



PROGRAMME  
DE RECHERCHE  
NUMÉRIQUE  
POUR L'EXASCALE

# ExaDoST - Work Package 1

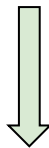
## Breakout session with SKA

# Feedback on SKA x WP1

- Measurement Sets for storing visibilities
    - New MSv4 format that should significantly improve the current version (MSv2)
    - Need to port current components to MSv4. XRadio lib to manage data
    - **An I/O benchmark using MSv4 files could be added to a I/O mini-apps repo**
      - Based on KillIMS?
    - PDI could be an abstraction layer on top of MSv4 for the pipeline components
      - More a WP2 task
  - Target applications
    - DDFacet is intended to change quite a lot (re-written in the next few months to leverage GPUs)
    - KillIMS-MPI, developed in Exa-AtoW/ECLAT, is a new good target for studying I/O behavior
      - Port to MSv4 first? Again, PDI could be the good abstraction here
      - **Need to identify an efficient software stack for monitoring I/O of Py-based apps**
        - Iheb, Luan and Jakob will work on that
        - Jean-Thomas (DDN) and Etienne (LAB) will also deploy and monitor KillIMS on DDN cluster.
        - Need to start discussions with the TOTO team for self-tuning (TADaaM)
-

# Feedback on SKA x WP1

- Concurrent accesses on the PFS from multiple pipelines
  - Behavior set to become more and more important at scale
  - The pipelines read similar data at the same time, creating contention
  - **Internship position open in Rennes to investigate that (NumPEX/ECLAT)**
- Fruitful discussions to refocus the role of WP1 for SKA
  - We have existing tools/libs in ExaDoST that could be used in the pipeline
    - PDI, TOTO, ...
  - There is a desire to propose **improvements that can be generalized.**
    - I/O optimizations in KillIMS -> limited scope
    - PDI-MSv4 plugin -> can benefit the whole radio-astronomy community
    - Methodology for profiling/monitoring I/O of Py-based applications -> useful in several communities
    - TOTO -> broader self-tuning tool targeting any type of I/O-intensive apps



# Detailed comments

# Motif inspired by SKA (from our previous AG)

An application generates a large amount of data and writes it into Measurement Set (MS). A MS file is a directory where a collection of files store diverse structured data and metadata. An MS can range in size from a few dozen MB to several hundreds of GB. For data analysis and processing, the MS is read multiple times. Data is distributed to processes according to several dimensions (temporal, frequency (channels), spatial (facets), resolution (baselines)). Following the data analysis/processing phase, new data can be added to a MS. Until all processing is complete, the integrity of the raw data in a MS must be guaranteed. We identified multiple issues with the MS format:

- The casa-core library used to access a MS is not designed for parallel I/O
  - Not using MS files does not seem to be an option. The MS format is widely used in the community.
  - A MS is directory containing small to large files (can be split at creation time? During a pre-processing phase? Depends on the resources available for the processing phase)
  - It would be best to use libraries that can adapt to hardware, instead of having to adapt the applications (because it can change, and we don't fully know how it will be yet)
  - The I/O time needs to be predicted in order to manage a storage tier whose space is very limited (at which timestamp you can get rid of a MS). Data placement is also very important.
  - shared namespace is important (like a PFS)
  - the final products of imaging are stored as a single or multiple FITS files (several or single channel per file). Mid image cubes are estimated to be ~15 TB
-

## MSv4 format is more adapted to cloud storage

- allows for loading partial data, only the chunks you need
- based on xarrays, it comes with API (xradio: <https://xradio.readthedocs.io/en/latest/>)
  - xarrays could also be a good structure for other data that is not in MS, in other steps
- it is early to say if it solves our problems
- the format is not final, still being discussed. it's happening inside SKAO
- it represents moving away from casacore
- MSv4 needs to be adopted by the data producers first (telescopes). Converters exists to translate from MSv2 to MSv4.

## PDI could be used to abstract the MS format/version

- from the discussion with WP2
- unclear if someone will do it, when, how, who

## WP1 will not work on this

## What is the target application (related to SKA) for WP1?

- DDFacet MPI
- KillMS MPI
- The DDF pipeline
- All used in production on LOFAR for instance
- but it does not make sense to put effort into optimizing MSv2-based code

KillMS could be adapted to MSv4. Good starting point to look at MSv4 limitations

- who? Iheb could start working on this. PDI-based solution is preferred
- when?

the actual software is quite complicated, should we start on a simpler piece of code (for proof-of-concept purposes)?

