running on several hosts, as many as needed to process rapidly the submitted jobs
  - Turning off multiple instances for servicing and upgrades is not disruptive for the overall operations
- Adding more instances assures future proofing with increase in Grid resources and job numbers
  - Higher focus on consistency as a result of increased parallelization



Introducing the new Job Optimizer in service

- Start of operation mid-March
  - Immediate 'disappearance' of waiting to be split jobs
- After an incident in April, a very large workload to be split, the service was further optimized
  - No further incidents registered
  - Stable operation

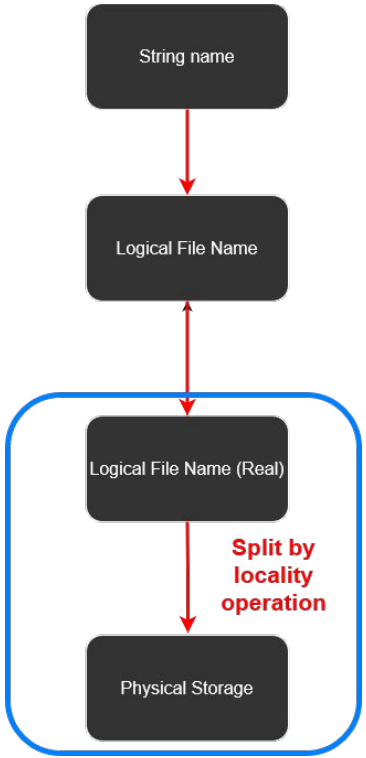


Further optimization of the Job Splitting algorithms

- In addition to the general improvement of the Job Optimizer, specific tuning of the job splitting algorithms was done
  - Part of the improvements are common for all jobs
  - Most complex case is splitting by data locality
    - Job attributes are added to match the jobs only with sites holding the dataset
  - This method involves multiple interactions with the file catalogue
    - **Data lookup is expensive**
    - **Furthermore, some jobs end up with very few files to process**
      - Several jobs can be merged together to improve efficiency and reduce splitting

Job splitting by data locality

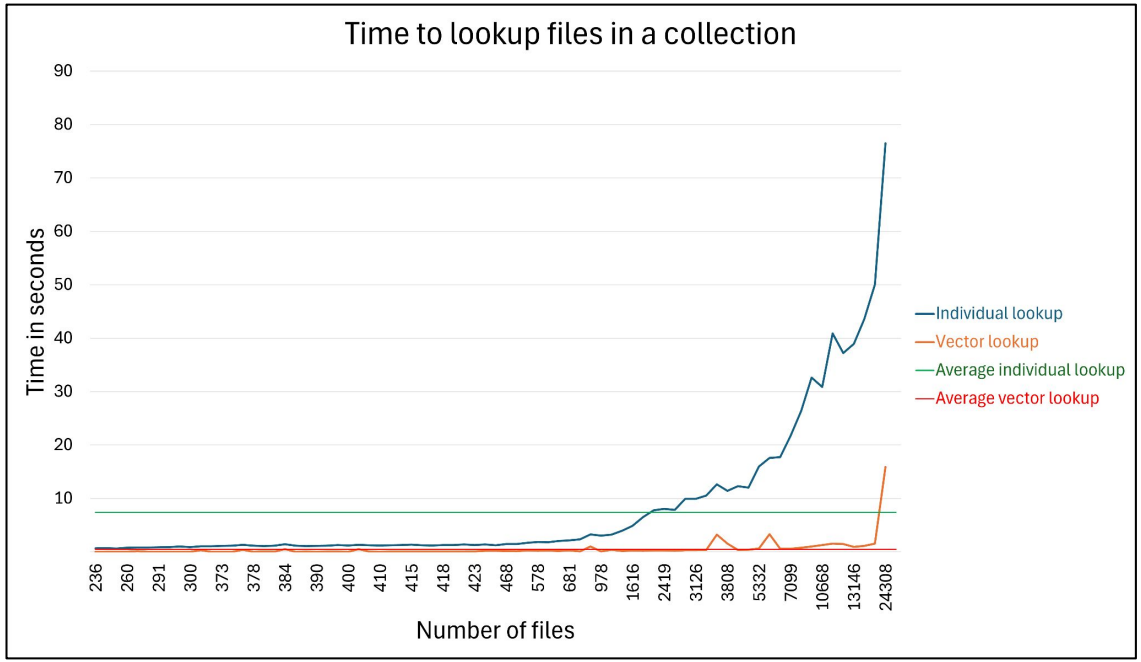
- Data lookup for physical location is a costly operation
  - Especially true for larger collections of above 1000 data files
  - All other splitting strategies require a single DB lookup
    - Splitting by data locality requires up to **3** DB lookups
- Introducing a partitioned DB, which holds the locality information
  - This is advantageous for faster lookups and faster indexing
- Take advantage of partitioned DB to further improve data collection lookup
  1. Divide the data collection by partition
  2. Do a bulk lookup query for each partition table



