



SME-LET Announcement of Opportunities 2009: Cal/Val and User Services -Calvalus

Technical Specification
Version 1.2
21. March 2011



Change log

Version	Date	Revised by	Change	Authors
0.1	11.06.2010	-	The initial draft version of this document.	Fomferra
1.0	16.11.2010		Document structure modified scenarios description architecture with hardware, interfaces and decomposition, Hadoop as concurrency engine Hadoop concurrency tradeoff analysis and process design	Fomferra, Zuehlke, Boettcher
1.1	04.01.2011	-	RID conclusions of ESA review of 1.0 integrated (see Excel list for RID decisions and status)	Fomferra, Zuehlke, Boettcher
1.2	21.03.2011		Improved a number of changes improving comprehensiveness: <ul style="list-style-type: none"> • Now using term "GAC QAA" instead of "QAA". • Now using term "Product file" over "Product", whenever we refer to single, physical files • 3.1.3: Explained role of GAC processors and mentioned SeaDAS I2gen as another possible processor. 	Fomferra

Table of Content

1	Introduction.....	- 1 -
1.1	Purpose and Scope	- 1 -
1.2	Terms and Definitions	- 1 -
1.3	Abbreviations	- 2 -
1.4	References.....	- 3 -
1.5	Document Overview.....	- 5 -
2	Overview.....	- 6 -
2.1	Cal/Val Application Domain.....	- 6 -
2.2	Hadoop Distributed Computing	- 7 -
2.3	Calvalus Approach for Concurrent Processing	- 10 -
2.4	Prototype System Context.....	- 10 -
3	Operational Scenarios	- 12 -
3.1	Production Scenarios.....	- 12 -
3.2	System Use Cases	- 22 -
3.3	Input Data Requirements	- 26 -
4	Hardware Environment	- 30 -
4.1	Hosted hardware vs. cloud computing	- 30 -
4.2	Hardware selection	- 30 -
4.3	Configuration.....	- 32 -
5	Concurrency Trade-off Analysis.....	- 33 -
5.1	Investigated Configurations	- 33 -
5.2	Measurement Results	- 36 -
5.3	Hadoop-Specific Findings	- 38 -
5.4	Conclusion	- 40 -
6	System Component Design	- 42 -
6.1	System Decomposition.....	- 42 -
6.2	User Portal.....	- 43 -
6.3	Catalogue and Inventory Service.....	- 44 -
6.4	Production Service.....	- 45 -
6.5	Ingestion and Staging Service.....	- 46 -
6.6	Hadoop Distributed File System (HDFS).....	- 47 -
6.7	Hadoop MapReduce Engine	- 48 -
6.8	Hadoop-to-Processor Adapter	- 48 -
6.9	Processor	- 50 -
7	External and Internal Interfaces	- 51 -
7.1	User Web Interface	- 51 -
7.2	Catalogue Interface	- 55 -



7.3	Inventory Interface	- 55 -
7.4	Production Interface.....	- 55 -
7.5	Ingestion Interface	- 56 -
7.6	Staging Interface.....	- 56 -
7.7	HDFS Interface.....	- 56 -
7.8	Hadoop Job Interface	- 57 -
7.9	Hadoop MapReduce Implementation Interface	- 57 -
7.10	Processor Interfaces	- 58 -
8	System Process Design	- 60 -
8.1	Concurrent L1-to-L2 Processing on HDFS.....	- 60 -
8.2	L2-to-L3 Processing with MapReduce	- 61 -
8.3	Further Processes	- 66 -
A.	Requirements Traceability	- 68 -

1 Introduction

Calvalus is a technology study that investigates the use of Hadoop Map-Reduce massive parallel processing on distributed data for the improvement of the cal/val cycle of processor development and validation. It also supports the evaluation of the temporal stability of the instrument by processing large time series for trend analysis.

1.1 Purpose and Scope

This document is the technical specification of the Calvalus system. It is a deliverable in the ESA project 22869/09/NL/VS. As an answer to the system requirements [AD 1] the document integrates the Hadoop MapReduce engine and the Hadoop distributed file system into a system of services for cal/val data and processing management.

The focus is on a selected set of processing scenarios that are typical for the cal/val cycle of instrument and processor validation: L1-to-L2 processing of satellite data, L2-to-L3 processing, match-up against in-situ data, and trend analysis. The L1-to-L2 processing algorithm usually is the one to be validated and improved. The process is repeatedly applied during algorithm improvement, and it is computationally expensive and time-consuming if larger datasets are used.

This is the challenge of Calvalus. The approach is to use massive parallel processing on distributed data with Hadoop as cluster scheduler, and to use the Hadoop map-reduce programming model for distributed processing where appropriate.

Unique for the application scenario is that the satellite data are more or less constant and can be distributed to a file system prior to the repeated processing cycles in order to utilise data locality. Data locality – i.e. to process where the data reside – is part of the Hadoop approach. Design decisions for data organisation and parallelised processing are also driven by a concurrency trade-off analysis that is described in this document.

The document is going to show the design for this approach. The design comprises a complete and useful system. Nevertheless, the focus in the implementation phase will be the proof of concept, not the completely operational system. As also shown in the section on hardware massive parallel for this study means an exemplary cluster of 20 machines, which is sufficient to demonstrate the concept.

The level of detail in this technical specification is selected to provide an overview, to elaborate on the approach and the main ideas, and to sketch a decomposition into a functional architecture with all interfaces identified as a good starting point for implementation. The specification does not dive into details of the implementation but prefers to leave open decisions that do not touch the overall approach to the implementation phase.

1.2 Terms and Definitions

The following terms describe concepts used in this document with a specific meaning. Though the terms may be common in the earth observation domain some of them may be used differently in other documents and contexts.

Term	Definition
Product Files	EO data used for both, input and output of a Processing Step . A product may be represented by one or more physical files.

Term	Definition
Product File Set	In this document, a product set denotes a named collection of references to Product files .
Production Request	A request to the processing system to produce something. A production request operates on one or more Product Sets and may produce a new Product Set.
Production Request Template	An incomplete Production Request that users may use as prototype for new processing requests, e.g. by altering single parameter values.
Production Job	The server-side process as a result of an accepted Production Request . The production job therefore has a limited lifespan.
Production Scenario	An operational end-to-end scenario implemented into the Calvalus system. The core system contains production scenarios for L1-to-L2 Bulk Processing , L2-to-L3 Bulk Processing , for a Match-up Analysis and Trend Analysis .
Production Recipe	Describes a Production Scenario in terms of its input values (name, type, value range, units), its output (name, type) and the actual Processing Steps to be performed.
Production Step	A single step in a Production Scenario that usually operates on Product Sets .
Processing Step	An atomic transformation of an input into an output by a Processor that usually operates on single Products .
Processor	An implementation of an algorithm that transforms an input into an output.

1.3 Abbreviations

The following abbreviations are used within the document without repeated definition.

Abbrev.	Expansion
API	Application Programming Interface
BC	Brockmann Consult GmbH
BEAM	Basic ERS & Envisat (A)ATSR and Meris Toolbox
cal/val	calibration and validation
CCI	ESA Climate Change Initiative
CEOS	Committee on Earth Observation Satellites
CPU	Central Processing Unit
DEM	Digital Elevation Model
ECSS	European Cooperation for Space Standardization
EO	Earth Observation
ESA	European Space Agency
FRS	Full resolution swath
GAC	GKSS Atmospheric Correction
GAC QAA	A QAA (see below) using as input the output of GAC
GPF	Graph Processing Framework of BEAM
HDFS	Hadoop Distributed File System

Abbrev.	Expansion
I/O	Input/output
IOP	Inherent optical properties
L1, L2, L3	Level 1, Level 2, Level 3 data (CEOS classification)
LET-SME	Leading Edge Technology - Small and Medium-sized Enterprises
NASA	National Aeronautics and Space Administration
OBPG	Ocean Biology Processing Group
OGC	Open GIS Consortium
QAA	Quasi-Analytical Algorithm, see [RD-8]
RR	Reduced resolution
SoW	Statement of Work
UML	Unified Modelling Language
WPS	Web Processing Service OGC specification

1.4 References

The following documents are applicable to this document.

ID	Title	Issue	Date
[AD 1]	SME-LET 2009 - CalVal and User Services, Offer A3067 in response to Invitation To Tender/AO-1-6037/09/F/VS, LET-SME Announcement of opportunities 2009	1.0	15.09.2009
[AD 2]	Cal/Val and User Services – Calvados Requirements Baseline, ESA SME-LET AoO 2009, Brockmann Consult	1.2	30.06.2010

The following documents are referenced in this document.

ID	Title	Issue	Date
[RD 1]	Fomferra, N.: The BEAM 3 Architecture; http://www.brockmann-consult.de/beam/doc/BEAM-Architecture-1.2.pdf	1.2	
[RD 2]	Brockmann, C., Fomferra, N., Peters, M., Zühlke, M., Regner, P., Doerffer, R.: A Programming Environment for Prototyping New Algorithms for AATSR and MERIS – iBEAM; in: Proceedings of ENVISAT Symposium 2007, ESRIN Frascati, Italy		2007
[RD 3]	Fomferra, N., Brockmann C. and Regner, P.: BEAM - the ENVISAT MERIS and AATSR Toolbox; in: Proceedings of the MERIS-AATSR Workshop 2005, ESRIN Frascati, Italy		2005
[RD 4]	Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung: The Google File System; in: 19th ACM Symposium on Operating Systems Principles, Lake George, NY (http://labs.google.com/papers/gfs.html)		Oct. 2003

[RD 5]	Jeffrey Dean and Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters; OSDI'04: Sixth Symposium on Operating System Design and Implementation; San Francisco, CA (http://labs.google.com/papers/mapreduce.html)		Dec. 2004
[RD 6]	Ariel Cary, Zhengguo Sun, Vagelis Hristidis, Naphtali Rish: Experiences on Processing Spatial Data with MapReduce; Lecture Notes In Computer Science; Vol. 5566 - Proceedings of the 21st International Conference on Scientific and Statistical Database Management - New Orleans, LA, USA (http://users.cis.fiu.edu/~vagelis/publications/Spatial-MapReduce-SSDBM2009.pdf)		2009
[RD 7]	R. Doerffer, H. Schiller: MERIS Lake Water Algorithm for BEAM and MERIS Regional Coastal and Lake Case 2 Water Project, Atmospheric Correction ATBD; ESRIN Contract No. 20436		June 2008
[RD 8]	Zhong Ping Lee, Kendall L. Carder, and Robert A. Arnone: Deriving inherent optical properties from water color: A multiband quasi-analytical algorithm for optically deep waters; APPLIED OPTICS, Vol.41, No.27		20.09.2002
[RD 9]	Bryan A. Franz, Sean W. Bailey, P. Jeremy Werdell, and Charles R. McClain: Sensor-independent approach to the vicarious calibration of satellite ocean color radiometry; APPLIED OPTICS Vol.46, No.22, 1		Aug. 2007
[RD 10]	Bryan Franz: Methods for Assessing the Quality and Consistency of Ocean Color Products; NASA Goddard Space Flight Center, Ocean Biology Processing Group http://oceancolor.gsfc.nasa.gov/DOCS/methods/sensor_analysis_methods.html		18.01.2005
[RD 11]	Janet W. Campbell, John M. Blaisdell, Michael Darzi: Level-3 SeaWiFS Data Products: Spatial and Temporal Binning Algorithms; SeaWiFS Technical Report Series, NASA Technical Memorandum 104566, Vol. 32		Aug. 1995
[RD 12]	K. Barker et al: MERMAID: The MERIS MATchup In-situ Database; ARGANS Limited (http://hermes.acri.fr/mermaid/doc/Barker-et-al-2008_MERMAID.pdf)		2008
[RD 13]	NASA OBPB: Ocean Color Level 3 Binned Products (http://oceancolor.gsfc.nasa.gov/DOCS/Ocean_Level-3_Binned_Data_Products.pdf)		
[RD 14]	CoastColour web site (http://www.coastcolour.org/)		
[RD 15]	ECSS-E-ST-40C ECSS Space Engineering - Software, European Cooperation for Space Standardization, ESA-ESTEC, Noordwijk, The Netherlands	C	06.03.2009
[RD 16]	Bryan Franz: OBPB l2gen User's Guide; (http://oceancolor.gsfc.nasa.gov/seadas/doc/l2gen/l2gen.html)		
[RD 17]	Web site of the ESA Climate Change Initiative (http://earth.eo.esa.int/workshops/esa_cci/intro.html)		
[RD 18]	OGC Web Processing Service Specification	1.0	08.06.2007

	(http://www.opengeospatial.org/standards/wps)		
[RD 19]	Case2R source code repository at https://github.com/bcdev/beam-meris-case2	1.4.3	ongoing
[RD 20]	QAA source code repository at https://github.com/bcdev/beam-meris-qaa	1.0.2	ongoing
[RD 21]	BEAM user manual (http://www.brockmann-consult.de/beam)	4.8	ongoing

1.5 Document Overview

After this formal introduction

- Chapter 2 introduces into the approach of using **Hadoop in the cal/val application domain**
- Chapter 3 describes operational scenarios to be implemented by the system, among them four typical **production scenarios** for cal/val and the **system use cases**. It also defines the **amount and selection of EO and reference data** to be used in the prototype.
- Chapter 4 defines the **hardware environment** of the Calvalus prototype cluster
- Chapter 5 describes the results of a **trade-off analysis** on data organisation on HDFS and parallel processing in Hadoop, to learn about the price of concurrency in form of overhead and to guide design decisions
- Chapter 6 describes the static architecture of the Calvalus system with its logical **decomposition** into components and with their **functional description**
- Chapter 7 identifies the **external and internal interfaces** of Calvalus and defines important structures and protocols of these interfaces
- Chapter 8 describes the **design of processing workflows** based on Hadoop parallelisation and map-reduce

Finally, Appendix A traces system requirements to the design in this document.

2 Overview

This chapter introduces into the approach of using Hadoop in the cal/val application domain. It explains main concepts of Hadoop MapReduce and the Hadoop distributed file system, describes the approach of this study to apply this in the cal/val domain, and defines the system context of the prototype to be developed.

2.1 Cal/Val Application Domain

Calibration of the measured EO sensor signal and validation of the derived data products is an extremely important task for efficient exploitation of the EO data and the basis for reliable scientific conclusions. In spite of this importance, the cal/val work is often hindered by insufficient means to access data, time consuming work to identify suitable in-situ data, matching the EO data, incompatible software and no possibility for rapid prototyping and testing of ideas.

The goal of Calvalus is to apply a leading-edge technology to develop an efficient processing and analysis system for EO satellite data. The technology manages massive EO datasets and will provide a large-scale, efficient, rapid, on-the-fly processing power to test concepts concerning instrument on-orbit characterization and its science algorithms. The instant feedback on the ideas will enable rapid prototyping and idea transfer to operations.

In Figure 2-1 the development cycle is shown, that is run through in order to improve the quality and consistency of EO data products.

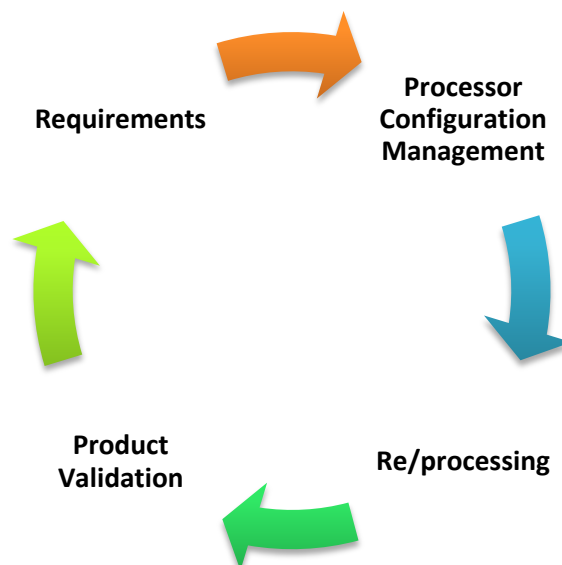


Figure 2-1: Development cycle for the improvement of EO data products

Requirements on higher level and value-added products originate from their user communities. They drive the initial development of instruments and of algorithms capable of producing the required data products. The products are generated by data processors which implement the algorithms that are used to transform the lower level (L1) input data to the higher level products. The resulting L2 and L3 products are then subject to validation, usually performed by scientists. The specific cal/val analyses include the processing of extracts corresponding to match-ups with ground observations and vicarious calibration sites as well as the processing of mission-long sampled global or regional

time series to evaluate the quality of long-term data records. Typically, the validation results generate new requirements on the processor configuration management, for example,

- modified parameterisation in terms of processing parameters and auxiliary data,
- algorithm (science code) adjustments,
- and Implementation of new algorithms

The ultimate goal of Calvalus is to accelerate the development cycles by allowing users to perform repeated processing of the same primary inputs with different algorithms or parameters and performing automated analyses on the resulting dataset. The validation activities such as

- comparison with reference data,
- inter-comparison with other sensors
- and detection of trends and anomalies

is supported by two automated standard analyses, namely the match-up and trend analyses.

The Calvalus study is envisioned to assist instrument quality working groups, validation teams, and ESA projects such as CoastColour [RD 14] and Climate Change Initiative (ocean_colour_cci) [RD 17]. The study will build a prototype of the system and demonstrate it on MERIS L2 and L3 processing and analyses.

2.2 Hadoop Distributed Computing

The basis of the Calvalus processing system is **Apache Hadoop**. Hadoop is an industry proven open-source software capable of running clusters of tens to ten thousands of computers and processing ultra large amounts of data based on massive parallelisation and a distributed file system.

2.2.1 Distributed File System (DFS)

In opposite to a local file system, the Network File System (NFS) or the Common Internet File System (CIFS), a distributed file system (DFS) uses multiple nodes in a cluster to store the files and data resources. A DFS usually accounts for transparent file replication and fault tolerance and furthermore enables data locality for processing tasks. Figure 2-2 shows the distribution and replication of a file splitted into several blocks.

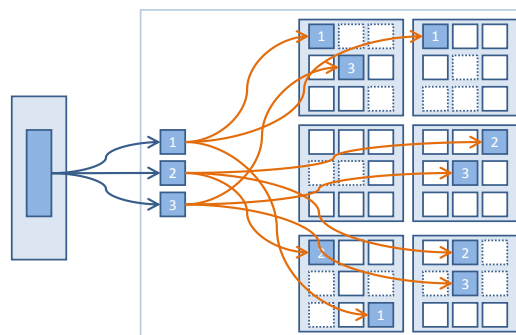


Figure 2-2: File blocks, distribution and replication in a distributed file system

Figure 2-3 demonstrates how the file system handles node-failure by automated recovery of under-replicated blocks. HDFS further uses checksums to verify block integrity. As long as there is at least one integer and accessible copy of a block it can automatically re-replicated to return to the requested replication rate.

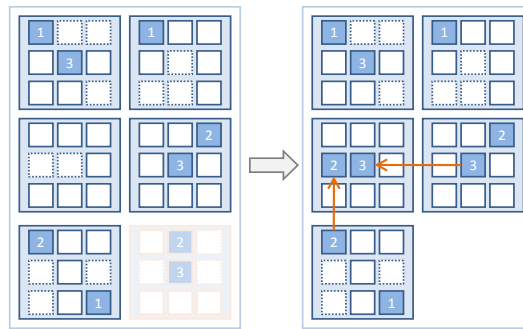


Figure 2-3: Automatic repair in case of cluster node failure by additional replication

Figure 2-4 shows how a distributed file system re-assembles blocks to retrieve the complete file for external retrieval.

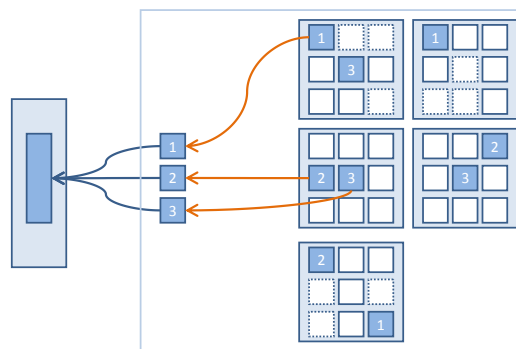


Figure 2-4: Block assembly for data retrieval from the distributed file system

2.2.2 Data Locality

Data processing systems that need to read and write large amounts of data perform best if the data I/O takes place on local storage devices. In clusters, where storage nodes are separated from compute nodes two situations are likely:

1. Network bandwidth is the bottleneck, especially when multiple tasks are working in parallel on the same input data but different compute nodes and/or storage nodes are separated from compute nodes.
2. Transfer rates of the local hard drives are the bottleneck, especially when multiple tasks are working in parallel on single (multi-CPU, multi-core) compute nodes.

A possible solution for these problems is to first use a cluster whose nodes are both, compute and storage nodes. Secondly, processing tasks will be distributed and executed on nodes that are “close” to the data, with respect to the used network topology (see Figure 2-5). By using a DFS, where large files are split into blocks and blocks are replicated over multiple nodes, input data can be processed on a number of nodes in parallel.

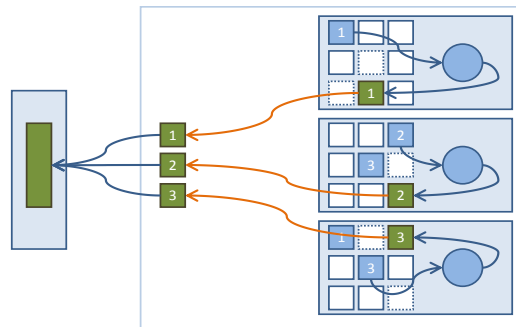


Figure 2-5: Data-local processing and result assembly for retrieval

2.2.3 MapReduce Programming Model

The MapReduce programming model has been published in 2004 by the two Google scientists J. Dean and S. Ghemawat. It is used for processing and generation of huge datasets on certain kinds of distributable problems using cluster. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model. Programs written in this functional style are automatically parallelized.

2.2.4 Hadoop

Apache Hadoop is a software framework that supports data-intensive distributed applications. It enables applications to work with thousands of computers (nodes), collectively referred to as a cluster, and petabytes of data. Hadoop was inspired by Google's MapReduce and Google File System (GFS) papers.



