An In-Memory Data-Cube Aware Distributed Data Discovery Across Clouds for Remote Sensing Big Data

Jie Song, Yan Ma, *Member, IEEE*, Zhixin Zhang, and Peng Liu, *Member, IEEE*

***Abstract*—With the booming of high-resolution Earth observation and open-data efforts, petabyte-scale Earth observation data have been available for free access. Due to the unprecedented availability of big data deluge, regional to global spatio-temporal analysis has been significantly challenged with the huge computational barriers, the tedious cycles of "download-preprocess-store-analyze" leading to excessive data downloading overhead, and the acquisition-oriented 2-D file-based structure, which is not fit for spatio-temporal analysis. The Earth observation data cube (EODC) paradigm revolutionizes the traditional way of storing, managing, and analyzing spatio-temporal RS data, and solves problems of easy-to-use of RS data to a certain extent. However, different EODC solutions are becoming "information silos." Therefore, the sharing and joint use of remote sensing (RS) data across EODCs have become extremely challenging. To address the abovementioned challenges, we proposed a method of in-memory distributed data cube autodiscovery and retrieval across clouds. We construct a distributed in-memory data orchestration across clouds to shield the heterogeneity of the EODC storage solutions, solving "information silos" problems, and we put forward a larger-sites-first and spatio-temporal aware RS data discovery strategy, which can automatically discover data across clouds for requirements. Based on the data cube paradigm, this article proposes a quality-first data filtering strategy, which can help users to filter out high-quality data covering the target spatio-temporal range from the huge amount of data, and solve the problem of data cube joint retrieval and efficient use across clouds. In addition, we have confirmed that our method is effective and efficient through comparative experiments.**

***Index Terms*—Clouds, data cube, data discovery, data integration, distributed computing, GEE, in-memory distributed file system, remote sensing (RS) big data.**

I. Introduction

THE continuous pressure on natural resources introduced by growing human activities have gradually aggravated global environmental changes [1] and led our planet onto an unsustainable pathway [2]. The Earth observation (EO) data [3] from satellite sensors facilitates regular and continuous monitoring of our planet from different facets [4]. The information and valuable patterns elaborated from EO data enable a great stride in understanding the Earth system dynamics [5] from the human side and mitigating environmental changes. Benefiting from high-resolution EO, especially the booming diversity of advanced sensors, makes it possible to achieve global coverage of the Earth's surface daily with higher spatial and spectral resolutions [6]. Up till now, the global archived EO data have undergone explosive growth and are approaching exabytes [7]. In particular, the U.S. Geological Survey (USGS) Landsat dataset [8] reaches 4.5 petabytes, whereas the European Space Agency (ESA) Sentinels data [9] are increasing with a daily volume of nearly six terabytes. Furthermore, due to the open-data efforts, petabyte-scale analysis-ready data (ARD) from the spatial agencies have become increasingly available for free access [10], [11]. Accordingly, the flood of EO data has been widely perceived as "big data" [12], [13]. In turn, the unprecedented availability of EO data also gives rise to the surging demand for larger scale resources and environmental changes monitoring [14], [15] (involving land [16], [17], atmosphere [18], oceans [19], etc.) over longer time spans [20], such as large-scale cropland mapping [21], global agricultural drought monitoring [22], and global forest mapping [23].

However, in the context of "big data" [24], regional to global research could be practically overwhelmed by the tremendous EO data deluge. Despite the great efforts laid in harnessing the full potential of the enormous multisensor and multitemporal EO data [25], [26], [27], spatio-temporal analysis at a much larger scale remains significantly challenged with several troubling issues. *First of all*, the major challenge goes with the tedious cycles of "download-preprocess-store-analyze" in the traditional paradigm [3]. Despite the abundant data available, it is significantly suffering for scientists to download all the required data into the local repository right before analyzing. Moreover, the excessive data transport may take several weeks and make the entire processing unfeasible when scaling to a larger region. *Second*, the other obstacle lies in the computational barrier resulting from the massive computing power demand introduced by the staggering scenes of EO data, which far transcend the capacity of mainstream computing resources. Even accelerated

Manuscript received 29 January 2023; revised 14 March 2023; accepted 1 April 2023. Date of publication 14 April 2023; date of current version 18 May 2023. This work was supported by the National Natural Science Foundation of China under Grant 42071413. *(Corresponding author: Yan Ma.)*

Jie Song and Zhixin Zhang are with the Aerospace Information Research Institute, Chinese Academy of Sciences, Beijing 100094, China, and also with the School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: songjie201@mails.ucas.ac.cn; zhangzhixin211@mails.ucas.ac.cn).

Yan Ma and Peng Liu are with the Aerospace Information Research Institute, Chinese Academy of Sciences, Beijing 100094, China, and also with the University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: mayan1133@gmail.com; liupeng@radi.ac.cn).

Digital Object Identifier 10.1109/JSTARS.2023.3267118

by several dominant and promising high-performance computing (HPC)-enabled solutions [28], traditional ways of processing remain considerably time-consuming, especially for global monitoring [29]. *What is more*, EO data processing commonly follows a scene-by-scene manner using images-as-a-temporal-snapshot data model [10]. However, the multidimensional-array data concepts have long been uptaken up for spatio-temporal analysis by remote sensing (RS) experts. It becomes quite clear that the acquisition-oriented 2-D file-based data structure is not a user-friendly granularity for spatio-temporal analysis. Moreover, the cumbersome data transformation for bridging the data structure gap could inevitably lead to extra overhead.

The EO data cubes (EODC) [30], [31], [32] as an innovative paradigm has revolutionized the way EO data are stored, processed, and the way users interact with large spatio-temporal datasets. Alternative to acquisition-oriented 2-D scene-file-based approaches, data cube is essentially an analysis-oriented multidimensional array solution [33] that facilitates data access along the spatial or temporal axis within one single data structure [10], [34]. Moreover, the sliced multidimensional array structure with inherent data parallelism could simplify data operations and make parallelization straightforward, especially for time series analysis [33]. Recently, with HPC-enabled or cloud-facilitated platforms, data cubes could serve massive spatio-temporal EO data in an analysis-ready way at high service quality and speed by lowering the big data barriers [35]. Eventually, it could offer web-based online analysis while avoiding massive data downloads [36].

Nevertheless, despite the increasing attention the EODC have gained [37], its data and service model is still rather confusing[34]. *Furthermore*, the thriving varieties of data cube infrastructures and initiatives also raise further interoperability challenges among different solutions for data cubes. With the prosperity of EODC, several typical varieties of data-cube implementations spring up, including the open data cube-based (ODC) initiatives, such as Australian Geoscience Data Cube (AGDC) [36], Swiss Data Cube (SDC) [35], and Colombian Data Cube [38], the DB-based solutions, such as EarthServer on Rasdaman [39] and E-sensing platform [40] on SciDB [41], as well as the cloud-based infrastructure, such as Google Earth Engine (GEE) [42] and Earth on Amazon web services (EAWS) [43]. However, this wide range of data cube solutions not only varies in data scales and resolutions but also differs in system infrastructures, software implementations, and user interfaces [44]. As a result, these differences will inevitably result in limited interoperability among existing data cube infrastructures. Accordingly, due to the incompatibility, these data cube infrastructures would tend to act as closed "information solis," where data sharing and joint data use across them would be considerably difficult [35], [45]. *Make it worse*, to take full advantage of different data cubes for a comprehensive harnessing of big EO data, the massive data transport among these infrastructures featured with various system architectures would be another challenging barrier. It would be significantly trivial and even unfeasible to fetch the data from other data cubes before joint analysis across different data cube platforms.

To properly tackle the challenging obstacles abovementioned, we propose an in-memory distributed data cube discovery method for RS big data across the Clouds with quality-first (QF) data filtering. *First*, this method employs Alluxio, a virtual file system, to conduct in-memory data orchestration across Clouds. It virtually mounts massive EO data from different Clouds or data cubes (such as Amazon Cloud, Ali Cloud, Tencent Cloud, and local Hadoop Distributed File System) onto Alluxio as a locally cached data catalog. Instead of conducting massive data transport among different Clouds, it offers transparent data prefetching from underlying heterogeneous platforms ahead to facilitate in-memory data access across Clouds. *Second*, we propose larger-sites-first (LSF) and spatio-temporal aware (LSA) data discovery strategy to optimize comprehensive and fast data discovery across clouds. By employing distributed crawler engine on multicore Hadoop-powered clusters, we implement LSA data discovery in parallel as bunches of distributed crawler tasks to search data throughout the virtually mounted in-memory data catalogs. In addition, for further enhancement, we adopt a QF data filtering strategy to filter out the data with high data quality while fitting the spatio-temporal coverage with minimal data cost. Eventually, benefiting from smart in-memory data cube discovery and quality-aware data filtering, we could expect fast and accurate data discovery out of the extremely big EO datasets across Clouds.

The rest of this article is organized as follows. The following section reviews the state-of-art works related to RS data management, sharing, and discovery. Then, Section III discusses the challenging issues of RS data discovery and joint use. In Section IV, we go into detail about the design and implementations. Then, Section V demonstrates the experiment result and comparative analysis. Section VI discusses the effectiveness and efficiency of the approach proposed based on the experimental result. Finally, Section VII concludes this article.

II. Related Work

A. Traditional RS Data Managing and Sharing on a Scene-File Basis

Traditionally, RS data are organized into scene-based units and stored in file systems in file formats. The distributed parallel file systems, such as GPFS, Lustre, PVFS, and OrangeFS [46], have been introduced to speed up the reading and writing of RS image data [47], [48], [49]. Although these file systems are optimized for data parallel access, they are not good at accessing RS data with multiple dimensions, such as time, space, and spectrum. The scene-file structure is not a user-friendly granularity for spatio-temporal analysis, and the cumbersome data transformation for bridging the data structure gap could inevitably lead to extra overhead.

On a scene-file basis, most data providers build web portals to share their data, such as EORS Earth Explore [50], the Copernicus Open Access Hub [51], and the FY Satellite Data Service Web Portal [52]. However, users have to log into each platform's website. Then, they have to manually search and filter data through the website portal, and download them into local storage, which is quite time-consuming. In addition,

different providers offer different types of data and with different spatio-temporal ranges, the data from a single data platform are not sufficient for large-scale data processing. So users have to download data from multiple platforms for large-scale processing at a large region or even globally [53]. Therefore, a long processing cycle of "download-pre-process-storage-analysis" is inevitable. In the case of large-scale data processing, large-scale data download is extremely time-consuming or even infeasible. This becomes a huge obstacle for large-scale analysis and limits the effective information extraction of RS big data.

