Design Lofar Batch scheduler

Version **0.1 21-5-2013** Wouter Klijn, Bob Dröge

# Introduction

Design for an alpha version of a LOFAR batch pipeline scheduler. It is based on discussions with Wouter, Bob, Alwin and a number of requirements workshops performed in the last half year [1].

Goals: With minimal impact on the existing LOFAR system and minimal effort extend the current LOFAR system with batch scheduling capabilities. Integrating the DPU scheduling system into the LOFAR system leading the way for automated scheduling of batches of pipelines

The following subjects are discussed:

* Problem description
* Software environment
* Introduction proposed Alpha system
* Details DPU side
* Details LOFAR side
* Parset description
* Estimation of work effort
* Roll out schedule, steps and estimation of impact
* Future work

# Problem description

The CEP batch scheduler is situated at the boundary of a number of (software)systems and responsibilities which leads to the following tally of problems:

1. With the current tools available for the operators of the LOFAR instrument it is time consuming to schedule large amounts of pipeline jobs [1]. There is a large amount of dependencies between jobs and there is uncertainty in the system resources used. Jobs are scheduled with large lead in and lead out times. Jobs are scheduled manually and it can take up hours of the support scientists time to schedule a set of jobs. The resource usage of the CEP2 cluster is as a result of this low [2] and the uncertainty in running times leads to failed jobs because downstream depending jobs are sometimes started to early. To resolve most of these issues a batch scheduler is needed. At a minimum it needs to be capable to standalone receive a large amount of pipeline jobs, manage dependencies and start/stop jobs on the CEP2 cluster.
2. The majority of the offline data processing is currently performed on the CEP2 cluster. There is time available in the GRID supercomputing network for LOFAR processing but the utilization of this resource is limited due to software and operational constraints. The resolve this issue an interface is needed between LOFAR and the GRID compute resources.
3. Time / manpower constraints. Within the LOFAR software team there is limited availability of manpower. Therefore it would be optimal if the bulk of development is performed by external partners.



Figure 1: Schematic (incomplete) overview of the interactions with the LOFAR systems of importance to the Pipeline framework. The inset details the workflow for specification of on pipeline type observation. The steps are numbered and can be found in the figure.

# Software environment

The CEP batch scheduler is not a solitary program/ function. It will need to operate in an existing software environment. Here follows a short introduction to the systems involved. Figure 1 displays a schematic overview of systems touching the offline processing part of LOFAR, details regarding online processing/station control/archiving are not included.

It is important to distinguish between two data flows in the system. First there is the recorded radio signal, the astronomical data as produces by Online processing system (also known as the BGP), and stored on the CEP2 cluster. In a next phase this raw data is processed using the functionality encapsulated in the pipeline recipes and finally it is offloaded to storage facilities of our external partners using the LTA system

The astronomical data is described with specification- and meta information. It contains details regarding the type of observation to perform, what is the target and various statistics produced along the whole processing chain. This information stream starts (normally) in MOM. An astronomer specifies a target and observation type. This is the stored in the MAC systems database: OTDB. Support Scientist then fill in the details of the observation using MOM. After all specifications are know the last parts of the information are added using the scheduler: A start-stop time and the locations of the input and output data products. Based on the start time MAC then start an Observation/Python control which in turn creates a connection to the head node of CEP2 calling StartPython.py, a small wrapper setting up environment variables that then starts the pipeline. It performs numerous operations on the RAW observation data. The pipeline produces statistics and general information regarding the run which is send back to MAC in a final step. Upload to the external partners is performed with the LTA system (not displayed in figure 1).

# Introduction proposed Alpha system

We currently envision the Alpha system as a combination of the Distributed Processing Unit (DPU) and a preprocessing step/module converting and combining the parsets as they are produced by the MAC system into a parset that can be used by DPU. The DPU would then start individual jobs/workloads on the CEP2 nodes. We do not envision any changes in MAC and minimal changes to the Pipeline framework. The individual jobs/workloads will start a full pipeline including all bells and whistles. This will lead to spurious computations, but the total running time should not be any larger

The DPU needs to be extended with new functionality which will detailed in the next part of this document, the work needed on LOFAR side is detailed in the subsequent one.

