Comet Report

Distributed processing orchestration

31 May 2022

[Tapez ici]

Table of Contents

1 Overview

1.1 Introduction

The use of patterns in the orchestration of data processing has grown significantly in recent years. Both in terms of orchestration and distribution of processing. These last years, the historical orchestration pattern was challenged by the concept of Event Sourcing, bringing with it the so-called “Choreography” pattern. This new approach was supposed to correct the failings of the orchestration with stronger steps/activities decoupling. But as often in architecture, it came with its difficulties and drawbacks; orchestration pattern solutions evolved to correct their historical flaws. At the same time, tools dedicated to resources management, such as can be found in HPC ecosystems, offer more and more native possibilities for orchestrating processing. The objective of this seminar is to try to take a step back from trend effects and to extract a state of the art through various feedback.

Why do we need to orchestrate data processing?

Build and integrate a data production system

- Chains together complex processing to produce a result from one or several inputs.
- Automate data processing
- Give operators the ability to monitor data processing
- Make data processing more robust (fail and retry, pause, restart, etc.)

Optimization of resources management

- Reduce time of processing
- Reduce & optimize resources usage

As good system design practice

- Uncouple steps
- Test more easily
- Being able to reuse steps
- Facilitate distribution of processing
- Being able to use "Cots"(commercial off-the-shelf)

1.2 Speakers

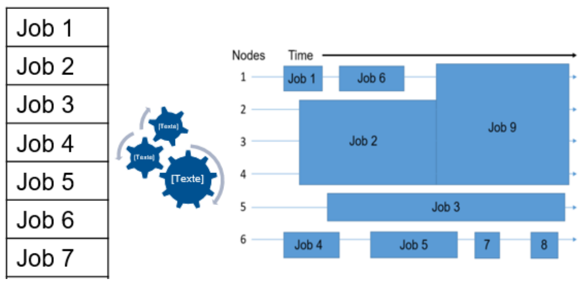
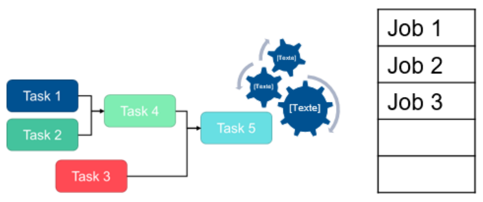
Christophe Arguel	Capgemini
Vincent Gresselin	AIRBUS
Sébastien Gonzalve	EASY MILE
Frédéric Tromeur	TELESPAZIO a LEONARDO and THALES company
Alain Montmory	THALES
Christophe Dabin Guillaume Eynard Hugues Larat	cnes CENTRE NATIONAL D'ÉTUDES SPATIALES

2 Resources management & Design Pattern

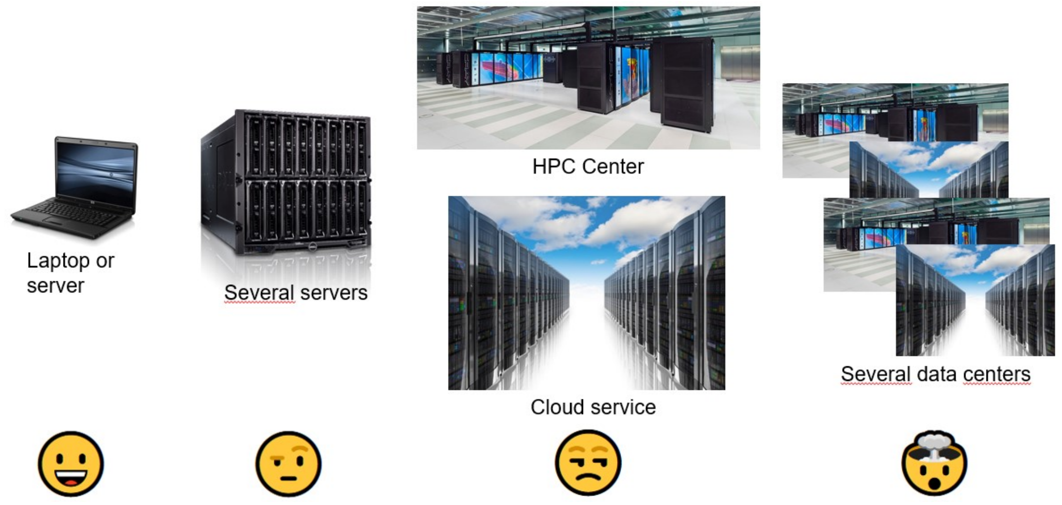
2.1 Resources management

2.1.1 Concepts

Two main concepts seem to emerge in the field of resource management. Either this resource management is specified programmatically (orchestration) or its automated management is delegated to a scheduler.