- how to define it?

differences between the sequential and parallel versions of ddf pipeline

the parallel version simply tackles more input (MS) files in parallel, so the I/O behavior does not really change between them (it just does more of it)

Iheb and Luan presented the profile

lots of threads: normal

the lack of asynchronous I/O is probably due to the configuration

lots of lseek calls during read phase, consistent with what Sunrise observed

- may indicate inefficient read, but it could come from the MS format

Focus on KillIMS could be a good strategy now (DDFacet may be re-written in the next few months)

how to organize the pipeline so I/O is efficient?

- killms on cpu and ddfacet on gpu?
- use cpu memory for staging?
- PDI could be a common interface to do gridding on cpu or gpu
  - data could also go to storage and be processed somewhere else

all facets depend on all the input MS files and of all frequency bins

each pipeline will continue to be small, but we will have many concurrent pipelines

- sometimes pipelines will form workflows

how to optimize the way parallel pipelines access resources?

- e.g. access to a shared parallel file system, contention
- focusing on running on specific SKA hardware
  - MS are distributed across multiple nodes
- -> ECLAT/NumPEX internship position is open on that topic (almost...)
- Concurrent accesses on the PFS creating heavy contention (Méline?)

Jakob, Luan and Iheb will look into what portions of data is read when

- the goal is to see the potential for optimizing the pipeline regarding data read (an optimal schedule of I/O accesses)
- data should be easily available from the Darshan DXT traces Iheb and Darshan have (Iheb fixed it, yay!)

work with MdIS people to use PDI, then use MSv2 and compare with the original code to see if it is good, and then do it for MSv4

- start with KillIMS because it is simpler (and ddfacet is going to be rewritten anyway). There are people in Rennes that developed the MPI version of KillIMS and could help
- Marwane already did this for another software, so he knows quite a lot about it and can help (should be kept in the loop thus)
- use the MPI version right away

containers make it difficult, there are spack recipes apparently

TOTO could be used

New version of DDFacet using GPUs is about to be released

All the MS files contribute to all the facets

Need to work on data pipelining between MS files and GPUs with limited memory

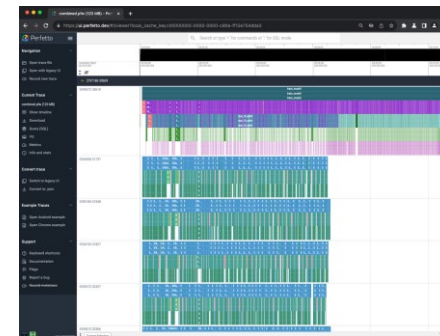
# Performance and Benchmarks

## Goals:

- Evaluate the performance of DDF-pipeline
- Understand the I/O behaviour
- Identify bottlenecks
- Measure the scalability of the pipeline

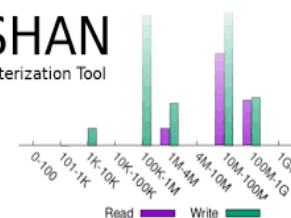
## I/O profiling tools:

- DFTracer
- Darshan



## DARSHAN

HPC I/O Characterization Tool

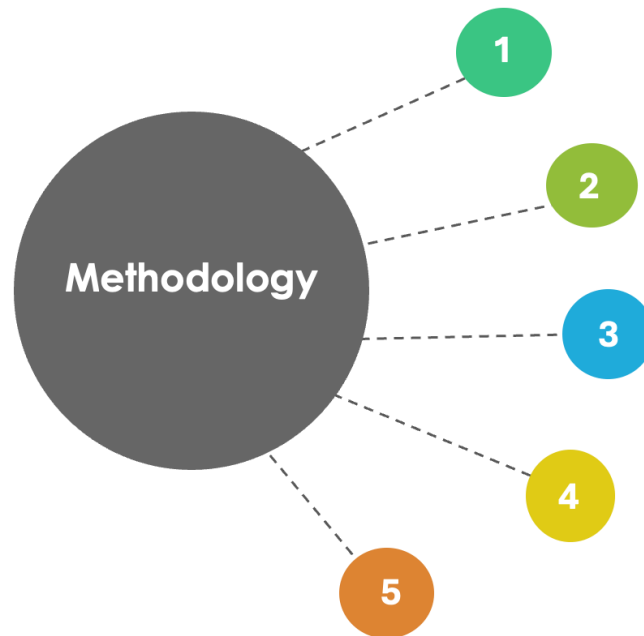


# Methodology

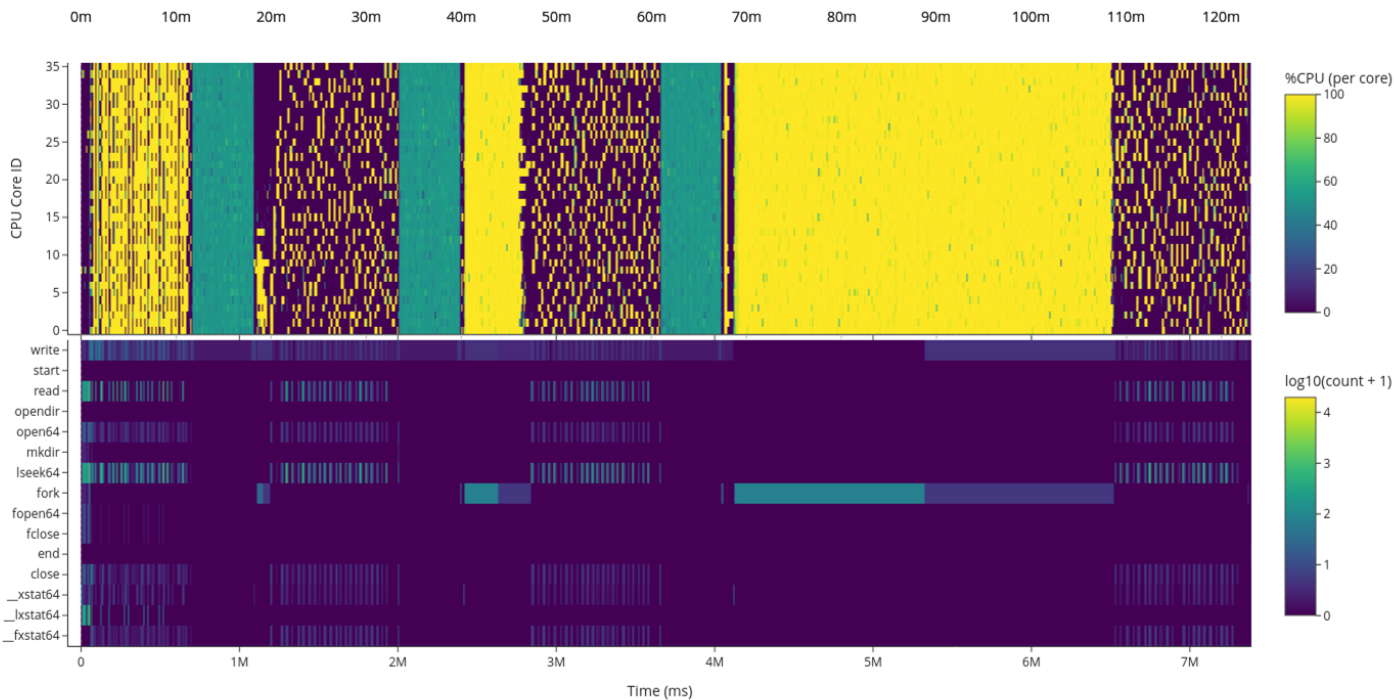
We developed a set of tools to process and visualize the traces.

All code and trace data are available at:

**[gitlab.inria.fr/lgouveia/ska\\_tracing](https://gitlab.inria.fr/lgouveia/ska_tracing)**



# DDFACET

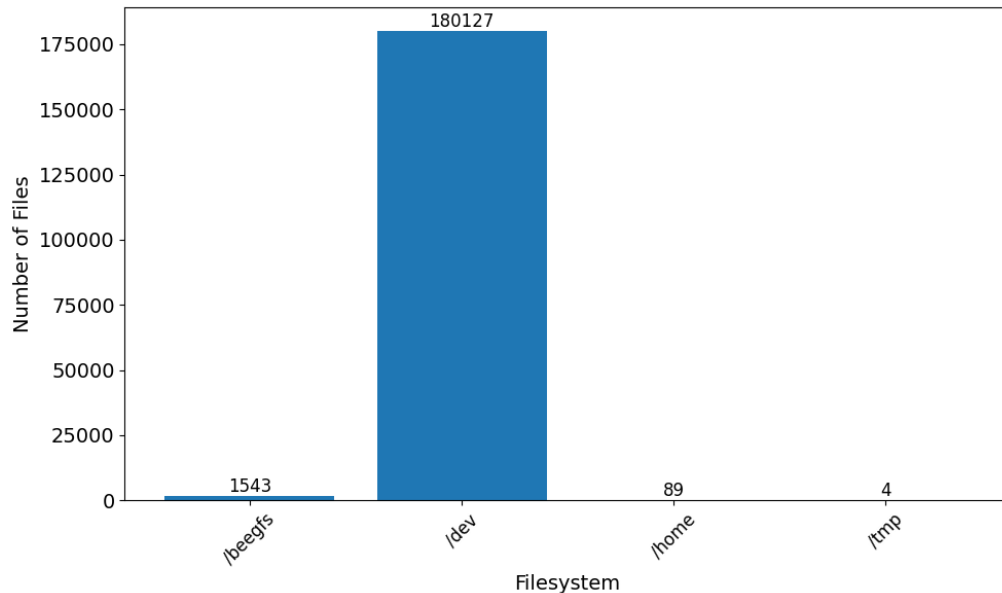


# DDFACET

**Analyzing the traces, we observe the following behavior:**

- DDFacet Alternates between I/O-bound and compute-heavy phases.
- CPU usage is low during I/O and high during compute.
- Phase transitions trigger bursts of fork and thread creation
- A single thread handles most data loading before parallel work begins



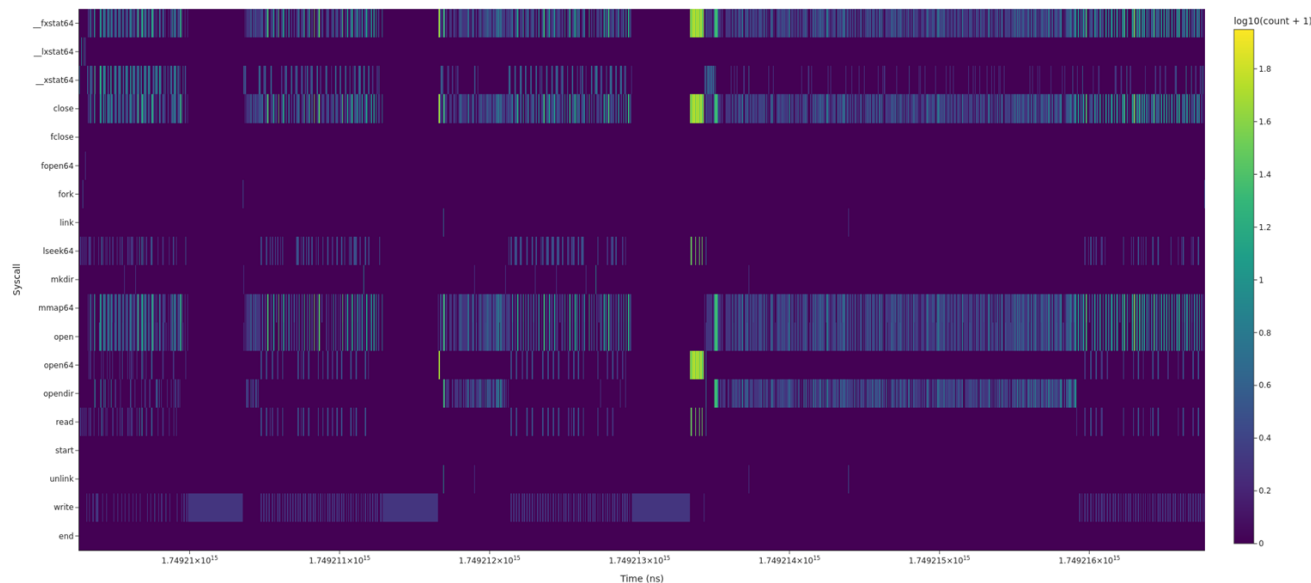


- In `/dev/shm` data is stored in volatile memory instead of a persistent storage device.
- The trace analysis shows that most of the files created by DDFacet are located in `/dev/shm`.

# DDFACET

## Syscall activity in /dev/shm is dominated by mmap64 and frequent open/close operations

Global Syscall Activity Heatmap (1000 ns bins)



**Intermediate data is stored in /dev/shm, resulting in high memory usage and frequent file mapping operations.**

## Key Findings

- DDFacet alternates between I/O-heavy phases and compute phases.
- A serialized data-loading step causes a slow startup and low CPU utilization during I/O phases.
- New threads are created at each compute phase, with one thread handling most data loading.
- Most intermediate files are stored in /dev/shm, improving speed but increasing memory usage.

## Next Steps

- Evaluate the memory footprint of /dev/shm usage and test scalability with larger datasets.
- Assess alternative I/O strategies (e.g., temporary scratch SSD or tuned PFS striping)
- Re-run experiments on multi-node setups to observe changes in I/O patterns and overheads.
- Execute the analysis across the entire pipeline.