The real benefit lies in the combination of the distributed file system and the MapReduce programming model. When a new job has been submitted on the cluster, first the distribution of the input data is evaluated. The logical parts of an input file are called splits and typically a split has the size of the blocks that store the data, but in contrast to a block which ends at arbitrary byte positions, a split always includes complete key/value pairs.

For each split a map task is scheduled. The Hadoop system tries to execute the mapper on nodes which have splits stored locally. Only when this is not possible the execution happens on another node to which the data has to be transferred over the network.

A single instance of the map function is invoked with all key/values pairs belonging to a split and can itself emit any number of key/value pairs. These pairs can have different types than the input pairs. The outputs of the map task are written as intermediate output to the respective local disks. If only one reduce task is executed all emitted pairs will be sorted by key and transferred to the node where the reduce task runs. There the key/value pairs from all map task are merged and fed to the reduce function. When multiple instances of the reduce task are executed the keys are divided into separate partitions. By default a hash function is used but a special implementation can be provided. The output of the reduce task is again stored in the HDFS. When multiple reduce tasks are executed each reduce task writes a part of the result. They have to be merged afterwards.

Tasks that fail or not respond within a given time period are executed again on another node before the whole job fails. Optionally tasks are speculatively executed a second time on idle nodes. The first

returning note contributes the result. This should prevent slower nodes from slowing down the whole cluster.

2.3 Calvalus Approach for Concurrent Processing

Hadoop map-reduce has been designed for efficient massive-parallel processing. But is this immediately applicable to EO data processing and does it help to solve a problem in this domain?

One of the salient challenges in EO is the large amount of data. Due to its processing power is a bottleneck in some cases and network transfer in others. In architectures with a central archive processing from archive involves transfer of all data for processing, especially if processing itself is distributed for parallelisation. The data is not local to the location of the algorithm, known as the data locality problem.

The theses of this study are:

- Bulk reprocessing of large product file sets on a Hadoop cluster is efficient and reliable.
- On-the-fly processing of a single product file can be parallelised on the cluster by splitting.
- L2-to-L3 processing can be parallelised by inputs and by geographic partitioning with the map-reduce approach

The selected sub-domain for this study is cal/val and its L1-to-L2 processing algorithm development and validation cycle. It is a computationally challenging problem to minimise the validation time for large product file sets, e.g. one year of satellite data. For computationally expensive algorithms this can only be achieved through parallelisation.

The L1-to-L2 processing itself is directly parallelisable due to its independence on the level of input products. The approach for this class of input-to-output processing is to use simple map-only workflows. Map-reduce is used for sorting-type steps like L2-to-L3 processing. It includes geographic sorting and spatio-temporal aggregation. Section 8 describes the process implementations in more detail.

2.4 Prototype System Context

The prototype system to be developed has the purpose to demonstrate parallel processing for the cal/val scenarios and its use via a portal. Figure 2-6 shows Calvalus with user portal and web services as front-end and the Hadoop cluster for processing and distributed data storage as back-end.

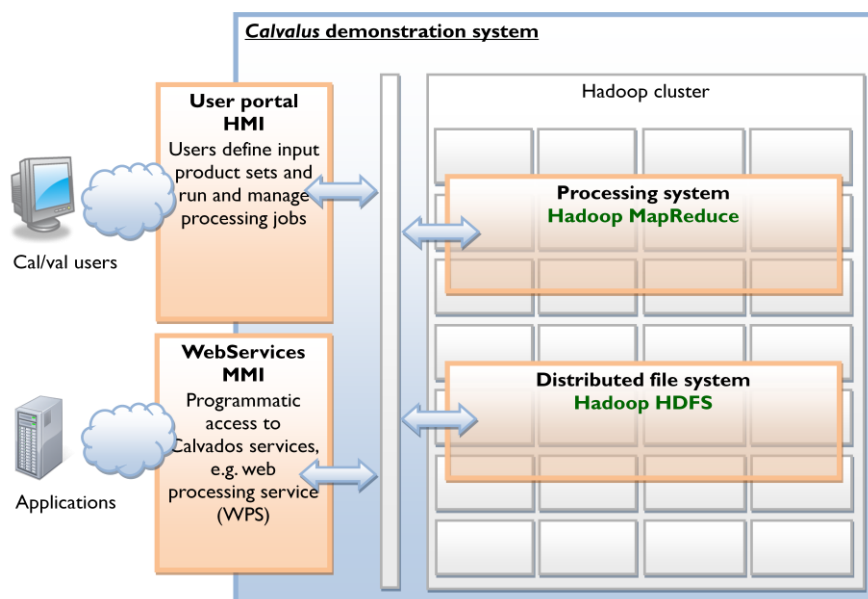


Figure 2-6: Hadoop cluster services and user interfaces in Calvalus

To demonstrate parallel processing and its usability the Calvalus prototype will implement

- a portal as user interface
- a set of services for data and processing management
- some existing, well-known L2 code as Hadoop-driven parallelized processors
- aggregation and analysis functions

In favour of the focus on parallel processing, other functions may be simplified (catalogue query, metadata schemata, online data access, web service interfaces) re-used from existing components, or implemented by simple shell scripts to be used by an operator (data ingestion). So, readers should not expect the Calvalus implementation to cover all functions described in this design. The technical specification nevertheless describes a complete view as this is clearer and easier to understand.

Figure 2-7 shows the Calvalus system in its context between user, EO data processor and operator.

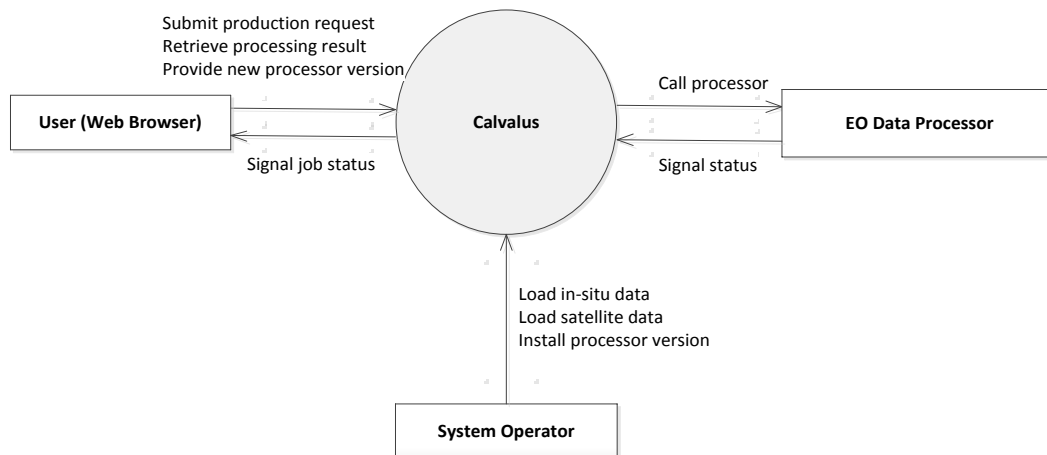


Figure 2-7: Calvalus system context

The next chapter on operational scenarios elaborates this in more detail.

3 Operational Scenarios

This chapter describes operational scenarios to be implemented by the system. Four typical production scenarios from the cal/val processor validation cycle are in the focus. In addition, system use cases from the user's point of view are defined.

3.1 Production Scenarios

The Calvalus processing system will have to realise at least four important scenarios triggered by EO and Cal/Val users. They are:

1. **Match-up Analysis** on water leaving reflectances, IOPs, and chlorophyll,
2. **Trend Analysis** on water leaving reflectances and chlorophyll,
3. **L1 to L2 Bulk-Processing** from L1b radiances to water leaving reflectances, IOPs, and chlorophyll,
4. **L1/L2 to L3 Bulk-Processing** from L1/L2 data to their temporal statistics.

As described in more detail in the following, the matchup analysis compares static reference data with L2 data that may be static or processed on the fly from L1b data. The trend analysis generates time-series from spatially and temporally integrated L2 data which also may be static or processed on the fly starting directly from L1b or from L2 data. The matchup and trend analyses produce comprehensive reports including diagrams and data tables.

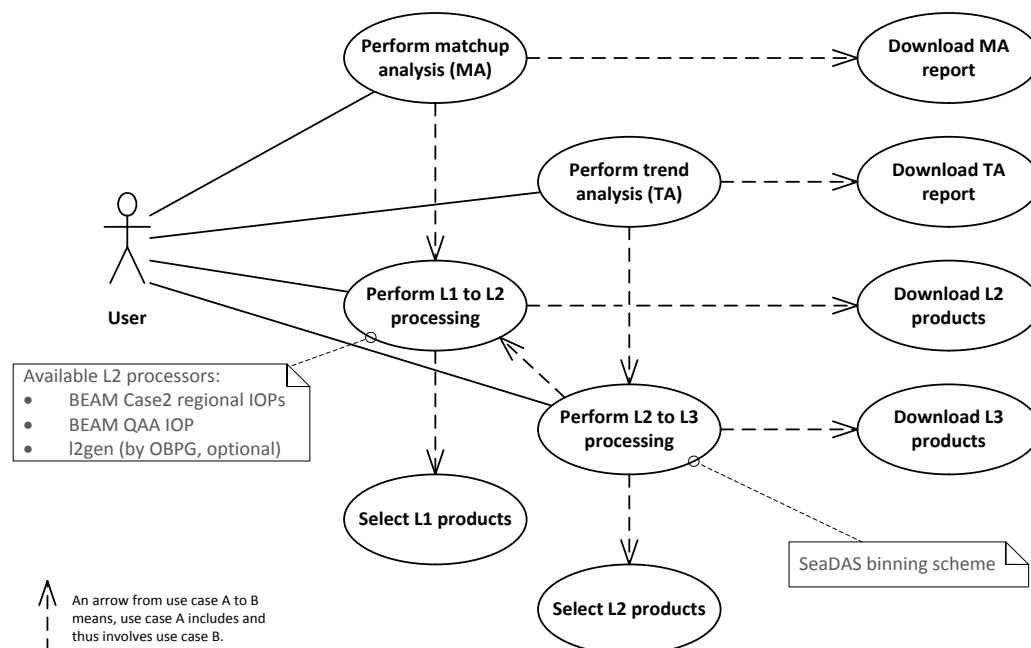


Figure 3-1: Top-level use cases

As shown in the use case diagram in Figure 3-1, the efficient generation of L2 and L3 data are important scenarios on their own. Users can select a number (or all) L1b data products and bulk-process them to L2 and L3 and finally download the generated data.

The major aim and the most challenging task of the Calvalus project is to implement an efficient processing system. We describe each of the production scenarios as control and data flows seen from point of view of the processing system. The flows are modelled as UML activity diagrams.

3.1.1 Scenario 1: Match-up Analysis

The match-up analysis is used to assess the quality of geophysical quantities retrieved from remote sensing data by comparing them with ground-truth measurements (in-situ data). The term match-up refers to data from one or more satellites and one or more in-situ measurements coincident in at least one variable and in space and time (1 match-up = dataset comprising all in-situ and satellite data at given phi, lam, t). The methodology used in Calvalus is based on the one used by the NASA OBPG [RD 10]. Besides in-situ data, Calvalus will also use other reference data for the match-up analysis. These are L2 data spatially and temporally related to the in-situ measurements from other sensors such as SeaWiFS and MODIS as well as L2 data of the MERIS standard products (processed by MEGS). In contrast to other systems that provide match-up data, Calvalus is capable of processing L1 data to L2 on-the-fly. Therefore, the match-up analysis will be the primary mechanism for Calvalus users to change the L2 processing parameters and/or processor algorithms and directly receive feedback in the form of an analysis report.

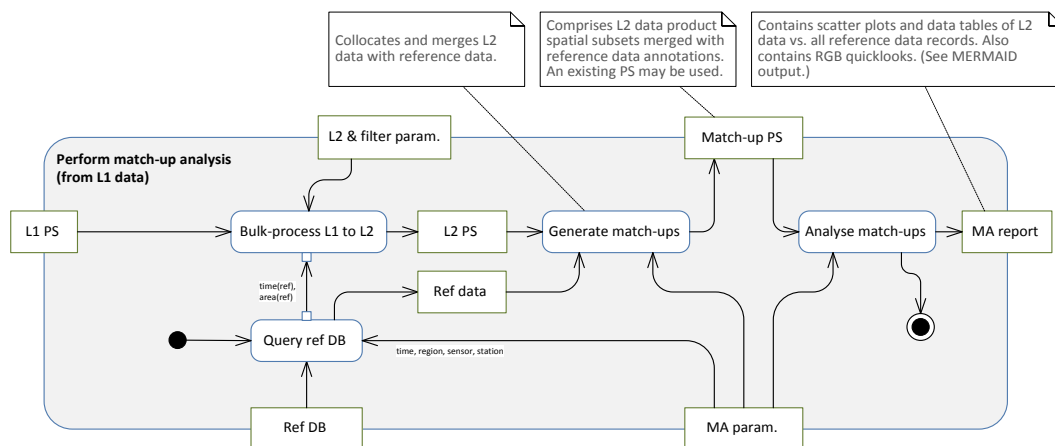


Figure 3-2: Match-up analysis from L1 data

The match-up analysis (Figure 3-2) starts with a **query** on the database of reference data. The query is defined by the user-defined match-up analysis parameters (MA param.) and may include time, region and sensor name. The query's result set defines an actual time range and geographic area which will be used to filter and clip the set of L1 data products (product set, PS) and finally process it to L2.

Filtering, clipping and processing to L2 is contained in the **L2 bulk-processing** action, a self-standing production scenario which is described in more detail below. Other data product filter parameters may be supplied by the user-defined L2 parameters, for example, a user might wish to process only scenes free of sun-glint.

The bulk processing results into an intermediate L2 product set that will be collocated and merged in **match-up generation**. The result of this step is a set of match-up data products, which is basically the clipped L2 data product accompanied with the collocated reference data. The set of match-up data products is input to the actual **match-up analysis** which generates the final analysis report. The report comprises a number of scatter plots, CSV data tables, summary statistics (min, max, stdev, mean, median) and RGB quick looks very similar to the output of the MERMAID database [RD 12].

A match-up analysis may also be performed on an existing L2 product set (Figure 3-3). This is especially useful when the need is to regenerate match-up files or to pre-filter the existing L2 product set. The L2 bulk-production step is now replaced by a filter and a subsequent clipping both still operating on entire product sets.

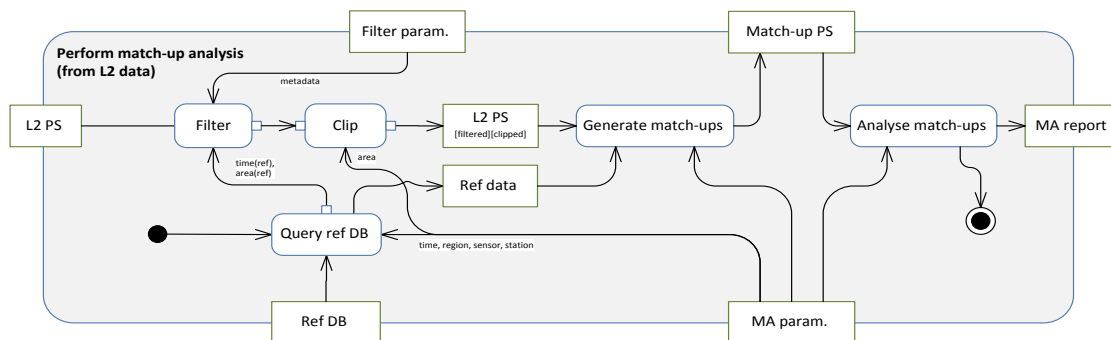


Figure 3-3: Match-up analysis from existing L2 data

Finally, the match-up analysis step may simply be applied to an existing set of match-up data products (Figure 3-4) in order to regenerate the report with different analysis parameters, e.g. by using a different pixel selection criteria or macro-pixel size.

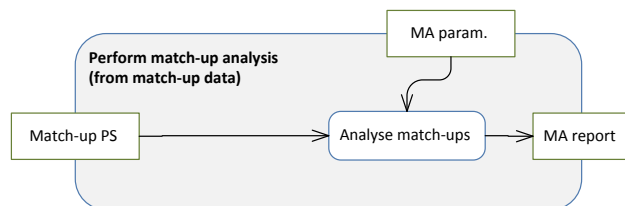


Figure 3-4: Match-up analysis from match-up data

3.1.2 Scenario 2: Trend Analysis

A trend analysis is used to assess the product consistency and sensor stability. Long-term time-series of L3 water leaving reflectances are used to indicate the trends. The time-series are generated by generating L3 composites from all L2 data of the first n days (n is usually 4 or 8) of consecutive m -day periods (usually a month or 32 day period).

The analysis is performed on global and regional spatial scales. Regions are usually selected due to their marine conditions, e.g. deep waters and oligotrophic, mesotrophic, eutrophic waters.

A usual trend analysis (Figure 3-5) starts from a set of L1 data products which is first processed up to a set of L3 data products. As the L2 bulk-processing, the **L3 bulk-processing** is considered a self-standing production scenario and described in more detail below. The final **trend analysis** takes the L3 product set as input and generates time-series plots, CSV data tables and summary statistics for each of the marine conditions passed in as parameter. The L2 and L3 product sets that have been generated for the analysis may be kept for later use.

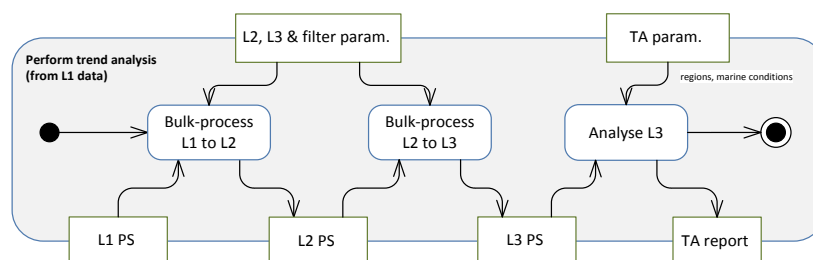


Figure 3-5: Trend analysis from L1 data

As shown in the following two diagrams Figure 3-6 and Figure 3-7, the trend analysis may also be started directly from L2 and L3 data. In the latter case, the L3 bulk-processing is replaced by a simple filter that is used to select a user-specified subset of the input product set for the analysis.

L3 analyses will include long-term temporal trending, temporal anomaly analyses, trend comparisons between different versions and types of MERIS processors and MERIS inter-comparisons with other missions, such as SeaWiFS and MODIS [RD-10].

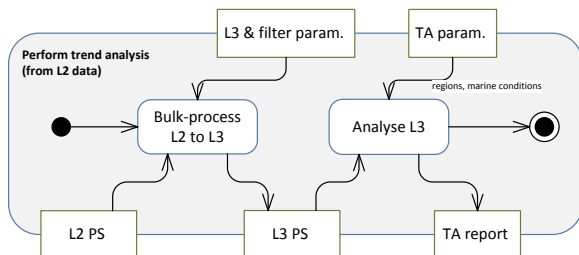


Figure 3-6: Trend analysis from L2 data

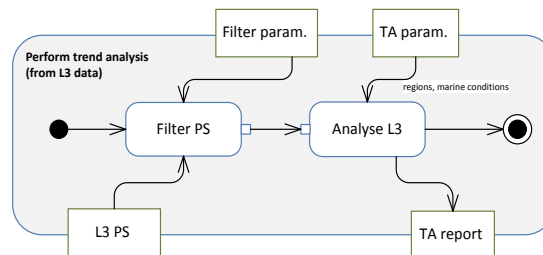


Figure 3-7: Trend analysis from L3 data

3.1.3 Scenario 3: Level 2 Processing

In this scenario, a set of L1 data products is bulk-processed to L2 as shown in Figure 3-8. The L2 processing parameters may contain product selection criteria which are used to filter the input L1 product set. It may also contain an area of interest which is used to perform a geometric clipping on the input data.

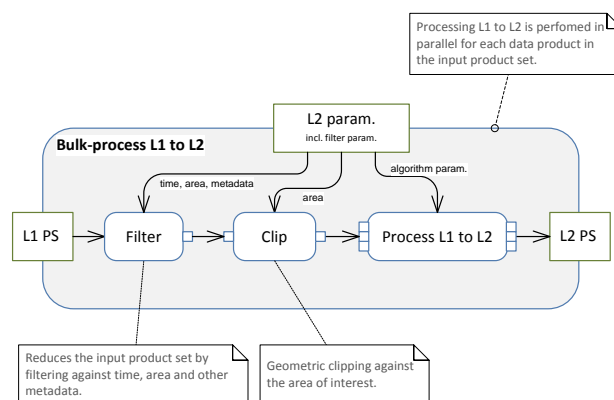


Figure 3-8: L1 to L2 processing

The Calvalus system offers at least two different L2 processors to be used, namely the *Case2R* processor [RD 7] and the *GAC QAA* processor [RD 8]. Both output a comparable set of IOPs and normalised water-leaving reflectances.

The QAA algorithm has originally been developed to operate on MERIS L2 reflectances. In the Calvalus context, the QAA gets its input from GAC, the “GKSS Atmospheric Correction” which uses a neural network approach to compute bottom-of-atmosphere reflectances from MERIS L1b top-of-atmosphere radiances (→ GAC QAA). The same is true for the Case2R processor.

The Case2R and the GAC QAA processors are available as BEAM processor plug-ins and are documented in the BEAM user documentation [RD 18].

3.1.3.1 GAC Processing (Atmospheric Correction)

For open ocean water robust methods have been developed for the correction of the atmospheric influence as well as the retrieval of phytoplankton chlorophyll. This is not so the case for some types of coastal waters as well as inland waters, where the case 1 water routines for the atmospheric correction and the retrieval of water constituents fail. Also the standard extension of the atmospheric correction to turbid coastal waters fails in some areas so that the case 2 water algorithm

for the retrieval cannot be applied. These areas have to be flagged and thus are lost for further use. [RD-7] describes the atmospheric correction procedure, which is used to calculate water leaving reflectances from top of atmosphere radiances. One requirement for this procedure is to include turbid case 2 waters into the scope of the algorithm. The standard MERIS atmospheric correction procedure for case 2 water, as implemented in the ground segment, is adapted to waters with only limited concentrations of suspended solids and yellow substances and leads under other cases to negative reflectances and artefacts. Thus, it was necessary to write a new procedure for BEAM. This opportunity was used to develop and test a new type of atmospheric correction method which is based on inverse modelling and its parameterisation by a neural network. It takes into account the effect of cirrus clouds, specularly reflected sun light (sun glint) and the water leaving radiance reflectances caused by all sort of water constituents.

