A Scalable Architecture for Rapid Terrain Analysis

Frido Kuijper and Ruben Smelik
TNO Defence, Safety and Security – The Netherlands
frido.kuijper@tno.nl | ruben.smelik@tno.nl

Rationale

Terrain databases are a *key prerequisite* for all simulations that seek to support training or decision support during military exercises and missions

The generation of terrain databases involves complex and extensive compute tasks on large amounts of geospatial data and models and is therefore *time consuming and costly*

Automatic Rapid Analysis shall significantly reduce terrain database generation lead times

Problem

Design a software architecture that

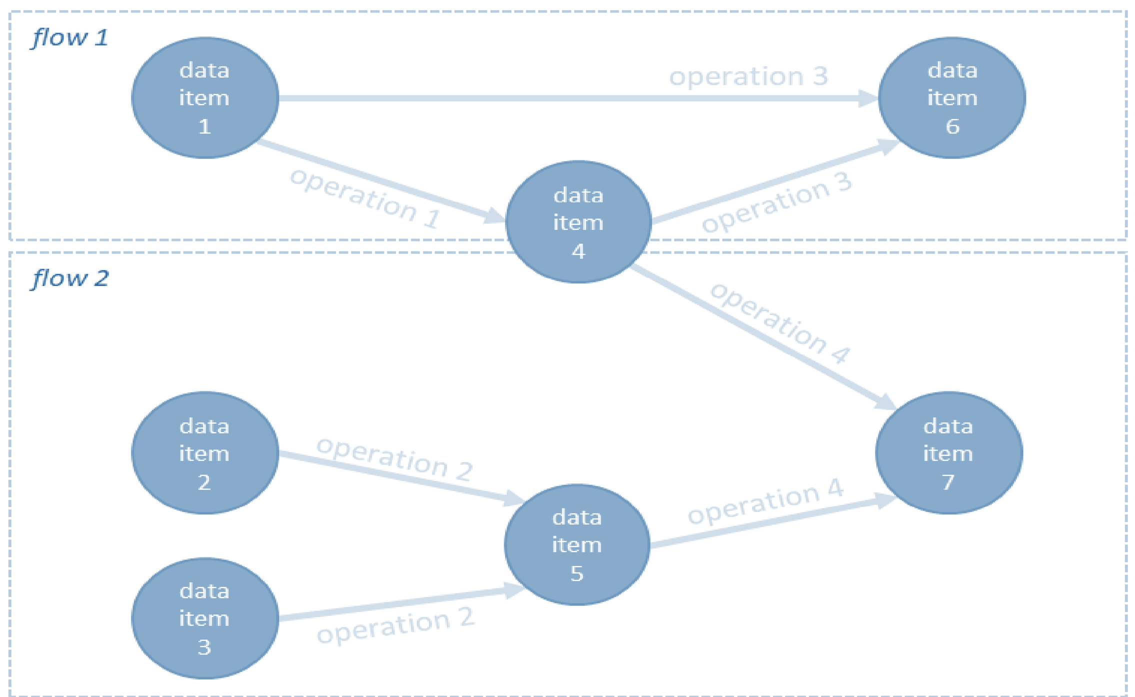
- facilitates development of complex compute flows for terrain analysis and modelling
- allows for execution of the compute flows on a scalable compute cluster

- improves maintainability and quality of the software
- simplifies production of large terrain databases
- drastically reduces compute times through scalable cluster deployment

unified in an architecture dedicated to rapid terrain analysis

Solution

The ARA architecture is dedicated to the flexible decomposition of terrain analysis tasks into smaller subtasks in order to spread the computation and data load across a cluster of compute nodes.



Key concepts

Software Tools are used to transform source Data Items into new Data Items. A Flow is a set of Operations that specify input/output Data Items and associated Tools. During execution, the architecture turns the Operations into Jobs on a compute cluster.

Job dependency management

The architecture knows how operations depend on changes in data items and tool code. Jobs are created as required and executed as soon as possible.

