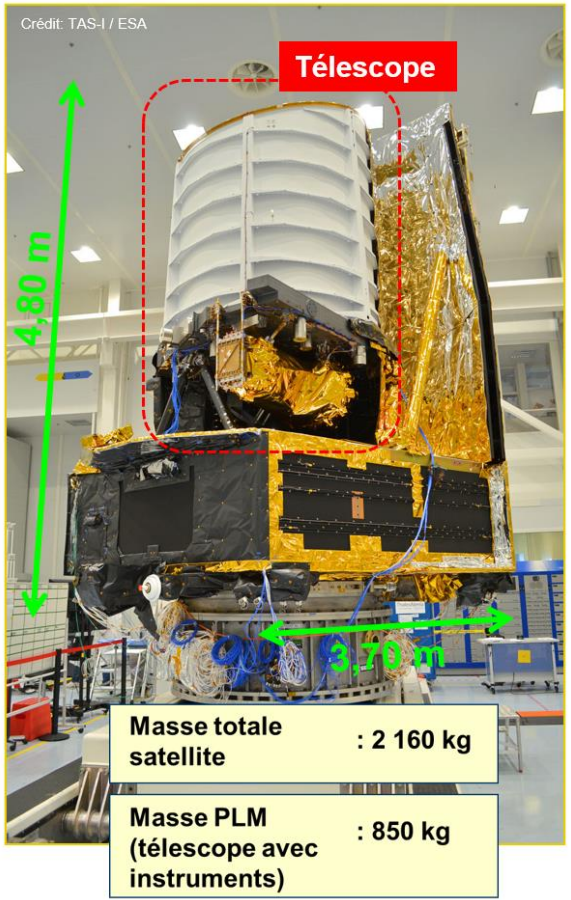
Orchestration Euclid

P. Casenove

Chef de projet Segment Sol Euclid

Mission Euclid

- Comprendre l'origine de l'accélération de l'expansion de l'Univers
  - Propriétés et nature de la Matière Noire et de l'Energie Sombre
  - A l'aide de 2 condes cosmologies:
    - Géométrie de l'Univers (Weak Lensing)
    - Historique de sa formation (Cluster de galaxies)
- Mission nominale de 6 ans
  - Relevé WIDE de 14500 deg²: environ 29k observations
  - 3 champs DEEP de 50 deg² + Calibrations : 16k observations
  - 20 observations par jour
  - Télescope:
    - Miroir M1 de 1.2m de diamètre
    - Imageur dans le visible de 600M pixels : VIS
    - Spectrophotomètre infrarouge de 70M pixels : NISP
  - Données de télescope Sol requises pour compléter les données
  - Trois Data Releases durant la mission:
    - Tir + 2 ans: 2500 deg², Tir + 4 ans: 7500 deg²
    - 14 mois après la fin de la mission: relevé complet, DR3
  - Date du tir en cours de consolidation suite à bascule Ariane 6.2



