# Master thesis proposal
# A container based self-adjusted auto-provisioning framework at resource level

**You Hu**
adolphus.hu@student.vu.nl
VU Amsterdam

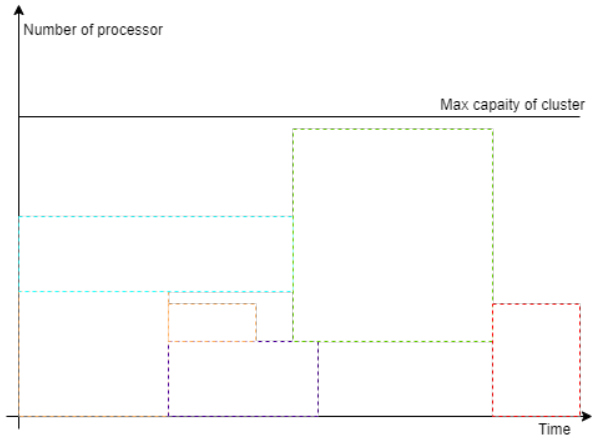
## 1 INTRODUCTION

The Netherlands eScience Center has developed solutions for calibrating imaged observation collected by LOw Frequency Array(LOFAR) telescope[1]. The LOFAR consists of 51 stations cross Europe and a typical LOFAR observation has the size of 100TB, after frequency averaging, the size can be reduced to 16TB. [6] Collectively, there are over 5 PB of data will be stored each year. [1] To calibrate the observation by given sky map, SAGECaL is invented and implemented for this purpose.[3] By given pre-processed observation data, sky model and parameters, the calibration can be done independently. However, it is a computation consuming application. Currently, eScience Center has developed GPU, MPI and Spark versions for acceleration. All of them have achieved great acceleration compared to the naive version.

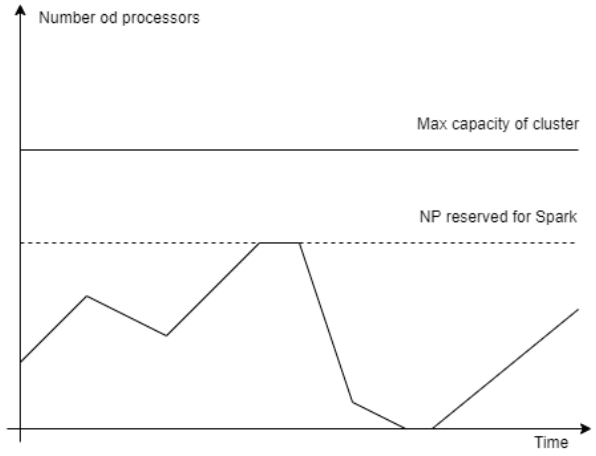
However, the solutions following horizontal scaling idea, MPI, and Spark, will lead to a waste of computation resources in non-dedicated clusters. In this project, we try to build up a system to achieve auto-provisioning at public clusters to drive the computation consuming applications. This solution can be applied to other more applications with the demand for auto-provisioning.

## 2 THE COMMON ISSUE ON CURRENT VERSIONS

Both MPI version[2] and Spark version[3] take SAGECaL process as a black box, it aims to schedule the task on multiple nodes to achieve acceleration. For the MPI version, the architecture is a master-worker like, but the task scheduling mechanism is very simple. In the Spark version, the tasks are carried by the driver with Java Native Interface. And each task is invoking an encapsulated C++ program. The common issue of MPI and Spark version is that the resources can be not be fully utilized in the non-dedicated cluster. The resource utilization of these two systems can be visualized as Fig. 1. As it is shown in Fig. 1a, MPI jobs are scheduled as fixed batch jobs which are colored boxes in the figure. It is very common to meet the situation that all jobs in the queue are too large and the number of idle resources is not enough for the jobs in the queue. For the Spark version, the possible situation can be visualized as Fig. b. The nodes should be reserved for Spark in advance and Spark handles the task scheduling. For typical Spark applications relying on RDD, the number of required executors can be up and down dynamically. Part of computation resources may be wasted since this computation power is exclusive for Spark.




**(a) Resources utilization of MPI batch jobs on cluster**




**(b) Resources utilization of Spark on cluster**

**Figure 1: The waste of compuation on cluster**

[1] http://www.lofar.org/
[2] https://github.com/nlesc-dirac/sagecal/tree/master/src/MPI
[3] https://github.com/nlesc-dirac/sagecal-on-spark

However, of course, the current Spark implementation for calibration is based on Driver mode and the granularity is one executor per task. In this case, the waste of resources won't happen within the Spark cluster. But when there are a lot of idle nodes in the cluster that not reserved for Spark at the beginning, they can not be used for Spark.

## 3 RESEARCH QUESTION STATEMENT

To tackle on the problems mentioned in previous sections,here, it can be summarized as the research question:

- Is it is possible to build up a system/framework equiped with auto-scaling stratagy at resource level under public clusters?

Moreover, the following topics can be considered together:

- portablility for utilizing remote (heterogeneous) resources
- data locality on cross regions
- fault tolerence

## 4 RELATED WORKS

The common solution for managing resources is utilizing cloud management platforms like OpenStack or OpenNebula. Tania et al. [4] listed multiple methods for auto-scaling upon VM based cloud. However, in our case, the application runs in clusters instead of the cloud environment. On the one hand, the VM is very heavy while our task is fixed. On the other hand, the middlewares need to be placed on nodes in advanced like Spark daemon, this leads to the same problem as Spark's. Of course, the strategies of resource management in this report and other papers[2][5] can be applied in our project.

## 5 METHODS AND TECHNOLOGIES

In the following subsections, the technologies that we intend to use will be described. And in the last subsection, a rough design will be illustrated.

