A High-Performance Cloud-based Remote Sensing Data Reprojection Method

Yi Dong1,2, Chen Xu* 3,4

1. Aerophotogrammetry & Remote Sensing Bureau of China Coal (ARSC)  
2. Shaanxi Engineering Technology Research Center for Geospatial Information  
3. Key Laboratory of Digital Earth Science, Aerospace Information Research Institute, Chinese Academy of Sciences  
4. University of Chinese Academy of Sciences

E-mail: xuchen20@mails.ucas.ac.cn

Abstract. Remote sensing (RS) data are the cornerstone of the digital earth. At present, the productions from Earth observation satellites are updated frequently with refined data quality. However, traditional methods are implemented mostly based on a single machine. When processing large-scale data in batches, there are limitations in computing power and storage, and the expansion is relatively cumbersome. This demands a more efficient and complex computation platform, such as cloud computation. Specifically, the reprojection procedure, unlike other data processing procedures, is both computationally intense and I/O-intense. This paper proposes a high-performance cloud-based RS data reprojection method called OCRM (optimised cloud-based reprojection method). First, the data process flow of reprojection was optimised by improving the I/O efficiency and computational efficiency. Second, with scheduling and controllers under cloud computation, high-performance massive RS data reprojection was achieved. The overall performance of the RS data reprojection module with the optimised algorithm in a single machine considerably outperformed that of the commercial software. The scheduling module can maximise the use of cloud resources to achieve high-performing reprojection calculations. The results indicate the stable performance of the proposed method, and we are working to expand the structure of OCRM to other computationally intensive remote sensing processes, such as data preprocessing and data mining with deep learning.

1. Introduction

Currently, RS data are widely used in the fields of earth science, disaster early warning, emergency dispatch, land management, etc. [1][2]. RS data are one of the most important data sources of the digital earth [3]. The era of remote sensing big data and digital earth big data has arrived, and thus, the demand for massive RS data computation has grown rapidly[4][5]. By this trend, specific regimes for RS big data preprocessing and cleansing [6] are expected to maintain the structural unity of RS data. Preprocessing includes geometric correction, radiometric correction, spatial and temporal reconstruction, quality evaluation, etc. [7]. Among them, the procedure that transfers multisource RS data of different projections into the same projection coordinate system is called reprojection. Reprojection is a typical computationally intensive and I/O intensive procedure that is widely used in...
various preprocessing steps. Its implementation efficiency deeply influences RS data preprocessing efficiency. However, different data providers use different projected coordinate systems. Different fields need RS data from different projection coordinate systems due to factors, such as technology stacks, demands, policies and equipment. The mismatch between providers and consumers needs services that can adapt highly dynamic data delivery demands and rapidly expand service capabilities. However, traditional commercial software (such as ArcGIS) can only support frame-level reprojection and can only run on a single machine. It is not able to cope with dynamic reprojection demands, process superlarge RS data, and rapidly expand to support large-scale data processing. To address these issues, this paper proposes a high-performance cloud-based RS data reprojection method called OCRM. To solve the problem caused by computational intensity, this research first optimised the calculation under basic efficiency and high dynamic demand, which improved the computational efficiency of reprojection computing. To minimize storage intensity, this research expanded the capacity through task scheduling and management, which optimised the migration ability and the efficiency of the transfer projection of superlarge data. This work is based on our previous work on ScienceEarth [8][9].

2. Methodology

![Figure 1. Technology Architecture of OCRM.](image)
Our technology architecture included three layers: the user layer, the compute layer, and the storage layer. The user layer received the user’s reprojection demands and then sent them to the computation layer. The compute layer used the subdivision method and divided the user’s requests into task groups. Then, the resource manager scheduled those task groups and sent each task to the compute nodes. The compute node was responsible for the reprojection computing procedure. The whole compute layer was based on a storage layer structured by Hadoop [10]. Our technology architectural design achieved computational and storage distributions. OCRM was implemented in conjunction with the technical improvements mentioned below.