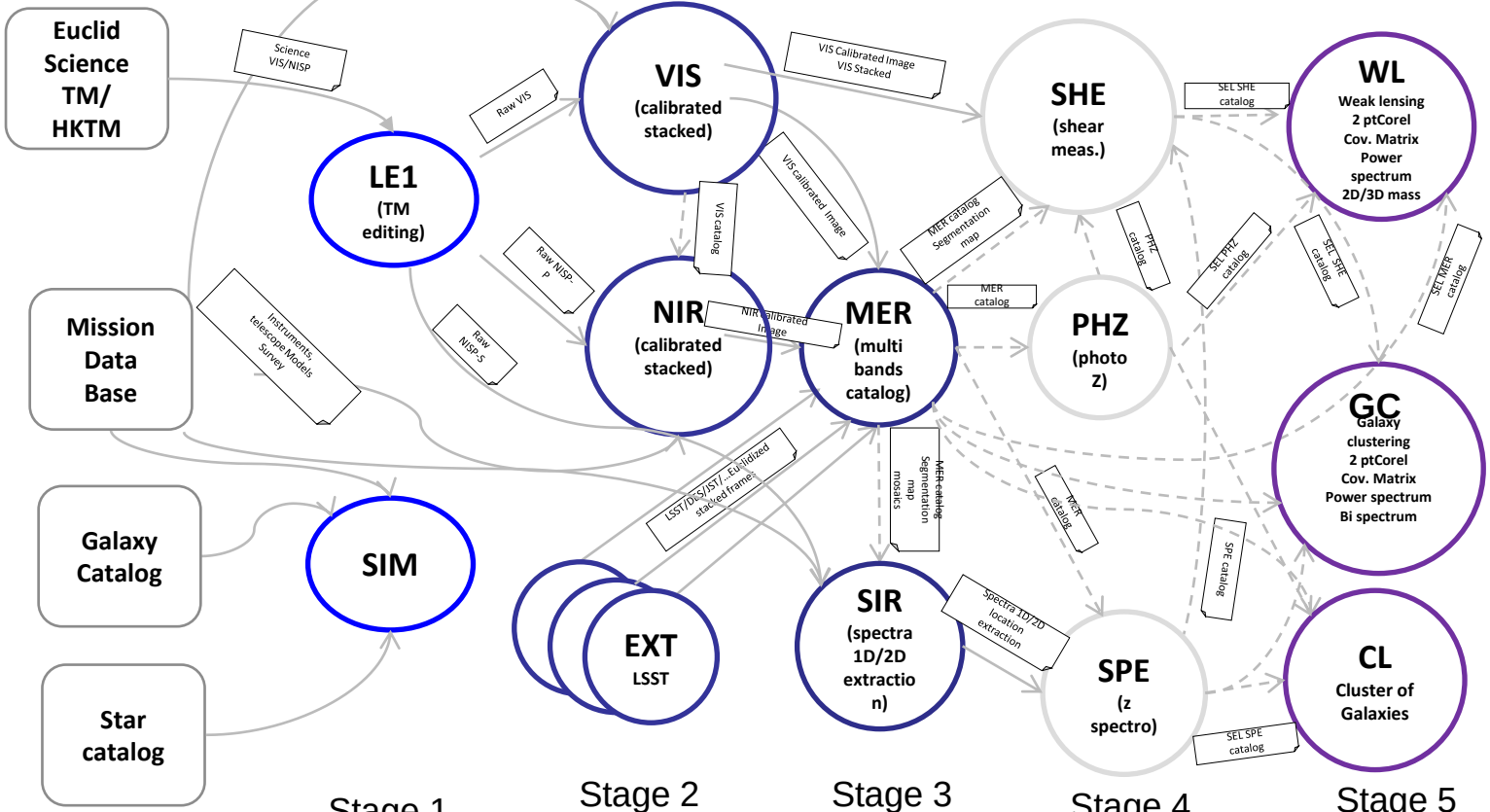
Architecture du segment sol

- Concepts clés du Scientific Ground Segment (SGS)
  - Composé de datacenters (SDC) répartis dans 9 pays différents
  - Les mêmes pipelines doivent tourner dans tous les SDCs
  - Le processing et les données sont distribuées: chaque SDC est aussi un nœud de stockage des données produites
  - Déplacer le code au plus près des données: minimiser les transferts de fichiers en entrée des pipelines

- Un système d'archive centralisé (Euclid Archive System)
  - Un repository central des metadata (DPS) : indexe, inventorie et localise toutes les données produites
  - Un Distributed Storage Service (DSS) dans chaque SDC qui enregistre les sorties et permet le transfert de fichiers inter-SDC
  - Un modèle de données centralisé (Euclid Common Data Model) définissant les interfaces entre les pipelines

- Infrastructure Abstraction layer (IAL): en charge de garantir l'exécution des pipelines sur tous les SDCs
  - Quelle que soit l'architecture SDC sous-jacente