B. RS Data Managing and Sharing in Data-Cube Paradigm

In the context of RS big data, the traditional processing way of "download–preprocess–store–analyze" is inefficient. The EODC paradigm has revolutionized the way of data storage, management, and analysis. It adopts the sliced multidimensional array structure with inherent data parallelism, which could simplify data operations and make parallelization straightforward [33], [34]. It supports bringing processing to data without large-scale data download [36]. At present, there are three main types of EODC solutions, such as cloud-based solutions, DB-based solutions, and ODC-based solutions.

The Cloud-based solutions provide spatio-temporal analysis tools and algorithms to support near-data computation, which avoids large-scale data transfer, such as GEE [42], EAWS [43], and DIAS [32]. They offer online access and on-the-fly analysis to petabytes of global spatio-temporal RS data in a cube-like format [54]. However, they have limited processing temporal and spatial scales [32]. The cube-like operations could be time-consuming for data retrieval and spatial aggregation over a large region. In addition, the uncertainty about the sustainability of these proprietary cloud platforms has become a major concern for users [45].

Array databases are specialized in managing large N-dimensional arrays and offer basic data operations focused on data cubes, such as aggregation on dimensions, which can simplify spatio-temporal RS data extraction, preparation, and analytical processing [55]. Current DB-based EODC implementations include EarthServer[39] with Rasdaman [56], EarthDB [57], and INPE Data Cube [58] with SciDB [59]. In addition, TileDB with a cloud-native array format at its core also has a promising future in RS storage and management [60]. These array databases provide relatively good performance and scalability for accessing and analyzing multidimensional spatio-temporal RS data, making them suitable for repeated access to regional data [61]. However, data ingestion and format conversion from a large number of image files to a multidimensional data cube can have an impact on the overall performance [62].

As an open-source, flexible, and promising data cube framework, ODC [63], has been used in numerous national or regional data cubes projects, such as SDC [35], CDCol [64], ARDC [65], and CDC [45]. It organizes and manages data with a large-scale multidimensional array [66], [67] that can support seamless spatial, temporal, spectral, and feature analysis [68], [69]. So users can access RS data based on spatio-temporal coordinates rather than the traditional single "scene" file [24], [30], [70]. Thus, the data cube structure enables the shift from "scene" file-based to pixel-based processing, greatly improving RS data organization, management, and analysis performance. However, the cube storage format, NetCDF4, has weak object storage access performance [71]. Although lightweight computing engine Dask has been used for memory computing [72] in the ODC, highly intensive data I/O and data transmission remain significant performance bottlenecks [73].

These diverse EODC implementations solve the problem of downloading large-scale RS data by bringing algorithms and processing to the RS data (near-data computing). They provide application programming interfaces (APIs) to facilitate data sharing and access, such as JavaScript and Python APIs of GEE, Python API of the AGDC [36], and Amazon's API of the Earth on AWS [74]. Instead of manually logging into the web portal, users can call the data platforms' APIs and retrieve and access data through different APIs, reducing manual efforts and improving data analysis efficiency. However, these APIs can only be used within a single data platform, limiting data sharing and usage across platforms. These diverse data platforms with different system infrastructures, software implementations, user interfaces, and data standards are currently closed and are becoming "information silos." The existing RS data managing and sharing systems could barely offer efficient data discovery across different cloud platforms or data cube platforms. This results in the inability to share and use data from multiple EODC platforms, and RS data analysis over large regions or even globally becomes extremely challenging.

C. RS Data Discovery Across Clouds

Traditional RS data sharing and access methods are using web portals and APIs of data platforms. However, both methods can only be used for data retrieval, download, and access within the single data platform, and cannot support data discovery across multiple cloud platforms. The latest version of ODC [75] only provides interfaces to access cloud storage buckets. Users have to manually find and filter the data needed for their applications from multiple cloud platforms separately. The data must be downloaded locally, parsed, and indexed into the ODC to support access to the data from cloud platforms. ODC only supports a small number of cloud storage types. In a short, some approaches could support indirect data access from multiple clouds, but they cannot yet support data discovery across Clouds.

In the managing and sharing of Internet data, the data orchestration function of Alluxio [76] enables the sharing and joint use of data from multiple cloud platforms. Some web crawler strategies can be used to discover and retrieve data from different cloud platforms. For example, the LSF [77], [78] strategy prioritizes access to important pages to traverse the most critical information as soon as possible, making it especially suitable for searching for large-scale data without consuming too much time and resources. Depth-first search (DFS) strategy [79] is to prioritize traversing the deep links of a web page until the deepest link is reached, then return to the previous level and continue traversing other links. This search method is usually used to find specific information in a short time, but it may fall into an infinite

loop, leading to a waste of resources. Although these web data search strategies support data discovery across Clouds, most of them rely on keyword searches for text data, which is not friendly to RS data discovery and search. Due to the multidimensional attributes, such as time, space, and spectrum of RS data, during the RS data discovery, users need to access the metadata file of each scene data to obtain information (including time, space, cloud coverage, etc.) to effectively discover the target data that meet application requirements. Therefore, these data discovery methods do not well support fast and accurate RS data discovery across Clouds.

On the whole, the existing RS data managing and sharing systems could barely offer efficient data discovery across Clouds. But for Internet data, there are ways to support data search across Clouds, such as LSF and DFS. Therefore, it is very necessary to study the RS data discovery across Clouds in order to fully utilize RS data at a large region even globally.

III. Challenges

The EODC paradigm is gaining increasing attention as an innovative solution to store, process, and manage EO data [35]. The novelty of the EODC paradigm has brought different innovative solutions, providing various RS datasets with different spatio-temporal scales. For regional and global RS data applications, it is a great challenge to discover and jointly use spatio-temporal RS data from cloud platforms and data cube platforms.

A. "Information Silos" Problem Among Cloud Platforms With Different Solutions

The thriving varieties of data cube infrastructures and initiatives raise interoperability challenges among different solutions for data cubes. Currently, there are various typical EODC implementations, including AGDC [36], GEE [42], EAWS [43], EarthServer [39], Earth System Data Cube [80], and Africa Regional Data Cube (ARDC) [81]. These different EODC implementations provide diverse satellite data, cover varying spatial scales, and support different resolutions. Furthermore, the lack of a commonly agreed definition of EODC has led to differences in system infrastructures (e.g., HPC, Google Cloud, and Amazon Cloud), software implementations (e.g., ODC, Rasdaman, SciDB), and user interfaces (e.g., Python, JavaScript, REST, and AWS CLI) [39], [81], [82]. As a result, these differences will inevitably result in limited interoperability among existing data cube infrastructures [70], [83]. Due to the incompatibility, these Data Cube infrastructures would tend to act as closed "information silos," making data sharing and joint use across them difficult [35], [45]. Moreover, most current EODC implementations, such as the EAWS, GEE, Digital Earth Africa, and Brazil Data Cube, host RS datasets on clouds [81], [84], [85]. In the coming years, it is expected that nearly all EO data will exist in some ARD data cube hosted on clouds and provided hosted analytic web services [31]. Therefore, how to support users to efficiently access and share multisource RS data across clouds is a challenging issue.

B. Data Discovery Challenges in the Context of Extremely Bulky RS Data Across Clouds

The diverse RS data platforms have emerged, offering different types of data with different spatio-temporal scales, and processing levels. The data from a single data platform is not sufficient for large-scale data processing [35], so users have to discover and download data from multiple platforms for large-scale processing at a large region or even globally [53]. Interoperability challenges among the diverse EODCs platforms exist due to the lack of uniform standards to support the description and disclosure of RS data, making it difficult to commonly search for data cubes. The traditional RS data sharing and access methods based on web portals and APIs of data platforms are only available within the data platform. Although data search strategies, such as DFS [79] and LSF [78], support data search across clouds, they are oriented to Internet data and do not apply well to RS data. Most data platforms host RS data on public clouds [45], but the existing RS data managing and sharing systems could barely offer efficient data discovery across clouds. It is extremely challenging to find high-quality data that meets the application requirements from the massively available data. These challenges severely limit the data utilization and hinder the full information potential of RS data [45], [86].

C. Performance and Scalability

As the demand for analyzing large-scale RS big data increases, users require efficient methods to discover and download the data needed for their applications from multiple data platforms. Traditional data discovery methods require users to interact with multiple data platforms separately, and understand each platform's system structure, service APIs, metadata standards, data download approaches, and more. This makes large-scale data discovery and downloading tedious and time-consuming. Data discovery through the web portal approach also demands significant user effort and time to search, select, and download the target data. Data discovery through APIs is typically a single-threaded process, which cannot meet the needs of real-time or near real-time data discovery and data access. Furthermore, large-scale multisource RS data discovery and access face huge network bandwidth and data I/O pressure. To provide higher performance and scalability while overcoming the I/O bottleneck, it is essential to explore the parallelism of data discovery.

IV. Design and Implement

A. Main Solution

With the increasing demand for analyzing and processing large-scale multisource RS big data, this article presents an innovative approach for in-memory distributed data cube autodiscovery and retrieval across clouds. The main solution of this article is shown in Fig. 1.

Our proposed solution consists of several key components. First, we establish a distributed in-memory data orchestration across clouds, allowing for the virtual mounting of multisource RS datasets. This provides users with a unified in-memory access

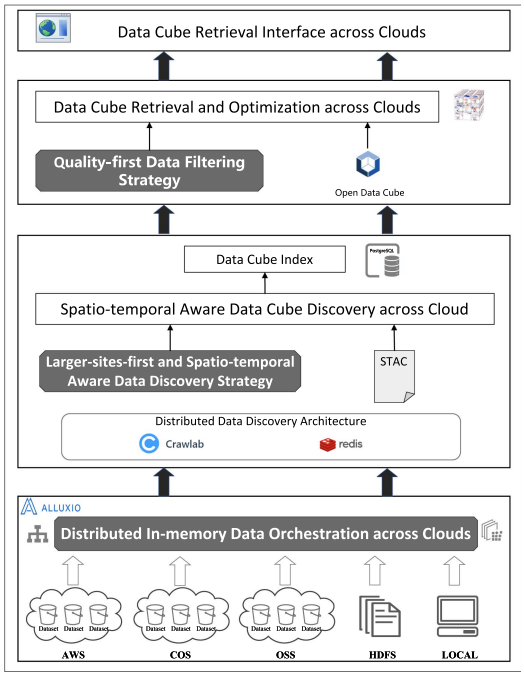


Fig. 1. Main solution.