Lookup required to find physical storage of a data file

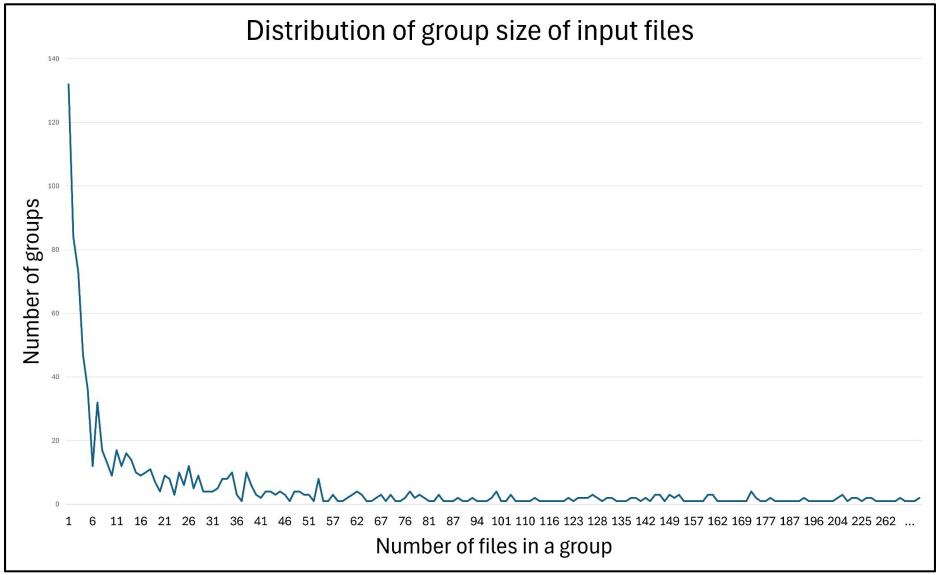
Lookup time as function of collection size

- With the old system, the time for data lookup increases exponentially with the size of data collection
  - Bulk data lookup time is significantly reduced and is essentially flat, irrespective of the data set size
- Time in seconds for the average sized data collection
  - Single lookup: **7,35s**
  - Vector lookup: **0,49s**
  - **Improvement of x15 for this case, significantly larger for large datasets.**