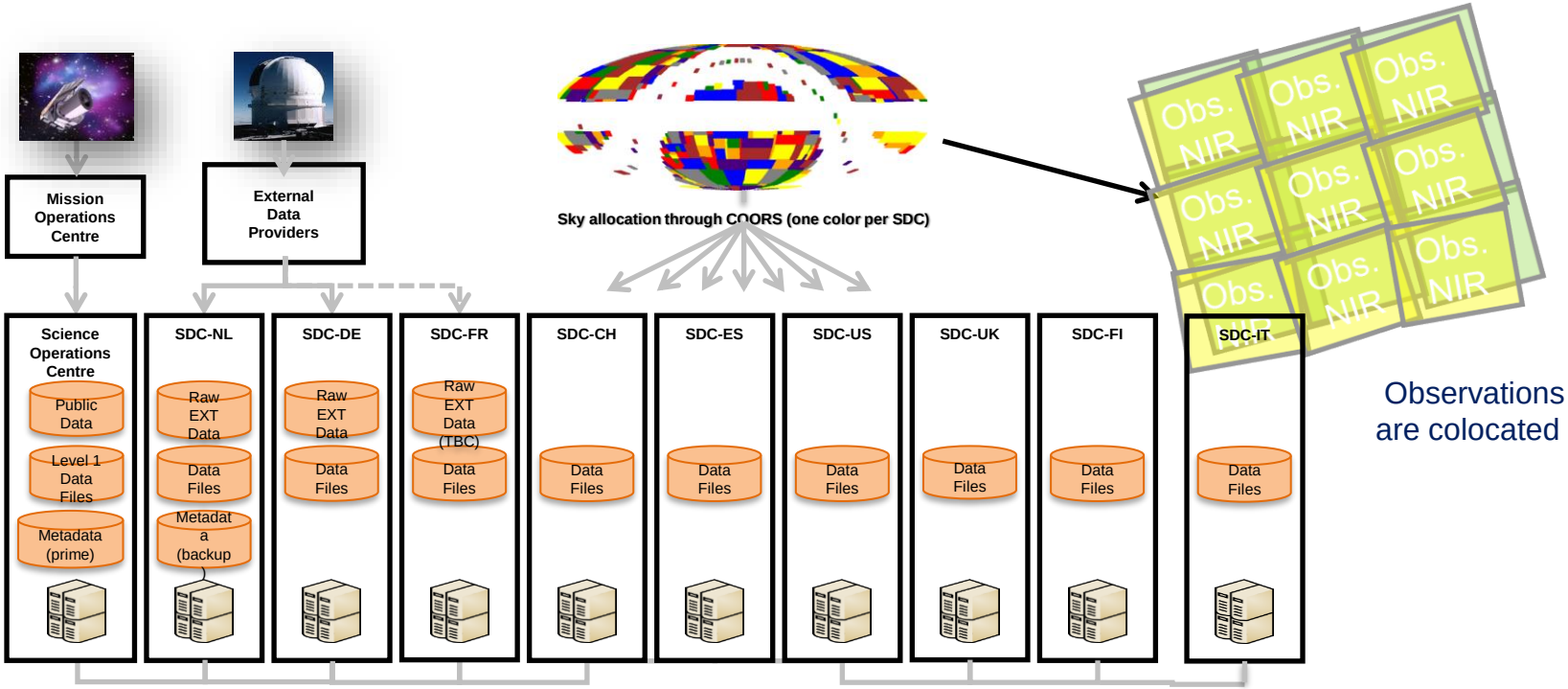
Pipeline


Euclid Science TM/ HKTM
Mission Data Base
Galaxy Catalog
Star catalog
Science VIS/NISP
Instruments, telescope,Models Survey
LE1 (TM editing)
SIM
Raw VIS
Raw NISP-P
Raw NISP-S
VIS (calibrated stacked)
NIR (calibrated stacked)
VIS catalog
NIR calibrated Image
VIS calibrated Image
MER (multi bands catalog)
LSST/DES/JST/..Euclidized stacked frames
EXT LSST
MER catalog Segmentation map mosaics
SIR (spectra 1D/2D extractio n)
Spectra 1D/2D location extraction
SPE (z spectro)
VIS Calibrated Image VIS Stacked
SHE (shear meas.)
MER catalog Segmentation map
MER catalog
PHZ (photo Z)
PHZ catalog
SEL SHE catalog
SEL PHZ catalog
SEL SPE catalog
SEL MER catalog
WL Weak lensing 2 ptCorel Cov. Matrix Power spectrum 2D/3D mass
GC Galaxy clustering 2 ptCorel Cov. Matrix Power spectrum Bi spectrum
CL Cluster of Galaxies
SPE catalog
SEL SPE catalog
Catalogs
Image
Stage 1
(daily cadence : LE1 or async: SIM)
Stage 2
(daily cadence : in sync with LE1 if calibration files available !)
Stage 3
(Requires adjacent Observations and EXT data)
Stage 4
(Upstream products available)
Stage 5
a.k.a LE3
(sky patch needed ~ 100 sq degrees)

Distribution des données

- Première orchestration au niveau du SGS complet
- Distribution faite à priori, basée sur le relevé de la mission
  - Chaque observation/tuile est assignée à un SDC primaire + Secondaire
  - Certains SDCs sont en charge des ingestions des données des téléscopes Sols
  - Informations de distributions stockées dans l'EAS
- Règles de distribution
  - Nombre d'observations/tuiles assignées est fonction de la taille (# cores, stockage) du SDC
  - Linéarisation de la distribution dans le temps : éviter de longues périodes sans réception de données
  - Minimiser la copie des données entre les SDC au niveau de la PF MER
- Et des contraintes opérationnelles…
  - Mise à jour / changement du survey du télescope
  - Maintenances planifiées des SDCs
  - Objectif de faire la distribution sur les 2/3 mois à venir