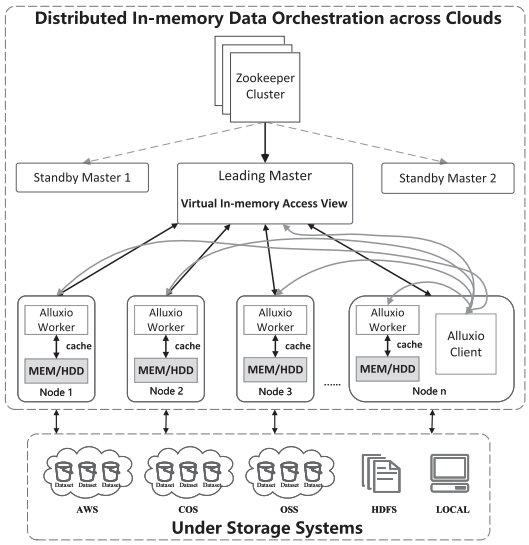


Fig. 2. Architecture of distributed in-memory data orchestration across clouds.

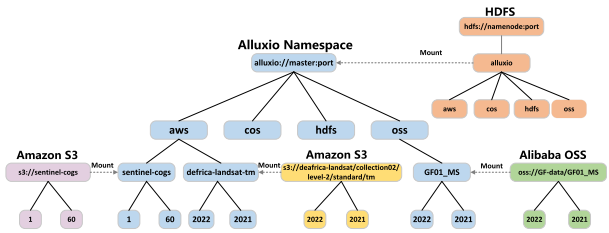


Fig. 3. Data in-memory access view across clouds.

view and operation interface, enabling the sharing and joint use of RS data across clouds. Second, we deploy a distributed data discovery architecture, leveraging STAC standards, to achieve data cube discovery and indexing across clouds. We propose the LSA data discovery strategy across clouds, which can quickly discover target RS data from multiple cloud platforms and improve the efficiency of spatio-temporal aware data discovery. Finally, we implement data cube retrieval across clouds using QF data filtering to select high-quality data from multiple cloud platforms covering a target spatio-temporal range. In addition, we provide a data cube retrieval interface across clouds, supporting large-scale RS analysis and applications.

B. In-Memory Data Orchestration Across Clouds With Alluxio

For the problem of "information silo" among cloud platforms of different EODC solutions, we construct a distributed in-memory data orchestration across clouds with Alluxio. This orchestration allows for the virtual mounting of RS data from various clouds, enabling the seamless access and sharing of multisource data across different cloud platforms. Alluxio (formerly known as Tachyon) [76] is a memory-centric distributed virtual storage system that bridges the gap between data-driven applications and various storage systems, making data more accessible from different storage systems, such as Amazon S3, Google Cloud Storage, OpenStack Swift, HDFS, Gluster FS, NFS, and Ceph.

In this article, we deploy the Alluxio in highly available mode via the Zookeeper cluster to construct an in-memory data orchestration across clouds, and the architecture is shown in Fig. 2. By virtually mounting RS datasets from different clouds including AWS S3, COS, OSS, HDFS, and LOCAL, we shield the heterogeneity of the underlying storage systems. This virtual data mounting mode provides users with a unified access view of all mounted datasets under the same namespace, which is structured as a directory tree, as shown in Fig. 3. This approach fully guarantees the data sovereignty of the data providers while allowing users to access and analyze data on demand directly from different clouds. The cloud platforms or data providers are responsible for maintaining and updating the original RS image data. When users initiate data access requests with clients, the master node locates the original data and notifies workers to read and cache the data from the remote clouds to the local in-memory (MEM) and hard disk drive (HDD). As a result, users do not need to keep a full copy of all data locally and can access and analyze data on demand directly from different clouds, avoiding the high requirement of local storage capacity for large-scale datasets.

The data orchestration across clouds presented in this article is capable of virtually mounting RS data from multiple cloud platforms and local file systems, providing a global virtual view of the data. Furthermore, it allows users to transparently cache frequently accessed data, especially from remote locations, which can provide memory-level data I/O throughput for subsequent data discovery, data cube retrieval, and data access across clouds. Overall, this data orchestration solution effectively addresses the challenges of sharing and accessing data across diverse EODC

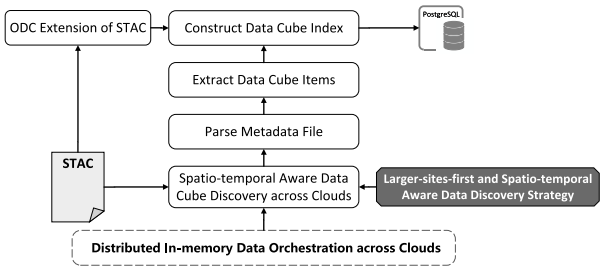


Fig. 4. Process of spatio-temporal aware data cube discovery across clouds.

solutions, thereby facilitating efficient access to multisource RS data across cloud platforms and data cube platforms.

C. Distributed Spatio-Temporal Aware Data Cube Discovery Across Cloud

To address the challenges of inefficient data discovery and joint use for regional or global large-scale RS data analysis, this article proposes a distributed spatio-temporal aware data cube discovery approach across cloud platforms, as illustrated in Fig. 4. First, we propose an LSA RS data discovery strategy, inspired by web data crawling strategies in the internet domain and taking into account the temporal and spatial attributes of RS data. The LSA strategy helps users quickly and accurately locate the target RS data for a particular application from multiple datasets mounted in a distributed data orchestration across clouds. Second, we parse the RS metadata files found in the distributed in-memory data orchestration based on the SpatioTemporal Asset Catalog (STAC) standard, extract key parameter information, and construct data cube indexes to store in a local database. The cube extension of STAC allows for the direct construction of data cube indexes from the original STAC Item files, eliminating the tedious and time-consuming process of index construction. To improve the performance and scalability of data discovery, a distributed data discovery architecture is deployed to manage and execute various data discovery tasks, including regular and incremental data discovery tasks.

The architecture of our distributed data discovery architecture is based on Crawlab and Redis. Crawlab [87] is chosen as the management platform for data discovery tasks. It is responsible for parallel task scheduling, interval task management, task status monitoring, and result summary and display. Redis [88] is used to store task queues and results to ensure communication and data synchronization among distributed task nodes. When users wish to perform data discovery across cloud platforms, they can select different data discovery tasks (regular or incremental), the number of task nodes, and the target discovery parameters on the Crawler master. The master schedules a specified number of nodes to synchronize tasks with workers and perform data discovery tasks concurrently. A thread pool is set for every worker to dynamically adjust the number of threads performing different analysis tasks during the data discovery process, thereby improving the performance and scalability of data discovery.

1) LSA Data Discovery Strategy Across Clouds: The in-memory virtual data orchestration across clouds enables a unified view (in the form of a directory tree) of all mounted datasets, eliminating the challenge of data sharing and collaboration across multiple clouds. When users want to discover the data required for applications from multiple cloud platforms, it is necessary to search the directory tree structure provided by the data orchestration across clouds and parse the metadata files of RS scene data in it to determine whether the scene data meet the target requirements. However, in the field of RS, there is currently no mature algorithm for efficient data discovery across clouds. After mounting terabyte and petabyte-scale datasets, traditional DFS or breadth-first search (BFS) algorithms become excessively time-consuming and complex for searching and discovering target data. To address this issue, the LSF crawling strategy [78] offers better scalability and is more suitable for large-scale distributed data crawling, which improves data discovery efficiency. However, this strategy does not consider the temporal and spatial attributes of RS data, and thus cannot be directly applied to RS data discovery. Given the abovementioned problems, this article proposes an LSA RS data discovery across clouds strategy. This strategy takes into account the temporal and spatial attributes of RS data and supports fast discovery across clouds, thereby providing an efficient and effective solution for RS data discovery.

The LSA data discovery strategy is presented in Algorithm 1 and illustrated in Fig. 5. The strategy comprises the following steps.

Step 1: Get the user's data discovery task parameters, including data type, spatio-temporal range, cloud coverage, etc.

Step 2: Create data discovery tasks based on the caching. Check the distributed in-memory database (Redis) to determine whether a data discovery task with the spatio-temporal intersection has been cached. If the cache hits, create an incremental data discovery task to search for updated data only within that region; otherwise, create a new regular data discovery task.

Step 3: Select the data sources of the "larger sites" cloud platforms for data discovery among all cloud platforms registered in the current system, following the LSF strategy. Cloud platforms with larger datasets, higher data access frequency, and access volume are deemed "larger sites" and searched first to locate target data and improve efficiency.

The steps involved are.

Step 3-1: Poll all registered data source sites (including AWS S3, COS, OSS, HDFS, and LOCAL) for virtual mount directories in the in-memory data orchestration.

Step 3-2: Calculate the site "rank" for all sites according to the dataset size, data access frequency, and access volume.

Step 3-3: Create new site data discovery tasks for each cloud platform data source site and add them to the directory task scheduling queue(stored in the zset structure of Redis, which is automatically sorted according to rank).

Algorithm 1: LSF and Spatio-Temporal Aware.