Parallel processing

Independent flows/jobs run in parallel. Mechanisms for data parallelism:

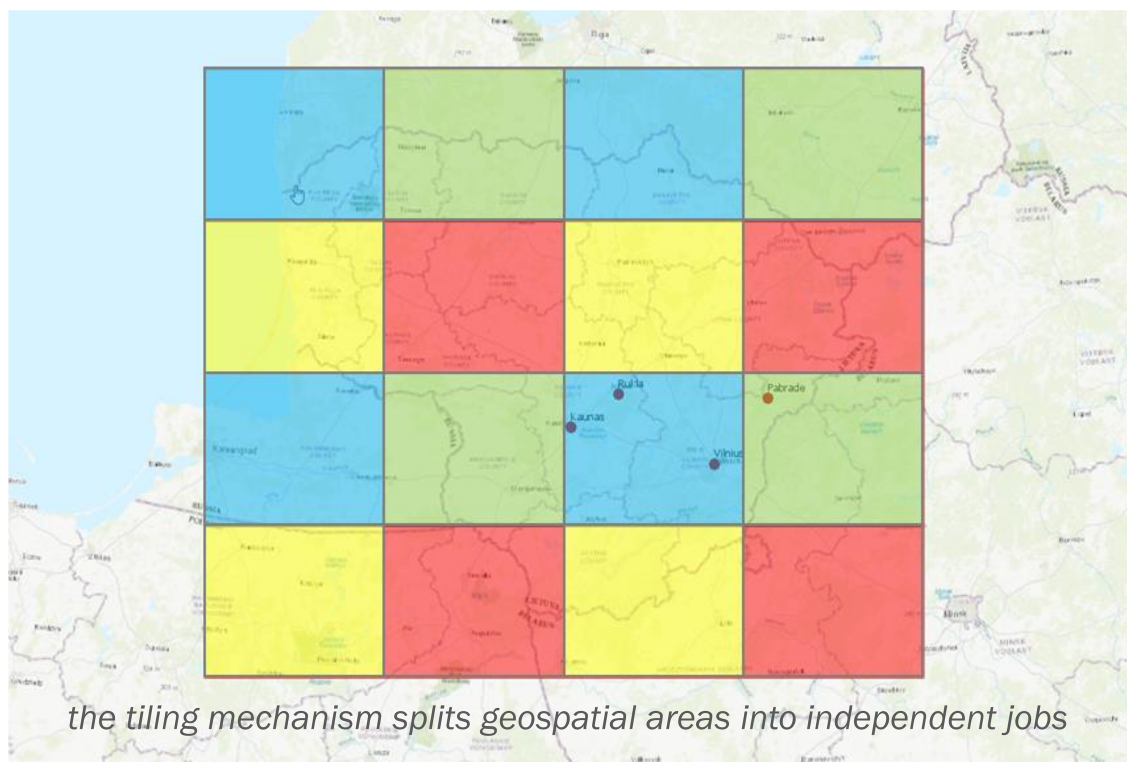
- batching operations
- tiling operations

Data transfer management

The architecture automatically prepares input data items for jobs on a specific node in a cluster and archives results on central storage as required.

Scalability

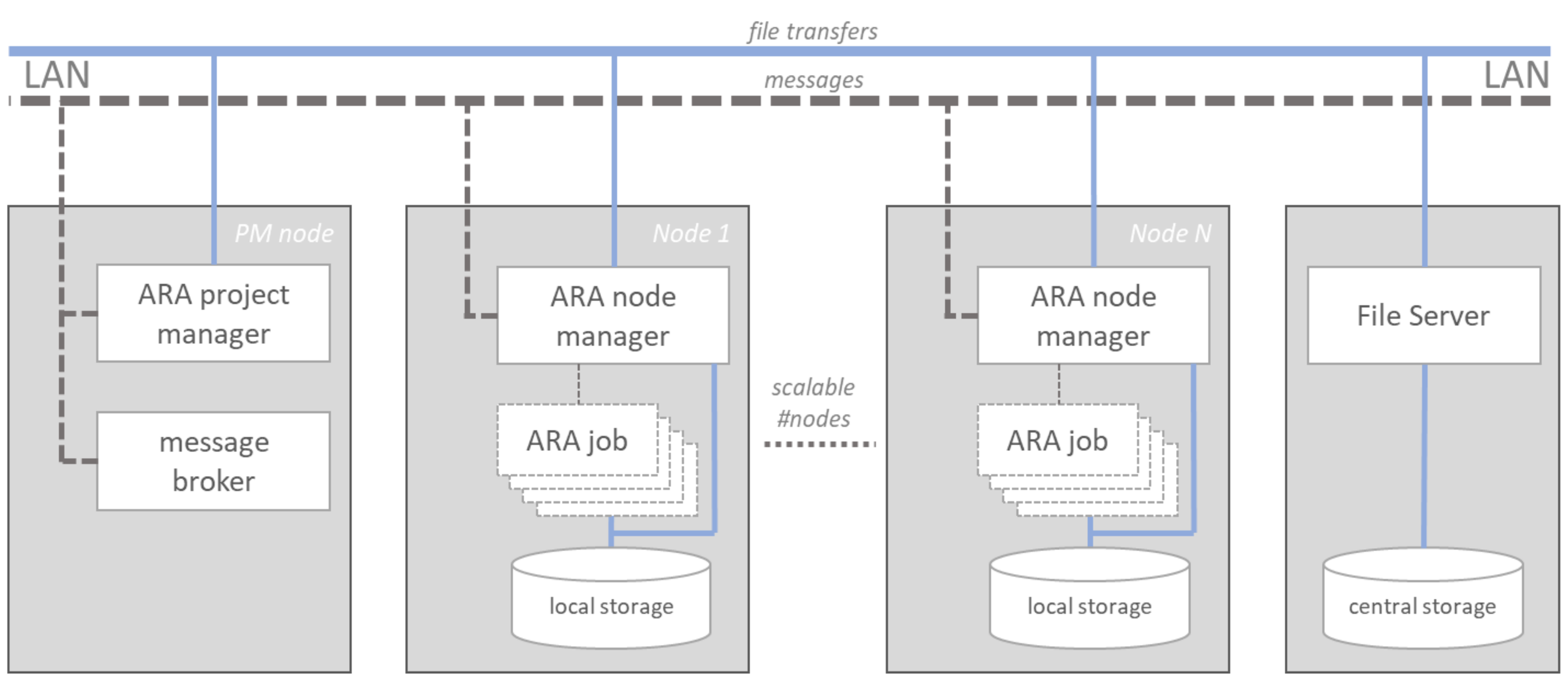
The architecture can be deployed on a scalable compute cluster. Tool code is agnostic to cluster configuration and the parallel processing mechanisms