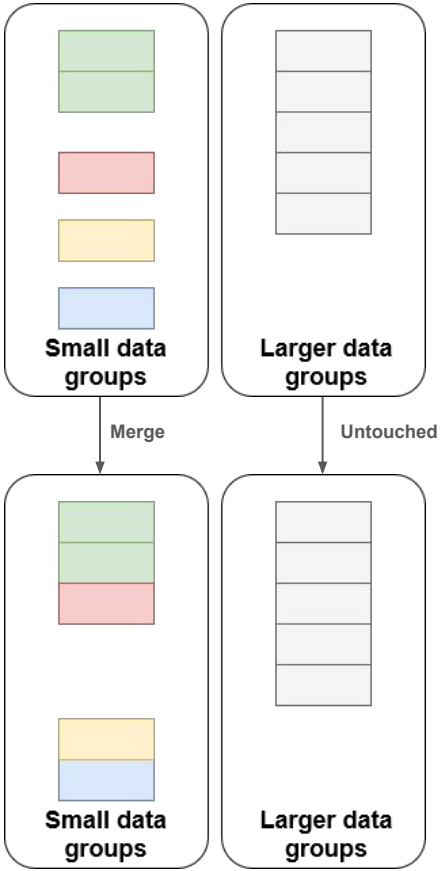
Job splitting based on data locality

- Data is grouped based physical locations
  - ALICE files typically have two replicas
  - Data required by a split job must be located on a given site for the job to be matched
- However an issue becomes apparent…
  - The number of files in a data group is a function of the size of the site storage element
  - Data group sizes can differ by **hundreds**, even **thousands** of files



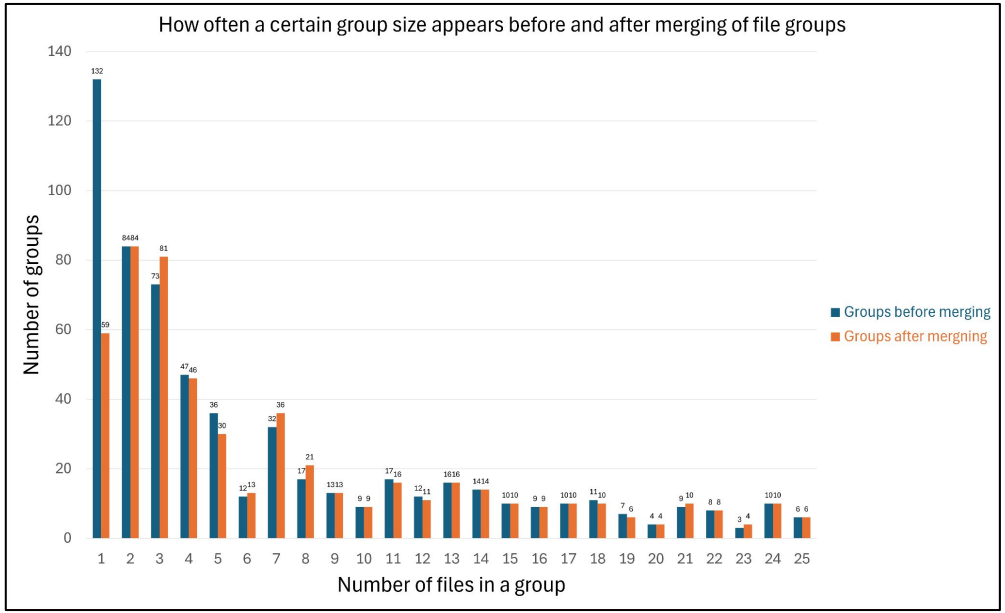
Merging of data locality groups

- Idea is to merge smaller data groups to create more balanced subjobs
  - Find all small data groups
  - Merge data groups based on one shared data locality and size
  - If no shared data locality base merge on distance between data localities
    - Another service keeps the distances up to date
- **Caveat:** Files are not always available locally on site after merging



Results of merging subjobs

- More balanced data groups
- Percentage of data groups needed to merge on average: **11.87%**
- Remote download of files does not increase significantly
  - Therefore no added burden on network



Site activity				
Site	Job eff.	All files	Local files	Remote files
TOTAL 9876 jobs	65.56%	160946 files 18.1 MB/s 315.9 TB	155941 (96.89%) 20.87 MB/s 307 TB	5005 (3.11%) 3.262 MB/s 8.941 TB