Figure 2: on the next page shows a schematic view on the design. It contains the Observation control/ Python control and MAC feedback module to be able to relate it to Figure 1. The StartPython module is this the start point of the system. In the old design it would start the pipeline toplevel recipe. It will will now send the parsets to a new module that collects these parsets. Collects dependency information and combines them together in a large parset to be consumed by a DPU parset interface. The DPU itself will schedule the individual jobs/workloads on CEP2 nodes. The feedback cannot be handled by the pipelines itself for they contain only information of individual jobs. The observation combiner will need to collect the feedback from the jobs and combine it in a single feedback file for MAC.



# Details DPU side

The DPU will be placed at the interface between two LOFAR modules. In the current situation the parsets supplied by the python controller are used by the StartPython script to select the pipeline script to start. The parset is then used as an argument. It at this exact junction that the DPU will be placed. There are a number of issues that first have to be resolved: Firstly the DPU does not have a parset interface. Secondly the DPU cannot distribute jobs to specific nodes in a cluster. Each issue will be addresses individually:

The LOFAR systems communicate using a homebrewed parameterset format. It is a human readable list of key value pairs with some additional automatic parameter expansion capabilities. Keys can be nested, values can be lists, strings or empty. The LOFAR tree contains a parset implementation for all major programming languages including Python. The details of the actual parset implementation are hidden and the DPU Parset interface can easily retrieve arguments normally supplied as command line arguments. The DPU Parset interface (the keys and values used) will need to be designed with the all possible use cases in mind and is therefore discussed in detail in a later part of this document. The interface will at a minimum contain the location, parameters and interdependencies for jobs to be scheduled and a set of parameters controlling the DPU.

The current standalone DPU version does not have the option to place a job on a specific node, a feature needed for CEP2 processing (The data size prohibits copying of data before the final data reduction in the imaging pipeline). There are addition software packages available, Torque/Maui [3], that are capable of performing this task. For the alpha version this would entail to much work and therefore DPU needs to be extended with this functionality.

# Details LOFAR side

The work to be perform on the LOFAR side entails three major topics. Firstly the creation of a parset containing all information of a set of interconnected jobs. Secondly the location of feedback to MAC must be moved from the Pipeline to the new Batch scheduler module. Finally functionality should be added that can combine the feedback of the individually running subbands into a single large feedback file, combining the logfiles should also be taken in account.

A single parset contains information regarding X parallel sub bands and ID’s linking it to predecessors. To make full use of the DPU capabilities, namely the creation of job lists containing dependent jobs, e.g. Two MSSS target runs depending on a MSSS calibrator, the information needs to be collected in a single parset. The creation of these combined parset is new functionality, although it would be preferred to have this functionality added to either MOM of the Scheduler, the amount of work would be big and the goal of this alpha system is to add functionality with minimal changes to the existing infrastructure.

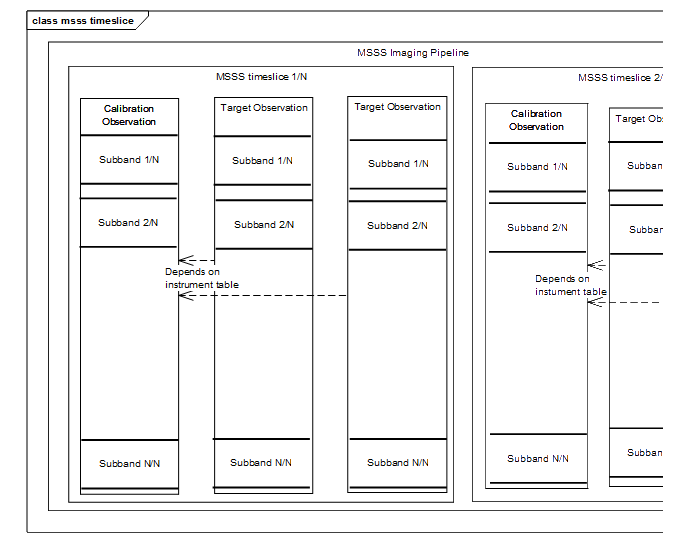
In the current Pipeline framework Top level recipes start a feedback recipe that collects information about the current run and sets up a connection to the MAC feedback, signaling a finished pipeline and providing statistics about the pipeline. Because we plan to run each individual subband in a full pipeline this functionality should be moved to the batch scheduler module. The information of the individual subbands should be combined and a single file should be used for feedback. To allow debugging from a single location the logfiles created at each processing node should also be collected and stored on a single node.