Scheduling	Orchestration
• Tasks list with attributes (cpu, memory, tasks duration) • Scheduling based on these characteristics: which task to execute • First Basic example: FIFO (or no scheduling)	• Task list defined with dependencies • Centralized description of a task sequence: workflow or pipeline • Automatic triggering • Relies on scheduling and resource management for execution
 Job 1 Job 2 Job 3 Job 4 Job 5 Job 6 Job 7 [Texte] [Texte] [Texte] Nodes Time 1 2 3 4 5 6 Job 1 Job 6 Job 2 Job 9 Job 3 Job 4 Job 5 7 8	 Task 1 Task 2 Task 3 Task 4 Task 5 [Texte] [Texte] [Texte] Job 1 Job 2 Job 3

The choice of solutions is mainly defined by the requirements of the software system, and the nature and constraints of the data processing application. And, by the requirements in terms of material distribution.



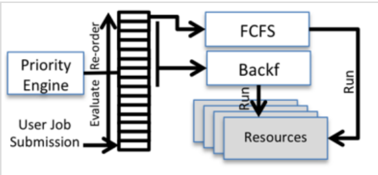
2.1.2 Technical usages & examples

Some technological illustrations for the management of resources were mentioned during the COMET. We selected the most common usage of scheduling technologies.

2.1.2.1 HPC Scheduling

Main goals

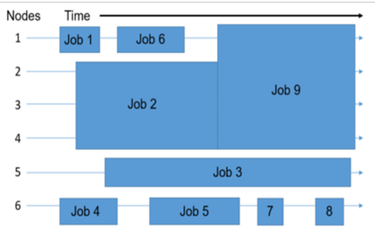
- Manage individual server resources
- Apply Wall-time
- Manage partitions
- Manage Project / Users access
- Configure Constraints / Options
- Ordering of jobs through the management of a global queue
- Offer partitions or QoS (groups of nodes, priorities, particular limits)
- Optimize Scheduling = fill the maximum of computing resources by satisfying user needs
    - Job priority management: according to partitions or QoS, fair-share, preemption, job age (starvation), job size .. etc ...
    - Back-filling = filling the holes once the priority has been evaluated (need the wall-time)
    - Preemption of jobs, stopping lower priority tasks



PBS Works™

slurm workload manager

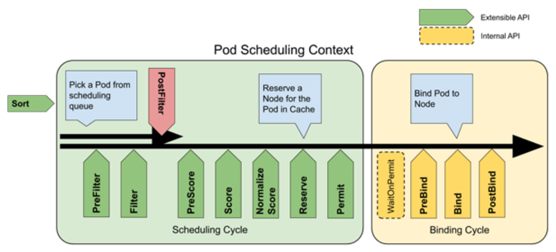
IBM Spectrum LSF

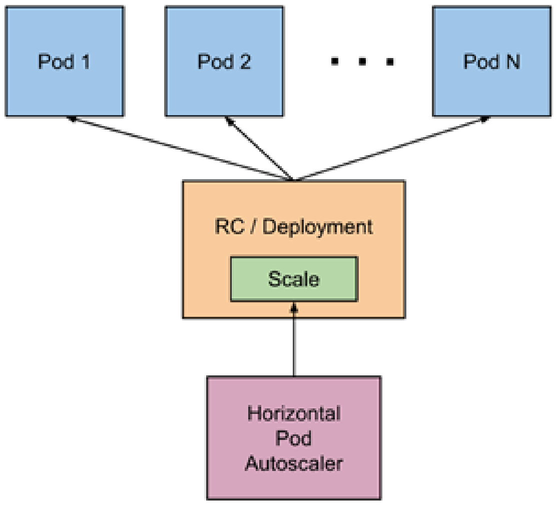


2.1.2.2 Scheduling containers with Kubernetes

Main goals

- Resources requirements & Management
- HW / SW / Policy / constraints
- Data Locality & POD State Full
- Find the best node to run a POD
    - Resilience: ensuring that a pod is in the desired state somewhere.
    - Manage pod distribution (Filtering, Rating, Affinity)
- POD duration management
- Scheduling Policies and Profiles
- Manage auto-scaling (Horizontal Pod Auto-scaling)





2.1.2.3 Orchestration

In this area, it seems to us that we have identified two subsets of applications: One orchestration usage is oriented towards the final user application development, giving high-level tooling to implement and automate the data processing pipeline, while the other is more specialized in optimizing the distribution of processing. In the end, we also noted that high level orchestration software solutions often come with interfaces to rely on the second subset of tooling for distributed processing.