2.1. Improvements of reprojection computing algorithm

2.1.1. Data read/write optimisation

To combat the inefficiency of I/O when random reading is reprojected, we used the locality of the projection algorithm\(^1\), which improved the aggregation of data reading. We performed a single point reprojection operation on the boundary required by the user to determine the original data range. The original data range was applied to read the range of image data into memory. Then, all calculations and copies of the data occurred in memory, and the reprojected data were written to disk after processing. This optimisation changed the random access to sequential reads and writes, which decreased the stress for the hard drive. This change greatly increased the speed of the I/O. To reduce the data size and better adapt to cloud computation, the COG format was also implemented in this study.

2.1.2. Memory addressing optimisation\(^2\)

Since reprojection calculations involve a large number of image band operations, such as data filling and cell interpolation, converting data from AOB (array of band) structures to AOC (array of colour) structures can effectively reduce the memory addressing time.

\(1\) Detailed discussion of locality at Appendix A.

\(2\) All examples in this section are 3 bands.
As shown in Figure 5, after rearranging the data according to AOC, each data address was near the neighbour. This process greatly reduced the possibility of page memory displacement and improved the efficiency of addressing.

2.1.3. Pre-building the environment

Building the environment while computing will consume time. However, the environment needed is stable for the same algorithm, and each computation only needs to change the input data. This feature leads us to try a prebuilt computation environment to compress the reprojection computation time. To ensure the correctness and generalisation use of the components, we chose GDAL [11] as the technical base.

2.2. Cloud-based tasks management design

For large-scale RS data, single machine performance may be constrained, which can be improved by calculating in parallel by scheduling. Since the range of user requirements does not often fall into individual RS data within the underlying data, converting the user's needs into several RS data reprojection subtasks is often needed. We proposed a division method for the reprojection of individual RS data. The file system HDFS and the task scheduling system, designed with the characteristics of reprojection tasks, achieved a dynamic expansion of resources and completed oversized reprojection. After the cluster was built according to this study, we proposed the following
cloud-based task management methods to ensure the high availability, stability and overall controllability of the system. The overall structure is shown in Figure 6.

2.2.1. Resources use function

To meet the controller's resource management needs, the manager needed to estimate the memory resources used by the task before running. It also prevented running instability when the physical machine was running out of memory. We found that the memory use of the task execution had the following mathematical relationships:

Input data is:
- Starting coordinates under the target projection: \( \text{Proj}_x, \text{Proj}_y \)
- Amount of change per unit under the target projection: \( \text{Proj}_{\Delta x}, \text{Proj}_{\Delta y} \)
- Width and height of the target picture: \( \text{Proj}_{\text{width}}, \text{Proj}_{\text{height}} \)
- Reproduction function: \( p(\text{Proj}_{kx}, \text{Proj}_{ky}) = (\text{Ori}_{xk}, \text{Ori}_{yk}) \)

The bounding box for the target projection can be calculated as follows:

\[
\begin{align*}
\text{Proj}_{x\text{End}} &= \text{Proj}_x + \text{Proj}_{\Delta x} \ast \text{Proj}_{\text{width}} \\
\text{Proj}_{y\text{End}} &= \text{Proj}_y + \text{Proj}_{\Delta y} \ast \text{Proj}_{\text{height}} \\
\text{Proj}_{\text{minx}} &= \min(\text{Proj}_x, \text{Proj}_{x\text{End}}) \\
\text{Proj}_{\text{miny}} &= \min(\text{Proj}_y, \text{Proj}_{y\text{End}}) \\
\text{Proj}_{\text{maxx}} &= \max(\text{Proj}_x, \text{Proj}_{x\text{End}}) \\
\text{Proj}_{\text{maxy}} &= \max(\text{Proj}_y, \text{Proj}_{y\text{End}})
\end{align*}
\]