Input: paramlist
Output: results
function DATA_DISCOVERY($paramlist$)
  $params \leftarrow getTargetParamas(paramlist)$ ▷ get target parameters
  if $params\ is\ in\ Redis's\ task\ cache$ **then** ▷ cretae data discovery tasks based on caching
    $create\ an\ incremental\ data\ discovery\ task$
  else
    $create\ a\ regular\ data\ discovery\ task$
  for $site\ in\ datasites$ **do** ▷ poll data site
    $rank \leftarrow CALCULATE\_RANK(site)$ ▷ calculate site rank
    $dir\_queue.add(site, rank)$ ▷ store and sort site rank
  while $dir\_queue\ is\ not\ empty$ **do**
    $dir \leftarrow dir\_queue.popmax()$ ▷ select the directory discovery task with the maximum rank
    $childlist \leftarrow alluxio.ls(dir)$ ▷ get a list of subdirectories through alluxio
    if $childlist\ is\ scene\ data$ **then** ▷ subdirectory is scene data
      $match \leftarrow CALCULATE\_MATCH(childlist[0], params)$ ▷ calculate scene data match
      for $child\ in\ childlist$ **do**
      $scene\_queue.add(child, match)$ ▷ store and sort scene data match
    $threadpool.submit(DATA\_ANALYSIS, params)$ ▷ execute a new thread to analyze scene data
  else ▷ subdirectory is normal directory
    for $child\ in\ childlist$ **do**
      $rank \leftarrow CALCULATE\_RANK(child)$ ▷ calculate subdirectory rank
      $dir\_queue.add(child, rank)$ ▷ store and sort subdirectory rank
function DATA_ANALYSIS$params$
  while $scene\_queueisnotempty$ **do**
    $scene \leftarrow scene\_queue.popmax()$ ▷ select the scene data analysis task with the maximum match
    $keyparams \leftarrow parseMetadata(scene)$ ▷ analyze scene data
    if $params = keyparams$ **then** ▷ compare with target parameters
      $results.add(scene)$ ▷ add to results set
      $updateRank(scene)$ ▷ update the rank of this scene's parent directories
      $updateMatch(scene)$ ▷ update the match of this scene's sibling nodes
  return $results$
function CALCULATE_RANKdir
  $level \leftarrow getLevel(dir)$ ▷ get the level of directory
  $site\_pr \leftarrow getDataSitePriority(dir)$
  if $level = 2$ **then** ▷ data site level
    $rank \leftarrow level * site\_pr$
  else if $level = 3$ **then** ▷ dataset level
    $type\_pr \leftarrow getDataTypePriority(dir)$
    $rank \leftarrow level * site\_pr * type\_pr$
  else ▷ normal directory level
    $type\_pr \leftarrow getDataTypePriority(dir)$
    $num \leftarrow db.ls(dir).length$
    $num\_pr \leftarrow 1 - 1/(num + 1)$
    $rank \leftarrow level * site\_pr * type\_pr + num\_pr$
  return $rank$
function CALCULATE_MATCH$scene, params$
  $time, bbox, type \leftarrow readMetadata(scene)$ ▷ get scene information
  $search\_time, search\_bbox, search\_type, \leftarrow readParam(params)$ ▷ get specific target parameters
  $time\_match \leftarrow calculateTime(time, search\_time)$ ▷ calculate data time_match
  $spce\_match \leftarrow calculateSpace(bbox, search\_bbox)$ ▷ calculate data space_match
  $type\_match \leftarrow calculateType(type, search\_type)$ ▷ calculate data type_match
  $order = getDataOrder(scene)$ ▷ get dataset major order
  $w1, w2 \leftarrow getWeights(order)$ ▷ set weight coefficient according to major order
  $match = (w1 * time\_match + w2 * space\_match) * type\_match$ ▷ calculate scene data match
  return $match$

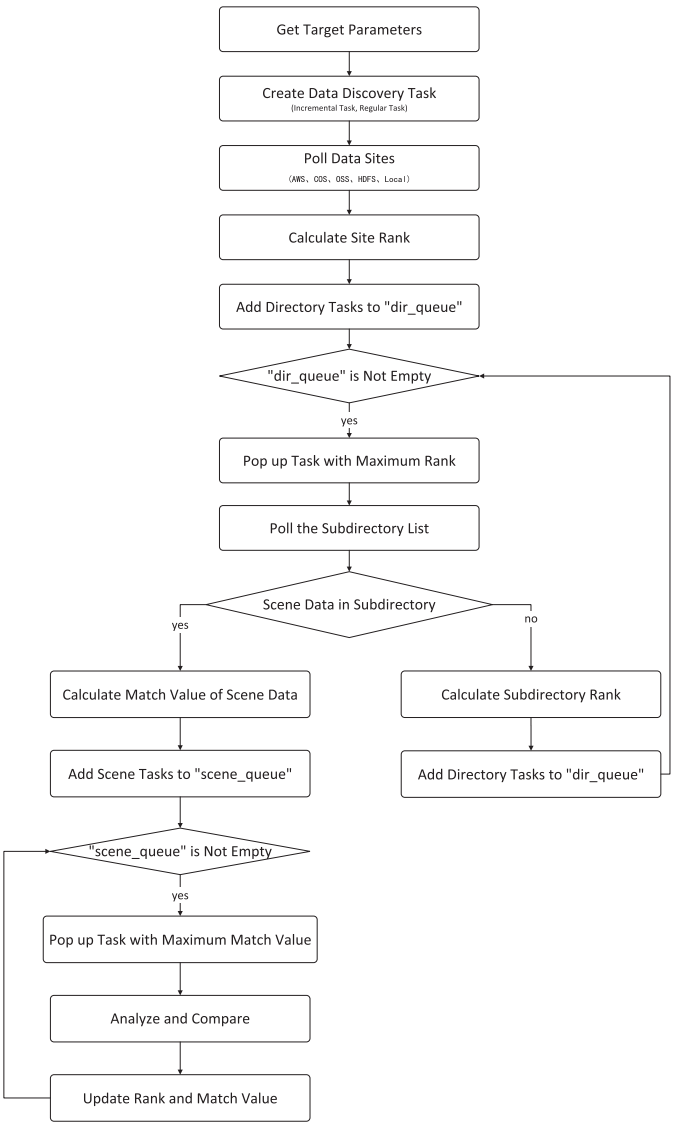


Fig. 5. Dataflow of LSA data discovery strategy cross clouds.

Step 4: Based on the spatio-temporal aware data search strategy, create multiple data search threads in the distributed cluster, which calculate the "match" value between the spatio-temporal data range of the RS scene data and the user's data discovery task. The scene data with a better "match" value is selected for further data analysis.

The steps involved are given as follows.

Step 4-1: Create multiple data search threads in the distributed cluster system.

Step 4-2: Each data search thread is used to select the site/directory data discovery task with the maximum "rank" from the directory task scheduling queue to execute.

Step 4-3: Perform recursive data search and discovery for the selected site/directory task. Poll all data subdirectories in the data directory corresponding to the site/discovery task and calculate the "rank" of these data subdirectories. Create new directory discovery tasks for these data subdirectories and add them to the directory task scheduling queue.

Step 4-4: Repeat steps 4-2 to 4-3 until the RS scene data are found in the directory. Then, randomly select one scene data from the directory to calculate the "match" value, which represents the "match" value of the remaining scene data in the directory. Create the new scene data analysis tasks for these scene data and added them to the scene data task scheduling queue (stored in the zset structure of Redis, which is automatically sorted according to match).

Step 4-5: Select the scene data analysis task with the maximum "match" value from the scene data task scheduling queue each time to perform further data analysis. Compare this scene data with the target parameter requirements and record the data path information that meets the data discovery requirements into Redis for storage, for subsequent data cube indexing and integration. Then, update the rank of the parent directory of this scene data and the match of the rest of the scene data.

Directory rank metric: With more and more petabytes of analysis-ready datasets being available free on the cloud, it is time-consuming and difficult to search all the datasets and analyze whether they meet target requirements. To address this challenge, we apply the LSF strategy, commonly used in web crawling, to RS data discovery. This strategy involves prioritizing the downloading of the most "important" pages "early" during the crawling [78]. In our case, we prioritize "larger sites" cloud platforms with larger datasets, higher data access frequency, and access volume as they are likely to contain the target data. We set a "rank" metric for the directory discovery tasks, which indicates the order in which they should be executed. Directory tasks with higher "rank" are expected to be executed first to help users discover the target data earlier. The "rank" is described by the following equation:

$$\text{rank} = \text{level} * \text{site\_pr} * \text{type\_pr} + \text{num\_pr}. \quad (1)$$

Level: It refers to the level of the data directory, which can be determined directly from the data path. The scene data are located on the leaf nodes of the tree. The deeper the level, the more likely to find the target data. As a result, rank and level are positively correlated, the deeper the level, the higher the rank.

site_pr: It refers to the access priority of the data site, which is a number in the range of [0,1]. The dataset size, data access frequency, and access volume of data sites of cloud platforms are different. For example, local data are accessible faster than data from remote clouds. Amazon Cloud's public RS datasets are maintained and updated by authoritative institutions or data platforms, such as the U.S. geological survey, digital Earth Africa, and INPE-Brazil data cube, which have higher access frequency. Data sites with larger datasets, higher data access frequency, and access volume have higher priority, the earlier we access them during the data discovery task. Therefore, rank and site_pr are positively correlated; the larger the site_pr, the higher the rank.

type_pr: It refers to the data type access priority, which is a [0,1] number. Based on the initial dataset information, if we explicitly know the dataset to contain the target data type, set

type_pr = 1 to prioritize access. If we explicitly know the dataset to not contain the target data type, set type_pr = 0 and drop the directory task. If the data type of the dataset is unknown and it is not certain whether it meets the target requirements, set type_pr = 0.5 and analyze it later.

num_pr: It refers to the number priority, which is a number in the range of (0,1). When accessing different directories from the same level, the same data site, and the same data type, we prioritize the directory containing more subdirectories for analysis, so that there is a greater possibility of finding more data. Rank and the number of subdirectories (num) are positively correlated, the larger the num, the higher the rank. Since num is a positive integer number, and the value is generally large, we normalize num to a number in the range of (0,1) as num_pr, with the equation as (2). In addition, the num_pr metric is mainly used to measure the priority of the directory from the same site, type, and level, and the impact on rank is smaller than the previous three metrics, so we use addition to num_pr in the equation of rank.

$$\text{num\_pr} = 1 - \frac{1}{\text{num} + 1}. \tag{2}$$

On the whole, we mainly refer to the LSF strategy for web crawling to design the "rank" metric. We computed level, site_pr, type_pr, and num_pr metrics, and all of them have a positive relationship with "rank." Among these metrics, level, site_pr, and type_pr have a greater influence on "rank," and num_pr has a smaller influence on "rank." A higher "rank" indicates earlier access to the directory during data discovery tasks, enabling us to discover more target data from larger sites earlier.