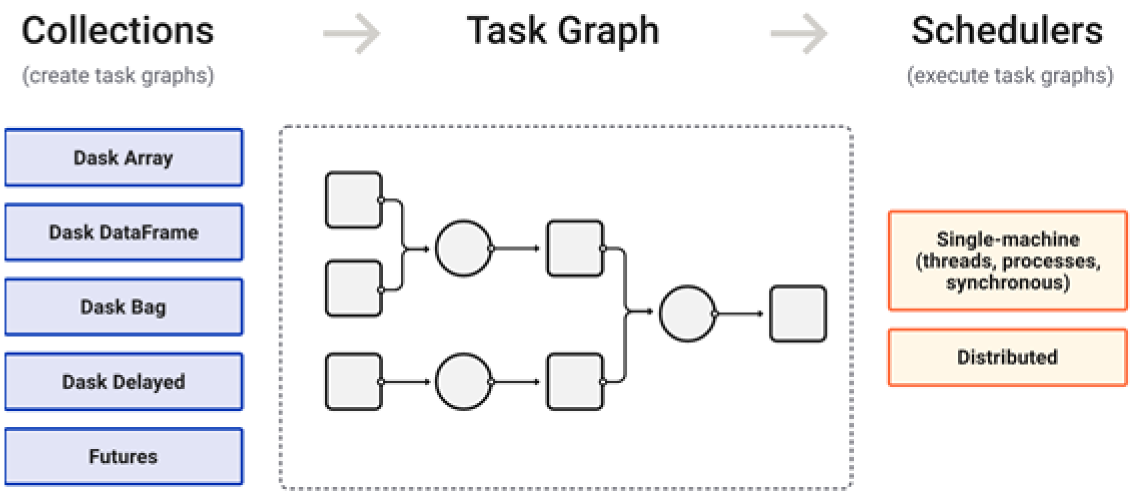
2.1.2.3.1 Distributed processing with Dask

Main goals

Be able to distribute a single data processing algorithm by dividing input data on several chunks. Doing that by defining a list of tasks to perform chained as a DAG.

- Define a list of all tasks with there dependencies
- Everything is a graph (DAGs), composed of fine tasks
- The scheduler is the master of the global graph, assigning tasks to workers

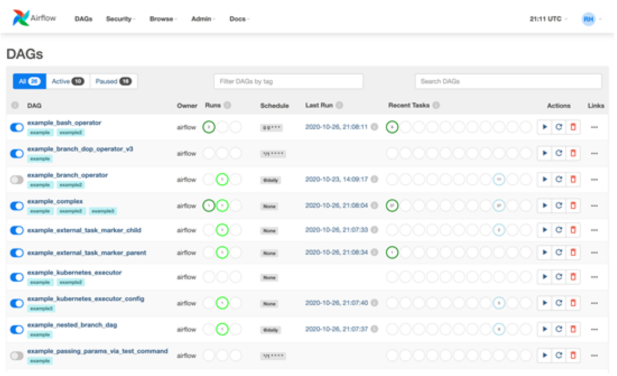
- Complex optimization of task scheduling to limit the overall memory footprint for example (depth first)
- Some orchestration features
  - Task stealing
  - Retry on error
  - Internal resources assignment (ex GPU)
- Point of attention : there is no complex mechanisms for pausing, replaying, triggering
- Interfaces for resources management: Kubernetes, YaRN, HPC, local (the Dask cluster can also run standalone)



2.1.2.3.2 Software orchestration with Airflow

Main goals

- Workflows/graphs/pipelines editor
- Complete GUI to follow workflows (sequence of tasks that can be re-executed).
- Task triggering (date, on events)
- Ability to stop/restart a workflow. Retry/Error
- Workflow described by a text file (Xml syntax, Yaml, Code, BPMN etc.), shareable
- Execution of tasks over short time (hours) or long time (days)
- Operation /production / automation oriented