New Job Optimizer advantages

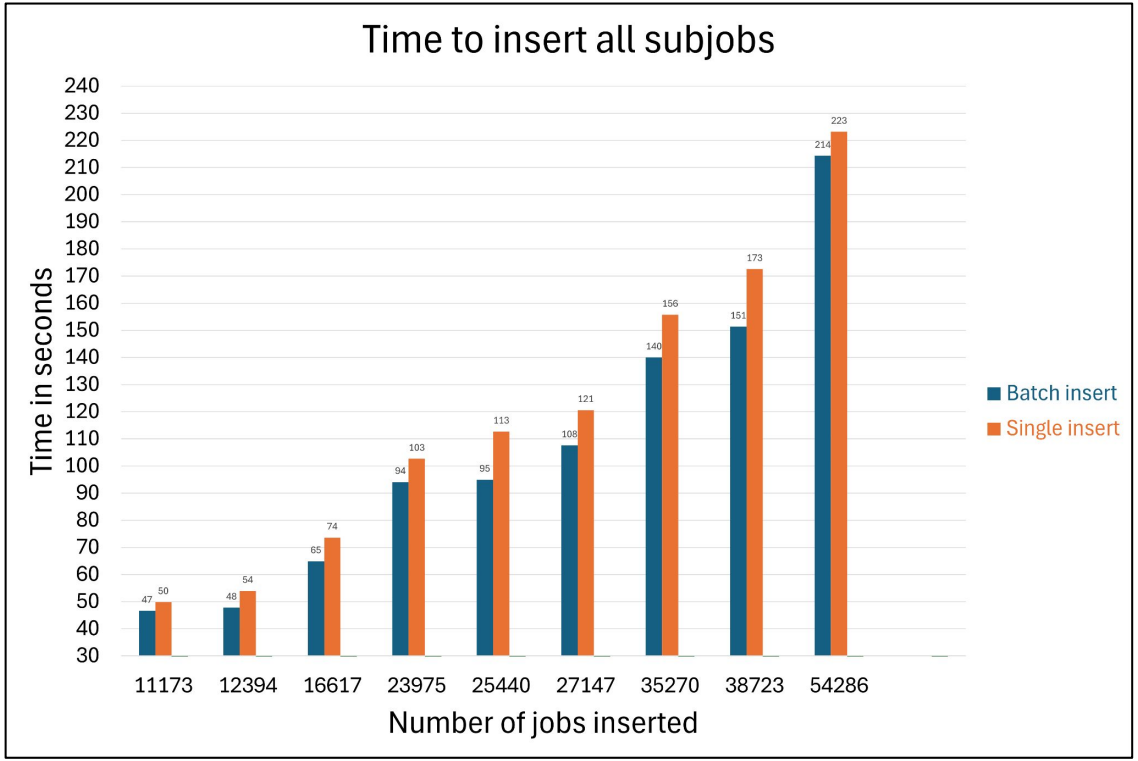
- With all functionality improvements introduced on the past slides, the operation of job insertion is done in a single DB transaction
  - Inserting does not require a lock blocking rows or table for other connections
  - However, to keep DB consistency a single write lock on a row is required
- Preprocess queries towards database before transaction to shorten connection usage
- Query failure related to locking of tables have to be handled separately with a retry

Batch Insert

- Large DB transactions for thousand of subjobs are slow if done in series
- To avoid this issue, the insertion is done in batches
  - Results in a significant performance boost
  - **Drawback -** batch is emptied both in case of success or failure
    - Failure related to table locking will trigger the creation of the batches again
    - Rare enough occurrence to negate the performance gain
  - Improvements only seen if number of subjobs are in the **thousands**

Result of batch insert

- Tangible difference in performance on large jobs
  - Improvement by about 5% for large job inserts
- Several seconds saved on shared connection usage
- Large portion of time is preprocessing job parameters



Summary and outlook

- Horizontally scalable Job Optimizer implementation solved the workload processing backlog issue in the ALICE job management system
- Large performance improvements by change of DB query methods and structure
  - Vector operations for file replica lookups
  - Less connection usage and as little locking as possible
    - This improves uptime for database connection for other services of JAliEn
- Significant further improvement can be achieved in the warehousing of split jobs
  - Every split job JDL is currently saved in its entirety, although most of the information is repeating and redundant
  - We are working on storing only a delta of the JDL containing the unique fields with respect to the original job JDL
    - **Advantage -** Will save more than 80% of the DB space currently used
    - **Disadvantage -** Requires more SQL queries
    - We are looking for the best compromise of the two