Scene data match value: Although "rank" metrics designed with the LSF strategy can effectively improve the efficiency of data discovery, it may not be entirely suitable for RS data discovery across clouds as it ignores the temporal and spatial attributes of RS data organization. Currently, most datasets are organized and stored in time or space order. For example, USGS's global Landsat and Digital Earth Africa's Landsat data are first divided into directories by different years, and then by path and row codes regarding the WRS. They organize the overall data in the order of "Year-Path-Row," which means the order of time before space. Similarly, for Sinergise's Sentinel2, Brazil Data Cube's Sentinel2, and Digital Earth Africa's Sentinel2, the data are first divided into directories by different spatial codes and then by different years. Therefore, we design a "match" value to indicate the data match value between the scene and the target. We randomly select one scene data from the parent directory, parse the metadata file to calculate the "match" value, and use the "match" value to represent the priority for the scene analysis task of other scenes in the same directory. The "match" value is described by the following equation:

$$\text{match} = w1 * \text{time\_match} + w2 * \text{space\_match}. \tag{3}$$

time_match: It refers to the match between the time of the scene data and the target time, which is a number normalized to the range of (0,1]. When time_match is closer to 1, it means the time of scene data is closer to the target time, and when time_match=1, it means the time of scene data fully meets the target time requirement. When calculating time_match, we mainly measure the time distance between the scene data time and the target time. In addition, the specific calculation should distinguish whether the data discovery target is a time point or a period to discuss. When the data discovery target is a period, time_match is described as follows:

$$\text{time\_match} = \begin{cases} 1 & T_s \le T_o \le T_e \\ 1 - \frac{T_s - T_o}{T_e - T_o} & T_o < T_s \\ 1 - \frac{T_o - T_e}{T_o - T_s} & T_o > T_e. \end{cases} \tag{4}$$

In the equation, T_s is the start time of the target time range, T_e is the end time of the target time range, and T_o is the time of the scene data. We use the ratio of the distance between the scene data time and the target time range to measure the time match, and normalized it. The smaller the time distance ratio is, the smaller the time distance is, and the larger the time_match is, the closer to the target time. Therefore, time_match is positively related to match, the larger the time_match, the larger the match.

When the data discovery target is a time point, we consider this point as the smallest period (one day). T_p is the target time point and T_o is the scene data time point. If $T_o < T_p$, set $T_s = T_p, T_e = T_p + 1$; if $T_o > T_p$, set $T_s = T_p - 1, T_e = T_p$. Bringing them to (4), the time_ match of the time point is calculated in (5).

$$\text{time\_match} = \begin{cases} 1 & T_o = T_p \\ 1 - \frac{T_p - T_o}{T_p + 1 - T_o} & T_o < T_p \\ 1 - \frac{T_o - T_p}{T_o - T_p + 1} & T_o > T_p. \end{cases} \tag{5}$$

space_match: It refers to the match between the spatial range of the scene data and the target space, which is a number normalized to the range of (0,1]. When space_match is closer to 1, it means the spatial extent of scene data is closer to the target spatial extent; when space_match=1, it means the spatial extent of scene data intersects with the target spatial extent. The scene data of RS are usually irregular polygon. To simplify data analysis, we often use the minimum bounding rectangle (MBR) of the scene to represent its spatial extent. When calculating the space_match of the data, the key is to compare the space distance between the MBR of scene data and the target bounding rectangle. space_match is described in the following (6) shown at the bottom of this page:

In the equation, rectangle B represents the MBR of the scene data, and rectangle S is the target spatial extent. B_x is the length of rectangle B in the *X* (longitude), B_y is the length of rectangle B in the *Y* (latitude), S_x is the length of rectangle S in the *X*, S_y

$$\text{space\_match} = \begin{cases} 1 & O_x \le (S_x + B_x)/2 \text{ and } O_y \le (S_y + B_y)/2 \\ \min\left(\frac{(S_x + B_x)/2}{O_x}, \frac{(S_y + B_y)/2}{O_y}\right) & O_x > (S_x + B_x)/2 \text{ or } O_y > (S_y + B_y)/2. \end{cases} \tag{6}$$

is the length of rectangle S in the *Y*, O_x is the distance between the center point of rectangle B and the center point of rectangle S in the *X*, and O_y is the distance between the center point of rectangle B and the center point of rectangle S in the *Y*. When the two rectangles intersect, the spatial extent of the scene data meets the target spatial requirement, and the space_match is 1. When the two rectangles do not intersect, we calculate the spatial distance ratios of the two rectangles separated in the *X* and *Y*, respectively, and choose the smallest distance ratio in the *X* and *Y* as the whole spatial distance ratio. The smaller the spatial distance ratio is, the smaller the spatial distance is, the larger the space_match is, and the closer to the target space. Therefore, the match increases with the increase of space_match.

The w1 and w2 are the associated weights of the two metrics, and w1 + w2 = 1. The w1 represents the weight of time_match, and w2 represents the weight of space_match. When the major order of the dataset is time, then w1 > w2; when the major order of the dataset is space, then w2 > w1.

On the whole, we fully consider the temporal and spatial attributes of RS data to design "match" metrics to achieve spatio-temporal aware RS data discovery. In the same directory, the "match" value of one scene data is used to represent the "match" value of its sibling nodes, and the "match" value is dynamically adjusted and updated during the scene analysis task. We use time_match and space_match metrics to measure the overall spatio-temporal data "match" between the scene data and the target. A higher "match" value indicates that we should perform earlier the scene analysis task, to discover the target data faster among millions of scene data and increase the efficiency of data discovery.

2) Data Cube Parsing and Indexing With STAC Standards: To address the challenge of the tedious and time-consuming process of constructing a data cube index for diverse RS datasets, we aim to implement automatic data parsing and data cube indexing for discovered RS data. There are various RS data products with different formats and standards, which makes it difficult to extract cube items and construct the data cube index. Therefore, this article focuses on parsing and data cube indexing of RS metadata adhering to the STAC standard. STAC is a standard that enhances access to information about spatio-temporal data, providing a standardized way to disclose and query RS datasets and making them more accessible and interoperable [89]. In addition, STAC supports data cube extensions, and most data cube platforms already use the STAC standard to index data or data cube products [90]. The files of STAC items can be directly parsed and converted to data cube format, which can improve the work efficiency without the user's tedious work of metadata parsing and data cube index construction.

First, we offer predefined data products, such as Landsat-5/7/8/9 and Sentinel-1/2, and support the parsing and indexing of most datasets using STAC standards. We also provide product definitions for Chinese domestic satellites GF-1/2/3, facilitating automatic parsing of their XML format metadata files and data cube index construction. Moreover, users can customize more data cube products and metadata formats, and we support the free extension of data cube products. Second, we use Alluxio to read metadata files and use the stac_transform function in the odc.apps.dc_tools library to directly parse the original STAC standard metadata file into the form of a data cube index. Third, we supplement the generated data cube index with the product type of the data cube and additionally add the attribute field of the data_source, which is used to record the different data sources of the RS dataset. Then, we need to perform virtual path mapping to map all the original data access paths into virtual access paths in Alluxio to support the subsequent data integration and access across clouds based on Alluxio. Finally, based on the data product definition, we use the index_update_dataset function of odc.apps.dc_tools.utils to load and store the complete data cube index information into the ODC's PostgreSQL database.

In this way, subsequent data applications can directly retrieve and access RS data stored in multiple cloud platforms and distributed file systems through the local data cube indexes, thus supporting unified data cube retrieval and data access across clouds.

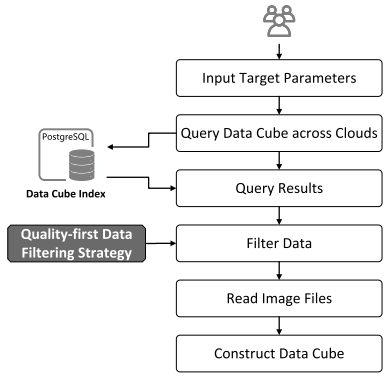


Fig. 6. Process of data cube retrieval and optimization across clouds.

D. Data Cube Retrieval Across Clouds With QF Filtering

The traditional scene file storage structure is complex and time-consuming for time series analysis of RS data. To address this, we expect that data can be processed in the multidimensional spatio-temporal array structure of RS data logically. This article deploys a data cube platform to manage spatio-temporal RS data using the widely adopted ODC [63] framework in various EODC implementations [32]. We have extended and optimized the ODC with the use of Alluxio to read data to support data cube indexing directly from different cloud platforms, allowing data cube retrieval across clouds without downloading metadata files locally. Fig. 6 shows the process for data cube retrieval and optimization across clouds.

In the process of data cube construction, users need to retrieve and filter out some data based on query conditions to provide support for subsequent data analysis and processing. However, the traditional RS data retrieval conditions are relatively simple, and a large number of retrieval results that meet the requirements will be obtained, and there may be a large amount of redundant data that overlap in time range or spatial region. If users want to finely filter out the ideal data for data analysis and calculation, they need to further filter the data manually, which is very time-consuming and laborious.

Algorithm 2: Qf Data Filtering.

Input: Q
Output: Results
function Data_Filtering$Q$
$Q_1, Q_2 \ldots Q_n \leftarrow getChunk(Q)$ ▷ the target area is chunked
for $i = 1$ *to* n **do** ▷ perform data filtering tasks in parallel
$datalist \leftarrow findDatasets(Q_i)$ ▷ query data cube across Clouds
for $data$ *in* $datalist$ **do**
$data.S \leftarrow Polygon(data.extent)$ ▷ get the polygon of the data space extent
$data.score \leftarrow getScore(data.cloud\_cover)$ ▷ set the scoring criteria for data sorting, default is cloud
coverage(score = - data.cloud_cover)
$Candlist \leftarrow datalist.sort(score)$ ▷ sort the data by data.score to form a candidate list
while $Q_i.area > \epsilon$ *and* $Candlist.size > 0$ **do** ▷ ϵ is a number close to 0, depending on the precision
$data \leftarrow Candlist.popmax()$ ▷ select the one with the highest score from the candidate list
$Inter \leftarrow data.S \bigcap Q_i$
if $Inter.area > \epsilon$ **then** ▷ determine if the selected data intersects with the target area
$Q_i \leftarrow Q_i - Inter$ ▷ subtracts the intersecting polygons from the target area
$Results.add(data)$ ▷ the selected data is added to the result set
return $Results$

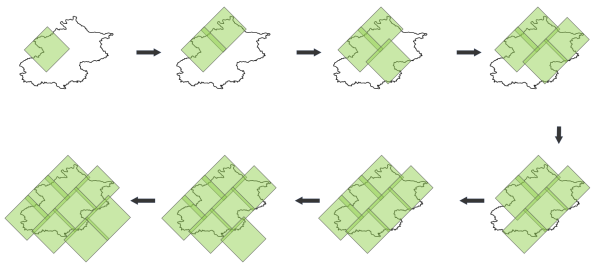


Fig. 7. Filtering process for data covering the target area.