Distribution des données

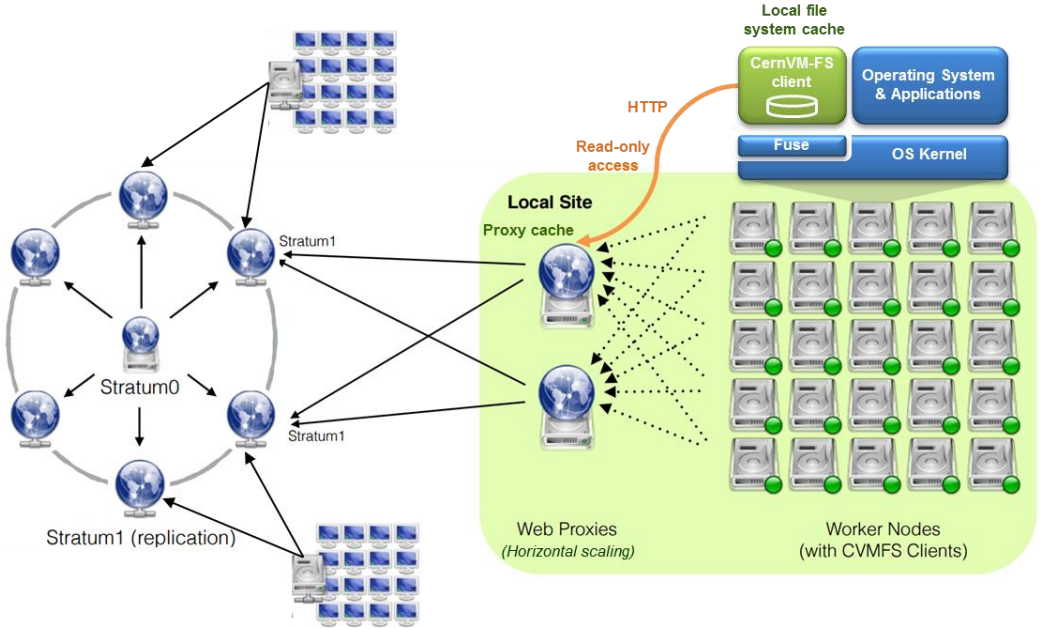


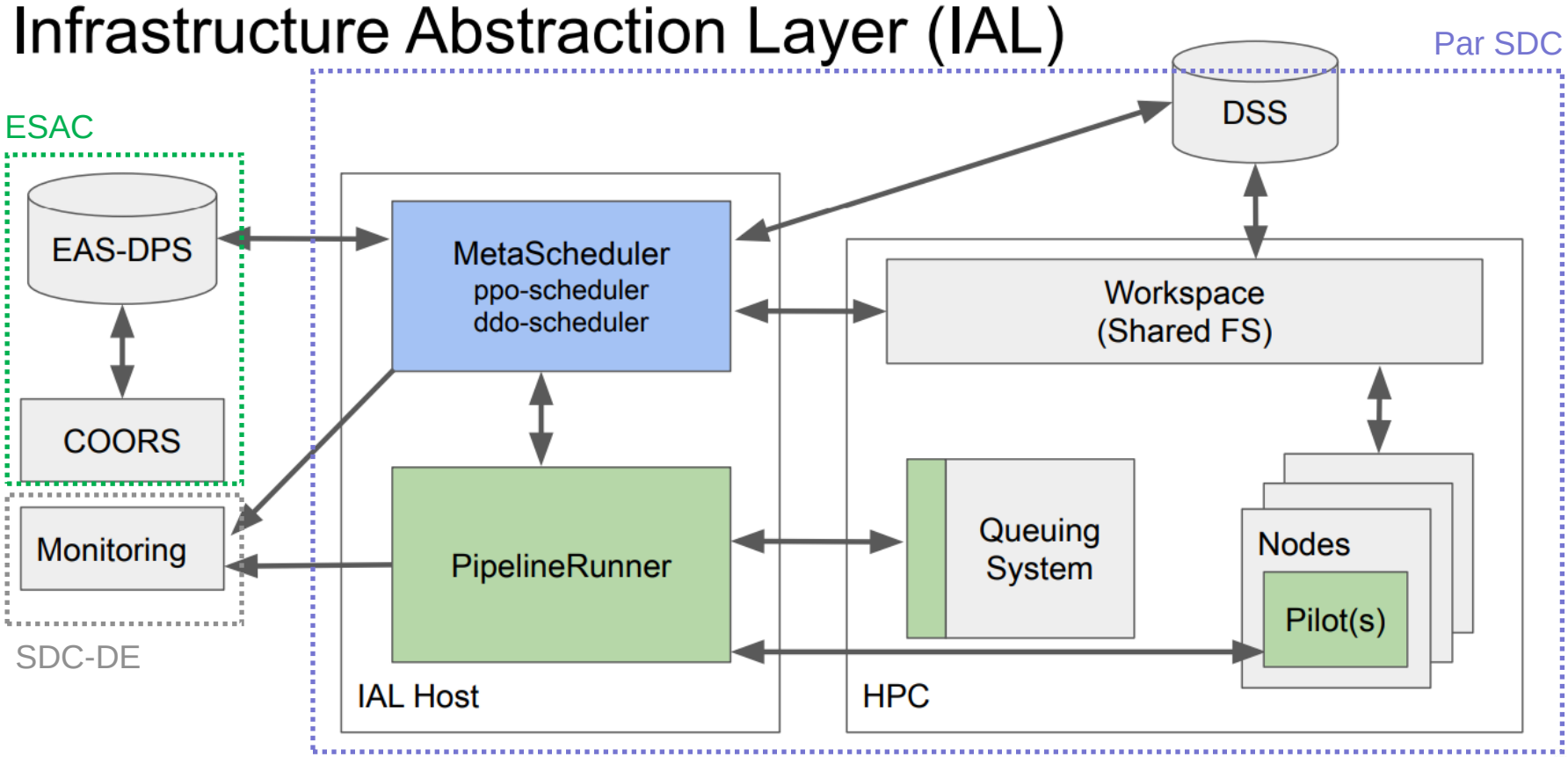
Observations with overlaps are colocated on a given SDC

Distribution du code

- Plateforme CI/CD (basée sur Jenkins) pour builder les codes

- Distribution du code dans tous les SDCs : CernVMFS
  - Déploiement des codes sur le « Stratum 0 »
  - Publication en moins de 10 mins sur tous les SDCs vis des stratum 1 puis des proxies Squid dans chaque SDC




Infrastructure Abstraction Layer (IAL)
Par SDC
ESAC
EAS-DPS
COORS
Monitoring
SDC-DE
DSS
MetaScheduler
ppo-scheduler
ddo-scheduler
PipelineRunner
IAL Host
Workspace
(Shared FS)
Queuing
System
Nodes
Pilot(s)
HPC