DAGs

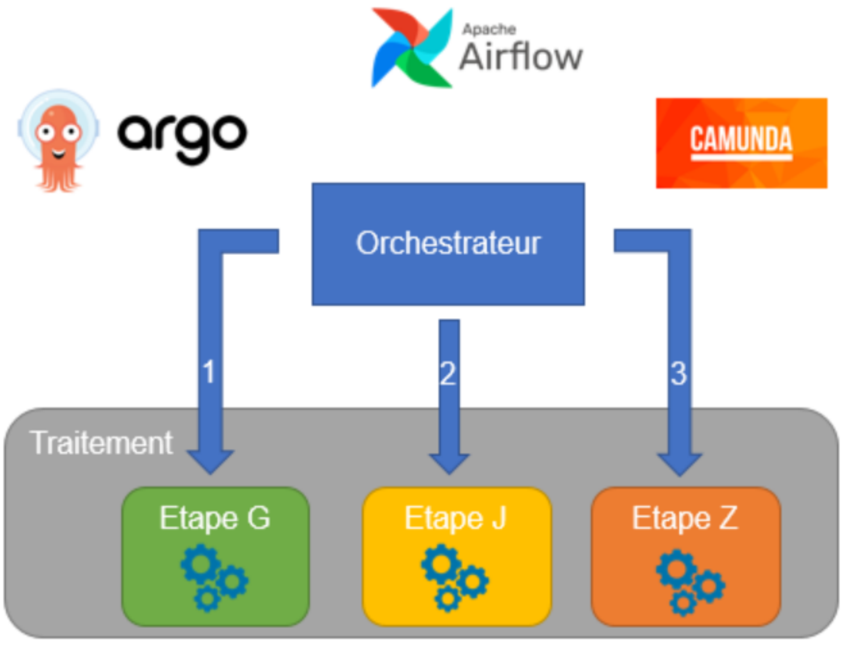
DAG	Owner	Runs	Schedule	Last Run	Recent Tasks	Actions	Links
example_bash_operator (example, example2)	airflow		0 0 * * *	2020-10-26, 21:08:11			
example_branch_dop_operator_v3 (example)	airflow		*/1 * * * *				
example_branch_operator (example, example2)	airflow		@daily	2020-10-23, 14:09:17			
example_complex (example, example2, example3)	airflow		None	2020-10-26, 21:08:04			
example_external_task_marker_child	airflow		None	2020-10-26, 21:07:33			
example_external_task_marker_parent	airflow		None	2020-10-26, 21:08:34			
example_kubernetes_executor (example, example2)	airflow		None				
example_kubernetes_executor_config (example3)	airflow		None	2020-10-26, 21:07:40			
example_nested_branch_dag (example)	airflow		@daily	2020-10-26, 21:07:37			
example_passing_params_via_test_command (example)	airflow		*/1 * * * *				

2.2 System Design – Orchestration vs Choreography

Until that point, we mainly discussed tooling for orchestration of data processing resource. In the field of design system, in terms of orchestration pattern we often find two patterns bringing their advantages and disadvantages named Orchestration and Choreography. As we concluded during the COMET, some concepts obviously overlap with the resource management ecosystem.

2.2.1 Orchestration pattern

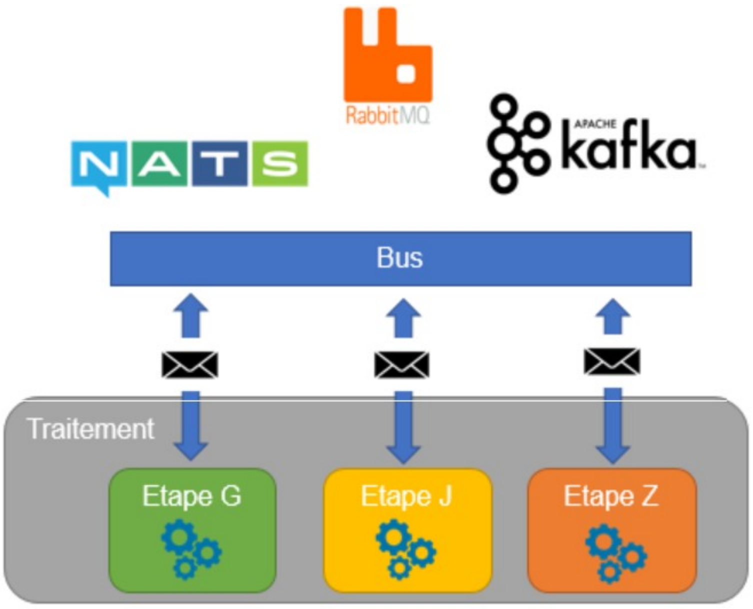
This pattern is the oldest and most known for system design. Every steps and global workflows are managed by a middleware (orchestrator)