As shown in Formula (7), the bounding box of the original data can be obtained by calculating the projection of the boundary points of the target projection:

\[
\text{Ori}_{\text{bounds}} = \begin{bmatrix}
p(\text{Proj}_x, \text{Proj}_y) & p(\text{Proj}_x + \text{Proj}_{\Delta x} \ast 1, \text{Proj}_y) & \cdots & p(\text{Proj}_{x\text{End}}, \text{Proj}_y) \\
p(\text{Proj}_x, \text{Proj}_{y\text{End}}) & p(\text{Proj}_x + \text{Proj}_{\Delta x} \ast 1, \text{Proj}_{y\text{End}}) & \cdots & p(\text{Proj}_{x\text{End}}, \text{Proj}_{y\text{End}}) \\
\vdots & \vdots & \ddots & \vdots \\
p(\text{Proj}_x, \text{Proj}_{y\text{End}}) & p(\text{Proj}_x + \text{Proj}_{\Delta x} \ast 1, \text{Proj}_{y\text{End}}) & \cdots & p(\text{Proj}_{x\text{End}}, \text{Proj}_{y\text{End}})
\end{bmatrix}
\]

\[
\begin{align*}
\text{Ori}_{\text{minx}} &= \min(\text{Ori}_{\text{bounds}}[x]) \\
\text{Ori}_{\text{miny}} &= \min(\text{Ori}_{\text{bounds}}[y]) \\
\text{Ori}_{\text{maxx}} &= \max(\text{Ori}_{\text{bounds}}[x]) \\
\text{Ori}_{\text{maxy}} &= \max(\text{Ori}_{\text{bounds}}[y])
\end{align*}
\]

We use the original unit change range read from the original data: \( \text{Ori}_{\Delta x}, \text{Ori}_{\Delta y} \). We calculate the width and height of the original data required as follows:

\[
\begin{align*}
\text{Ori}_{\text{width}} &= \frac{(\text{Ori}_{\text{maxx}} - \text{Ori}_{\text{minx}})}{\text{Ori}_{\Delta x}} \\
\text{Ori}_{\text{height}} &= \frac{(\text{Ori}_{\text{maxy}} - \text{Ori}_{\text{miny}})}{\text{Ori}_{\Delta y}}
\end{align*}
\]

The use of memory resources can be calculated with the following formula:

\[\text{memory} = (\text{Ori}_{\text{width}} \ast \text{Ori}_{\text{height}} + \text{Proj}_{\text{width}} \ast \text{Proj}_{\text{height}}) \ast B \ast U + C\]

B: number of bands
U: number of bytes occupied by data type
C: constant memory for the environment (in our environment \( C = 2\text{MB} \))

2.2.2. Subdivision method

The task will be divided if memory consumption exceeds the threshold (we set it to 750 MB). The subtask will be divided later if the subtask still fails to meet the threshold. For the subdivision method, we divide the image according to the configuration, as shown in Figure 7.
By applying this method, the read/write conflict area in the image task can be controlled at the segmentation boundary. The parallel executability of the overall task can be improved by cooperating with the scheduling strategy.

2.2.3. Scheduling strategy

After the user's task goes through the division process, if there is no division, the task is directly put into the computing node pool to wait for the allocation of computing nodes. If the split task group has only one layer, all of the subtasks in this layer are sequentially placed into the computing node pool. If multilayer division occurs, the batches are divided into n layers, and then each batch is scheduled in a single-layer manner.
3. Result

3.1. Implementation environment introduction
This research has two implementation environments: Table 1 for a single machine and Table 2 for cloud computation.

**Table 1. Implementation environment single machine.**

| System       | Windows                                      | Linux                                      |
|--------------|----------------------------------------------|--------------------------------------------|
| CPU          | 11th Gen Intel(R) Core (TM) i5-1145G7        | 11th Gen Intel(R) Core (TM) i5-1145G7     |
| RAM          | 32GB 3200                                   | 16GB 3200                                 |

**Table 2. Implementation environment cloud computation.**

| Node type | Computing node | Management node | Storage Node |
|-----------|----------------|-----------------|--------------|
| CPU       | 8 cores        | 8 cores         | 4 cores      |
| RAM       | 16GB           | 8 GB            | 8GB          |

3.2. Single machine efficiency comparison
The single machine efficiency experiment using an RS data original projection is EPSG:4513, the size is: 4518 * 4732, and compares two common usage scenarios:

**Table 3. Single machine efficiency comparison table.**

|               | Mission1<sup>a</sup> | Memory<sup>a</sup> MB | Mission2<sup>b</sup> | Memory<sup>b</sup> MB |
|---------------|----------------------|------------------------|----------------------|------------------------|
| Arcgis        | 9226.99999809        | NOT Available          | NOT Available        | NOT Available          |
| GdalPY        | 4610.871826171       | 178.72                 | 185.95874023         | 66.69                  |
| Gdal          | 4017.36              | 192.60                 | 114.705              | 38.87                  |
| OCRM          | 6054.84              | 173.60                 | 33.35                | 39.10                  |

<sup>a</sup> Convert the overall original image to other projections

<sup>b</sup> Extract 256*256 target range from the original image to other projections

The efficiency comparison source code can be found at <https://gitee.com/ddyy_1213/tile-extract>. The project experiment can be reproduced under the Linux environment with Gdal installed.

In Mission 1, our implementation was better than that of ArcGIS, but it was slower than that of Gdal, which will be discussed in Section 4.2.

In Mission 2, our implementation outperformed Gdal and GdalPY by an order of magnitude in time consumption and was almost the same as the C version of that of Gdal in memory consumption.
It can be seen from Table 4 that after the above optimisation, the reprojection operation still occupies the main body of the overall execution.

### 3.3. Cloud computation experiment

During the study, we received a request to process the latest satellite remote sensing data from a certain place into usable RS products. We divided the satellite data into three batches, each of which contained one hundred images. The data volume per image was approximately 200 G (uint8, 5 band), and the data diversities were almost the same. We tested the traditional reprojection method with the three images in the first batch and processed them for an average of approximately 19 minutes. Traditional reprojection has difficulty achieving parallel processing. The processing efficiency of a batch is estimated to be far more than 60 hours if considering the single hard disk capacity limitation and data transfer time. This efficiency makes it difficult to complete the task on time. A large amount of manual participation is extremely error-prone. We successfully processed data after applying OCRM and provided the required RS products.

### 4. Discussion

#### 4.1. Optimisation effects

From Figure 9, after I/O improved, the execution efficiency on Mission2 steadily surpassed that of Gdal. However, the execution time was still generally more than 50 ms. Later, with the implementation of the prebuilt optimisation, the execution time was reduced to less than 50 ms, but the performance was still not stable enough. Finally, after the introduction of AOC optimisation, the execution time was stable in the 30-50 ms interval.
4.2. Why mission 1 OCRM had a worthy performance

Mission 1 OCRM was slower than Gdal. The preliminary analysis shows that the implementation of the OCRM is point-by-point reprojection, while the implementation of Gdal is to take a certain interval for reprojection calculation, and the missing points in the middle are supplemented by interpolation. The reprojection calculation was most time-consuming, as shown in Table 4. Gdal achieved the overall efficiency improvement by reducing the number of reprojection calculations.

In Mission 2, the performance of OCRM was far ahead of Gdal. Therefore, we believe that purely from the perspective of the reprojection process, the optimisation scheme of this study already
optimised the execution time of reprojection. If point-by-point reprojection is also used in Gdal, OCRM should perform better than Gdal on the entire image reprojection.

4.3. Cloud computation practice
In the cloud computation practice in Section 3.3, since the total data volume exceeds 120 TB, only two of our storage nodes could safely store project data. Due to the limitation of storage nodes, too many computing nodes would have been stalled due to I/O speed. After considering the calculation speed and the I/O speed, we only used 3 computing nodes to process the data. If there was no storage limitation, with the expansion of computing nodes and storage nodes, the task execution time could have been shortened.