The GAC is currently part of the Case2R processor. Details of the GAC algorithm are provided in [RD 7] and [RD 19].

3.1.3.2 GAC Output Product

GAC specific geophysical variables and datasets:

Name	Description
toa_reflec_	SMILE-corrected TOA radiance reflectance in band number =1...15
reflec_	Fully normalised water leaving radiance reflectance in band number =1...10, 12, 13
l1_flags	Quality flags dataset from L1b product
l2_flags	Quality flags dataset generated during GAC processing

GAC specific flag coding of the **l2_flags** dataset:

Bit#	Name	Description
1	RAD_ERR	Top-of-atmosphere radiance out of valid range
2	LAND	land pixels
3	CLOUD_ICE	cloud or ice
4	SUNGLINT	sunlint risk
5	ANCIL	missing or out-of range auxiliary data
6	TOSA_OOR	Top-of-atmosphere reflectance out of range
7	WLR_OOR	Water leaving reflectance out of range
8	SOLZEN	solar zenith angle high
9	SATZEN	satellite zenith angle high
10	ATC_OOR	atmospheric correction out of range
11	-	Unused
12	OOTR	Water leaving reflectance out of training range
13	WHITECAPS	Whitecaps pixels

14	-	Unused
15	-	Unused
16	-	Unused
17	INVALID	Pixel not valid (LAND and CLOUD_ICE and l1_flags.INVALID)

3.1.3.3 GAC Processing Parameters

The Case2R processing parameters are given in the following table.

Parameter	Default Value	Description
atmCorrNn	"25x30x35x40_4016.9.net"	Paths and names of the neural network files
polCorrNn	"18_518.1.netPolEffekt"	Paths and names of the neural network files
inputValidMask	"not l1_flags.INVALID"	An arithmetic expression which defines the valid pixels for the processing.
landWaterSeparationExpression	"toa_reflec_10 > toa_reflec_6 AND toa_reflec_13 > 0.0475"	Expressions used to separate land from water. This is the default value. (More effective expressions may be specified from more advanced classification products.)
cloudIceDetectionExpression	"toa_reflec_14 > 0.2"	Default expressions used to separate cloud and ice. This is the default value. (More effective expressions may be specified from more advanced classification products.)
performSmileCorrection	true	Switch the smile correction on and off.
performPolCorr	false	Controls if the polarisation correction should be included in the atmospheric correction procedure.

3.1.3.4 Case2R Processing

The Case2R Level 2 processor has been developed as a joint effort between GKSS Research Centre, Institute for Coastal Research, and Brockmann Consult, under contract of the European Space Agency ESA. Co-investigators in form of sub-contractors are the Norwegian Institute for Water Research (NIVA) and the Institute of Ocean Sciences (IOS), BC, Canada.

The processor produces Level 2 data from MERIS Level 1b data for regional coastal waters. This includes an algorithm for atmospheric correction (GAC, see above) and an algorithm to derive properties of the water such as the inherent optical properties (IOP's) absorption and scattering and the concentrations of total suspended matter (TSM) and chlorophyll from the water leaving radiance reflectance spectra (i.e. after atmospheric correction).

Input to the Case2R algorithms is the water leaving reflectances of 8 MERIS bands. These data are the output of the atmospheric correction [RD-7]. The algorithms derive data of the inherent optical properties total scattering of particles (total suspended matter, tsm) **b_tsm**, the absorption coefficient of phytoplankton pigments **a_pig** and the absorption of dissolved organic matter **a_gelb** (gelbstoff), all at 443 nm (MERIS band 2). From these IOPs the concentrations of phytoplankton chlorophyll and of total suspended dry weight are determined. Furthermore, the attenuation coefficient for the downwelling irradiance, k , at the wavelength with maximum transparency **k_min** is determined as well as the **z90** signal depth, which indicates the water depth from which 90% of the reflected light comes from. The algorithm is based on a neural network (NN), which relates the bi-directional water leaving radiance reflectances with these concentration variables. The network is trained with simulated reflectances. The bio-optical model used for the simulations is based on a data set collected in different lakes in Finland and Spain. Two NN's are trained with simulated reflectances:

- Inverse NN to emulate the inverse model (reflectances, geometry) \rightarrow concentrations, and
- Forward NN to emulate the forward model (concentrations, geometry) \rightarrow reflectances.

The inverse NN is used to obtain an estimate of the concentrations which is used as a first guess to start a minimization procedure, which uses the forward NN iteratively to minimize the difference between the calculated reflectances and the measured ones. The procedure is fast as it takes advantage of the Jacobian, which is a byproduct of the NN calculation.

Details of the algorithm for the retrieval of water constituents are provided in [RD 7] and [RD 19].

3.1.3.5 Case2R Output Product

Case2R specific geophysical variables and datasets:

Name	Description
reflec_	From GAC: Water leaving radiance reflectance in band number =1...10, 12, 13
a_pig	Absorption coefficient at 443 nm of phytoplankton pigments
a_gelb	Absorption coefficient at 443 nm of yellow substance and of all particles after
b_tsm	Scattering coefficient of all particles at 443 nm
a_total	Total absorption of water constituents at 443 nm
k_min	Down-welling irradiance attenuation coefficient
z90_max	Inverted value of k_{min}
chl_conc	Chlorophyll concentration (mg m^{-3}). $\text{chl_conc} = \text{chlConversionFactor} \cdot \text{a_pig}^{\text{chlConversionExponent}}$
tsm_conc	Total suspended matter dry weight (g m^{-3}). $\text{tsm_conc} = \text{tsmConversionFactor} \cdot \text{b_tsm}^{\text{tsmConversionExponent}}$
ang_443_865	Total suspended matter dry weight (g m^{-3})
ang_443_865	Angstrom coefficient
tau_550	Spectral aerosol optical depth
chi2	A low value in the product indicates a higher success in the retrieval and that the conditions, which have led to the measured spectrum, are in (sufficient) agreement with the conditions and the bio-optical model used in the simulations for training the neural network. A value above a threshold of <code>spectrumOutOfScopeThreshold</code>

(default is 4.0) triggers the out of training range == out of scope flag.

l1_flags Quality flags dataset from L1b product

l2_flags Quality flags dataset generated during GAC and Case2R processing

Flag coding for the **l2_flags** dataset:

Bit#	Name	Description
1	RAD_ERR	TOAR out of valid range
2	LAND	land pixels
3	CLOUD_ICE	cloud or ice
4	SUNGLINT	sunglint risk
5	ANCIL	missing/OOR auxiliary data
6	TOSA_OOR	TOSA out of range
7	WLR_OOR	WLR out of scope
8	SOLZEN	large solar zenith angle
9	SATZEN	large spacecraft zenith angle
10	ATC_OOR	atmos. correct. out of range
11	CONC_OOR	concentration out of training range
12	OOTR	RLw out of training range
13	WHITECAPS	Whitecaps pixels
14	FIT_FAILED	fit failed
15	SPAREFLAG06	spare flag 06
16	SPAREFLAG07	spare flag 07
17	INVALID	not valid

3.1.3.6 Case2R Processing Parameters

Parameter	Default Value	Description
waterNnInverse	"meris_bn_20040322_45x16x12x8x5_5177.9.net"	Pathname of the inverse IOP neural network file
waterNnForward	"meris_fn_20040319_15x15x15_1750.4.net"	Pathname of the forward IOP neural network file
tsmConversionFactor	1.73	Factor and ...
tsmConversionExponent	1.0	... exponent for converting particle scattering into total suspended matter dry weight.
chlConversionFactor	21.0	Factors and ...
chlConversionExponent	1.04	... exponent for converting absorption of phytoplankton pigment at 443 into chlorophyll concentration.

radiance1AdjustmentFactor	1.0	Factor for the adjustment of TOA reflectance MERIS band 1 (412 nm).
spectrumOutOfScopeThreshold	4.0	The out of scope (= out of training range) flag will be set if the chi2 value is above the given threshold.

3.1.3.7 QAA Processing

Details of the QAA algorithm can be found in [RD 8] and [RD 21].

3.1.3.8 Processing Output Product

QAA geophysical variables:

Name	Description
qaa_a_<λ>	Total absorption coefficient in band <λ> (1/m), where <λ> is one of the spectral wavelengths 412, 443, 490, 510, 560, 620 nm
qaa_bb_<λ>	Backscattering coefficient b_b in band <λ> (1/m), where <λ> is one of the spectral wavelengths 412, 443, 490, 510, 560, 620 nm
qaa_aph_<λ>	Absorption coefficient a_{ph} of phytoplankton in band <λ> (1/m), where <λ> is one of the spectral wavelengths 412, 443, 490, 510, 560, 620 nm
qaa_adg_<λ>	Absorption coefficient a_{dg} of detritus and gelbstoff in band <λ> (1/m), where <λ> is one of the spectral wavelengths 412, 443, 490, 510, 560, 620 nm
analytical_flags	Quality flags dataset. Description provided below.
l2_flags	Quality flags dataset generated during GAC processing .Copied from GAC. Description provided above.

Flag coding for **analytical_flags** dataset:

Bit #	Name	Description
1	NORMAL	A valid water pixel.
2	IMAGINARY	Classified as water, but an imaginary number would have been produced.
3	NEGATIVE_ADG	Classified as water, but one or more of the bands contain a negative a_{dg} value.
4	NON_WATER	Not classified as a water pixel (land/cloud).

3.1.3.9 QAA Processing Parameters

Parameter	Default value	Description
aLowerBound	-0.02	Lower bound of a (1/m)
aUpperBound	5.0	Upper bound of a (1/m)
bbLowerBound	-0.02	Lower bound of b_b (1/m)
bbUpperBound	5.0	Upper bound of b_b (1/m)
aphLowerBound	-0.02	Lower bound of a_{ph} (1/m)

aphUpperBound	3.0	Upper bound of a_{ph} (1/m)
adgUpperBound	1.0	Upper bound of a_{dg} (1/m)
divideByPI	true	Divide source reflectances by π

3.1.3.10 Other L2 Processors

During Calvalus feasibility studies, the integration of the *SeaDAS l2gen* L2 processor [RD 16] has been evaluated. The tests were very promising, so that best efforts will be made to fully integrate l2gen as a possible L2 processor in Calvalus.

3.1.4 Scenario 4: Level 3 Processing

Due to its generic nature, L3 data can be generated either from L2 data (or from L1 data if some normalisation is applied to it before binning). Calvalus L3 data products are composites computed by temporally aggregating L1/L2 pixels into spatial equal-area bins (binning). The bin cells in a L3 product are arranged in an integerised, sinusoidal grid (ISIN) which is compatible to the one used for other ESA MERIS L3 products [RD 13] and the MODIS L3 products generated by the NASA OBPG [RD 10]. The complete description of the L3 generation methodology can be found in [RD 11]. Figure 3-9 depicts the processing scenario.

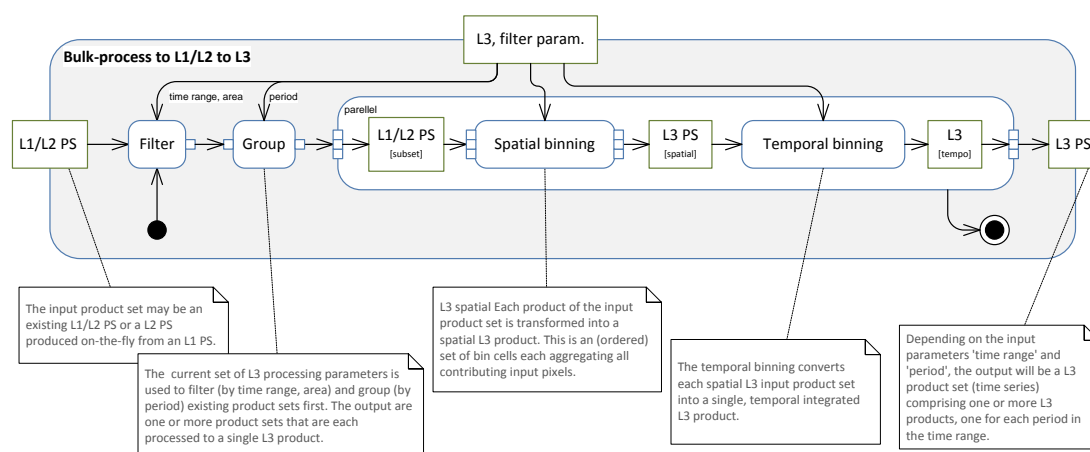


Figure 3-9: L1/L2 to L3 processing

Input to the L3 bulk-processing is a set of either L1 or L2 data products. The L3 processing parameters include the geophysical variables to be “binned”, the time range of products to be considered, the aggregation period, an area of interest, pixel masks and finally the bin size (in km). The temporal and regional constraints are used to **filter** in the input product set. A subsequent operation **groups** the input product set into a number of smaller product sets according to the aggregation period. Each of the smaller L1/L2 product sets are then binned to a single L3 product.

The binning process is twofold: First, to each product of the input product set a **spatial binning** is applied in order to generate an intermediate, spatial L3 product. It only comprises those bin cells of the final L3 product that are covered or intersected by the considered pixels of the input product and which are valid with respect to a given mask set. Masks are the flags which are interpreted to mask pixels in the L3 binning process so that pixels carrying these flags are not binned to the L3. All valid input pixels are aggregated into their associated bins. The final **temporal binning** creates a L3 composite generated from all spatial L3 products by taking into account (weighting) the number of contributing pixels in each bin cell of the spatial L3 products. This binning scheme is widely accepted and frequently applied in the EO data community.

3.2 System Use Cases

The production scenarios described above define a core functionality of the system. They serve for deriving common system use cases and concepts, which will allow for a consistent user experience. Deriving commonality will also ease the later extension of the system by new production scenarios and force reuse of existing components and services.

Figure 3-10 is a view of the system from a human actor's perspective. It shows the identified use cases required for an operational realisation of the production scenarios. (The use cases from Figure 3-1 are generalised here.)

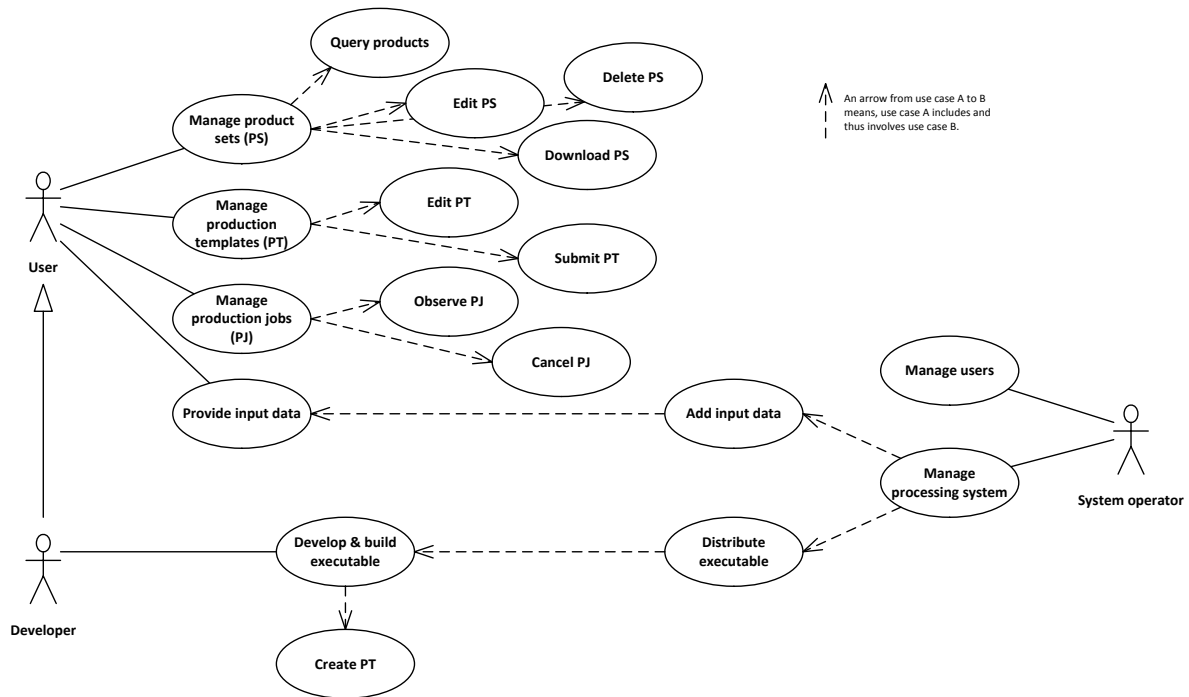


Figure 3-10: System use cases

Before the use cases shown in the diagram are explained in more detail, some general concepts and the system's actors are introduced.

3.2.1 General Concepts

The most obvious concept is the **product file set**, since all production scenarios perform at least some kind of bulk-processing on a number of input data products. We consider a product set to be a logical construct. It is basically a list of references to the actual product files physically stored in the Calvalus data archive.

Production scenarios are (server-side) work-flows each comprising one or more configurable production steps. In order to put a specific work-flow into operation, users first select a **production template**, specify the input product set and optionally edit some of the processing parameter default values.

Once submitted to the processing system, production template translates into a **production job**, whose status may be further observed by the user. Successfully executed production jobs may finally output a new product set, which could be the input for another production scenario.

After processing and eventual retrieval of the output products the system automatically removes the outputs after some **retention time**. Product sets that no longer reference any archived product are

automatically removed as well. The system may provide an option to keep user-generated data for a longer period.

3.2.2 Actors

Common **users** of the system are scientists deeply involved in the development and validation of remote sensing algorithms producing these geo-physical quantities and their time series, as well as engineers interested in calibration and validation of the EO sensors. During phase 1 of the Calvalus study, it has been decided to support particularly the cal/val activities performed in the ESA CoastColour project and CCI [RD 14].

Developers are supported by a dedicated Calvalus API that lets them modify existing or write new processor algorithms. Advanced developers will also have the possibility to develop new work-flows and develop a new production template for them.

The management and maintenance of the processing system is done by **system operators**. They are responsible adding new users, adding input data provided by them (e.g. new in-situ data, new data product sets) and for distributing the executable code provided by developers.

3.2.3 Use Case Descriptions

It is assumed that all use cases associated with the actor **user**, are realised as a web application with restricted access. Users thus need to log-in, before they can use the Calvalus system.

Use case	Actor	Description
Query products (in archive)	User	<p>Purpose: Lets a user query data products that are stored in the Calvalus archive. The resulting product set can be given a name and saved for later use.</p> <p>Motivation: Define a product set that can be used as input to various production scenarios.</p> <p>Input: Users can enter search criteria such as an area of interest, a time range, a data product type or a simple name pattern.</p> <p>Output: A named product set owned by the user.</p>
Manage product sets	User	<p>Purpose: Lets a user select, view, edit, delete and download own product sets. Viewing includes display of product set properties, such as the number of products and the total size of the associated product files.</p> <p>Motivation: Users must be able to manage their owned product sets. An owned product set may result from an archive query or as output of a production job.</p>
Edit product set	User	<p>Purpose: A user renames a product set owned by this user. (Useful, but optional: add/remove selected products of the set.)</p> <p>Motivation: Users must be able to change owned product sets.</p> <p>Input: Selected, owned product set.</p> <p>Output: The renamed product set.</p>
Delete product set	User	<p>Purpose: A user deletes an owned product set. Deletion of referenced files in the archive is implicit for product sets which</p>



		<p>are production results.</p> <p>Motivation: It must be possible to delete owned product sets. It is also important to remove user-associated files from the archive, that are not required anymore.</p> <p>Input: Selected, owned product set.</p>
Download product set	User	<p>Purpose: A user downloads a selected product set. May be realised by FTP-pull, or using HTTP “wget” to download multiple, zipped output files. (Calvalus does not offer operational data distribution services yet, but its design takes it into account that common users may download results generated by other users.)</p> <p>Motivation: It must be possible for users to obtain data stored in the Calvalus data archive.</p> <p>Input: Selected, owned product set</p> <p>Output: A ZIP archive comprising the files of the product set.</p>
Manage production template	User	<p>Purpose: Lets a user load & save, create, edit, delete and submit owned production templates. Creating a new template is basically cloning an existing one. Users can select from pre-defined production templates (see use case “Define system production template” below) related to all considered production scenarios including match-up & trend analysis, L2 & L3 processing.</p> <p>Motivation: Different production scenarios are handled in consistent and generic manner.</p>
Edit production template	User	<p>Purpose: Lets a user rename a selected production template and edit its configuration. The configuration usually comprises specification of input product set, processing parameters, e.g. for flags and masks, and the name (pattern) of the output product set to be generated.</p> <p>Motivation: Users need to invoke production scenarios with different configurations, which may be stored for later use.</p> <p>Input: Selected or newly created production template.</p> <p>Output: A modified production template.</p>
Submit production template	User	<p>Purpose: Once a production template is complete in terms of mandatory parameters, it may be submitted to the processing system. Successfully submitted production templates create a server-side production job.</p> <p>Motivation: Users shall be able to submit completely configured production templates in order to execute a production scenario on the specified input product set.</p> <p>Input: Completely configured production template.</p> <p>Output: An observable, server-side production job.</p>

Manage production job	User	<p>Purpose: Let users observe and cancel their created production jobs.</p> <p>Motivation: Production jobs are server-side processes that need to be manageable by users that invoked them and by the operator.</p>
Observe production job	User	<p>Purpose: Users can inspect the progress, status (submitted, started, running, ended) and error codes of server-side production jobs.</p> <p>Motivation: Retrieve status of running production jobs.</p> <p>Input: Selected production job.</p>
Cancel production job	User	<p>Purpose: A user cancels a running production job. Cancelled jobs may be restarted or deleted.</p> <p>Motivation: It shall be possible to manually cancel jobs that have been accidentally submitted, or that are badly configured or simply log-running.</p> <p>Input: Selected production job.</p>
Provide input data	User	<p>Purpose: Users provide data that serves as input for existing production scenarios, newly acquired or updated (reprocessed) MERIS L1b data, processor related auxiliary data or new reference data (in-situ) records.</p> <p>Motivation: The system shall be extendible in terms of new EO, auxiliary or reference data.</p> <p>Output: Various types of input data</p>
Develop and build executable	Developer	<p>Purpose: Developers can use the Calvalus API to change existing algorithms or develop new (L2) processors. After building an executable, it is provided to the system operator who will install the code in the processing system (see use case "Install executable"). Includes definition of a system production template.</p> <p>Motivation: Ability to change the (L2) processors that are used in various processing scenarios.</p> <p>Input: Calvalus library and API</p> <p>Output: Processor executable</p>
Define system production template	Developer	<p>Purpose: The Calvalus system offers to users a number of predefined, system production templates. A system template is associated with a certain processor executable and comprises default values for all optional processing parameters. Values of missing mandatory parameters must be entered later by users (see use case "Edit production template"). Users clone system production templates in order to derive their own product templates.</p>

		<p>Motivation: Since templates are specific to a certain processor executable, they are defined by developers wishing to extend the Calvalus system by new production scenarios.</p> <p>Input: Processor executable.</p> <p>Output: Processor executable bundled with associated system production template.</p>
Manage processing system	System operator	<p>Purpose: System operators configure and maintain the Hadoop cluster including hardware and software. It also includes data ingestion and the distribution of executables required for the realisation of specific production scenarios.</p> <p>Motivation: The Calvalus system needs continuous administration and maintenance.</p>
Manage users	System operator	<p>Purpose: System operators can add and remove users to the Calvalus system. They also edit user specific properties such as log-in data or access rights.</p> <p>Motivation: The Calvalus demonstration system is not for public use. Only registered users can access the processing system.</p>
Install executable	System operator	<p>Purpose: A System operator installs the processor executable build by a developer. Installation involves distribution of the code in the Hadoop cluster so that each cluster node can execute it.</p> <p>Motivation: Processor executables must be distributed to all nodes in the Hadoop cluster.</p> <p>Input: Processor executable.</p>
Add input data	System operator	<p>Purpose: System operators ingest the data provided by users into the Calvalus system. (EO data acquisition may be automated later.)</p> <p>Motivation: Input data needs to be transferred into dedicated system locations and an inventory (database, file index) will need to be updated.</p> <p>Input: Various types of input data.</p>

3.3 Input Data Requirements

Calvalus is demonstrating its EO data processing and cal/val capabilities using the data of a single sensor, namely MERIS on board of ESA's Envisat satellite. Practical experience with the system is gathered by the cal/val needs of the ongoing ESA CoastColour [RD 14] and the ocean_colour_cci [RD 17] projects. In both projects, water leaving reflectances and IOPs are derived from MERIS radiances. While CoastColour aims at the remote sensing of IOPs in regional, coastal waters from MERIS FR data, the focus in ocean_colour_cci project is the generation of high-quality, consistent, long-term IOPs over marine waters, water leaving reflectances and chlorophyll from global MERIS RR data (and other sensors).