IAL PipelineRunner

- Développement de l'IAL pris en charge par l'université FHNW à Zurich
- Historique
  - Premier prototype en 2015 (Challenge 2), première version stable en 2017 , pour le Challenge 3
  - Développement continu, avec plusieurs refactoring et livraisons majeures :
    - Release 1.5 – Juillet 2018: multi process, ajout d'une couche de persistance, refonte de la configuration
    - Release 1.7 – Octobre 2019: robustesse et correction de bugs
    - Release 2.2 – Janvier 2021 : réécriture complète, introduction d'un bus à message interne
    - Release 2.4 – janvier 2022 : passage en python 3.10
    - Release 3.0 - été 2022 : ressources dynamiques, TMPDIR, GPU, définition de pipeline macro
- Fonctionnalités principales du Pipeline Runner
  - Traduire le Pipeline Script en un Data Flow Graph
  - Soumission des jobs unitaires permettant l'exécution du flow graph
  - Interfaçage avec les DRM des SDC (Slurm, SGE, PBS, HTCondor) ou run en local
  - Supervision des jobs et notification du MetaScheduler
  - Profiling des jobs
  - Prend en charge l'orchestration d'un pipeline donné, pas celle au niveau SGS

Description des pipelines 1/2

- Définition d'une interface pour permettre aux développeurs de définir:
  - Leurs tâches unitaires et les ressources associées: Package Definition
  - Leur pipeline complet, appelant les différentes tâches: Pipeline Script
  - Implémentation en python

- Package Definition: décrit un ensemble d'Executable
  - Nom de la commande
  - Paramètres d'appel d'entrée et de sortie
  - Ressources max utilisées: CPU/RAM/VMS/Walltime
  - Nouveautés 2022:
    - Ressources dynamiques: en fonction des sorties de la tâche précédente, passage des ressources ajustées via un fichier JSON
    - TMPDIR: utilisation d'espaces disques temporaires pour soulager les FS partagés
    - GPU: accès à un ou plusieurs GPU sur les SDC le permettant

```
ialtest_correct_dark=Executable(command="ialtest_correct_dark",
                                inputs=[Input("image"), Input("master_dark"), Input("control_params")],
                                outputs=[Output("corrected_image"),
                                         Output("darkmap", lineage=['image', 'master_dark'])],
                                resources=ComputingResources(cores=2, ram=2.0, walltime=4.0, vms=20.0)
                                )
```

Description des pipelines 2/2

- Pipeline Script: décrit le pipeline complet
  - Implémentation sous la forme de module Python
  - Appel des tâches définies dans le Package Definition
  - Décorateur Python pour définir les tâches à paralléliser (@parallel)
    - PR déduit le nombre de jobs des sorties de la tâche
  - Décorateur Python pour définir la fonction principale du pipeline (@pipeline)
  - A venir : chainage de plusieurs pipelines – Macro Pipeline

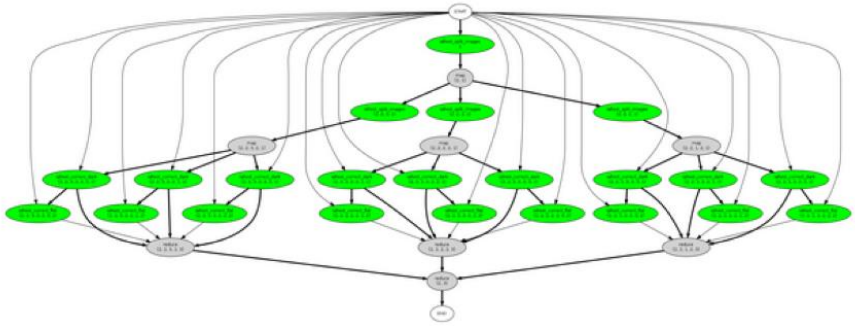
```python
from euclidwf.framework.workflow_dsl import pipeline, parallel
from ialtest_package import ialtest_correct_dark, ialtest_correct_flat,\
    ialtest_split_images, ialtest_combine_images

@parallel(iterable='image')
def correct(image, master_dark, master_flat, control_params):

    image_dark_corr, dark_corr_quality = ialtest_correct_dark(
                                            image=image,
                                            master_dark=master_dark,
                                            control_params=control_params)
    image_flat_corr, flat_corr_quality = ialtest_correct_flat(
                                            image=image_dark_corr,
                                            master_flat=master_flat,
                                            control_params=control_params)
    return image_flat_corr, dark_corr_quality, flat_corr_quality

@pipeline(outputs=('corrected_images', 'quality'))
def parallel_split(images, master_dark, master_flat, control_params):
    images_splitted = ialtest_split_images(images=images)
    images_corr_output = correct(image=images_splitted, master_dark=master_dark,
                                 master_flat=master_flat, control_params=control_params)
    images_corr, quality = ialtest_combine_images(inputlist=images_corr_output)
    return images_corr, quality
```

- Construction du Data Flow



Concept des “Pilot Job”