To address the abovementioned problems, this article proposes a QF filtering strategy for data cube retrieval across clouds, which solves the problem of redundant data. This strategy can filter out data with high quality (i.e., less cloud coverage) that covers the target spatio-temporal range to construct the data cube. This data filtering strategy prevents poor-quality data from affecting data analysis results and also avoids loading duplicate data, which can affect the efficiency of data analysis. The strategy is outlined in Algorithm 2, and the data filtering process that covers the target area is depicted in Fig. 7.

Step 1: Chunk the target query area (spatial range) to get several target subspace regions Q1, Q2,... Qn to filter data in parallel.
Step 2: Query data cubes across clouds based on each target subspace to obtain a list of data (each scene data includes information, such as time, spatial range, and cloud coverage) that meet the target requirements.
Step 3: According to the boundary of the spatial range of each scene data in the list, create the corresponding polygon S of scene data, and sort all the data according to the specified scoring criteria to form an ordered candidate list. In this article, data quality is selected as the scoring criterion (score = – cloud_cover), that is, the less cloud coverage, the higher the data quality and the higher score. Of course, users are free to customize this scoring criterion for data sorting and filtering according to the actual application needs.
Step 4: When the candidate list is not empty and the area of subspace Qi is larger than ϵ (ϵ is a number close to 0, depending on the precision), the data with maximum score (the best data quality) is selected from the candidate list each time.
Step 5: Judge whether polygon S of the selected data intersects the target subspace Qi. If they intersect, the data will be added to the result set, and the intersecting polygons (S intersects Qi) will be subtracted from Qi.
Step 6: Repeat Step 4 and Step 5 until the target subspace Qi is completely covered.

Overall, the QF filtering strategy aims to select high-quality and nonoverlapping data from a large amount of available data in order to cover the target area with minimal overlap. This strategy could help improve the data processing efficiency and ensure better data analysis results.

V. Experiments

In this article, we present the method of in-memory distributed data cube autodiscovery and retrieval from RS big data across clouds. For performance and scalability analysis, we conducted multiple groups of experiments, given as follows:

1) several performance comparative experiments on different data discovery methods (ODC-based, BFS, DFS, I-BFS, I-DFS, and LFS) and LSA data discovery strategy;
2) several scalability experiments on LSA data discovery strategy;
3) a scalability experiment on data cube retrieval across clouds, and a performance experiment on retrieval with and without QF data filtering.

We conducted these comparative experiments in a HPC cluster environment equipped with 13 nodes. Three nodes are configured with Zookeeper service to support the high availability of Alluxio masters and ten nodes are Alluxio workers, providing

TABLE I
RS DATASETS MOUNTED TO IN-MEMORY DATA LAYER FROM ACROSS THE CLOUDS

Virtual Path	Data Site	Original Path	Dataset Description
/aws/sentinel-cogs	Amazon Cloud	s3://sentinel-cogs/sentinel-s2-l2a-cogs	Global Sentinel-2, PB scale
/aws/deafrica-landsat-oli-tirs	Amazon Cloud	s3://deafrica-landsat/collection02/level-2/standard/oli-tirs	African regional Landsat-8/9, TB scale
/aws/deafrica-landsat-etm	Amazon Cloud	s3://deafrica-landsat/collection02/level-2/standard/etm	African regional Landsat-7, TB scale
/aws/deafrica-landsat-tm	Amazon Cloud	s3://deafrica-landsat/collection02/level-2/standard/tm	African regional Landsat-5, TB scale
/cos/GF/GF01_PA	Tencent Cloud	cos://RSdata/GF01_PA	China's regional GF-1, GB scale
/cos/Landsat-8	Tencent Cloud	cos://RSdata/Landsat-8	African regional Landsat-8, GB scale
/oss/GF01_MS	Ali Cloud	oss://RSdata/GF01_MS	China's regional GF-1, GB scale
/oss/Landsat-8	Ali Cloud	oss://RSdata/Landsat-8	African regional Landsat-8, GB scale
/hdfs/GF02_level2	HDFS	hdfs://clu08:9090/GF02_level2	China's regional GF-2, GB scale
/hdfs/GF03_level2	HDFS	hdfs://clu08:9090/GF03_level2	China's regional GF-3, GB scale
/hdfs/Landsat-8	HDFS	hdfs://clu08:9090/Landsat-8	African regional Landsat-8, GB scale

a total of 20 TB of storage capacity through a tiered storage mechanism. Each node has 8 core CPUs and 8 GB memory, and the operating system is Centos7.9.

A. Experimental Datasets

In the data orchestration with Alluxio, we mounted several RS datasets from Amazon Cloud, Ali Cloud, Tencent Cloud, and HDFS. The details of the RS datasets mounted to the in-memory data layer from across the clouds are given in Table I.

B. Performance Experiments on Data Discovery

To test the effectiveness and efficiency of the LSA data discovery strategy, three groups of comparative experiments have been conducted in this section. The first group is a comparative experiment on ODC-based data access and LSA. Because the existing RS data managing or sharing platforms could barely provide direct and effective RS data discovery across clouds, whereas the ODC could offer data accessing and indexing across clouds through APIs of cloud storage buckets. Therefore, we use ODC to simulate data access across clouds, called the ODC-based data access method in the experiment, to compare with LSA data discovery. The second group is comparative experiments with some typical tree traversal strategies, such as DFS) BFS, and improved DFS (I-DFS) and improved BFS (I-BFS). This is because Alluxio can provide in-memory directory trees once we mount datasets from multiple clouds, which causes data discovery across clouds to look like a directory tree traversal process. So we compared LSA with the mainstream directory tree traversal methods. The third group is a comparative experiment on LSA and LSF data crawling strategy used for Internet data.

1) Comparative Experiment on ODC-Based Data Access and LSA Data Discovery: We have conducted five comparative experiments of data discovering between the ODC-based method and our LSA strategy, as the number of data scenes scales from 10, 50, 100, and 200 to 500, whereas the ODC-based data accessing follows a manual way. We have to download and parse the metadata of the entire huge datasets including our requested data from the cloud platform using s3 API ("s3-to-dc" command of odc_apps_dc_tools package) offered by ODC based on user experiences. Then, the data must be indexed into ODC to support data access across cloud platforms. The experimental data are the Landsat8 datasets in a temporal span of the year 2019 and a spatial range of $0° - 30°N, 0° - 30°E$.

TABLE II
TIME OVERHEAD OF ODC-BASED DATA ACCESS AND LSA DATA DISCOVERY

Number of discovered data(scene)	data access ODC-based time(s)	LSA data discovery time(s)
10	410s	18s
50	1284s	21s
100	2510s	24s
200	5940s	55s
500	18080s	157s

According to the experimental results in Table II, the time overhead of both approaches goes up nearly linearly, as the amount of discovered data increases. The time performance of the LSA data discovery is 22 times better than that of the ODC-based data access. As the amount of data increases, the time gap between the two approaches becomes even larger. Especially in the case of discovering 500 scenes, the time performance of the LSA data discovery is 115 times better than that of the ODC-based data accessing. The main reason is that the data accessing, filtering, and huge data downloading operations are mainly done in a manual way in ODC-based accessing. This could inevitably result in extra higher time overhead when compared with our

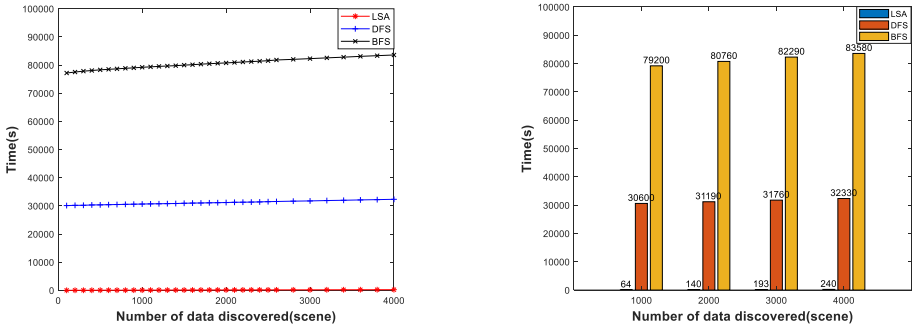


Fig. 8. Time overhead of DFS, BFS data search, and LSA data discovery.

LSA approach. In contrast, the LSA data discovery strategy can support automatic data searching among large RS datasets across clouds based on application requirements, without manually searching, filtering, and downloading. Because Alluxio virtually mounts multiple datasets across clouds, it can support local virtual data through data prefetching. The LSA data discovery strategy is spatio-temporal aware, which can narrow the searches in the data discovery process and find the target data more quickly from a large number of mounted datasets. Apparently, the performance of LSA is much better than the ODC-based method. When the data scale became larger, the performance of LSA turned out to be even more outstanding than ODC-based data access.

2) Comparative Experiment on DFS, BFS, I-DFS, I-BFS, and LSA Data Discovery: The data orchestration across clouds with Alluxio supports unified data access across clouds in the form of a directory tree. To test the performance of the LAS data discovery strategy, we compare it with DFS and BFS directory tree traversal strategies, comparing the time overhead of the different approaches to discover the same amount of data. We implemented data discovery experiments to find Landsat8 data with a time in 2018, a spatial range of $0^\circ - 30^\circ N, 0^\circ - 30^\circ E$, and a cloud coverage of less than 50%. It is an experiment on small datasets of terabyte scale.

According to the results shown in Fig. 8, the time overhead of all three data discovery methods is increasing linearly as the amount of discovered data increases. Among them, the time overhead of LSA is increasing more gently, DFS is the second, and BFS has the most drastic increase in time overhead. When discovering the 4000 scene data, LSA took 6 min, DFS took over 8 h, and BFS took over 23 h. At this point, the time overheads of DFS and BFS are 80 and 230 times higher than that of LSA, respectively. The reason for this significant time gap is that during the data discovery across clouds, the directory discovery and scene analysis tasks have different execution priorities in different data discovery strategies. DFS and BFS data search strategies do not perform any sorting or filtering of directories and scene data during data discovery. So they need to discover and analyze all data in mounted datasets, which is very time-consuming and difficult to find target data. In contrast, the LSA strategy can sort the directories based on the importance and amount of directories and sort the scenes based on the spatio-temporal match of the data and target to narrow the data searches, enabling faster target data discovery. In short, the time performance of the LSA data discovery strategy is better than that of DFS and BFS.