3.3.1 EO Input Data

CoastColour: The input data for CoastColour comprises MERIS FR product slices that are cut out above 22 coastal sites. The data has been acquired during 5 years, from 2005 to 2010. The total size of the CoastColour MERIS FR product set is approximately 25 TB (5 years x 5 TB/year).

ocean_colour_cci: For ocean_colour_cci, global data coverage is important so that an effective trend and consistency analysis can be performed on L2 variables remotely sensed above a number of marine water types. The global product set comprises MERIS RR full orbits from 5 years, from 2004 to 2008. The total size of this dataset is approximately 15 TB (5 years x 3 TB/year). MERIS L1 file sizes usually vary between 20 (some) and 600 MB (most of them).

The total amount of file system space is determined by the hard drive space on the 19 data nodes of the Calvalus cluster. Each node provides 4.5 TB, and the overall space usable by the HDFS is 76 TB (19 nodes x 4 TB/node, see chapter 4 Hardware Environment). By using a default replication rate of 3, the effective storage space is reduced to 25 TB. In order to leave sufficient space for user-generated L2 and L3 products, we have decided to use 20 TB for input data and 5 TB for user generated outputs.

For demonstration purposes, we prefer having a complete, global coverage MERIS RR dataset over the regional MERIS FR dataset so only a subset of the CoastColour coastal sites will be used. Only sites are considered, that have associated in-situ data and which are most representative for the coastal water type.

The overall storage space is finally divided as follows:

MERIS L1 RR 2004-2008:	15 TB
MERIS L1 FR 2005-2010 (slices over selected sites):	5 TB
User space:	5 TB
Total:	25 TB

3.3.2 Auxiliary Data

The following table provides an overview of the auxiliary data used within the various production scenarios. It only comprises those auxiliary data which are either subject to change over time or have a non-negligible size. In both cases, these auxiliary data files may not be part of a processor's software installation. Instead, the preferred location of the files is the HDFS and processors would be configurable in terms of final locations of their auxiliary data.

MERIS central wavelengths and Sun Spectral Fluxes	MERIS radiometric correction: Smile effect correction	Hosted in BEAM source code repository https://github.com/bcdev/beam/	Plain text files
MERIS recalibration coefficients	MERIS radiometric correction: Recalibration	Ludovic Bourg (ACRI-ST) and ESA (http://earth.eo.esa.int/services/auxiliary_data/meris/), also hosted in BEAM source code	Envisat N1 format



		https://github.com/bcdev/beam/	
MERIS equalisation coefficients	MERIS radiometric correction: Degradation equalisation	Marc Bouvet (ESA ESTEC). Hosted in BEAM Case2R source code repository https://github.com/bcdev/beam/	Plain text files
Diverse, regionalised neural nets.	Sun glint/atmospheric correction, Case2R L2 processor. Match-up and trend analysis.	Roland Doerffer (HGZ). Hosted in Case2R source code repository https://github.com/bcdev/beam-meris-case2	Plain text files
Reference data incl. in-situ, MODIS, SeaWiFS, MERIS (MEGS) L2 data.	Match-up analysis. See further description below.	Assembled by various team members of the CoastColour project. Not available yet (as of 2010-11-08).	Plain text files
Bathymetry (or DEM)	Trend analysis.	E.g. http://www.gebco.net/	GeoTIFF

At the time being, all auxiliary data files are assumed static with respect to the MERIS L1 input data (no dependency in time). They are therefore considered to be part of the installation of the processor executables, so it will be distributed to each node within the Hadoop cluster. This may change at any time for auxiliary data that have a dependency in time.

3.3.3 Reference Data

The reference data used by the Calvalus system for the match-up process is originating from the a number of data sources.

- In-situ data from various sensors at the CoastColour cal/val sites
- MERIS L2 products (from standard L2 processing using MEGS)
- MODIS L2 products
- SeaWiFS L2 products

All reference data will be stored in a database accessible by the Calvalus processing system. The table structure will be the same for all sensors.

Attribute	Units	Type	Description
site	-	S	Name of the site.
sensor	-	S	Name of the sensor that provided the measurement.
time	UTC*	I	Time of the measurement.
latitude	deg	F	WGS-84 latitude of the site's location.
longitude	deg	F	WGS-84 longitude of the site's location.
thetas	deg	F	Solar zenith angle.

rho_wn_λ	1	F[9]	Normalised water-leaving reflectances in 9 MERIS bands. λ is the spectral band wavelength in nanometres.
Chl	mg/ m ³	F	Chlorophyll concentration.
TSM	mg/ m ³	F	Total suspended matter.
YS	mg/ m ³	F	Yellow substances.
AOT_870	1	F	Aerosol optical thickness.
alpha	1	F	Aerosol Angstrøm exponent.

* UTC is given by ISO 8601. The pattern is `yyyymmddThhmmssZ`

The number of measurements per site will vary between less than 10 and several hundreds. The reference dataset is considered to be dynamic. It is expected that new in-situ measurements or that new EO reference data sources will be added throughout the project.

4 Hardware Environment

Calvalus is considered to be a self-standing, self-contained demonstration system. It relies heavily on the Hadoop technology, which in turn is supposed to be operated on a Linux cluster.

The hardware for Calvalus will be procured and hosted at BC for the duration of the project and, depending on ESA decision, it will be operated and maintained for 2 years after the official project end.

4.1 Hosted hardware vs. cloud computing

The project allocates **25k€** for the procurement of hardware. Since this is a limited budget with respect to the expected performance of the system, it has also been considered to configure a test cluster in a suitable cloud environment. At the time of this writing, there were two major arguments against the cloud approach:

1. Upload of the entire EO data archive into the cloud is an expensive and time consuming task.
2. Within cloud infrastructures, data locality is usually lost.

For Hadoop running on Amazon MapReduce using EC2 instances and the S3 file system, the Hadoop wiki says

“There are two ways that S3 can be used with Hadoop's Map/Reduce, either as a replacement for HDFS using the S3 block filesystem (i.e. using it as a reliable distributed filesystem with support for very large files) or as a convenient repository for data input to and output from MapReduce, using either S3 filesystem. In the second case HDFS is still used for the Map/Reduce phase. Note also, that by using S3 as an input to MapReduce you lose the data locality optimization, which may be significant.” (<http://wiki.apache.org/hadoop/AmazonS3>)

4.2 Hardware selection

The hardware system shall be representative for a larger system supporting multiple users with multiple, simultaneously running jobs. It shall also comprise enough nodes making it possible to adequately test the scalability and reliability of the system. With this requirement the following selection criteria for the cluster hardware have been established:

- Prefer a higher number of servers over the performance of each server
- Prefer a high performance of single servers over their fail-safety

To fulfil these criteria a set of different configurations have been examined. The alternatives are listed in Table 4-1.

Table 4-1: Hardware alternatives

Solution	Type	CPU	HDD	RAM	Network	Price (€)	Uptake (Watt)	Size
MEDION AKOYA X7351 D	Desktop	I7-860, 4x 2.8GHz	3 TB	6 GB	Gigabit	1000		2)
Intel SR1630HGP Server	19" Barebone	Xeon UP X3450, 4x 2.66GHz	4.5 TB	6 GB	2x Gigabit	1123	350	1U
Supermicro	19"	Xeon UP X3450,	4.5 TB	8 GB	2x	1450	280	1U

Barebone 5016I-MTF	Rackmount	4x 2.66GHz			Gigabit			
Lenovo IBM ThinkServer RS110	19" Rackmount	Xeon UP X3450, 4x 2.66GHz	1)	2 GB	2x Gigabit	700	350	1U
Dell	19" Rackmount	Xeon UP X3430, 4x 2.4GHz	4 TB	4 GB	2x Gigabit	2065	375	1U
Dell	Tower	Xeon UP X3430, 4x 2.4GHz	4 TB	4 GB	2x Gigabit	1777	375	2)

U: Unit in a 19-inch rack, equals to 44.45 mm

1): Drives have to be ordered from IBM because drive carriers are not available separately.

2): Standard desktop computer which cannot be installed into a rack.

The current cluster has been built from rack-mounted barebones. Compared to standard desktop computers they have the advantage of being made of components of server quality that are designed to run 24/7. Furthermore space occupied by 20 desktop cases is significant larger than a single server rack with the possibility of hosting up to 42 servers.

The ordered servers from Supermicro are very similar to the Intel barebones but feature a 4th drive bay for future expansion of the storage space as well as KVM over LAN for remote hardware maintenance. In comparison to the barebone servers from Intel they are delivered fully assembled.

The full specification of the procured servers:

- Supermicro Barebone 5016I-MTF
- 1HU Rackmount with 280W power supply
- 4x HotSwap SATA trays
- 1x Intel Xeon X3450 Processor with 2,66 GHz Quad Core, 8 MB Cache
- 6 Memory Slots (max. 32GB) - 8 GB Memory (4x 2 GB DDR3 reg. ECC)
- 2x Gigabit Ethernet controller onboard with RJ-45 LAN connector.
- IPMI 2.0 incl. KVM over LAN Expansion Slots: 1x PCI-Express x16
- 3x 1,5 TB S-ATA Seagate Disks, 7,2K UPM, 32 MB Cache ST31500341AS (one disk tray remains empty)

All 20 servers are connected using a Gigabit Ethernet switch. They are installed in a rack as shown in Figure 4-1.



Figure 4-1: Calvalus cluster hardware

4.3 Configuration

The operating system on the servers is “Ubuntu Server 10.04 LTS (Long Term Support), 64bit”. We currently have a configuration with one server being the dedicated master (*namenode* and *jobtracker* in Hadoop terminology) for the cluster and 19 servers operating as slaves (*datanode* in Hadoop).

5 Concurrency Trade-off Analysis

This chapter describes the results of a trade-off analysis done to prepare design decisions on data organisation in HDFS and parallel processing in Hadoop. The analysis shall answer the questions

- How to organise level 1 input data on the distributed file system best for parallel processing?
- How much speedup by parallelism can be achieved, despite overhead and bottlenecks?

Besides this, the analysis has identified some Hadoop parameters relevant for optimising data-intensive and computationally-intensive earth observation data processing on the cluster.

5.1 Investigated Configurations

To analyse the influence of data-organisation and data-locality on processing time the measurements have considered different data configurations.

The MERIS L1B data to be stored in the distributed file system is already aggregated into "product files", i.e. files containing a spatio-temporal range of data in an ENVISAT-specific format "n1". The granularity of the files are full orbits for MERIS reduced resolution (RR), which is up to 550 MB, and parts of orbits something smaller than 2 GB for MERIS full resolution swath (FRS). The n1 format is composed of a header followed by data of the first band, then data of the second band etc. (band-interleaved).

Hadoop proposes to organise data into blocks of 64 MB. Several blocks may make up a file, but each block may be stored on a different node of the cluster. The intention is that processing of the different splits of a file may be distributed, where splits should match blocks to utilize data locality. Deviations from the proposed block size of 64 MB are possible. In order to meet this one of the configurations uses a reorganised line-interleaved file format for the input (Figure 5-1).

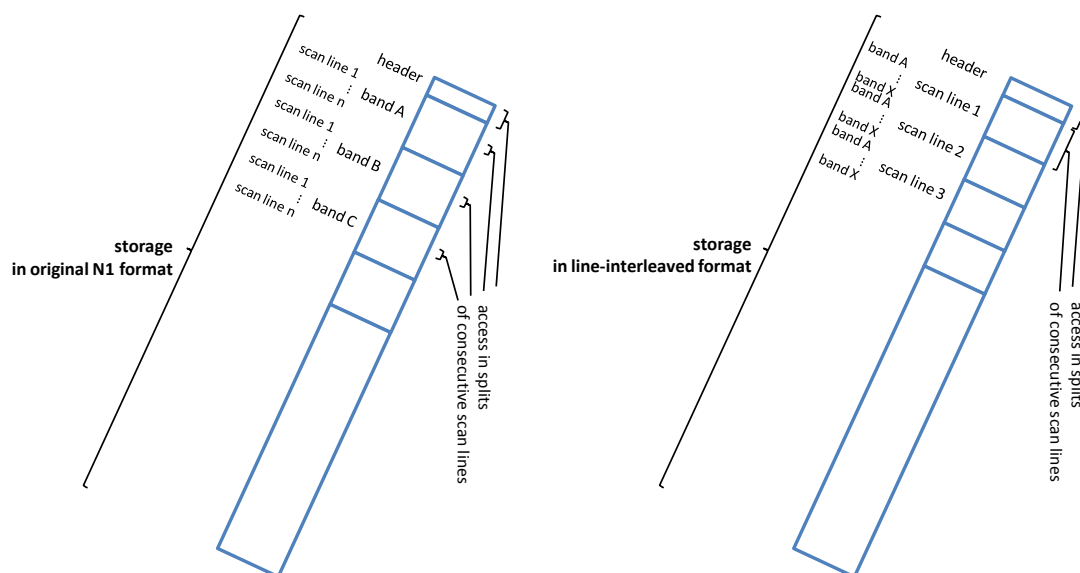


Figure 5-1: Original band-interleaved N1 format and reorganised line-interleaved format

The line-interleaved format can be split into blocks at any line. Blocks are distributed to the nodes. Access to a set of consecutive lines can be data-local.



Figure 5-2: Variants A to E for distributed data organisation and access

The data formats considered (Figure 5-2) are:

- A. (n1) MERIS L1B files are stored and accessed in a single block each in the original n1 format. A block may be 550 MB in this case, and the complete file will be stored on the same node.

The advantage of this format is that no conversion is necessary. Replication ensures that each product is available on 3 nodes (3=replication factor).

- D. (slices) Child products of 64 MB of the MERIS L1B files are generated, each with its own header. The advantage of this format is that granules for concurrent data-local processing are already built at data upload.
- E. (line-interleaved) MERIS-L1B files are stored in blocks of ~64MB, but with a reorganisation of the band data. The data are ordered by scan lines instead of bands. The advantage of this format is that it can be split at any scan line. Blocks will be distributed, and only the header must be read from remote for blocks other than the first one.

Two additional access patterns have been analysed as they are interesting for concurrency:

- B. (n3) Storage is as in A, but access is in 3 slices. The idea is to use as many slices as there are replicated files and to generate different parts of the output on different nodes concurrently.
- C. (n18) Storage is as in A, but access is in 18 slices. This gives up data locality in favour of concurrent processing. Some of the slices will be processed on nodes that have a replication of the input, others may be processed on nodes that access the input over the network.

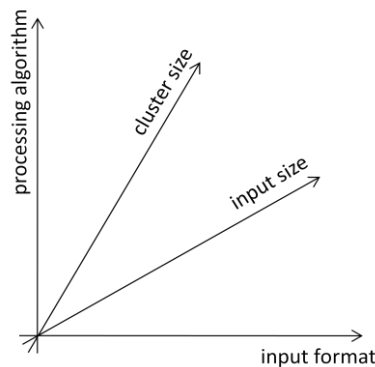


Figure 5-3: Variable dimensions analysed by measurements

Orthogonal to this data configuration three dimensions have been varied for the measurements (Figure 5-3):

- Processing algorithm:
 - NDVI (short-running, IO-intensive)
 - Radiometric correction (one of the processing steps of the L2 processor, increased computational effort compared to NDVI)
 - Case2R (computationally intensive)
- Cluster size:
 - 9 nodes, 2 processes on each node (maximum concurrency is 18)
 - 18 nodes, 2 processes on each node (maximum concurrency is 36)
 - 18 nodes, 4 processes on each node (maximum concurrency is 72)
- Input selection:
 - single MERIS RR product (550 MB)
 - one week of MERIS RR products (53 GB)
 - single MERIS FRS product (1.7 GB)
 - one week of MERIS FRS products (307 GB)

The single product stands for a scenario of on-the-fly processing of a single input while the week of inputs stands for a reprocessing scenario. Each of the processing algorithms have been configured to run single-threaded to get comparable results.

For different combinations of variants, measurements have been done for data uploaded into the HDFS and for processing on the Hadoop cluster. The results of the measurements are listed in the next subsection.

5.2 Measurement Results

This section presents runtimes of data ingestion and of processing for the variants named in the previous section. Data ingestion is the process of transferring level 1 input data to the HDFS cluster. The data is transferred as complete N1 files, as child products of 64 MB, and as complete reorganised line-interleaved file. The times include the conversion effort.

5.2.1 Data Ingestion

Figure 5-4 compares the transfer times for different data formats, showing times for a week of products, for a single product, and for RR and FRS. The transfer times to a Unix file system by FTP without the replication overhead are listed for reference. FTP is one of the faster protocols over TCP and serves as a lower bound for comparison. The times are measured on a gigabit network.

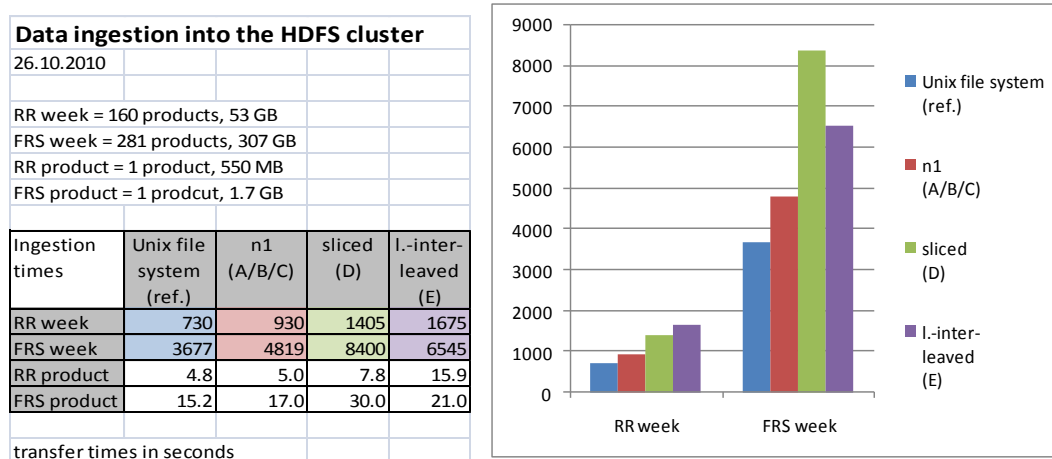


Figure 5-4: Data ingestion times for different target data formats

The figure shows that there is an overhead of transferring data to HDFS (e.g. 930 seconds to transfer a week of RR to HDFS compared to 730 seconds for the simple transfer to a Unix file system by FTP). Further, the reorganisation into the line-interleaved format consumes time (e.g. 1675 seconds to generate and transfer line-interleaved in comparison to 930 seconds to transfer the unmodified n1 of the same size. Finally, also slicing requires additional effort, both for reorganisation and for more files. The time for FRS for this format is larger related to the other times (also for a single product).

5.2.2 Processing a Single Product

In order to investigate the influence of the data format and data access on processing times, several measurements have been done on the cluster of 18 nodes with 2 processes on each node, resulting in an overall concurrency of 36. Each measurement has been repeated 3 times, investigated for low variation, and the median value has been selected. Figure 5-5 shows processing times for a single product with processors with different time demand. The processing times on a single node in one single-threaded process without Hadoop have been added for reference.