Origine des Pilots Jobs

- Mise en œuvre au CERN, pour l’expérimentation LHC
- Adresse la problématique des temps d’attente pour les petits jobs dans les centre de données partagés

Concept des Pilots Jobs

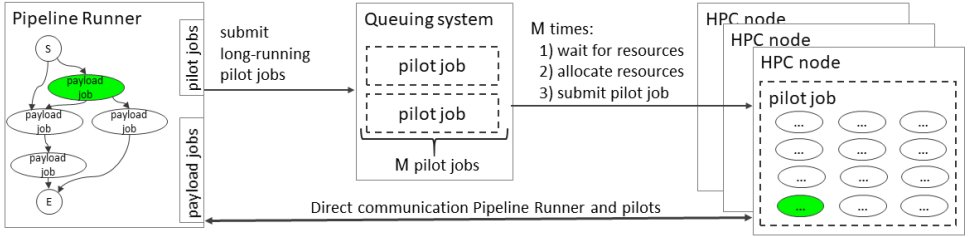
- Le Pipeline Runner réserve et soumet au DRM local un « gros » job
- Communication entre le PR et les pilots via le bus à message ZeroMQ
- Une fois ce job alloué, les PR soumet les jobs au sein de ce Pilot:
  - En fonction des ressources déclarées pour cet Executable
  - Dans la limite des ressources définies pour un pilot donné

```
# mc huge template pipelinerunner.pilots.mc_huge.maxInstances=0 to disable
pipelinerunner.pilots.heavy.CPUcores=22
pipelinerunner.pilots.heavy.rssInMB=66000
pipelinerunner.pilots.heavy.walltimeInMin=10080
pipelinerunner.pilots.heavy.maxInstances=100
pipelinerunner.pilots.heavy.maxQueued=50
```

Conséquences

- Le DRM Local ne gère que les jobs Pilot: en cas de dépassement de ressources, le Pilot est tué et tous les jobs associés
- Le Pipeline Runner doit gérer
  - Un pool de Pilot et les re-soumettre régulièrement
  - Une priorisation des jobs à soumettre au sein des Pilots

Pilot job assignment model

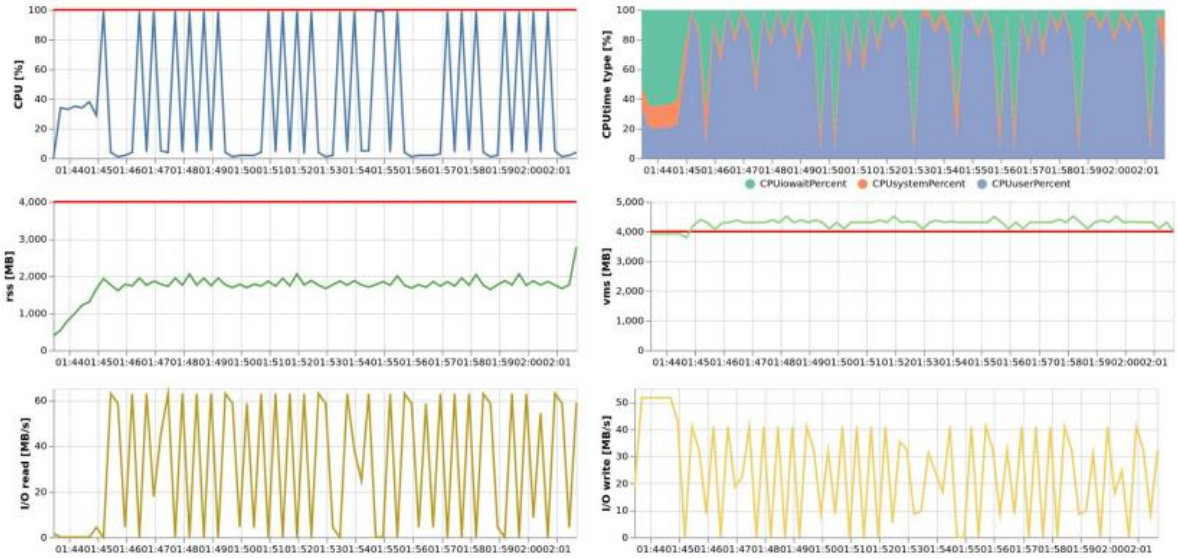


Planification des jobs

- Le Pipeline Runner doit gérer le scheduling des Pilots et des Payload Jobs
- Pilot Jobs
  - Ressources configurées par l'administrateur système en fonction de son DRM, de sa ferme et de ses règles
  - Pilots soumis uniquement à l'arrivée de Payload Jobs
  - Soumission de Pilots plus gros que les ressources demandées par les Payload Jobs
  - Configuration possible du nombre max de pilots soumis par passe, du nombre de pilots par file d'attente du DRM, nombre max de pilots d'un même type, …
- Payload Jobs
  - Calcul d'un score, à partir d'une fitness function
    - Ratio de consommation CPU/RAM/VMS/Walltime par rapport au total de chaque Pilot
    - Temps d'attente dans la queue
    - Poids de chaque ratio configurable par l'administrateur système
  - Ajout de règle en provenance de l'exécution :
    - administrateur Système peut manuellement augmenter la priorité
    - Un retry d'un payload job en erreur a une priorité haute par défaut
    - La moitié des Pilots tourne en mode starving: priorité aux plus gros jobs, le reste étant complété par les plus petits jobs
  - Dans le futur: gestion de priorité au niveau système (via les PPO), à propager aux Payload Jobs