Pros	Cons
• Centralization of the process • Easy observability • Error handling and "retry" • Transactional approach	• Need of dedicated middleware • The orchestrator can be a Single Point Of Failure • Design more often coupled, because steps can end up to have dependencies with the orchestrator

2.2.2 Choreography pattern

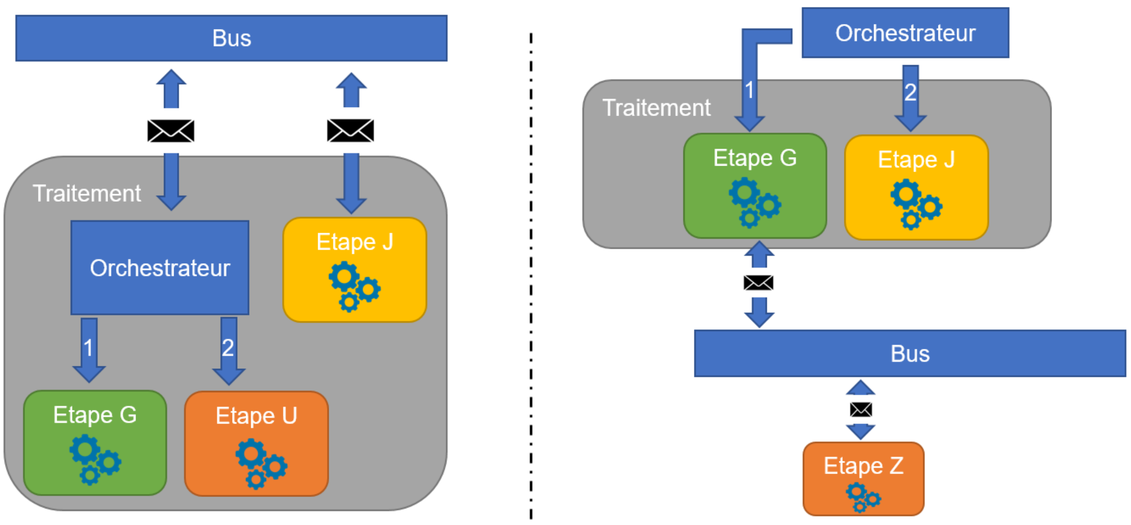
As explained previously, Choreography pattern comes from "Event Sourcing" concept and aims to solve Orchestration pattern issues. It's often use in micro services architecture with a middleware as messages bus.



Pros	Cons
• Loose-coupling • No single point of failure	• Observability is more complex • Retry or transaction management is complicated to achieve • A common monitoring and UI configuration for the workflow difficult to build/get

2.2.3 Hybrid design

Today, we also end up with a mix of these patterns. In addition, our architecture can be driven by our software design (the middleware or framework we use).



2.3 In Practice

During the comet, the participant was offered to complete a technological radar via Klaxoon.

2.3.1 Choreography

Choreography seems to be mostly used when we need a continuous stream processing. This pattern offers also the easiest way to add or remove steps with the minimum of disruption. However, many return of experience seems to indicate a lack of tools/solutions regarding the observability.

2.3.1.1 Presentations

EUMETSAT L2PF: https://www.comet-cnes.fr/sites/default/files/ressources/04%20-%20L2PF-MicroserviceChoregraphie-Comet-CNES-31-05-2022.pdf

Easy Miles: https://www.comet-cnes.fr/sites/default/files/ressources/07%20-%20CNES-COMET-EasyMile_pr%C3%A9sentation.pdf

STORM: https://www.comet-cnes.fr/sites/default/files/ressources/06%20-%20COMET-SIL-%2031%20MAI%20-%20STORM%20et%20Chor%C3%A9graphie%20-%20TPZF.pdf