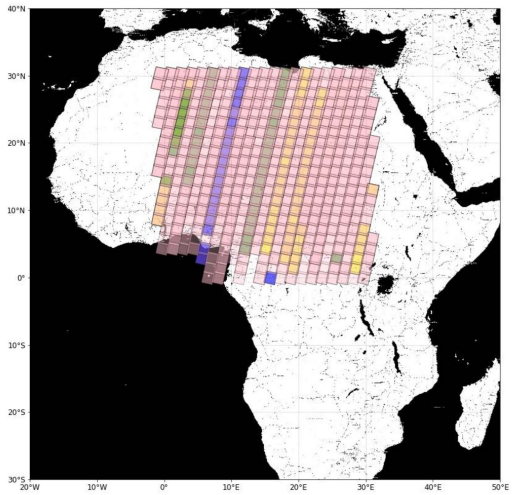


Fig. 9. Region and source of data discovery results across clouds on the map.

Fig. 9 displays 4000 scenes of data obtained from the above-mentioned data discovery experiment on a map. Each small box on the map represents one scene of RS data, with varying colors indicating different data sites (cloud platforms, distributed file systems, local storage, etc.). Pink boxes indicate data sourced from AWS, blue boxes represent data from COS, green boxes indicate data from OSS, and yellow boxes represent data from HDFS. The various colored boxes in the map demonstrate the multiple data sites involved in the data orchestration across clouds in the experiments and confirm that the LAS strategy enables data discovery across clouds.

Considering the organization and directory division of RS data, most datasets are organized by different years. We have improved the BFS (I-BFS) and DFS (I-DFS) strategies to filter directories that do not match the target year. We conducted comparative experiments between LSA and I-BFS, I-DFS data discovery strategies. One experiment was implemented on a small terabyte-scale dataset to find Landsat8 data in the year

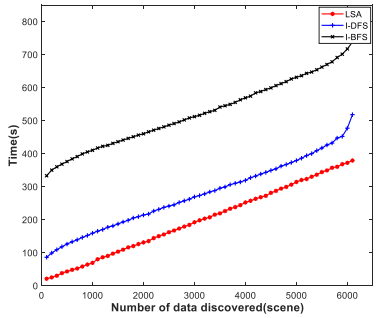


Fig. 10. Time overhead of I-DFS, I-BFS, and LSA data discovery for small datasets.

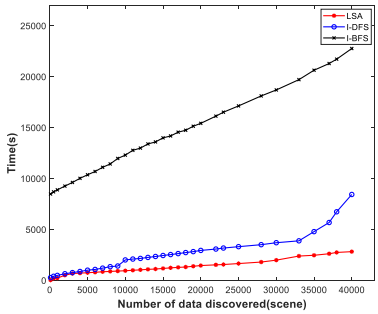


Fig. 11. Time overhead of I-DFS, I-BFS, and LSA data discovery for large datasets.

2020, with a spatial range of $5°S - 20°N, 5°W - 20°E$, and a cloud coverage of less than 80%. The other was implemented on a large petabyte-scale dataset to find Sentinel2 data in the year 2021, with a spatial range of $10°S - 80°N, 25°W - 170°E$, and a cloud coverage of less than 40%.

The experimental results of Figs. 10 and 11 show that LSA has the least time overhead, I-DFS the second least, and I-BFS the most when discovering the same amount of target data. Based on the small-scale data discovery results in Fig. 10, the time overhead of LSA is about 20% less than that of I-DFS and about 50% less than that of I-BFS when discovering 6000 scenes of data. Based on the large-scale data discovery results in Fig. 11, the time overhead of LSA is about 66% less than that of I-DFS and 87% less than that of I-BFS when 40 000 scenes of data are found. This indicates that the performance of LSA turned out to be even more outstanding than I-DFS and I-BFS in the larger-scale data discovery experiments. Because the LSA strategy makes full use of the spatio-temporal information of RS data, the spatio-temporal matching of data and target is used to narrow down the data search, so that the target data can be found quickly from the huge amount of data. While the I-DFS and I-BFS strategies can only filter out the data year directories, the rest of the data directories still need to be traversed and parsed sequentially, which is very complicated and time-consuming. On the whole, the time performance of the LSA data discovery strategy is better than that of the I-DFS and I-BFS methods in large-scale and small-scale datasets, and has greater performance advantages in large-scale data discovery.

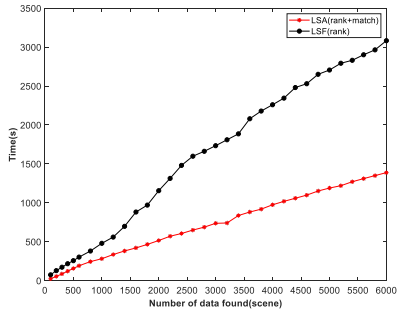


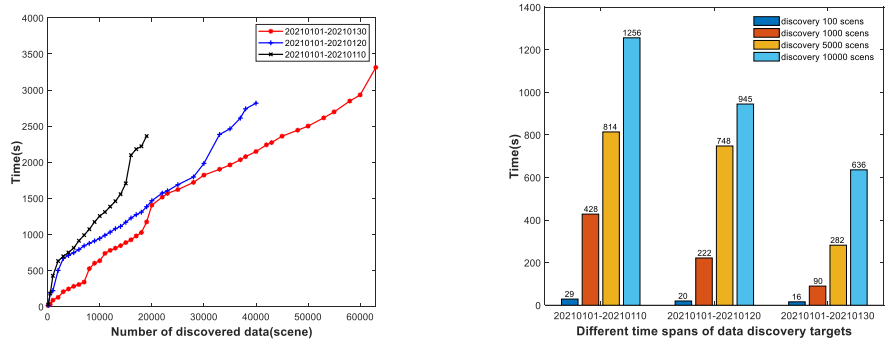
Fig. 12. Time overhead of LSF data search and LSA data discovery.

3) Comparative Experiment on LSF and LSA Data Discovery: Generally, the LSF data crawling strategy ranks the websites based on the number of uncrawled pages as the priority for picking a website. In this experiment, we use the LSF strategy to sort the data directory for data discovery. Unlike the single rank metric of LSF, our LSA strategy uses two metrics to accelerate RS data discovery. During data discovery, we use "rank" values based on the LSF strategy to sort the directories and use "match" values based on spatio-temporal matching of RS data to sort the scenes in the directory. The experiment was implemented to conduct a data discovery task of searching Landsat8 data in a temporal span of the year 2019 and a spatial range of $0° - 30°N, 0° - 30°E$ among all the mounted datasets from clouds, and the data quality requirement was a cloud coverage of less than 60%.

From the experimental results in Fig. 12, we can tell that the time overhead of LSA and LSF is increasing linearly as the amount of discovered data goes up. The time overhead of the LSA strategy could nearly be 50% of that of the LSF Internet data discovering method, as the data scale from 10 to 6000. This indicates that the LSF strategy does not work well for RS data discovery. Because the LSF strategy is mainly used for crawling text data on the Internet, and it does not take multidimensional attributes, such as time and space of RS data into account. The LSF is blind to the spatio-temporal attributes of the data, which could lead to the inefficiency of the whole discovery process. In contrast, our LSA strategy is based on LSF, while taking into account the spatio-temporal attributes of RS data. It narrows the data search by calculating the spatio-temporal match between the scene data and the target, so the target data can be found faster during the data discovery. Overall, the time performance of LSA data discovery is twice as good as the LSF strategy, and the LSA is more efficient in RS data discovery.

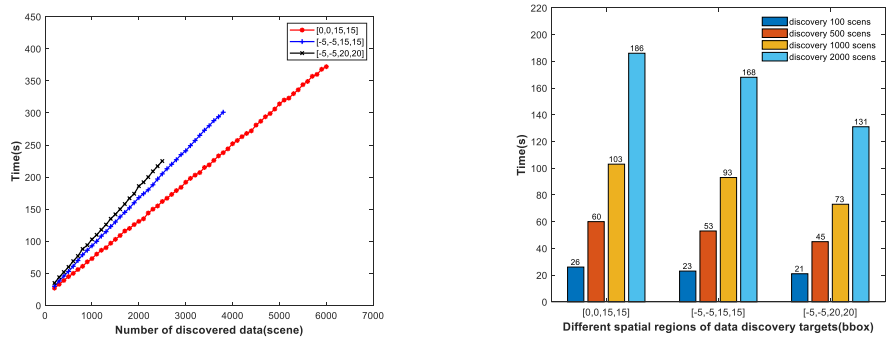
C. Scalability Experiments on Data Discovery

1) Experiment on Spatio-Temporal Scalability of LSA: We tested the scalability of different time spans and spatial regions for LSA data discovery. The first experiment was conducted to discover Sentinel2 data in the Asian region and different time spans (10 days, 20 days, and 30 days), with cloud coverage of less than 50%. The second experiment was conducted to discover Landsat7 data in a temporal span of the year 2020 and different



(a) (b)

Fig. 13. Time overhead of LSA data discovery in different time spans.



(a) (b)

Fig. 14. Time overhead of LSA data discovery in different spatial regions.

spatial regions ($15^\circ \times 15^\circ$ grid, $20^\circ \times 20^\circ$ grid, and $25^\circ \times 25^\circ$ grid), with cloud coverage of less than 50%.

According to Fig. 13(b), when the target time span is scaling from 10 days, 20 days to 30 days, the time overhead of discovering 10 000 scenes of data is reduced by 24% and 32% in order. According to Fig. 14(b), when the target spatial region is scaling from a $15^\circ \times 15^\circ$ grid, a $20^\circ \times 20^\circ$ grid to a $25^\circ \times 25^\circ$ grid, the time overhead of discovering 2000 scenes of data is reduced by 9% and 22% in order. These experimental results prove that the time overhead of discovering the same amount of target data decreases as the time span or spatial region of the discovery is expanded. Because the larger the time span or spatial region of discovery leads to a larger amount of data that meets the requirements, the easier it is to locate the target data from a large amount of mounted data, and the time overhead to discover the same amount of target data decreases. Based on the results of Fig. 13(a), when the time span was from 10 days to 20 days, the total amount of data found doubled and the total time overhead increased by 20%. Based on the results of Fig. 14(a), when the spatial region from a $15^\circ \times 15^\circ$ grid to a $20^\circ \times 20^\circ$ grid, the total amount of data found increased by half and the total time overhead increased by approximately 28%. This is because with the expansion of time span or spatial region, more data of mounted datasets meet requirements, and it takes more time to find all the target data. Overall, as the time span or spatial region of the data discovery target increases within a certain range, the LSA strategy can still discover the target data in a short time with better performance. However, if the time span and spatial region of the data discovery target are expanded to the full spatio-temporal range of the dataset (when all the data of the dataset match the target requirements), the LSA strategy will no longer have the performance advantage. In this case, it would be more appropriate to use the DFS strategy to perform the data discovery task.