Profiling

- Récupération de nombreuses métriques Systèmes durant l'exécution des Jobs
  - Implémenté en python via psutil
  - Suivi CPU/RAM/VMS/Ios, avec report des ressources demandées
- Investigation de problèmes de performances
- Données brutes centralisées dans une base ELK SGS



IHM PipelineRunner

- Le Pipeline Runner implémente aussi une interface graphique d'administration
  - Suivi des pipeline en cours d'exécution, ajustement des priorités
  - Affichage des données de profiling

Pipeline Run Server

Pilots Jobs Runs Configuration

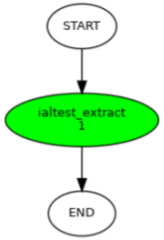
Details for Run with ID=RUN_20201006_075122_332571_PF_SingleStep: COMPLETED ###### Tuesday, 06. October 2020 10:02AM

Run Id:	RUN_20201006_075122_332571_PF_SingleStep
Processing Status:	COMPLETED
Submission Date:	10-06 07:51:23
End Date:	10-06 07:51:53
Duration:	30.094747
Workdir:	ialtest/PF_SingleStep
Pipeline Source File:	PF_SingleStep.py
Message:	None

Tasks

TICK	PILOT	STATUS	DURATION	OUTDIR	LOGDIR	MESSAGE
1	Pilot_Id.generic_heavy.20201006_074759_449877	COMPLETED	24.160559	ialtest_extract	logdir/ialtest_extract	None

Dataflow Graph



Inputs

portname	path
image	inputs/image_input1.xml
catalog_spec	inputs/catalog_spec_input2.xml

Outputs

portname	path
catalog	ialtest_extract/spectra.xml
quality	ialtest_extract/xquality.xml

Lineage

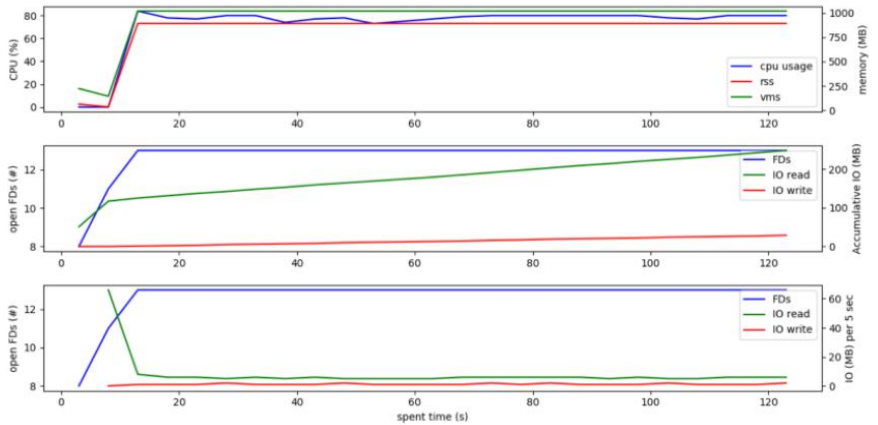
Output Port	Output Product	Used Inputs
catalog	ialtest_extract/spectra.xml	['inputs/catalog_spec_input2.xml', 'inputs/image_input1.xml']
quality	ialtest_extract/xquality.xml	['inputs/catalog_spec_input2.xml', 'inputs/image_input1.xml']

Pipeline Run Server

Pilots Jobs Runs Configuration

Details for Job with ID=RUN_20201006_145757_784911_PF_ParallelSplitConsumeResources_2_2_3_2: COMPLETED ###### Tuesday, 06. October 2020 03:09PM

PayloadJob Id:	RUN_20201006_145757_784911_PF_ParallelSplitConsumeResources_2_2_3_2
Processing Status:	COMPLETED
Created:	2020-10-06 15:00:53.699821
Ended:	2020-10-06 15:03:14.021789
Pilot:	Pilot_Id.generic_heavy.20201006_145000_220347
Retry:	0 attempts
Walltime in Min (used / requested):	2 min used of 60 requested
%CPU (avg. / cores requested):	72 % used of 1 requested cores
MB RSS (max / requested):	894 used of requested 1000
MB VMS (max):	1022
MB accumulated IO (read / write):	248 / 29
Execution Command:	cd /home/simon/euclid-ial/workspace/ialtest/PF_ParallelSplitConsumeResources && source /cvmfs/euclid-dev.in2p3.fr/CentOS7/EDEN-2.1/bin/activate && time E-Run -b x86_64-conda_cos6-gcc73-o2g ST_IalTestPipelines_ialtest_correct_flat --consume_cpu --consume_memory --consume_io --run_for_sec=120 --workdir=/home/simon/euclid-ial/workspace/ialtest/PF_ParallelSplitConsumeResources --logdir=logdir/correct.iterations.3.ialtest_correct_flat_consume --image=correct.iterations.3.ialtest_correct_dark_consume/corrected_image.xml --master_flat=inputs/master_flat_input3.xml --control_params=inputs/control_params_input4.xml --corrected_image=correct.iterations.3.ialtest_correct_flat_consume/corrected_image.xml --deadmap=correct.iterations.3.ialtest_correct_flat_consume/deadmap.xml