5. Conclusion
This paper proposes a high-performance cloud-based RS data reprojection method called OCRM. First, we optimised the process of the reprojection algorithm of RS data. After I/O, addressing, prebuilding and other optimisation methods, the time of reprojection calculation is effectively reduced. To ensure the accuracy and versatility of the results, as well as the ease of use in engineering, we used the reprojection algorithm provided by Gdal. Cloud-based computing is used to optimise the computing efficiency of a large amount of RS data, avoiding the difficulty of expanding computing resources and storage resources. The experiment shows that the overall performance of the optimised algorithm when running the RS data reprojection module on a single machine is considerably better than that of the current commercial software. Second, the dispatch system can maximise the use of cloud resources to achieve high-performance reprojection calculations. This research supports single-machine deployment or multi-node deployment, fully considering the availability issues in different computing environments and the scalability of the overall system. Possible follow-up optimisation directions include: 1. We may research the implementation of reprojection algorithms and try to propose more efficient algorithms. 2. We may try to implement the reprojection algorithm with more efficient software and hardware. 3. We may try to explore the method of taking points at intervals to reduce the number of reprojections.

6. Appendix

Appendix A: Spatial locality of reprojection
Spatial locality is defined as the feature that immediately accesses its neighbouring addresses when it is likely that a program accesses a memory address. The reprojection operation is executed sequentially in the original projection space, which has spatial locality. When performing interpolation operations in the postprojection space, it also visits 8 directional points of the postprojection point, which has spatial locality.

Appendix B: System migration ability
Different modules were separated, which enhanced the expansion capability of the overall system when interacting with each other by protocol. When facing other kinds of data sources, the mismatched management node or computing node can be replaced in a modular manner to adapt the diversity of the data. In addition, our reprojection method is only a module of this architecture to achieve cluster reprojection, and other reprojection methods or other calculation methods can also be incorporated into the system as cloud-based tasks after the control protocol has been extended.
Appendix C: Software version

| Software | Version | Software | Version | Software | Version |
|----------|---------|----------|---------|----------|---------|
| Arcgis   | 10.03   | PROJ     | 8.0.0   | Curl     | 7.75.0  |
| gcc      | 9.3.0   | SQLite   | 3350300 | Python   | 3.8.10  |
| Gdal     | 3.2.2   | Tiff     | 4.2.0   | PYGdal   | 3.2.2   |

Funding: This work is supported by the Innovation Drive Development Special Project of Guangxi (GuikeAA20302022), the National Natural Science Foundation of China (41974108) and the Strategic Priority Research Program of Chinese Academy of Sciences, Project title: CASEarth (XDA19080103).

References

[1] N. Pahlevan, S. K. Chittimalli, S. V. Balasubramanian, and V. Vellucci, “Sentinel-2/Landsat-8 product consistency and implications for monitoring aquatic systems,” Remote Sens. Environ., vol. 220, no. October 2018, pp. 19–29, Jan. 2019.
[2] H. Huang et al., “Mapping major land cover dynamics in Beijing using all Landsat images in Google Earth Engine,” Remote Sens. Environ., vol. 202, pp. 166–176, Dec. 2017.
[3] M. F. Goodchild et al., “Next-generation Digital Earth,” Proc. Natl. Acad. Sci., vol. 109, no. 28, pp. 11088–11094, Jul. 2012.
[4] H. Guo et al., “Big Earth Data science: an information framework for a sustainable planet,” Int. J. Digit. Earth, vol. 13, no. 7, pp. 743–767, Jul. 2020.
[5] B. Zhang et al., “Remotely sensed big data: evolution in model development for information extraction [point of view],” Proc. IEEE, vol. 107, no. 12, pp. 2294–2301, Dec. 2019.
[6] C. Yang et al., “Big Earth data analytics: a survey,” Big Earth Data, vol. 3, no. 2, pp. 83–107, 2019.
[7] A. Lewis et al., “The Australian Geoscience Data Cube — Foundations and lessons learned,” Remote Sens. Environ., vol. 202, pp. 276–292, Dec. 2017.
[8] C. Xu, X. Du, Z. Yan, and X. Fan, “ScienceEarth: A Big Data Platform for Remote Sensing Data Processing,” Remote Sens., vol. 12, no. 4, p. 607, Feb. 2020.
[9] C. Xu et al., “A Modular Remote Sensing Big Data Framework,” pp. 1–11, 2021.
[10] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The Hadoop Distributed File System,” in 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010, vol. 454, no. 3, pp. 1–10.
[11] O. R. G. GDAL, “GDAL—Geospatial Data Abstraction Library.” Open Source Geospatial Foundation, 2012.