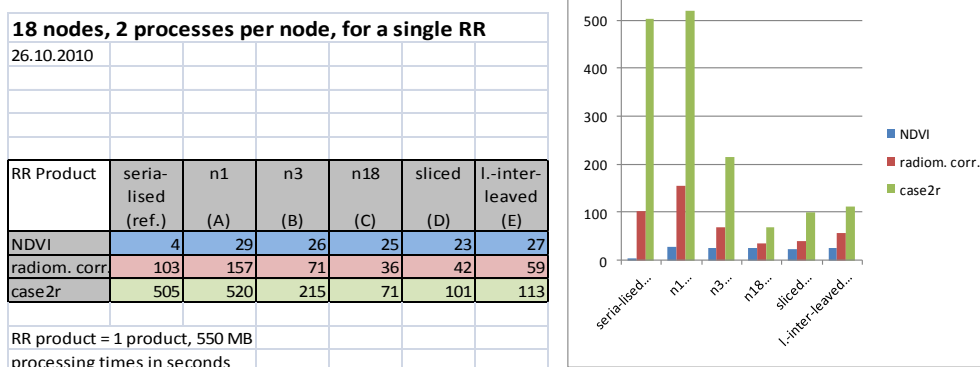


Figure 5-5: Processing times for a single product with different data formats

The figure shows that there is an overhead for running processes on the cluster (difference between n1 and ref, as both run on a single node only). For short running processes like NDVI concurrent processing of parts of the product does not gain much. This is the case for long running processes like Case2R:

- n3 (B) generates three parts of the result concurrently in one third of the time plus some overhead.
- Sliced (D) for RR means ca. 6 slices, increasing concurrency further.
- This is comparable to line-interleaved (E) that uses about the same 64 MB splits. But the time is higher than for sliced.
- n18 (C) computes parts of the result in 18 concurrent splits. Due to the overhead this does not cut the time to 1/18.
- "n36" and "n72" (on an extended cluster with 4 processes per node) result in 56 seconds and 59 seconds respectively. The increase of concurrency to n36 gains speed, but the further increase does not (see measurements with different cluster configurations below).

Increasing concurrency in processing a single input gains speed but not to a linear extent.

5.2.3 Processing a Week of Inputs

When processing a week of 160 inputs the starting situation is different as the data set is inherently "split" into products already, even in case n1 (A). Figure 5-6 shows processing times for the week of data for the same processors and variants.

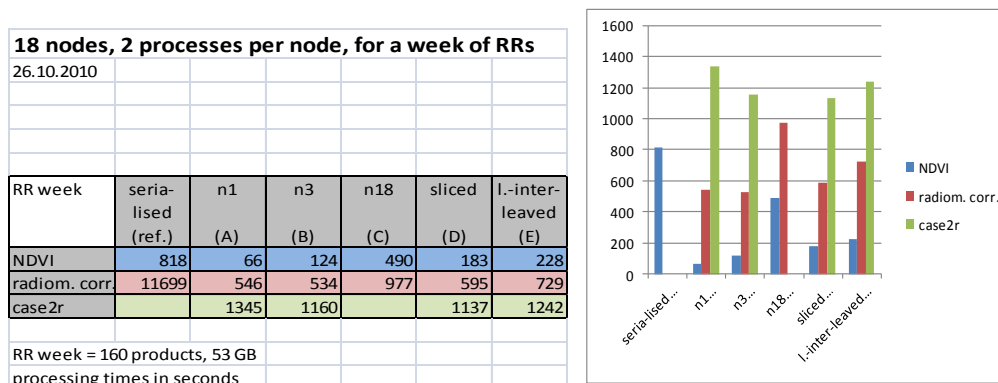


Figure 5-6: Processing times for a week of products with different data formats

The figure shows that

- for short running processors splitting below the level of products increases the times (66 seconds for the week for n1, all other figures higher, n18 worst with 490 seconds).
- For longer running processors splitting the inputs moderately leads to lower figures due to an effect of concurrency: Single longer running processes of the last processing wave determine the overall runtime. Split inputs make shorter waves, allowing for more concurrency at the end of the flood.

n1, n3 and sliced all perform relatively homogeneous. n18 for the large set of inputs does not increase concurrency but only overhead. Also line-interleaved does not perform as well as the others. All of them beat sequential processing on a single node, but none of them by a factor equal to concurrency.

5.2.4 Varying the Cluster Configuration

Finally, processing measurements on three configurations of the cluster allow observing scalability. Figure 5-7 shows processing times for a reduced cluster of 9 nodes, a full cluster of 18 nodes with 2 concurrent processes on each node, and a full cluster with 4 processes on each node.

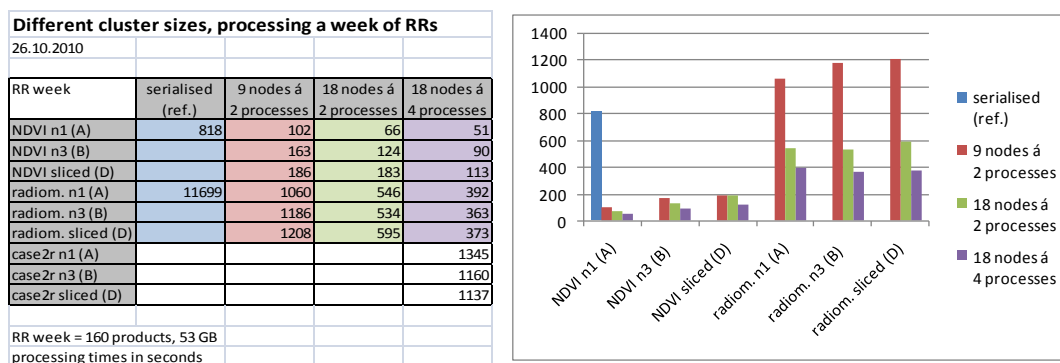


Figure 5-7: Processing times for a week of products on different cluster configurations

The figure shows that for longer running processes increasing concurrency by adding nodes cuts down overall processing time (1060 seconds on 9 nodes, 546 seconds on 18 nodes). Duplicating the number of processes per node again gains some speed (392 seconds) but does not cut the time by 1/2. Short running processors do not gain as much from parallelization as longer running processors.

5.3 Hadoop-Specific Findings

The first measurements with processing on the Hadoop cluster immediately lead to questions regarding the overhead introduced. Another observation was that the figures for the same processing configuration varied substantially between measurement runs.

Investigating the overhead immediately revealed that Hadoop constructs and dispatches tasks for activities, and each task requires some time even if the activity is an empty operation. By removing e.g. the empty reduce task from the processing chain gained about 10 seconds.

A second observation was that single task jobs and in some cases also jobs with a few tasks are scheduled on remote hosts instead of data-local hosts. It turned out that the reason for this is the Hadoop approach to pull tasks by slaves that poll for them at the master when they are idle. The first slave that polls gets a task, regardless of whether it must access the data remote or local. If there are enough tasks it prefers one that is data-local such that data-local processing is achieved for large sets of splits/tasks. For larger sets of inputs only at the end of the processing wave tasks are scheduled remotely. Then, the same situation as for jobs with few tasks arises that a node pulls a task and there is no data-local task left.

What remains still contains overhead as shown in the numbers of the previous section. Figure 5-8 lists a log file excerpt used to investigate this on an exemplary case in detail (time lag of client, master and slave04 with some uncertainty due to maybe weak synchronisation by NTP).

```

2010-10-21 15:31:11.9 client submits request
[...]
2010-10-21 15:31:17.742 INFO cvmaster00 Initializing job_201010081626_0153
[...]
2010-10-21 15:31:17.964 INFO cvmaster00 Input size for job job_201010081626_0153 = 508419147. Number of splits = 1
2010-10-21 15:31:17.964 INFO cvmaster00 tip:task_201010081626_0153_m_000000 has split on node:/default-rack/cvslave01.bc.local
2010-10-21 15:31:17.964 INFO cvmaster00 tip:task_201010081626_0153_m_000000 has split on node:/default-rack/cvslave02.bc.local
2010-10-21 15:31:17.964 INFO cvmaster00 tip:task_201010081626_0153_m_000000 has split on node:/default-rack/cvslave03.bc.local
[...]
2010-10-21 15:31:18.400 INFO cvmaster00 Adding task 'attempt_201010081626_0153_m_000002_0' to tip task_201010081626_0153_m_000002, for tracker
'tracker_cvslave04.bc.local:localhost/127.0.0.1:48991'
2010-10-21 15:31:18.402 INFO cvslave04 In TaskLauncher, current free slots : 2 and trying to launch attempt_201010081626_0153_m_000002_0
2010-10-21 15:31:19.235 INFO cvslave04 In JvmRunner constructed JVM ID: jvm_201010081626_0153_m_-429863639
[...]
2010-10-21 15:31:20.097 INFO cvslave04 Task attempt_201010081626_0153_m_000002_0 is done.
2010-10-21 15:31:21.405 INFO cvmaster00 Task 'attempt_201010081626_0153_m_000002_0' has completed task_201010081626_0153_m_000002 successfully.
2010-10-21 15:31:21.405 INFO cvmaster00 Adding task 'attempt_201010081626_0153_m_000000_0' to tip task_201010081626_0153_m_000000, for tracker
'tracker_cvslave04.bc.local:localhost/127.0.0.1:48991'
2010-10-21 15:31:21.405 INFO cvslave04 In TaskLauncher, current free slots : 2 and trying to launch attempt_201010081626_0153_m_000000_0
2010-10-21 15:31:21.490 INFO cvslave04 In JvmRunner constructed JVM ID: jvm_201010081626_0153_m_-951408644
2010-10-21 15:31:22.475 INFO cvslave04 attempt_201010081626_0153_m_000000_0 starts processing of split
hdfs://cvmaster00:9000/data/experiments/n1/MERIS-RR-product/MER_RR_1PNPDE20100901_065519_000023902092_00321_44463_9218.N1:0+508419147
[...]
2010-10-21 15:31:33.119 INFO cvslave04 attempt_201010081626_0153_m_000000_0 stops processing of split
hdfs://cvmaster00:9000/data/experiments/n1/MERIS-RR-product/MER_RR_1PNPDE20100901_065519_000023902092_00321_44463_9218.N1:0+508419147
after 10.641688501 sec
2010-10-21 15:31:33.132 INFO cvslave04 Task attempt_201010081626_0153_m_000000_0 is in commit-pending, task state:COMMIT_PENDING
2010-10-21 15:31:33.410 INFO cvslave04 Received commit task action for attempt_201010081626_0153_m_000000_0
[...]
2010-10-21 15:31:34.205 INFO cvslave04 Task attempt_201010081626_0153_m_000000_0 is done.
2010-10-21 15:31:36.413 INFO cvmaster00 Task 'attempt_201010081626_0153_m_000000_0' has completed task_201010081626_0153_m_000000 successfully.
2010-10-21 15:31:36.413 INFO cvmaster00 Adding task 'attempt_201010081626_0153_m_000001_0' to tip task_201010081626_0153_m_000001, for tracker
'tracker_cvslave04.bc.local:localhost/127.0.0.1:48991'
2010-10-21 15:31:36.413 INFO cvslave04 In TaskLauncher, current free slots : 2 and trying to launch attempt_201010081626_0153_m_000001_0
2010-10-21 15:31:36.436 INFO cvslave04 No new JVM spawned for jobId/taskid: job_201010081626_0153/attempt_201010081626_0153_m_000001_0. Attempting to
reuse: jvm_201010081626_0153_m_-951408644
2010-10-21 15:31:37.713 INFO cvslave04 JVM with ID: jvm_201010081626_0153_m_-951408644 given task: attempt_201010081626_0153_m_000001_0
[...]
2010-10-21 15:31:37.828 INFO cvslave04 Task attempt_201010081626_0153_m_000001_0 is done.
2010-10-21 15:31:39.456 INFO cvmaster00 Task 'attempt_201010081626_0153_m_000001_0' has completed task_201010081626_0153_m_000001 successfully.
2010-10-21 15:31:39.553 INFO cvmaster00 Moving file:/usr/lib/hadoop-0.20/logs/history/cvmaster00_1286547979383_job_201010081626_0153_conf.xml to
file:/usr/lib/hadoop-0.20/logs/history/done
2010-10-21 15:31:39.917 INFO cvslave04 JVM : jvm_201010081626_0153_m_-951408644 exited. Number of tasks it ran: 2
2010-10-21 15:31:40.309 INFO cvslave04 JVM : jvm_201010081626_0153_m_-429863639 exited. Number of tasks it ran: 1
[...]
2010-10-21 15:31:39.8 client receives final status (by polling every second)

```

Figure 5-8: Log file excerpts for a simple NDVI processing job

The listing shows

- The client writes its log entry of job submission at 15:31:11.9
- The master has created the job at 17.7, i.e. after 5.8 seconds.
- The master knows where to submit the preparation task to at 18.4, i.e. after 0.7 seconds. This time can be explained with the polling for tasks ("heartbeat") by the slaves.
- The preparation task is done at 20.0, and the master gets knowledge about this at 21.4, i.e. after 1.4 seconds.
- The master submits the processing task to slave04 and slave04 recognises this at 21.4, i.e. without delay.
- Processing really starts at 22.4, after 1 second starting and initialising a new JVM.
- Processing finishes at 33.1, what the master gets at 36.4 after some commit activity and some delay.
- A cleanup task is started immediately and is done at 37.8. The master gets this at 39.4.
- The client receives this by polling at 39.8.

The overall delay is introduced by mainly four factors:

- The communication from slave to master is not immediate after a state change. Together with the pull strategy for tasks by slaves this introduces delays.
- The preparation and cleanup is delegated to tasks as well, leading to a constant overhead.
- The master does not immediately create the job but seems to require some time for this.

- The client uses polling to retrieve the final status.

As named in the beginning of this section another issue was that the observed times required for processing varied much between processing runs. It turned out that this is caused by output replication. Initial measurements were done with the default replication of 3 also for the outputs. This means that each output product is written to the local node, replicated to another node, that in turn replicates it to a third node. When the cluster is under heavy load this seems to require a substantial time budget. It caused unpredictable pauses in the processing stream of several seconds several times during processing (see Figure 5-9).

```
[...]
2010-10-27 16:17:07,396 INFO com.bc.calvalus: written y=7680 h=64
2010-10-27 16:17:07,955 INFO com.bc.calvalus: written y=7744 h=64
2010-10-27 16:17:08,497 INFO com.bc.calvalus: written y=7808 h=64
2010-10-27 16:17:09,057 INFO com.bc.calvalus: written y=7872 h=64
2010-10-27 16:17:09,597 INFO com.bc.calvalus: written y=7936 h=64
2010-10-27 16:17:10,163 INFO com.bc.calvalus: written y=8000 h=64
2010-10-27 16:17:25,179 INFO com.bc.calvalus: written y=8064 h=64
2010-10-27 16:17:25,924 INFO com.bc.calvalus: written y=8128 h=64
2010-10-27 16:17:26,478 INFO com.bc.calvalus: written y=8192 h=64
2010-10-27 16:17:27,035 INFO com.bc.calvalus: written y=8256 h=64
[...]
```

Figure 5-9: Processing trace with anomaly, observed in case of output replication

This has been eliminated by setting output replication to "1" for this measurement such that the output is no longer replicated to other nodes. In general, the decision to replicate outputs will depend on whether the outputs are to be long-term archived.

5.4 Conclusion

Table 5-1 shows which degree of concurrency the different approaches allow.

Let

n	be the number of slave nodes in the cluster	e.g. 9
p	be the number of concurrent processes per node	e.g. 2
r	be the replication factor of input data	e.g. 3
s	be the split factor of input file size / 64 MB	e.g. 6

Table 5-1: Degree of concurrency for data format and access variants

	n1 (A)	n3 (B)	n18 (C)	sliced (D)	line-inter-leaved (E)
Single input processing	1 <i>data-local</i>	3 <i>data-local</i>	n * p, e.g. 18 <i>only r * p of them data-local</i>	s splitting slices would get n * p <i>data-local</i>	n * p <i>data-local</i>
Bulk processing	n * p	n * p <i>better than n1 only for small sets</i>	n * p <i>small tasks, not recommended</i>	n * p <i>more tasks than n1, comparable to n3</i>	n * p <i>no advantage over n1</i>

The table shows that the maximum concurrency can be achieved with each input data format, and that splits are helpful to process single products or small sets of inputs concurrently.

As a conclusion the measurements have shown how distributed processing with different configurations performs well on a cluster. They also have contributed to an understanding of some strengths and weaknesses of Hadoop. The main results are:

- Storing the original format (n1) performs well for bulk processing. The granularity of products in this case is the basis for parallelisation.



- Due to a fast network, the original format is acceptable for single input processing. The approach to produce the result concurrently (n3, n18, n...) is feasible. Note that this may not hold for larger clusters where concurrent access to the input via the network may be the bottleneck.
- Hadoop introduces some overhead that cannot be neglected especially for short-running jobs for single input processing. The overhead is mainly caused by features of Hadoop that (shall) support scalability, like the heartbeat-timed pulling of tasks by slaves.

The design of Calvalus will initially be based on storing the N1 files in their original format.

6 System Component Design

This chapter describes the static architecture of the Calvalus system. It logically decomposes the system into functional components and describes each of the components with its identification, purpose, functions, interfaces and the data it maintains (aspects taken from ECSS-E40 [RD 15]).

6.1 System Decomposition

The Calvalus demonstration system comprises basically an EO data processing system based on Hadoop, a number of dedicated service components and a user portal. The UML component diagram in Figure 6-1 shows the identified system components and the interface dependencies between them.

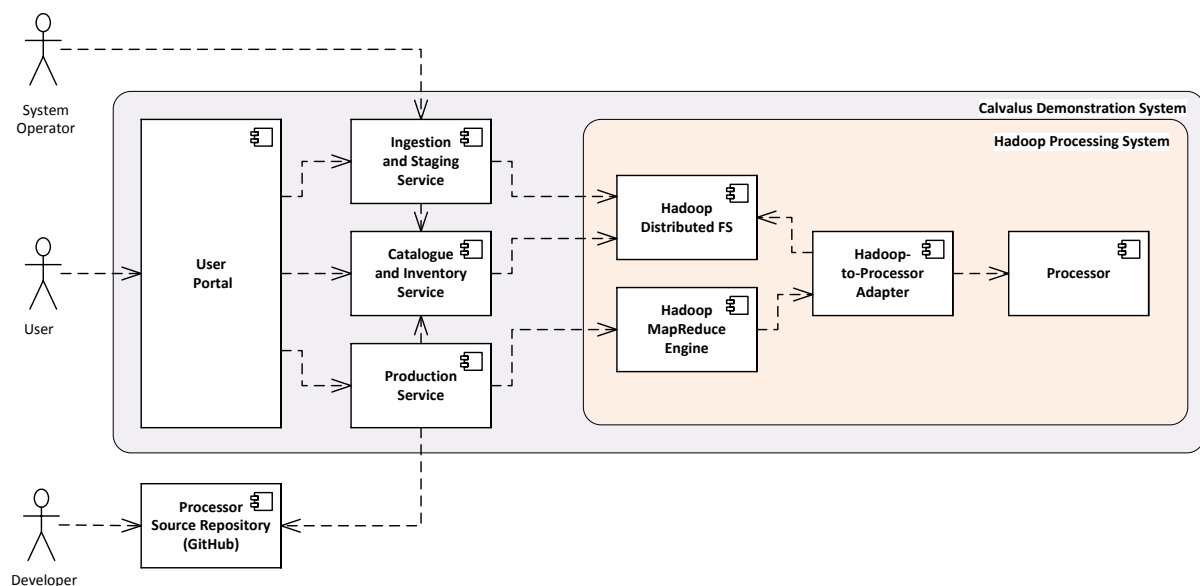


Figure 6-1: Calvalus system decomposition

Registered users interact with the processing system solely via the user portal. The portal is a usual web application, accessible through typical internet browsers. It communicates with the processing system exclusively via the Calvalus services. Beyond the user portal, developers submit updates to existing processors and new processors that are stored and maintained within the system. Operators monitor operations of the entire Calvalus system.

The processing system comprises the MapReduce engine, the HDFS and the actual processors to be executed by the processing system. The MapReduce engine and the HDFS are both part of the Hadoop software. HDFS will also serve as the one and only Calvalus data archive. The processors realise the various Calvalus processing scenarios and are actually independent on the MapReduce API of Hadoop. A dedicated Hadoop-to-Processor adapter is used to invoke the processors in a MapReduce-specific style.

The interfaces of the service components shown above abstract from Hadoop-specific concepts, such as those introduced by the Hadoop MapReduce engine and the HDFS, by using ones that are more common to the EO domain, e.g. data product files, product file ingestion, catalogue, inventory, production and staging.

The following sections describe each of the components by the following categories:

Identifier	Section heading with name of the component in the Calvalus system
Purpose	Characterisation of the role of the component within the system
Function	Computational service of the component, list of functions
Interfaces	Identification of interfaces implemented by the component and of interfaces used by the component. The interface items and protocols will be defined in chapter 7.
Data	Data encapsulated and maintained by the component, the component's complex state

The description starts with the user side followed by the first layer of services in the backend.

6.2 User Portal

This section describes the logical component *user portal* of Calvalus.

6.2.1 Purpose

The Calvalus user portal is the main human-machine interface to the Calvalus system. For registered users, it provides an intuitive access to the internal Calvalus services such as data catalogue query, data inventory manipulation, production control and staging of output data. The entry page of the portal is public and provides to visitors a detailed description of the Calvalus study.

6.2.2 Function

The computational service provided by the user portal is:

- user authentication, user management
- presentation of catalogue, production and result access by graphical user interfaces
- presentation and handling of user-related product sets or production request templates

As a web application, the portal is accessed by entering its URL in a usual internet browser. The entry page of the portal is public and provides to guest visitors a detailed description of the Calvalus study. In order to use the Calvalus data services users must be registered and signed-in. After signing in, users see product sets they own and the status of jobs they are currently running. From a menu users can choose to perform a data catalogue query, to manage product sets and production requests. The menu items lead to dedicated portal pages.