the tiling mechanism splits geospatial areas into independent jobs

Implementation

Main components ARA Project manager and ARA Node Manager implemented for Windows compute clusters. Jobs interface through Python API to ARA architecture.

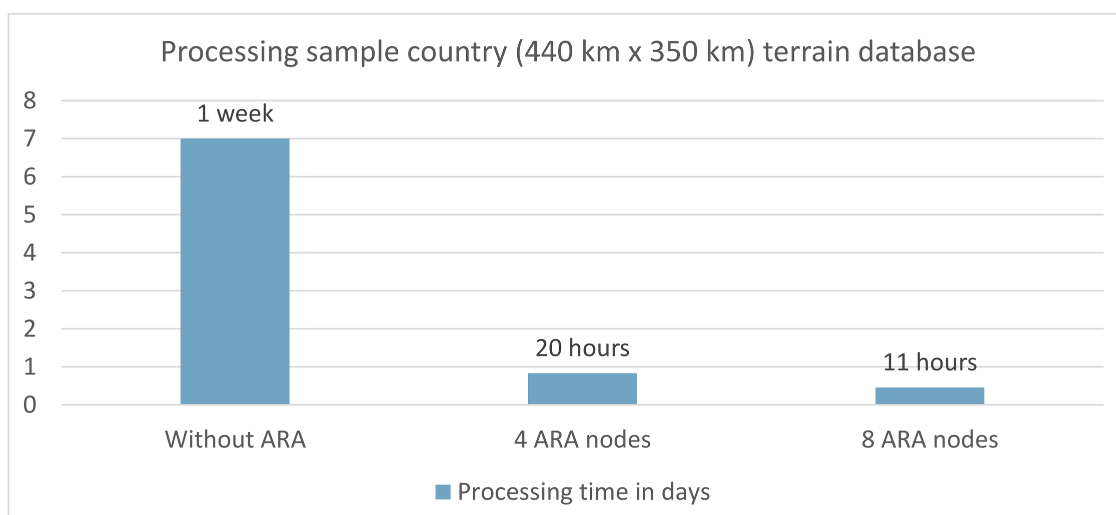



```
# ARA tools are modular, relatively small pieces of Python code
# using the ARA API to access the architecture’s features.
# The parallel processing mechanisms are fully transparent in tool code.

class Tool(BaseTool):

  def RunTool(self):
    self.logInfo('*** Elevation - compute DTM ***')

    # Get local filenames
    input_dsm = self.getInputLocalPath('input_dsm')
    output_dtm = self.getOutputLocalPath('output_dtm')

    # Get parameters
    threshold = self.getParameter('threshold')

    # Perform the DTM filtering
    self.RunDsm2Dtm(input_dsm, output_dtm, threshold)

    # Register output
    self.registerOutput('output_dtm')

    # Signal tool completion
    self.exitNoError('Done')
```

Results

The ARA architecture was tested on a sample project for country size terrain database generation (440 km x 350 km, 1.2 TB source data)

- Processing times reduced from weeks time to over-night
- Scaling of number of nodes practically linearly reduces processing time
- Modular tool code structure improves maintainability
- Job dependency and data management improves user experience when iteratively developing terrain projects



innovation for life