### Ibis Portability Layer(IPL)

Ibis[7] is a programming environment that combines Java's "run everywhere" portability both with flexible treatment of dynamically available networks and processor pools, and with highly efficient, object-based communication.[4] IPL can be used for communication between executors. The advantages of IPL are the cross-domain router, builtin leader election mechanism, and user-friendly interface. All of these help to implement a parallel application.

### Docker container

A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, and settings.[5] In our case, the executors are based on containers which encapsulate all related environment. It is more lightweight than VMs and easier to manage.

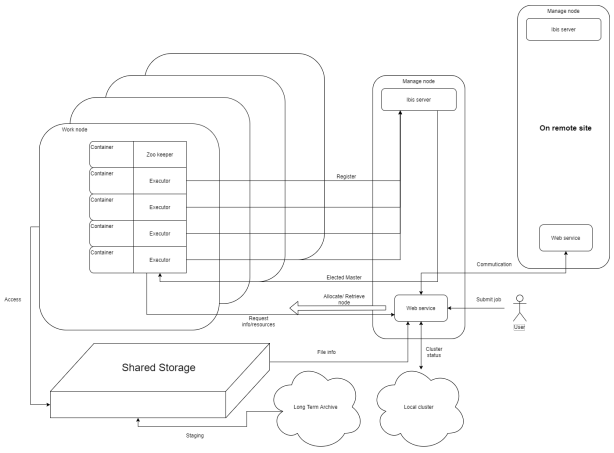



**Figure 2: An over view design for the system**

### Xenon

Xenon is a middleware developed by eScience Center that tends to provide a single programming interface to many different types of remote resources, allowing applications to switch without changing a single line of code.[6] This tool enables the program to communicate with the cluster and to manage resources automatically.

### Rough design

Here, a rough design is purposed and shown in Fig. 2. The idea of our solution is that for each cluster, we assume there is one node never crash conveys ibis-server and service for: job submitting; computation node allocation and release; communication with other clusters. Once a node is assigned, it will join/create the docker swarm or Kubernetes cluster. The containers will execute the same code, and elect ibis master. The master receives a job list and allocates tasks to workers. It also monitors the job list to send scaling up or down demand to the server( this part can be discussed in detail later). The job list or task arrangement can be persisted by something like Zookeeper to reduce the cost of the crash of the master.

## 6 EVALUATION

There are a few kinds of features that need to be evaluated. The core is the resource utilization of the cluster. The average usage rate and overall running time for given tasks can

[4]https://www.cs.vu.nl/ibis/javadoc/ipl/index.html

[5]https://www.docker.com/resources/what-container

[6]https://github.com/xenon-middleware/xenon

be measure for old versions(MPI and Spark) and the new version. Since here we would use JNI to invoke SAGECaL function, we can replace it by any computation consuming and data-intensive application. For data locality features, we can compare the time spent between the remote access version and the version with locality optimization. For fault tolerance, we can randomly force kill nodes to test this feature.

## 7 PLAN

- Phrase 1 - literature study March 30th- April 27th
  - Week1: literature study on resources management algorithm
  - Week2: literature study on container-based auto-scaling systems
  - Week3: literature study on cross fields
  - Week4: summarizing and reporting
- Phrase2 - implement prototype : April 27th - May 1st
  - Week1-2: implement a prototype with simple scaling mechanism in one cluster
  - Week3 add fault tolerance feature
  - Week4-5: design a benchmarking framework for different scaling mechanism
- Phrase3 - add&test features : May 1st - May 29th
  - Week1-2: test different(2-3) mechanisms
  - Week3: add cross region features
  - Week4: enhance testing system for cross region requirement
- Phrase4 - final evaluation : May 29th - June 26th
  - Week1-2: evaluate the performance and try to compare to existed versions
  - Week3-4: debuging and optimizing
- Phrase5 - organize thesis paper : June 29th - July 24th
  - Week1-2: make the draft
  - Week3-4: review and improve

## REFERENCES

[1] 2020. PROviding Computing solutions for ExaScale ChallengeS. 777533 (2020), 1–163.

[2] Hyejeong Kang, Jung In Koh, Yoonhee Kim, and Jaegyoon Hahm. 2013. A SLA driven VM auto-scaling method in hybrid cloud environment. *15th Asia-Pacific Network Operations and Management Symposium: "Integrated Management of Network Virtualization", APNOMS 2013* (2013).

[3] S. Kazemi, S. Yatawatta, S. Zaroubi, P. Lampropoulos, A. G. de Bruyn, L. V.E. Koopmans, and J. Noordam. 2011. Radio interferometric calibration using the SAGE algorithm. *Monthly Notices of the Royal Astronomical Society* 414, 2 (2011), 1656–1666. https://doi.org/10.1111/j.1365-2966.2011.18506.x arXiv:1012.1722

[4] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A. Lozano. 2014. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments. *Journal of Grid Computing* 12, 4 (2014), 559–592. https://doi.org/10.1007/s10723-014-9314-7

[5] Ming Mao and Marty Humphrey. 2011. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. *Proceedings of 2011 SC - International Conference for High Performance Computing, Networking, Storage and Analysis* (2011), 1–12. https://doi.org/10.1145/2063384.2063449

[6] Hanno Spreeuw, Souley Madougou, Ronald Van Haren, Berend Weel, Adam Belloum, and Jason Maassen. 2019. Unlocking the LOFAR LTA. (2019), 467–470. https://doi.org/10.1109/eScience.2019.00061

[7] Rob V Van Nieuwpoort, Jason Maassen, Rutger Hofman, Thilo Kielmann, and Henri E Bal. 2002. Ibis: an efficient Java-based grid programming environment. In *Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande*. 18–27.