# Parset description

The exact details of the parset to be presented to the DPU module depends for a sertain extend on implementation details. It should at a minimum contain the following:

* The parsets to be provided to pipelines, without dataset information.
* The dataset information and node locations
* A set of DPU parameters.
* Dependency information

The parset and dataset information should be structured in such a way that it is possible for the DPU parset wrapper to simply grab the appropriate parset and add the dataset information at job launch time.



# Estimation of work effort

|  |  |  |
| --- | --- | --- |
| **Task** | **Hours** | **System** |
| Node specific job placement | 30 | DPU |
| Node specific job using Torque + Roll out | 30 + 30 | BatchScheduler |
| DPU parset wrapper | 40 | DPU |
| Maximum running time for jobs | 20 | DPU |
| DPU output collection, parset and logging | 20 | DPU |
| Parset Combining | 20 | BatchScheduler |
| Observation queues | 20 | BatchScheduler |
| MOM DPU control parameters (optional?) | 30 | MOM |
| Logfile combiner | 10 | BatchScheduler |
| MAC feedback | 20 | BatchScheduler |
| BatchScheduler daemon | 40 | BatchScheduler |

# Roll out schedule, steps and estimation of impact

A number of steps can be taken to limit the impact of adding the batch scheduler to the LOFAR software environment. Firstly the system will be tested on the CEP1 cluster which is not used in production. Secondly a test on the CEP2 test cluster running a small LOFAR environment. In a final step the BatchScheduler should be released in the full system. Optionally using a switch already present in the StartPython.py script and in a later stage as the default method of scheduling pipelines. Because the control and management mechanisms will not be changed it should be possible to roll back to the old situation in case unforeseen complications arise.

# Discussion and future work

The design described in this document focusses on a simple alpha system adding batch scheduling capability to the current LOFAR framework. It also provides a proof of concept of DPU mediated launching of pipeline kernels. The impact on the operational system is low. The amount of effort needed is limited while the usability of the system greatly increases.

A number of improvements or extensions can be added. Firstly, a Torque/Maui system could be added. This combination of systems would allow advanced resource based scheduling increasing the usage of the current CEP2 cluster. This would also allow interfacing with a to be build CEP3 cluster.

Secondly, the DPU could interface with the GRID. It would then be possible to offload a substantial amount of the processing to remote sites. This option would need substantial work for a feedback mechanism would have to be created that is not dependent on MAC: GRID jobs would be added to long queues. The jobs would in this case stay in a non-finished state for long times. A solution could be the addition of an extra state ‘offloaded to GRID’ etc. Also the fact that data is not located on the node where computations are perform adds complications.

A third application of the current system is the creation and running jobs in the archive. This could be used to perform pipeline runs on archived data without the need of the LOFAR control system

# Bibliography

|  |  |
| --- | --- |
| [1] | W. Klijn, „CEP batch scheduling,” 10-01-1013. |
| [2] | W. Klijn, „CEP2 load: Visualization and analysis,” 11-01-2013. |
| [3] | W. Klijn, „Investigation of the Maui and Torque job scheduling tools,” 8-03-2013. |