2.3.1.2 Technologies watch

kafka
RabbitMQ
PULSAR
NATS
elasticsearch
redis

2.3.2 Orchestration

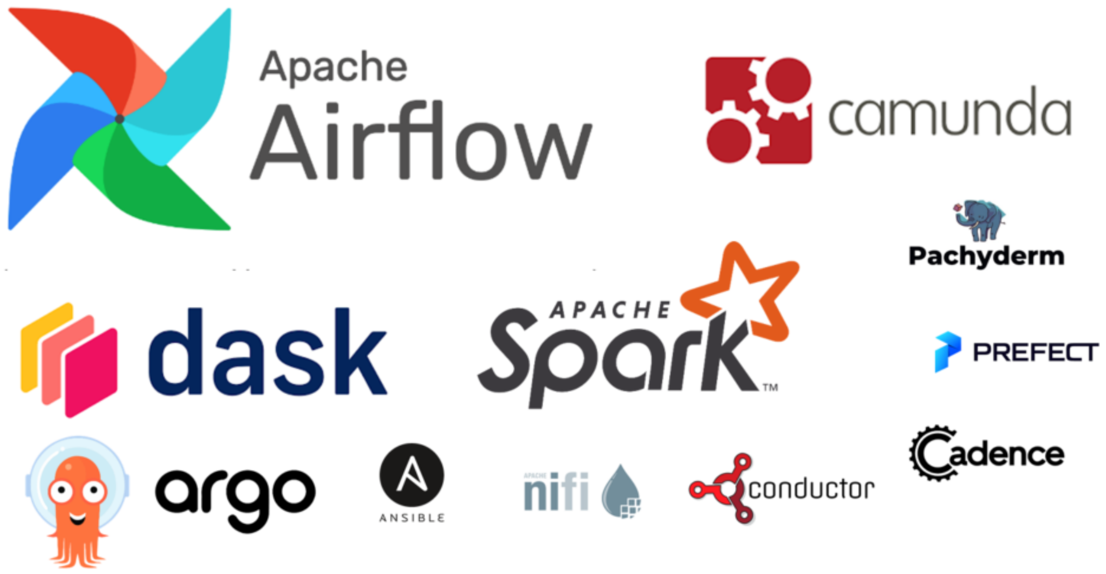
Orchestration software improved there reliability and resilience to reduce the risk of SPOF for the system design. So as we see during the COMET, it can be used in various use case of orchestration. The usage of orchestration middleware is often due to monitoring needs shared between user and a clear workflow management (editor, viewers ... etc...).

2.3.2.1 Presentations

EUCLID: https://www.comet-cnes.fr/sites/default/files/ressources/03%20-%20COMET_SIL_Euclid_PipelineRunner_v1_0.pdf

CO3D: https://www.comet-cnes.fr/sites/default/files/ressources/08%20-%20COMET-orchestration-processing-image-CO3D.pdf

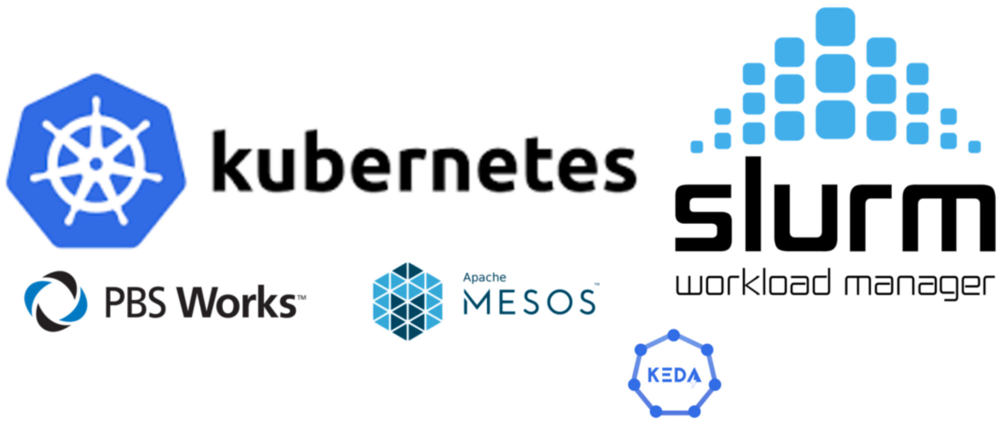
2.3.2.2 Technologies watch



2.3.3 Resources allocation management

Resources allocations tools seems to be used by the upper layer (orchestration software) to optimize their resources usages or alone by the users when you do not need specific tools to manage/share your workflow.

2.3.3.1 Technologies watch


kubernetes
slurm workload manager
PBS Works
Apache MESOS
KEDA