The **catalogue query page** lets a user query data products by criteria based on the product's catalogue metadata. The resulting product set can be given a name and saved for later use. The new product set that can be used as input to various production scenarios. Users can enter search criteria such as an area of interest, a time range, a data product type or a simple name pattern. The output of a catalogue query is a named product file set owned by the user.

The **manage product file sets page** lets a user select, view, edit, delete and download owned or public product sets. Viewing includes display of product set properties, such as the number and the total size of the associated product files. Owned product sets may result from a catalogue query or as output of a production job. After selecting a product file set, the page offers the following functions

- View a product set.
- Rename an owned product set.
- Delete an owned product set
- Download a product set

The **manage production requests page** lets a user load & save, create, edit, delete production (request) templates and finally submit production requests. Creating new requests is basically done

by cloning existing production templates. Users can select from pre-defined production templates related to all considered production scenarios including match-up & trend analysis, L2 & L3 processing. After selecting an owned production template, the page offers the following functions

- Edit and save production template (rename, change configuration parameters)
- Select a standard processor or a new or updated code or executable
- Select and set processing parameters (e.g. pixel mask, glint and cloud thresholds, vicarious calibration gains), or load key value parameter text file to set parameters
- Submit production request

After acceptance of the production request by the system, a server-side production job is created. The portal delegates to the **manage production jobs page**. It allows users to track their instantiated production jobs. The page offers users to

- Observe the status and details of a running production job
- Cancel a running production job

Successfully completed production jobs may result in new, user owned product sets.

6.2.3 Interfaces

The portal offers a number of external, graphical user interfaces which are described in section 7.1. The portal depends on the interfaces exposed by

- The Catalogue and Inventory Service (see 7.2, 7.3),
- the Production Service (see 7.4) and
- the Staging Service (see 7.6).

6.2.4 Data

The user portal maintains user-related information:

- user authentication information
- user contact information, e-mail

6.3 Catalogue and Inventory Service

This section describes the logical component *catalogue and inventory service* of Calvalus.

6.3.1 Purpose

The catalogue and inventory service is the place for metadata and collection information in Calvalus. It hosts metadata of EO products, of reference data and of auxiliary data, serves queries, and it maintains predefined collections and user-defined product sets. Besides temporal and spatial coverage the metadata comprise product file information available from the respective product file types. For L1, this includes e.g. percentages of land/ocean or of bright pixels. If L2 product files are catalogued, also percentage of pixels marked by L2 flags like cloud and glint may be provided.

6.3.2 Function

The computational service provided by the catalogue and inventory service is:

- product file identification, each product file gets a unique identifier in Calvalus
- catalogue search based on metadata including temporal and geo-spatial criteria and on predefined collections or user-defined product sets
- presentation of results with detailed metadata records, thumbnails

- inventory lookup to locate products in the archive, translate from identifier to physical archive location
- creation of product sets based on query results (virtual sets) and as container for processing outputs (output sets)
- automated retention of product sets after retention time
- maintenance of catalogue product metadata (get, add, remove/mark, update)
- maintenance of inventory product locations (get, add, remove/mark, update)
- maintenance of collection and product sets (get, add, remove/mark, update)

6.3.3 Interfaces

The catalogue and inventory service exposes two interfaces:

- Catalogue interface with functions for search and present, addition, update, deletion of products and of product sets (see 7.2)
- Inventory interface with functions to locate, to add, update and delete an entry (see 7.3)

It uses one interface:

- Archive (HDFS) interface to delete files after retention time

6.3.4 Data

The catalogue and inventory service hosts metadata, product sets and archive locations:

- Catalogue of product metadata with database for efficient search and optional files with original metadata record and thumbnails
- Catalogue of reference data, in-situ database(s), auxiliary data, maintained like products
- Catalogue of collections and product sets with collection metadata and product membership, retention information
- Inventory of references to files in the archive, HDFS URLs to files or directories

The data is encapsulated into the catalogue and inventory service and is accessible via its interfaces.

6.4 Production Service

This section describes the logical component *production service* of Calvalus.

6.4.1 Purpose

The production service manages and controls production processes for the generation of new products within Calvalus. It handles production requests from users, maintains production recipes and organises processing chains, and ensures cataloguing and inventorisation of results.

6.4.2 Function

The computational service provided by the production service is:

- Production request handling, generation of production jobs, maintenance and provision of their states, command handling (cancellation)
- Production job execution by translation into one or more processing steps, driven by production recipes
- Issue of processing requests for steps to the Hadoop MapReduce engine and monitoring
- Interaction with catalogue and inventory service to resolve product sets, to get product file locations, to create result product set, to catalogue and inventorise results
- Production failure handling

- Maintenance of production request templates (get, add, remove, update) to be used for request composition by users in portal
- (optional) Automated retrieval of requested processor version from repository and deployment on Hadoop cluster
- Maintenance of processor updates and processor versions

6.4.3 Interfaces

The production service exposes one interface:

- Production interface with functions to submit requests, browse and control jobs, retrieve and user-define production request templates (see 7.4)

It uses interfaces:

- Hadoop job interface for submission and monitoring (see 7.8)
- Catalogue and inventory interfaces for product set resolution, product location determination, product result set creation, and cataloguing and inventorisation of results (see 7.2, 7.3)
- (optional) Code repository for versioned processor installation package download

6.4.4 Data

The production service maintains current production jobs and information related to production:

- current production jobs that are results from submitted requests, with state and progress information, ongoing processing steps
- history of done jobs
- Production recipes that implement production scenarios
- Production request templates, system-provided and user-defined

The data is encapsulated into the production service and is accessible via its interfaces.

6.5 Ingestion and Staging Service

This section describes the logical component *ingestion and staging service* of Calvalus.

6.5.1 Purpose

The ingestion and staging service is the data gateway of Calvalus. It implements both ingestion of new EO products and reference data into the system and access to produced and archived data by staging into a user-accessible download area.

6.5.2 Function

The computational service of the ingestion and staging service provided for ingestion is:

- Extraction of metadata
- Validation of inputs
- Thumbnail generation
- Archiving rules application to determine archive location
- Consistent archiving, inventorying and cataloguing

The computational service provided for staging is:

- Data retrieval from archive
- Formatting of output products from distributed concurrently generated part files

- Data analyses, plot generation, statistics generation, provided by plug-ins (see also section 6.9 Processor)
- Data provision on staging area (in order to isolate the archive from direct user access)
- Notification of data provision
- Deletion of data from staging area after successful retrieval

The formatting function in particular converts temporary partial outputs into user formats like NetCDF, GeoTIFF, BEAM-DIMAP.

6.5.3 Interfaces

The ingestion and staging service exposes two interfaces:

- Ingestion command interface with functions to ingest a product or a reference data item (see 7.5)
- Staging interface with functions to retrieve a result from archive with optional formatting (see 7.6)

It uses interfaces:

- Hadoop HDFS interface for directory and file access (see 7.7)
- Catalogue interface for catalogue insertion, product set update (see 7.2)
- Inventory interfaces for product location registration (see 7.3)

6.5.4 Data

The ingestion and staging service maintains only configuration information:

- Archiving rules to determine archive path from product metadata
- Configured formatters for different result part formats

6.6 Hadoop Distributed File System (HDFS)

This section describes the logical component *Hadoop distributed file system (HDFS)* of Calvalus.

6.6.1 Purpose

The Hadoop distributed file system serves as archive for primary and auxiliary input and output data products. On the data provider and user side the data in the archive is encapsulated by the ingestion and staging service. On the processor side it is accessed locally or remote via the Hadoop-to-processor adapter in a controlled way.

6.6.2 Function

The computational service of the HDFS is:

- File system functions to store files, to organise them in directories (create, read and delete files; create, list and delete directories)
- Data replication to different nodes to improve fail safety and to support data locality
- Distributed data access to support data locality

The functions are accessible by the Hadoop namenode and a client API.

6.6.3 Interfaces

The HDFS exposes

- HDFS interface via a Java API and command line tools (see 7.7)

The HDFS does not use any other service's interface.

6.6.4 Data

The HDFS service maintains the archive data which comprises:

- The directory tree
- The product files

The archive stores a product either as a single file or as a directory of files. The latter is the case at least for all outputs generated by concurrent mappers and reducers on the cluster. For these products a formatter in the ingestion and staging service composes the user product.

6.7 Hadoop MapReduce Engine

This section describes the logical component *Hadoop MapReduce engine* of Calvalus.

6.7.1 Purpose

The Hadoop MapReduce engine is the cluster scheduler and the workflow engine for the map-reduce programming model. It distributes tasks to the cluster of computing nodes in a way that obeys data-locality.

6.7.2 Function

The computational service of the Hadoop MapReduce engine is:

- Parallelisation, creation of concurrent tasks for a Hadoop job with a set of inputs
- Distributed processing, scheduling of tasks on the cluster of processing nodes
- Data-locality, considering data-locality for scheduling
- Orchestration of map and reduce tasks, partitioning and sorting (re-shuffle) of intermediates
- Monitoring of task execution, status handling
- Failure handling with automated retry (failover)
- Speculative execution (preventive failover)

6.7.3 Interfaces

The Hadoop MapReduce engine exposes:

- Hadoop Job API with functions to submit and monitor Hadoop jobs (see 7.8)

It uses:

- Hadoop MapReduce implementation interface with functions for data access organisation and to run processing (see 7.9)

6.7.4 Data

The Hadoop MapReduce engine maintains current Hadoop jobs and their distributed status on the cluster:

- current Hadoop jobs with state and progress information, ongoing Hadoop tasks (processing steps)
- history of done Hadoop jobs

The data is encapsulated into the Hadoop MapReduce engine and is accessible via its interfaces.

6.8 Hadoop-to-Processor Adapter

This section describes the logical component *Hadoop-to-processor adapter* of Calvalus.

6.8.1 Purpose

The adapter integrates existing processors into Calvalus and binds them to the Hadoop MapReduce engine. The adapter is foreseen in two variants, for BEAM GPF processors and for executable/shell script processors. The adapter further serves as an example or pattern for the implementation of specific Calvalus processors that directly interface to the Hadoop MapReduce engine.

6.8.2 Function

Figure 6-2 shows two variants of the Hadoop-to-processor adapter in order to bind different types of processors to the MapReduce engine: one for BEAM GPF operator processors and one for executable shell script processors.

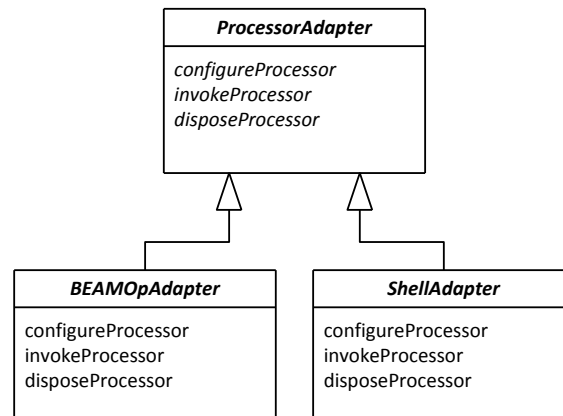


Figure 6-2: BEAM and shell script variants of the Hadoop-to-processor adapter

The computational service of the Hadoop-to-processor adapter is:

- Transformation of the Hadoop MapReduce implementation interface to interfaces of existing processors
- Invocation, control and monitoring of the processor
- Parameter provision as method parameters or parameter file
- Input data product provision as input streams or local copies of files
- Output data product archiving provided as output streams or local files
- Preparation of the environment before processing and cleanup of local files after processing
- Status propagation

6.8.3 Interfaces

The Hadoop-to-processor adapter exposes:

- Hadoop MapReduce implementation interface with functions for data access organisation and to run processing (see 7.9)

It uses:

- HDFS file system Java API with access (input, output) to data via streams (see 7.7)
- BEAM GPF operator interface for BEAM processors (see 7.10.1)
- Unix shell interface in order to integrate existing non-BEAM processors (see 7.10.2)

6.8.4 Data

The Hadoop-to-processor adapter maintains only configuration information:

- Processor recipes to define type information for expected and optional inputs, outputs, and protocol information (BEAM operator, shell script)

6.9 Processor

This section describes the logical component *processor* of Calvalus. Two instances of Level 2 processors are Case2R [RD 7] and QAA [RD 8], another optional processor is l2gen [RD 16]. Furthermore, other processors will perform data analyses, plots and statistics if parallelisation is applicable to their respective function.

6.9.1 Purpose

The processor implements the algorithm to transform input data into output data. It shall be executed on the cluster to perform a processing step. There are different types of processors for different algorithms. Processors are versioned.

6.9.2 Function

The computational service of the processor is:

- Transformation of inputs to one or more outputs in a processing step considered as atomic. Outputs may be EO products of a higher level or reports.
- Data analyses, plot generation, statistics generation (if parallelised)
- Status provision

Processors should provide metadata for their output, e.g. as part of it.

6.9.3 Interfaces

The interface of the processor depends on its type. It exposes:

- BEAM GPF Operator interface for BEAM processors (used for Case2R and QAA, see 7.10.1)
- Unix shell interface in order to integrate existing non-BEAM processors (optionally used for l2gen, see 7.10.2)

The processor does not use any other service's interface.

6.9.4 Data

The processor maintains only configuration information:

- Processor configuration
- Auxiliary data
- Input/output parameters and product files of the current processing request

7 External and Internal Interfaces

This chapter identifies the external and internal interfaces between users and Calvalus and between Calvalus components. Main aspects are identification, characterisation of the interface by functions provided or interface items exchanged, and the exchange protocol where helpful. Purpose of the section is to provide enough details to explain the interaction between components. As most of the interfaces are internal interfaces, decisions about details are intentionally left open to the implementation.

The most visible interface of Calvalus is the graphical user interface of the user portal.

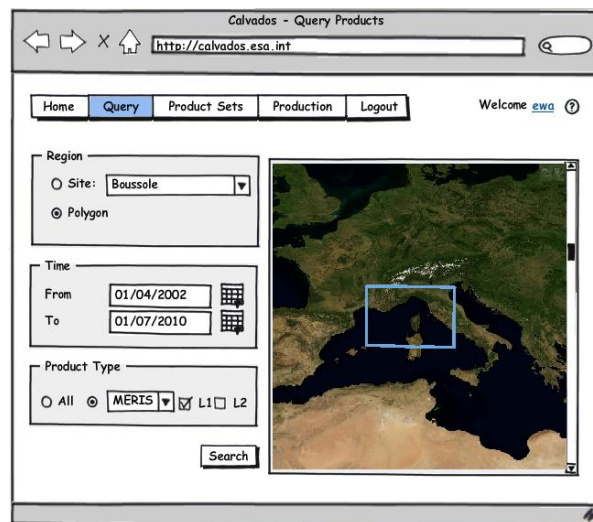
7.1 User Web Interface

The Calvalus user portal enables the user query the available EO data products, build up product sets, specify and run processing jobs and special analyses on these product sets and download the results of these processing jobs.

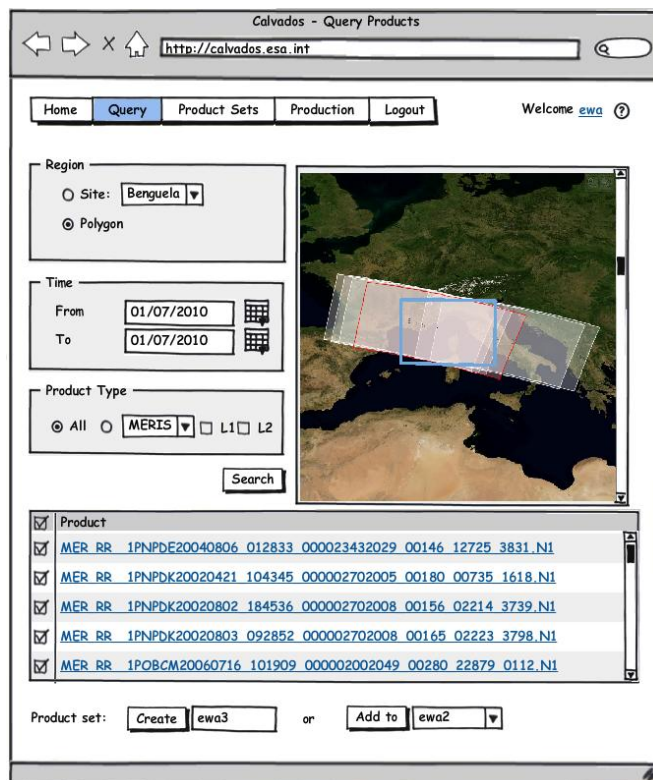
At the writing of this document, the user interface has not been specified in great detail. Thus, the user interfaces presented here are mock-ups that shall give a hint on how they may look like in the final system.

7.1.1 Query products

To build up the product set that is used for the processing the user enters his constraints on the region, the time range and the product type. The region of interest (ROI) may e.g. be entered as a list of lat/lon coordinates. The same ROI can further be used as a processing parameter to extract and process only this region from the inputs.



After pressing the search button, the results are displayed:



Calvados - Query Products

http://calvados.esa.int

Home Query Product Sets Production Logout Welcome ewa

Region

Site: Benguela

Polygon

Time

From: 01/07/2010

To: 01/07/2010

Product Type

All MERIS L1 L2

Search

Product

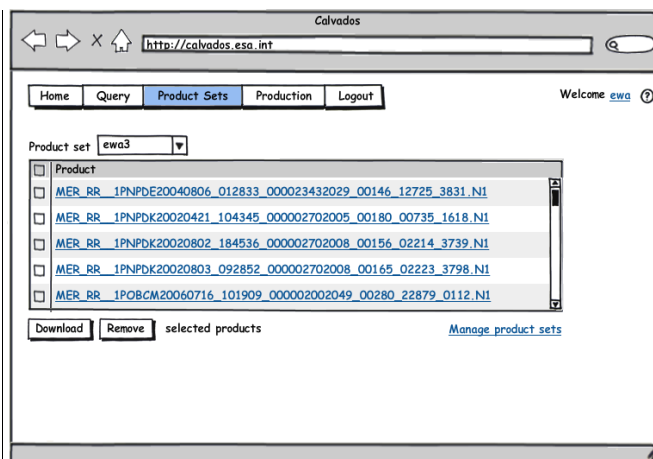
<input checked="" type="checkbox"/>	MER_RR_1PNPDE20040806_012833_000023432029_00146_12725_3831.NI
<input checked="" type="checkbox"/>	MER_RR_1PNPDK20020421_104345_000002702005_00180_00735_1618.NI
<input checked="" type="checkbox"/>	MER_RR_1PNPDK20020802_184536_000002702008_00156_02214_3739.NI
<input checked="" type="checkbox"/>	MER_RR_1PNPDK20020803_092852_000002702008_00165_02223_3798.NI
<input checked="" type="checkbox"/>	MER_RR_1POBCM20060716_101909_000002002049_00280_22879_0112.NI

Product set: Create ewa3 or Add to ewa2

The search result can be either converted into a new product set or added to an existing one, optionally checking whether the products of the set are all of a common type. The individual products making up such a result can be listed on a separate page.

7.1.2 View product set

This interface is used to view the details, namely the names of contained products, of a selected product set. Users may download the entire product set or single products.



Calvados

http://calvados.esa.int

Home Query Product Sets Production Logout Welcome ewa

Product set ewa3

Product

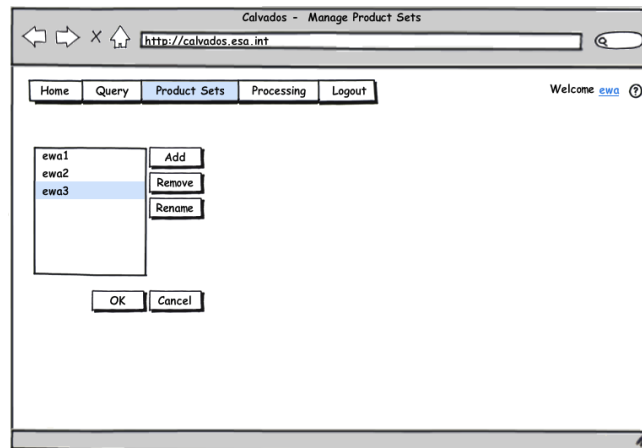
<input type="checkbox"/>	MER_RR_1PNPDE20040806_012833_000023432029_00146_12725_3831.NI
<input type="checkbox"/>	MER_RR_1PNPDK20020421_104345_000002702005_00180_00735_1618.NI
<input type="checkbox"/>	MER_RR_1PNPDK20020802_184536_000002702008_00156_02214_3739.NI
<input type="checkbox"/>	MER_RR_1PNPDK20020803_092852_000002702008_00165_02223_3798.NI
<input type="checkbox"/>	MER_RR_1POBCM20060716_101909_000002002049_00280_22879_0112.NI

Download Remove selected products Manage product sets

Clicking on single products displays individual metadata.

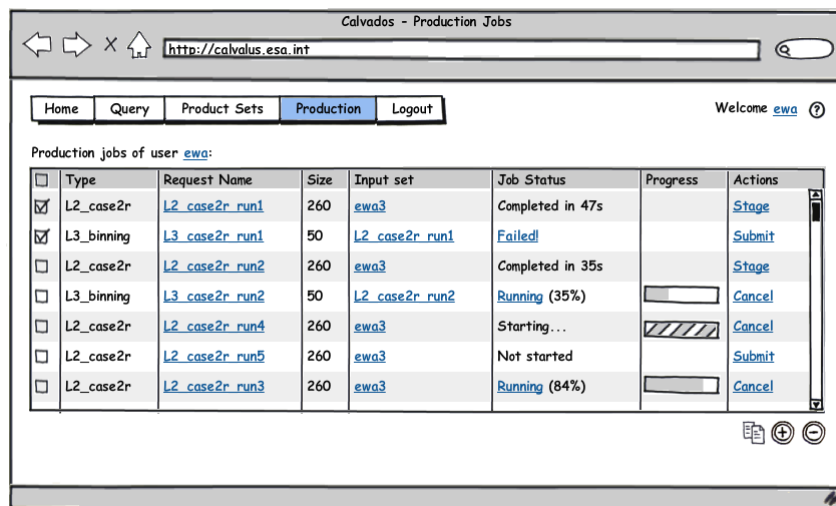
7.1.3 Manage product set

Users can manage their own products sets using the standard operations add, remove and rename:



7.1.4 Manage production jobs

The following form is used for managing owned production jobs. It displays the result of completed jobs, the error message for jobs that failed and the completion ratio for running jobs.