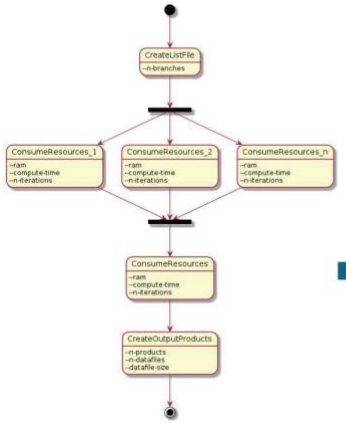
Etude Moteur de workflow générique

- Etudier la possibilité de remplacer le PipelineRunner par des COTS
  - Assurer la maintenance long terme
  - Etude faite courant 2021 avec la société CS

- Périmètre de l'étude
  - Utilisation du langage CWL (Common WorkFlow Langage) en remplacement du système PipScript/PkgDef
  - Moteur de workflow: comparaison avec TOIL, Airflow natif et Airflow-CWL
  - Utilisation du « Dummy Pipeline »:
    - ensemble de pipelines de test utilisant toutes les fonctionnalités du Pipeline Runner
    - utilisé pour valider le bon fonctionnement de chaque DataCenter après une montée de version du segment sol

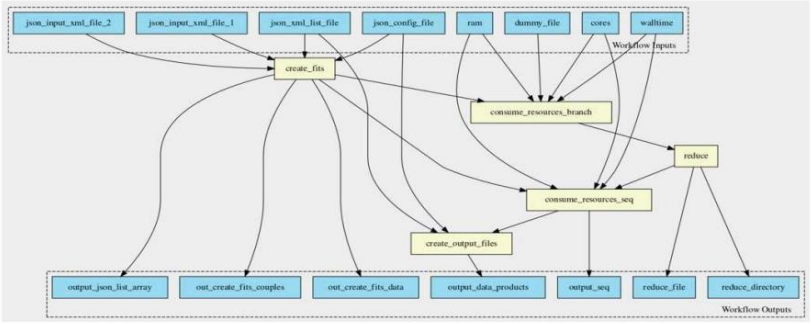
CWL

- Description
  - CWL permet de décrire des commandes et de les lier ensemble
  - CWL est une spécification à implémenter dans les moteurs



- Implémentation
  - Utilisation des formats YAML ou JSON
  - Une description CWL par étape

- Conclusions
  - CWL permet de couvrir les différentes notions implémentées dans les Pipelines Scripts et Package Definition



Moteurs de Workflow

- TOIL
  - Compatibilité complète avec CWL
  - Exécution du Dummy Pipeline sans problème majeur
    - Ajout d'étape pour garantir l'unicité des noms des fichiers/dossiers de sortie des tâches parallèles
  - Pas de trigerring intégré dans TOIL → lancement à faire par un outil externe

- Airflow-CWL
  - Extension à Airflow pour supporter CWL, en retard sur les dernières version officielles Airflow
  - Pas compatible du Dummy Pipeline, sorti de l'étude

- Airflow natif
  - Migration des langages Pipeline Script / Package Definition dans le langage Airflow
  - Compatible du Dummy Pipeline

Résultats

- TOIL
  - 500 workflows en parallèle sur une VM 8cpu/32G (en utilisant Mesos)
  - Migration vers CWL
  - Interfaçage de l'IAL et de TOIL à mettre en place pour maintenir l'IHM et déclencher les workflows

- Airflow natif
  - 1000 tâches de workflow lancées, mais max 16 en parallèle (problème pour Euclid)
  - Migration vers le langage Airflow
  - Interfaçage avec les DRM à développer

	TOIL-CWL	CWL-AIRFLOW	AIRFLOW
LAST RELEASE	2022-01-10	2021-09-21	2021-12-2021
HMI	NO	YES	YES
DATA CIRCULATION	YES	YES	NO
PARALLEL EXECUTION	YES	NO	NO/YES
DAG	YES	YES	YES
WORKFLOW CATALOG	NO	YES	YES
TASK CATALOG	NO	YES/NO	YES
WORKFLOW TRIGGERING	EXEC BIN	TIME/API	TIME/API
WORKER TYPE	Mesos/PBS/Slurm/HTCondor/LSF/ TORQUE/GridEngine/KUBERNETES	CELERY/DASK/KUBERNETES/LOCAL/SEQUENTIAL	CELERY/DASK/KUBERNETES/ LOCAL/SEQUENTIAL
DYNAMIC RESOURCES	NO	NO	YES
PLUGINS	NO	YES	YES
SCHEDULING	NO	YES	YES
LINEAGE	NO	NO	YES Can be bind https://openlineage.io/