Each production job generates as its result either a new product set of L2 or L3 products or a test report. The name of the job is equal to the name of the processing request that created the job. The name of the resulting product again is equal to the name of production job.

7.1.5 Edit production request

The individual pages for editing the parameters of a processing request are dependent on the processing type. After a request has been submitted, the parameters cannot be changed anymore. The following mock-ups show the forms for editing a L2 and a L3 request. Production requests may be saved as a template for later use. The following mock-up shows the form used to edit a L2 production request. The parameters shown are examples only. Different algorithms have different parameters that will be shown in the respective form.

Calvalus - Edit Production Request

http://calvalus.esa.int

Home Query Product Sets **Production** Logout Welcome ewa ?

Recipe: L2_case2r

Name: l2_case2r_run5

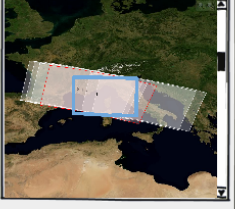
Description: Some text
A second line of text

Input set: ewa3

Region

Site: Boussole

Bounding box



Processing Parameters - L2 Case2r

Inverse NN file: meris_bn_20040322_45x12x8x5_5177.9.net

Reverse NN file: meris_fn_20040319_15x15x15_1750.4.net

☒ Perform atmospheric correction

Atm. corr. NN file: atmo_net_20091105_25x30x35x40_11415.7.net

☒ Perform polarisation correction

Pol. corr. NN file: 18_518.1.netPolEffekt

Input mask: NOT l1_flags.INVALID

Land water mask: toa_reflec_10 > toa_reflec_6 AND toa_reflec_13 > 0.0475

Cloud ice mask: toa_reflec_14 > 0.2

Save As Template Submit

The next mock-up shows the form used to edit a L3 production request.:

Calvalus - Edit Production Request

http://calvalus.esa.int

Home Query Product Sets **Production** Logout Welcome ewa ?

Recipe: L3_binning

Name: L3_case2r_binning_run2

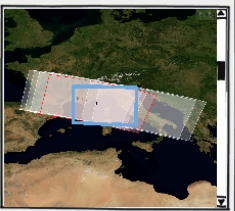
Description: Some text
A second line of text

Input set: l2_case2r_run5

Region

Site: Boussole

Bounding box



Processing Parameters - L3 Binning

Averaging period: 3 days

Grid cell size: 9.28 km

☐ Remove input product set after processing

Variable	Valid expression	Aggregation	Weight coeff.
a_gelbstoff	NOT l2_flags.INVALID	Arithmetic Mean	1
a_pig	NOT l2_flags.INVALID	Arithmetic Mean	1
a_tsm	NOT l2_flags.INVALID	Arithmetic Mean	1

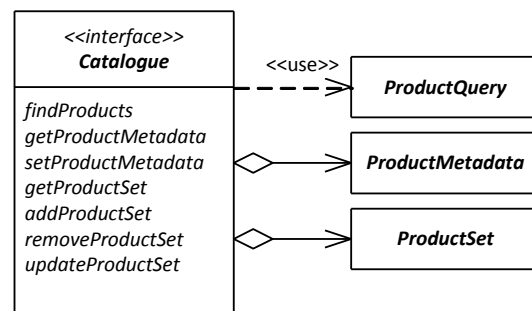
Save As Template Submit

7.2 Catalogue Interface

The Calvalus catalogue links metadata to each registered product in the Calvalus system. This allows for fast data products look-ups given a query that comprises various metadata search criteria (operation findProducts).

The interface also allows for getting and setting the metadata associated with each EO data product (operations getMetadata, setMetadata).

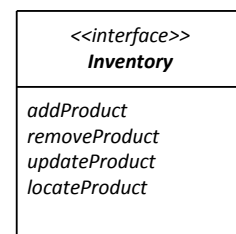
Furthermore, the catalogue hosts all product sets known in the system. Thus, the interface also provides functions to manage product sets (operations getProductSet, addProductSet, removeProductSet, updateProductSet).



7.3 Inventory Interface

The Inventory comprises references to all registered EO data products in the archive. The interface allows for adding, removing and updating registered products (operations addProduct, removeProduct and updateProduct).

Another very important function of the Inventory interface is to translate a product reference into a physical product path pointing into the Calvalus archive (operation locateProduct).



7.4 Production Interface

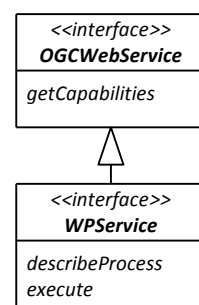
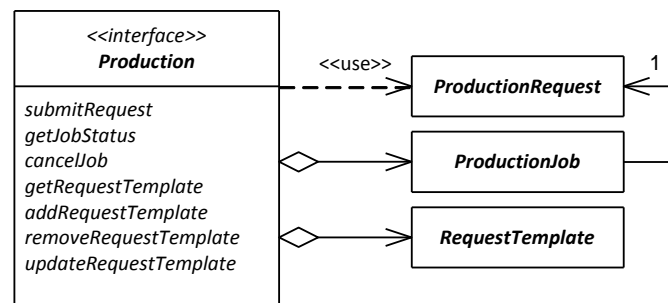
The basic function of the Production interface is to let clients submit production requests (operation submitRequest). A production request identifies the production recipe to be used and the values for all required input parameters.

The response for accepted production

component provides a reference to a production job. The production interface allows for retrieving detailed status information about that job and also allows for the cancellation of that job (operations getJobStatus and cancelJob).

Finally, the Production interface is used to manage templates for production requests (operations get-, add-, remove- and updateRequestTemplate). A template request basically a more or less incomplete requests with respect to the required parameter values.

It is envisaged to wrap the final version of the production component by an OGC Web Processing Service [RD 18], brief WPS, allowing Calvalus to be easily incorporated into existing geospatial infrastructures based on a web service oriented architecture (e.g. ngEO - Next Generation User Services For Earth Observation, in planning). A WPS defines a standardized interface that facilitates the publishing of geospatial processes, and the discovery of and binding to those processes by clients. It basically comprises three operations that all WPS must implement. First, an operation that allows a client to request



metadata about the general service's capabilities and the processes that it supports (operation `getCapabilities`, common to all OGC web services). Second, an operation that allows a client to request a detailed description of a named process (operation `describeProcess`) and third, an operation that executes such a process (operation `execute`).

7.5 Ingestion Interface

Calvalus system operators use the *ingest tool* to register and archive EO data products in the Calvalus system. The tool comprises a simple command-line interface whose arguments are the source directory that are scanned for EO data products:

```
ingest [<options>] <dir1> <dir2> ...
```

The interface provides options that control the final location in the HDFS of each EO data product found. There are also various options that control the behaviour of the ingestion, e.g. whether to overwrite existing products or to perform a dry run for testing purposes.

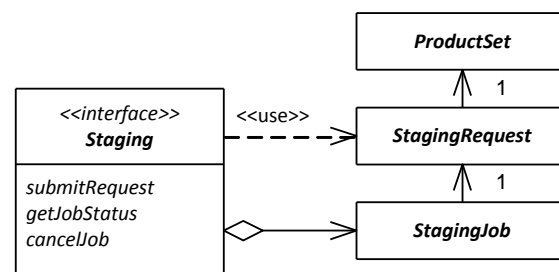
The ingestion process first extracts metadata and performs some validation on each source product. In order to re-ingest data products that are already in the archive, the ingestion tool may be run on existing directories in the HDFS:

```
reingest [<options>] <dir1> <dir2> ...
```

The re-ingestion is an important tool to be applied to existing data after the metadata extraction has been changed and/or after structural changes in the catalogue database.

7.6 Staging Interface

The Staging interface lets clients submit a staging request that identifies the product set to be delivered and optional staging parameters, e.g. the user notification scheme or the delivery format (operation `submitRequest`). The response for an accepted staging request provides a reference to a staging job. The Staging interface allows for retrieving detailed status information about that job and also allows for its cancellation (operations `getJobStatus` and `cancelJob`).



7.7 HDFS Interface

The HDFS is accessible via a dedicated Java API and a POSIX-like shell command-line interface (CLI). It is described in detail on the Apache Hadoop project page hadoop.apache.org:

HDFS Java API: <http://hadoop.apache.org/hdfs/docs/r0.21.0/api/>

HDFS CLI: `hadoop fs [<generic-options>] [<command-options>]`
 where `<generic-options>` are described at http://hadoop.apache.org/common/docs/current/commands_manual.html
 and `<command-options>` at http://hadoop.apache.org/common/docs/current/file_system_shell.html

7.8 Hadoop Job Interface

The Hadoop Job interface comprises dedicated Java classes in the Hadoop MapReduce API. The API is described on the Apache Hadoop project page hadoop.apache.org.

MapReduce Java API: <http://hadoop.apache.org/mapreduce/docs/current/api/>

The actual Java Job interface is provided by the class

```
org.apache.hadoop.mapreduce.Job
```

and its associated classes and interfaces. Clients create a Job object and configure it with their mapper, reducer and partitioner functions as well as with their input and output formats. Then they call the Job's submit method (Figure 7-1).

```
Job job = new Job(configuration);
job.setJobName("MyJob");
job.setJarByClass(MyJob.class);
job.setInputClassName(MyJob.MyInputFormat.class);
job.setOutputClassName(MyJob.MyOutputFormat.class);
job.setMapperClass(MyJob.MyMapper.class);
job.setReducerClass(MyJob.MyReducer.class);
job.submit(true);
```

Figure 7-1: Code snippet of a Job configuration

After compilation of the Java code that submits a Job to the Hadoop cluster, the Job may be directly executed on the cluster by using the Hadoop Job command-line interface (CLI).

Hadoop Job CLI: `hadoop job [<generic-options>] [<command-options>]`

where <generic-options> are described at

http://hadoop.apache.org/common/docs/current/commands_manual.html

and <command-options> at

http://hadoop.apache.org/common/docs/current/commands_manual.html#job

7.9 Hadoop MapReduce Implementation Interface

We refer to the Hadoop MapReduce "implementation" interface as the set of abstract Java classes that developers must derive from, in order to implement a Job's executable code. Together with the Job class, these classes are all part of Hadoop MapReduce API described on the Apache Hadoop project page hadoop.apache.org.

MapReduce Java API: <http://hadoop.apache.org/mapreduce/docs/current/api/>

The actual Java MapReduce implementation interface is provided by the following classes of the package `org.apache.hadoop.mapreduce`. The most important interfaces clients must implement are the Mapper and Reducer classes. Applications typically extend them to provide the map and reduce functions.

Mapper	Maps input key/value pairs to a set of intermediate key/value pairs.
Reducer	Reduces a set of intermediate values which share a key to a smaller set of values.
Partitioner	Partitions the key space.

Other core classes of the API are dedicated to the input and output formats that a specific Job reads and writes.

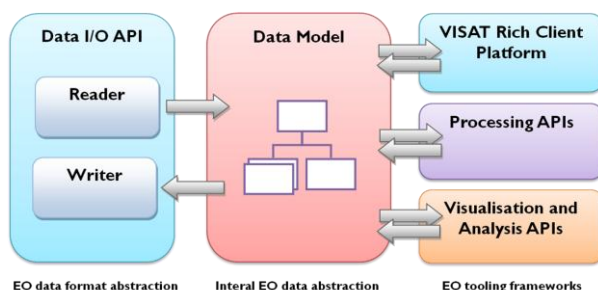
InputFormat	Describes the input-specification for a Map-Reduce job in terms of InputSplit and RecordReader.
InputSplit	Represents the data to be processed by an individual Mapper.
RecordReader	Breaks the data of an InputSplit into key/value pairs for input to the Mapper.
OutputFormat	Describes the output-specification for a Map-Reduce job in terms of OutputCommitter and RecordWriter.
OutputCommitter	Describes the commit of task output for a Map-Reduce job.
RecordWriter	Writes the output <key, value> pairs to an output file.

7.10 Processor Interfaces

Calvalus supports two types of data processors that take a single input product and generate a single output product. They are the BEAM GPF Operator Interface and a simple shell interface, which are described in more detail below.

7.10.1 BEAM GPF Operator Interface

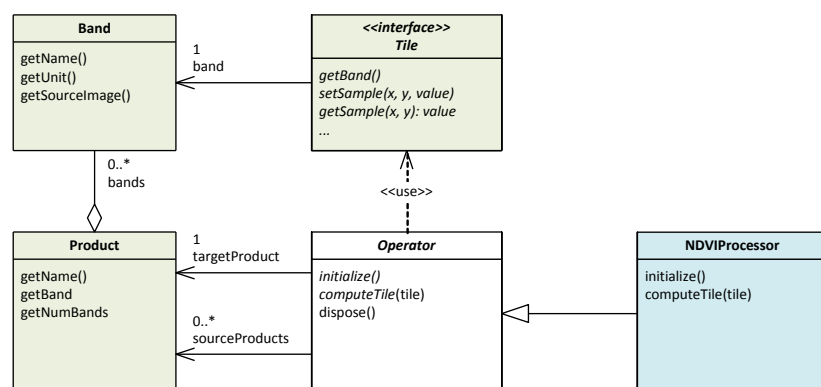
The BEAM GPF Operator interface is part of the official BEAM development platform. It allows developers to easily implement new data processors using a very effective programming model. Processors developed against this interface are bound to MapReduce using the Calvalus BEAM Adapter.



One of the most important concepts of the BEAM development platform is the simple abstraction of remote sensing products from their external file format. Product readers create instances of a product data model. Once the data model is instantiated in memory, various higher-level BEAM APIs and frameworks can use it for data processing, analysis and visualisation.

One of those higher-level APIs is the BEAM Graph Processing Framework (GPF). It allows for developing new operators that can later serve as nodes in processing graphs. Developing a new operator is basically implementing the Operator interface of the BEAM GPF. The interface comprises basically three operations

that are called by the framework. The first initialises the operator and defines a target product including all target bands (variables) to be computed (operation initialise). The second operation is used to compute all the pixels for a given tile. The tile



represents a part of a target band's raster data (operation computeTile). The last operation is called by the framework in order to allow an operator to release its allocated resources (operation dispose), e.g. file pointers to auxiliary data.

7.10.2 Shell interface

The shell interface allows for the incorporation of executables that can be invoked from a commands line shell and that do not have any user interactions. The interface comprises a process descriptor file. This is a plain text file (XML) that describes the inputs files (name and type), the processing parameters (name, type, value range), the output file (name and type) and provides a template for the command-line that is used to invoke the executable.

Executables that provide a process descriptor are bound to MapReduce using the Calvalus Shell Adapter.

8 System Process Design

This chapter describes the design of processing workflows on the system based on Hadoop parallelisation and map-reduce. The two most important since time-consuming processes are L1-to-L2 processing and L2-to-L3 processing. The approach for L1-to-L2 is simple parallelisation of map-only tasks. The approach for L2-to-L3 is to use map-reduce for parallelising on inputs and on geography.

8.1 Concurrent L1-to-L2 Processing on HDFS

This chapter describes the generation of L2 products from L1 products on the Hadoop Distributed File System (HDFS).

The processing of a set of L1 input product into a set of corresponding L2 output products belongs to the class of problems that can be parallelized over the input data without the need for data aggregation afterwards. Special care has to be taken to also utilize the power of the Hadoop cluster, when only processing a single input file.

8.1.1 Approach

The distribution of the workload for the map function on a Hadoop cluster is done by the split function. This function partitions the data that is stored in blocks on HDFS into chunks of data that are processed by a single mapper.

Our analysis has shown that when processing a whole set of products from L1 to L2 the best approach is to process a single product by a single mapper. In this case the split function is trivial as the input product files are stored in one HDFS block each. The resulting split contains a complete input product that is then processed by a single mapper (see Figure 8-1). The computation will be executed, whenever possible, on a cluster node that has a replica of the data.

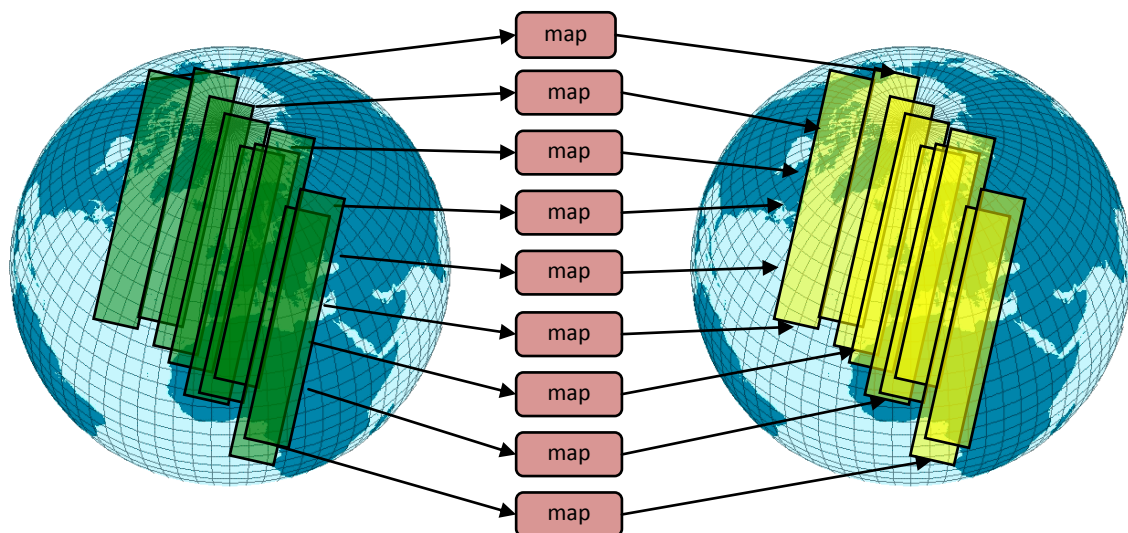


Figure 8-1: Concurrent mappers for L2 processing of a product set

When only one product has to be processed this approach would result in a single mapper processing the input product on a single node. So there would be no advantage of using a cluster. In this case multiple splits are created to foster parallel processing on multiple nodes (see Figure 8-2).

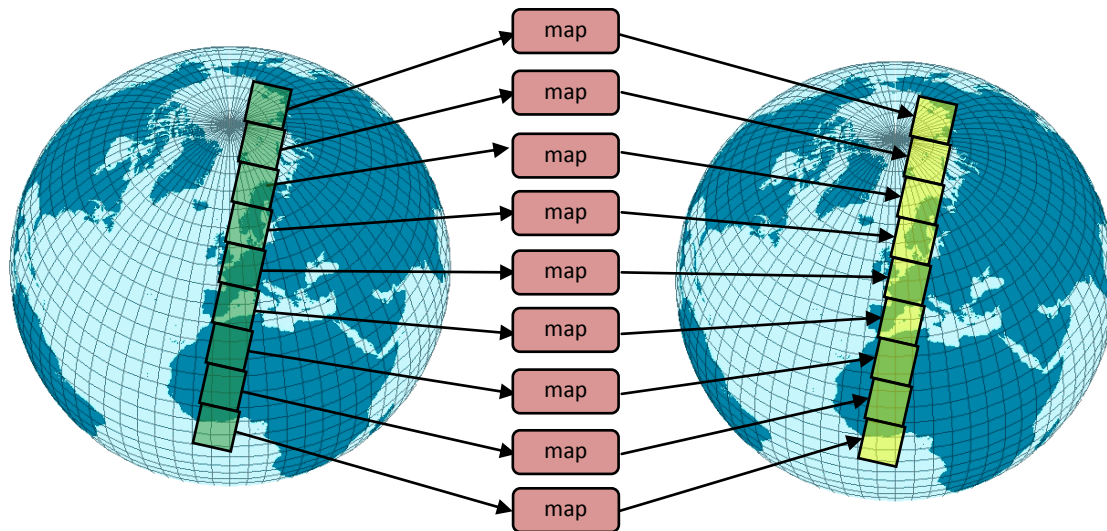


Figure 8-2: Concurrent mappers for L2 processing of a single product

As this leads to many nodes processing parts of the input product that have no data stored locally this is only useful when the computation time outweighs the time for the data transfer. Our study has shown that for reasonably complex algorithm like the Case2R this splitting is a performance improvement.

8.1.2 BEAM Processors

The mapper has direct access to the data stream, which can be used by the standard BEAM reader for Envisat N1 data products. In the next step the GPF operator will be created with the given parameters. And the input product or the created subset will be used for the input product. The resulting output product of the operator is then directly written into the HDFS.

In the case where the input product is processed in multiple parts (as described above) the processing differs a bit. From the input product created by the Envisat reader a subset covering only the required region is created which is then processed by the operator. Each mapper will write a part of the product to a unique output file. The product will be merged into one product once it is copied into the staging area.

The mapper is generic in respect to the utilized BEAM GPF operator. Any GPF operator that takes a single input product can be used. The differentiation can be done using configuration.

8.1.3 Shell Executables

In contrast to processors written in the Java programming language, operational processors that are available as C code or as an executable rely on the local file system for reading its input product and writing its output product. These processors are supported using a shell interface. The executable must be available on all nodes of the cluster and it must read the native Envisat N1 product format. First the input product is copied from HDFS onto the local disk before the executable is invoked using the local product copy and the specified parameters as arguments. Afterwards the resulting output product is copied back into the HDFS.

8.2 L2-to-L3 Processing with MapReduce

This section describes the design of the L3 composite generation as concurrent map-reduce process. It implements the kernel part of the Level 3 processing scenario (section 3.1.4, Figure 3-9). The data aggregation is based on the spatial and temporal binning as described in [RD 11]. The binning

generates a product in sinusoidal projection [RD 13]. It carefully considers how to aggregate and weight values. Because of its large unordered set of inputs and its geographically ordered output the process is a good candidate for map-reduce.

8.2.1 Approach

Figure 8-3 shows how concurrency can be utilised to generate a single L3 product.

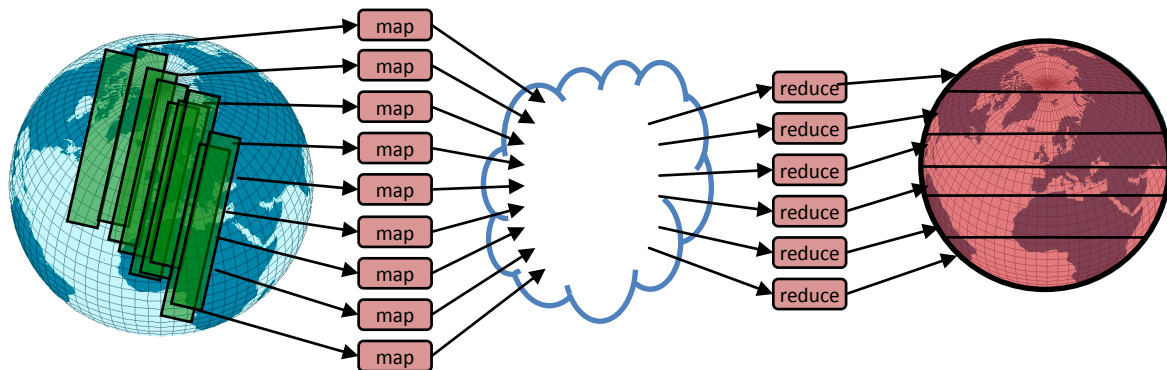


Figure 8-3: Concurrent mappers for inputs and concurrent reducers for regions

The approach is characterised by

- a *mapper* for each L2 input doing spatial binning, that generates intermediate data for bin cells with weight information, using the bin cell ID as key
- *partitioning* into ranges of bin cells, ranges to cover the region of interest
- a *reducer* for each partition doing temporal binning for every bin cell in the range, writing a segment of the output
- a *formatting* tool to create the user output from the distributed output segments

This is depicted in Figure 8-4 that also characterises the structure of the intermediate data.

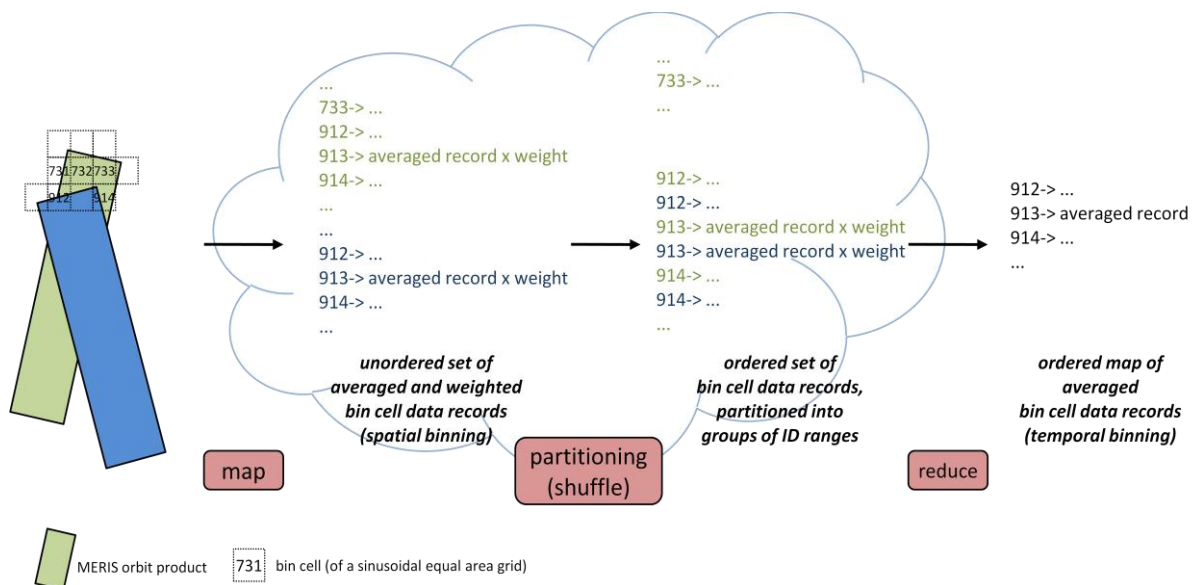


Figure 8-4: Inputs, intermediates and outputs of map and reduce for L3 binning

8.2.2 Data structures

Figure 8-5 shows an example of a request to generate a L3 product.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<wps:Execute service="WPS"
  version="1.0.0"
  xmlns:wps="http://www.opengis.net/wps/1.0.0"
  xmlns:ows="http://www.opengis.net/ows/1.1"
```

```

    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 ../wpsExecute_request.xsd">
<ows:Identifier>L3-generation</ows:Identifier>
<wps>DataInputs>
  <wps:Input>
    <ows:Identifier>input</ows:Identifier>
    <ows:Title>binning input</ows:Title>
    <wps:Reference xlink:href="#hdfs:/MER_RR__1P/v1.0/2010/08/23/MER_RR__1PNPDE20100823_013513_000026322092_00189_44331_6800.N1" />
  </wps:Input>
  <wps:Input>
    <ows:Identifier>input</ows:Identifier>
    <ows:Title>binning input</ows:Title>
    <wps:Reference xlink:href="#hdfs:/MER_RR__1P/v1.0/2010/08/23/MER_RR__1PNPDE20100823_031550_000015262092_00190_44332_6793.N1" />
  </wps:Input>
  <wps:Input>
    <ows:Identifier>input</ows:Identifier>
    <ows:Title>binning input</ows:Title>
    <wps:Reference xlink:href="#hdfs:/MER_RR__1P/v1.0/2010/08/23/MER_RR__1PNPDE20100823_034040_000011402092_00190_44332_6825.N1" />
  </wps:Input>
  <wps:Input>
    <ows:Identifier>input</ows:Identifier>
    <ows:Title>binning input</ows:Title>
    <wps:Reference xlink:href="#hdfs:/MER_RR__1P/v1.0/2010/08/23/MER_RR__1PNPDE20100823_045626_000023622092_00191_44333_6826.N1" />
  </wps:Input>
  <wps:Input>
    <ows:Identifier>input</ows:Identifier>
    <ows:Title>binning input</ows:Title>
    <wps:Reference xlink:href="#hdfs:/MER_RR__1P/v1.0/2010/08/23/MER_RR__1PNPDE20100823_053510_000003062092_00191_44333_6860.N1" />
  </wps:Input>
  <wps:Input>
    <ows:Identifier>input</ows:Identifier>
    <ows:Title>binning input</ows:Title>
    <wps:Reference xlink:href="#hdfs:/MER_RR__1P/v1.0/2010/08/23/MER_RR__1PNPDE20100823_063702_000023002092_00192_44334_6853.N1" />
  </wps:Input>
  <wps:Input>
    <ows:Identifier>aoi</ows:Identifier>
    <ows:Title>area of interest to be binned</ows:Title>
    <wps>Data>
      <ows:BoundingBoxData><ows:LowerCorner>120.0 20.0</ows:LowerCorner><ows:UpperCorner>140.0 60.0</ows:UpperCorner></ows:BoundingBoxData>
    </wps>Data>
  </wps:Input>
  <wps:Input>
    <ows:Identifier>resolution</ows:Identifier>
    <ows:Title>size of binning cell</ows:Title>
    <wps>Data>
      <ows:LiteralData uom="meter">9000</ows:LiteralData>
    </wps>Data>
  </wps:Input>
</wps>DataInputs>
<wps:ResponseForm>
  <wps:ResponseDocument storeExecuteResponse="true">
    <ows:Output asReference="true">
      <ows:Identifier>l3-parts</ows:Identifier>
      <ows:Title>part files with averaged records of bin cells</ows:Title>
    </ows:Output>
  </wps:ResponseDocument>
</wps:ResponseForm>
</wps:Execute>

```

Figure 8-5: Example request for L3 processing

The following paragraphs make use of an algebraic specification of data structures using sets, cross products $set1 \times set2$, functions $argset \rightarrow targetset$, subsets $subset \subseteq set$, and elements $element \in set$.

- An instance of a data structure is represented by a tuple of values. Tuples may be nested.
- A set defines a value range for a data structure. Sets of tuples define value ranges for tuples. Sets of tuples correspond to record data structure definitions in programming languages.
- A cross product $set1 \times set2$ is the set of tuples of a value from the first set and a value of the second set. The cross product contains all combinations.
- A function $argset \rightarrow targetset$ is a set of tuples of a value from the argument set and a value of the target set with the restriction that only one tuple for each argument value is contained. The typical implementation of a function in a programming language is a map.
- A subset $subset \subseteq set$ of a set is an enumeration of values of the set.

If e.g. the set is a function $argset \rightarrow targetset$ a subset is a partial function mapping only some of the argument values to some target value.

The output of each mapper is a set of pairs of bin cell IDs and (spatially aggregated) cell data including weights. For MERIS L2 the data comprises the aggregated pixel data for each channel (\bar{X}_{mle} in [RD 11]) and the number of contributing pixels (n in [RD 11]):

$$mapperOutput \subseteq cellId \rightarrow (spatiallyBinnedValues \times n)$$

The reducer's output is a sorted set of bin cells with temporally aggregated data and the number of contributing input products (N in [RD 11]), total contributing pixels (n), and the sum of weights (W [RD 11] in [RD 11]):

$$reducerOutput \subseteq cellId \rightarrow (temporallyBinnedValues \times N \times n \times W)$$

The binned values also comprise statistical information. The concrete representations of these data structures is described together with the functions below.

8.2.3 Input Handling Function

The input is handled by an input format. The input format for L3 processing shall simply use splits as large as each input file. It is assumed that inputs are full orbits as else the files of the same orbit.

The input format further gets the set of file paths from the job and provides the corresponding data streams.

8.2.4 Map Function

The mapper reads its L2 input, loops over the pixels, and aggregates pixels contributing to the same bin. it writes the aggregated value together with the bin ID to the output. In order to avoid too much intermediate memory the algorithm may use the feature of bin areas to be convex and coherent, and it may "open" a bin with the first pixel found to fall into it and "close" a bin cell with the first scan line that does not contribute to it. Figure 8-6 lists the pseudo code of the mapper for spatial binning.

```

algorithm spatialBinning is
  input: MERIS L1B (or L2) orbit product with record of bands per pixel, product
           optional area of interest, geo-object, aoi
           binning grid resolution in meters, binningGridResolution
  output: set of tuples of
           (grid cell ID, tuple of sums of ln values per band, tuple of sums of
            squares of ln values per band, number of contributing records),
           spatialCells
  intermediate: map of grid cells to sets of pixels (y,x) in product, openCells
                  marker map of grid cell ID touched in current line, touchedCells

  initialise spatialCells := empty
  initialise openCells := empty
  for each line y of product do
    initialise touchedCells := empty
    processColumn(y, product, openCells, touchedCells)
    closeUntouchedCells(openCells, touchedCells, product, spatialCells)
  closeUntouchedCells(openCells, empty, product, spatialCells)
  return spatialCells

algorithm processColumn(y, product, openCells, touchedCells, resolution) is
  for each column x in product do
    cellId := determine grid cell ID for x, y of product considering resolution
    skip column if cellId does not intersect aoi
    if openCells does not contain cellId add openCells(cellId) := empty set
    add pair (y,x) to set openCells(cellId)
    add cellId to touchedCells

algorithm closeUntouchedCells(openCells, touchedCells, product, spatialCells) is
  for each cellId in openCells do
    if touchedCells does not contain cellId
      spatiallyBinnedRecord := compute spatial average of openCells(cellId)
      add pair (cellId, spatiallyBinnedRecord) to spatialCells
      remove cellId from openCells

```

Figure 8-6: Map function

The mapper writes its output using standard Hadoop means, i.e. with `IntWritable` for the key and some specific "SpatiallyBinnedWritable" to represent *spatiallyBinnedValues* $\times n$.

8.2.5 Partitioning and Sorting Functions

The partitioning function for L3 processing defines how bin cell IDs are grouped to determine the reducer that will handle them. The function must be parameterised by information from the AOI in order to get reasonable partitions.

For simplicity the maximum and minimum latitude of the AOI determines the range of bin cells that may be covered. This range is equally divided into the number of partitions configured for the job (that should be something less than the number of reducer slots available).

The mapper output shall be sorted ascending by bin cell ID (not a hash of it).

8.2.6 Reduce Function

The reducer gets the outputs of mappers ordered and grouped by the bin cell ID. The reducer loops over the bin cell IDs of the partition it handles and consumes the (ordered) bin cell records it has received. It aggregates the data of each cell with the temporal binning as described in [RD 11]. The result is written to an output file in the output directory. Figure 8-7 lists the pseudo code for temporal binning of the reducer.

```

algorithm temporalBinning is
  input: ordered set of tuples of
           (grid cell ID, tuple of sums of ln values per band, tuple of sums of
            squares of ln values per band, number of contributing records),
           spatialCells
  output: map of grid cell ID to temporally binned record of a value per band,
           temporalCells

  group spatialCells by grid cell ID
  for each group of spatialCells do
    temporallyBinnedRecord := compute temporal average of group
    add pair (cellId, temporallyBinnedRecord) to temporalCells
  return temporalCells

```

Figure 8-7: Reduce function

The concrete data format for the output is internal. It is not necessarily a Hadoop-specific format. It may be organised already similar to the user file format for the output, but arranged in parts.

The output file is a "part" of the overall result. The sum of parts, i.e. the directory containing these parts makes up the complete data output.

8.2.7 Result Formatter Function

The result formatter converts the partitioned output of the reducers into a user product in a suitable file format. It is run independently on request when a user retrieves the output.

The result formatter gets the directory of the map-reduce process as input and creates a file in the local file system as specified in the request. The file format is one of the formats named in section 6.5.

8.2.8 Expected Performance

For large datasets the approach is expected to perform better than processing on a single node:

- Processing on the cluster is expected to be faster by a factor limited by the number of concurrent mappers and the number of concurrent reducers. It is expected that the number

of mappers is more important since the mappers have to read more input (depends on bin cell size, see below).

- Some overhead for distributed tasking can be expected, some overhead for data transfer between mappers and reducers. As binning is not a computationally intensive process the overhead cannot be neglected.
- The process uses data-locality, as the inputs are distributed and mappers should be scheduled on data-local nodes
- The amount of data transferred to reducers is by a factor smaller than the overall input size that corresponds to the ratio between pixel size and bin cell size.

The implementation will show to which extent the expectations can be met.

8.3 Further Processes

The two processing scenarios described in the previous sections are handled as a single production step that involves massive-parallel processing to generate an output product set. The other two production scenarios match-up and trend analysis are of a different kind. They involve larger production chains that usually include a step for bulk L1-to-L2 or L2-to-L3 processing. Such chains are controlled by the Production Service using configured production recipes.

For matchup the production steps include:

- input filtering with respect to the selected reference sites
- bulk L1-to-L2 processing in one production step
- match-up generation
- report generation

For the match-up generation step the input data is distributed on the cluster, and the output may either be a single file, or it may be generated in parts. Generating it in parts corresponds to understand match-up as a sorting-type problem:

- mappers read L2 inputs and write records for the output sorted by reference data site
- reducers combine all records for a reference data site into a single matchup record. Different reducers handle ranges of reference data sites. The result will be part files of the match-up in a directory.

The final step of report generation is not distributable because a single document is generated. It accesses data from the cluster but may run on any node. It is not computationally expensive.

For trend analysis the production steps include

- bulk L1-to-L2 processing
- bulk L2-to-L3 processing to generate several L3 of a time series
- report generation

Like for matchup, such a chain of production steps is controlled by the production service. The computationally expensive steps are the two bulk processing steps. Depending on the number of L3 products to be generated parallelisation by map-reduce for each single L3 output makes sense as it parallelises better if there are fewer L3 than the cluster overall concurrency. So, the parallelisation for trend analysis is the one described in the previous two sections. The final report generation step generates a single output and need not be distributed.

There are some other cal/val problems that may be tackled with the approach:



- Concurrent processing of the same set of products with a variation on processing parameters and subsequent analysis of the differences. This is very similar to trend analysis, not temporal but in the dimension of some varied processing parameter. Massive parallelisation does allow the variation at low cost (measured in time the user has to wait).
- Satellite data inter-comparison is very similar to match-up analysis but with coverages of reference data instead of single in-situ measurements. Each pixel of the reference data set may serve as reference measurement. For this computation becomes more demanding such that parallelisation with map-reduce as described above can help.

As a summary, massive parallel processing considers the transformation of even larger product sets as a single relatively fast production step. This may allow for new approaches for validation.

A. Requirements Traceability

ID	Title	Section	Comment
General LET-SME Requirements			
R-SOW-1.1	Adequate Technology	§2.3 Calvalus Approach for Concurrent Processing §2.2 Hadoop Distributed Computing	
R-SOW-1.2	No Overlap or Extension	n/a	Hadoop for EO as unique feature
R-SOW-1.3	Feasibility and Prototyping	§1.1 Purpose and Scope	
R-SOW-1.4	Application to Space	§1.1 Purpose and Scope	
R-SOW-1.5	Added Value	§1.1 Purpose and Scope §2.3 Calvalus Approach for Concurrent Processing	
Specific SoW Requirements (LET-SME Solicitation Topic)			
R-SOW-2.1	Support of Cal/Val Activities	§1.1 Purpose and Scope §2.1 Cal/Val Application Domain	
R-SOW-2.2	Goals of ESA Cal/Val, GMES, CCI	§2.1 Cal/Val Application Domain	
R-SOW-2.3	Web-based Front End	§6 System Component Design §6.2 User Portal	
R-SOW-2.4	Application on MERIS Data	§3.3 Input Data Requirements	restriction to 4 days per month dropped
R-SOW-2.7	GECA Interface	-	(optional)
Functional Requirements			
General EO-User Portal Requirements			
R-FN-1.1	Portal Services	§6.2 User Portal	
R-FN-1.2	Guest Users	§6.2 User Portal	(desired)
R-FN-1.3	Registered Users and Admins	§6.2 User Portal	Configuration not described on this level
Data Management Requirements			
R-FN-2.1	Query User Interface	§7.1 User Web Interface §6.2 User Portal	
R-FN-2.2	Query Response Presentation	§7.1 User Web Interface §6.2 User Portal	(desired)
R-FN-2.3	Data Download	§6.2 User Portal §6.5 Ingestion and Staging Service	
R-FN-2.4	Stored Queries	§7.1 User Web Interface §6.2 User Portal	Stored queries replaced by concept of user-defined product set
Data Processing Requirements			
R-FN-3.1	L2 and L3 Processing Requests	§3.1 Production Scenarios	

		§6.4 Production Service §7.1 User Web Interface §8.1 Concurrent L1-to-L2 Processing on HDFS	
R-FN-3.2	Different L2 Processors	§3.1 Production Scenarios §6.9 Processor	
R-FN-3.3	L2 Processor Candidates	§3.1 Production Scenarios §6.9 Processor	
R-FN-3.4	Processing Parameter Specification	§3.2 System Use Cases §6.2 User Portal §6.4 Production Service	concept of production request templates
R-FN-3.5	Processing Request Submission	§6.2 User Portal §6.4 Production Service §7.1 User Web Interface	
R-FN-3.6	Processing Request Cancellation	§6.2 User Portal §6.4 Production Service	
R-FN-3.7	Processing Request Status Monitoring	§6.2 User Portal §6.4 Production Service	
R-FN-3.8	L3 Processing Requests	§3.1 Production Scenarios §8.2 L2-to-L3 Processing with MapReduce	
R-FN-3.9	L2-L3 Processing Chain	§3.1 Production Scenarios §6.4 Production Service	
R-FN-3.10	Stored Processing Requests	§3.2 System Use Cases §6.2 User Portal §6.4 Production Service	concept of production request templates
R-FN-3.11	Hosted Processing and Processor API	§7.10 Processor Interfaces	
R-FN-3.12	JAVA API compatible to BEAM GPF	§7.10 Processor Interfaces	
R-FN-3.13	C or C++ API	§7.10 Processor Interfaces	
R-FN-3.14	Public Repository for Processors	§6.1 System Decomposition	(desired) manual deployment in prototype
R-FN-3.15	Configurable Output Format	§6.5 Ingestion and Staging Service	
Analysis Requirements			
R-FN-4.1	Test (Analysis) on Processed Data	§3.1 Production Scenarios	tests match-up and trend analysis are covered
R-FN-4.2	MERMAID Reference Data Sets	§3.3 Input Data Requirements	
R-FN-4.3	Match-up Generation and L3 Time Series	§3.1 Production Scenarios	
R-FN-4.4	Match-up Generation	§3.1 Production Scenarios	
R-FN-4.5	Time Series Generation	§3.1 Production Scenarios	
R-FN-4.6:	Statistical Analysis	§3.1 Production Scenarios	
R-FN-4.7:	Analysis Request	-	(optional)



Non-Functional Requirements			
Performance Requirements			
R-NF-1.1	Fractional Amount of Time	§5 Concurrency Trade-off Analysis	addressed in trade-off
R-NF-1.2	Linear Scalability	§5 Concurrency Trade-off Analysis	addressed in trade-off
R-NF-1.3	Storage Capacity for 2 Years of MERIS RR	§4 Hardware Environment §3.3 Input Data Requirements	
Architecture			
R-NF-2.1	Portal and Backend Services	§6.1 System Decomposition §7 External and Internal Interfaces	
R-NF-2.2	Generic Data Models and Services	§6 System Component Design §7 External and Internal Interfaces	
R-NF-2.3	Hadoop MapReduce Technology	§2.2 Hadoop Distributed Computing	
R-NF-2.4	Hadoop Distributed File System	§2.2 Hadoop Distributed Computing	
R-NF-2.5	12 Machines Cluster	§4 Hardware Environment	
R-NF-2.6	Unix Operating System	§4 Hardware Environment	
Demonstration to the Community			
R-NF-3.1	Demonstration System for CCI-OC and Cal/Val Users	n/a	
R-NF-3.2	Links to Communities	n/a	