2) Experiment on Parallel Scalability of LSA: We deployed a distributed data discovery architecture to manage and execute data discovery tasks. To test the parallel scalability of the LSA data discovery strategy, we execute the same data discovery task with a different number of threads (each node with a thread pool of up to ten threads). We implemented data discovery experiments to discover Landsat7 data in 2020 with a spatial range of $0^\circ - 30^\circ N, 0^\circ - 30^\circ E$ and a cloud coverage of less than 50%.

From the results in Fig. 15, we can tell that the more threads execute tasks, the better the performance of LSA data discovery. The largest performance improvement in data discovery was found when the number of threads was increased from 10 to 20, with a 54% performance improvement. The performance improvement was between 30% and 20% for each additional ten threads as the number of threads increased from 20 to

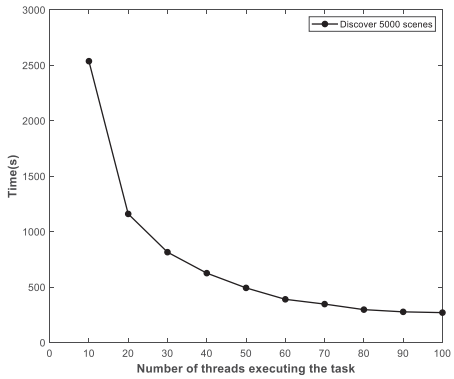


Fig. 15. Time overhead of LSA data discovery with different number of threads.

60. When the number of threads reaches 60, the performance improvement is less than 10% for each additional ten threads and the performance improvement is getting smaller, which may be limited by the network communication or bandwidth of the distributed system, etc. Therefore, using 60 threads (six nodes) can achieve optimal execution efficiency without wasting resources under the current experimental conditions. Overall, the LSA data discovery strategy deployed on a distributed architecture can scale well and improve data discovery efficiency by increasing the number of threads (nodes).

D. Retrieval Experiments With QF Filtering

1) Experiments on Spatio-Temporal Scalability of Data Retrieval: For RS data discovered from multiple cloud platforms, we construct data cube indexes for them and store them in ODC's database, thus supporting subsequent data cube retrieval and access across clouds. To test the scalability of data cube retrieval across clouds in different time spans or spatial regions, we conducted data cube retrieval experiments based on two million data cube indexes in the database. As shown in Fig. 16, we tested the time overhead of data cube retrieval in different time spans (time span: 1 month, 3 months, 6 months, 9 months, and 12 months) and Shandong Province, China. As shown in Fig. 17, we tested the time overhead of data cube retrieval in January 2020 and different spatial regions (spatial region: Shandong Province, Heilongjiang Province, Xinjiang Province, Inner Mongolia Province, and China).

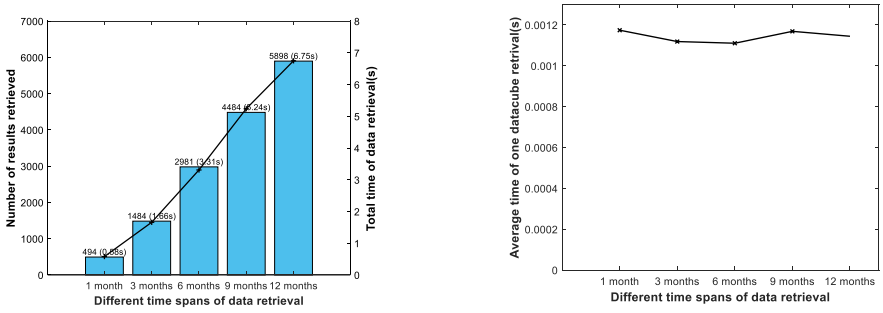
Based on the results in Fig. 16(a), the amount of data retrieved for the 12 months was about 11.9 times that of the one month, and the total time overhead was about 11.6 times. From the results in Fig. 17(a), the amount of data retrieved for the China region was about 73.7 times than that of the Shandong province, and the total time overhead was about 71.84 times. These results indicate that as the time span or spatial region of data retrieval expands, the total amount of data retrieved increases, as does the total time overhead. The reason is that as the time span or spatial region of the retrieval increases, the more data in the database that meet the requirements, the larger the amount of data retrieved. It takes a lot of time to load these retrieval results into memory and return them to the user at one time, so the total time overhead of data retrieval increases. Based on the results in Figs. 16(b) and 17(b), the average time overhead of data retrieval (total time overhead/number of results) remains relatively stable in the range of 0.0010 to 0.0012 s as the time span or spatial region of data retrieval expands. This indicates that the average time overhead of data cube retrieval is hardly affected by the time span and spatial region of retrieval, maintaining a good retrieval performance. Overall, the total time overhead of retrieval is increased by the number of results when the time span or spatial region increases, but the average time overhead of retrieval remains almost constant, which confirms the good spatio-temporal scalability of data retrieval.

2) Comparative Experiment on Data Retrieval With and Without QF Filtering: This article introduces a QF data filtering strategy for data cube retrieval across clouds. To test the performance of this filtering strategy, we constructed a comparison experiment on data cube retrieval with and without the QF filtering strategy. The retrieval experiment was implemented to search Sentinel2 data in January 2021, with a spatial range of $25^\circ - 45^\circ N, 95^\circ - 115^\circ E$.

Fig. 18(a) displays the results of direct data retrieval without data filtering on the map. It took 4 s to retrieve 5812 scene data, which have a large amount of overlap on the map. Fig. 18(b) displays the results of data retrieval with QF data filtering on the map. It took 34 s to retrieve and filter out 894 high-quality scene data, which maximized coverage of the target area with a small overlap. The experimental results show that the QF data filtering strategy can help us filter out a small amount of high-quality data covering the target area from a large amount of data at a small time overhead. This strategy avoids a large number of overlapping data retrieved, thereby improving the efficiency of subsequent data processing and analysis. In addition, it prevents poor-quality data from affecting the accuracy of the data analysis results.

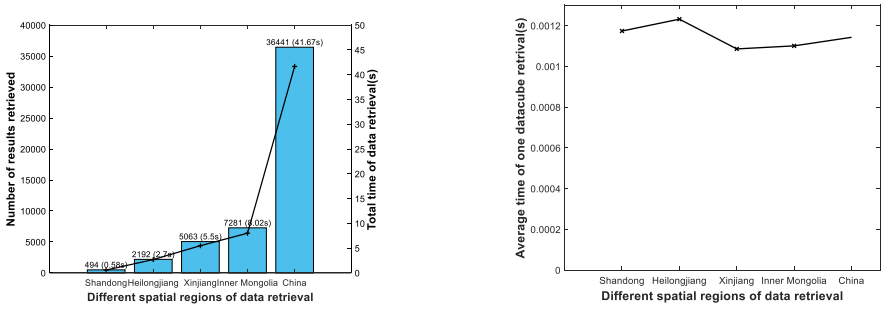
VI. Discussion

In this article, we conduct several comparison experiments to evaluate the performance and scalability of the LSA data discovery and data cube retrieval across clouds. The experimental results show that the methods are effective, efficient, and scalable. From the comparative experiments on LSA and ODC-based data access, the time overhead of the ODC-based approach is tens or even hundreds of times higher than the that of LSA discovery strategy, and the LSA data discovery is more efficient. In the comparative experiments on DFS, BFS, I-DFS, I-BFS, LSF, and LSA discovery, the time overhead of LSA is about 20% less than that of I-DFS and about 50% less than that of I-BFS in the small-scale data discovery experiments; the time overhead of LSA is about 66% less than that of I-DFS and 87% less than that of I-BFS in large-scale data discovery experiments; the time overhead of the LSA strategy is 50% less than that of the LSF strategy. These experimental results show that the LSA strategy is more effective than other data discovery strategies. From the scalability experiments on data discovery, as the time span or spatial region of the data discovery target increases within a certain range, the LSA strategy shows good



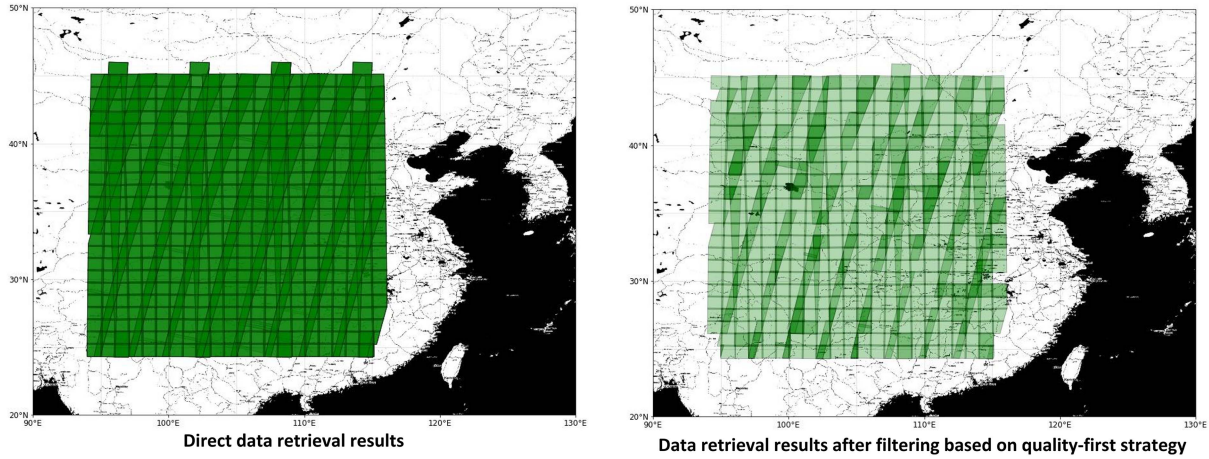
(a) (b)

Fig. 16. Time overhead of data cube retrieval in different time spans.



(a) (b)

Fig. 17. Time overhead of data cube retrieval in different spatial regions.



(a) (b)

Fig. 18. Data cube retrieval results with and without QF filtering.

spatio-temporal scalability. In addition, the LSA data discovery can scale well as increasing the number of threads, and using 60 threads (six nodes) can achieve optimal execution efficiency without wasting resources under the current experimental conditions. The experiments on spatio-temporal scalability of data retrieval show that data cube retrieval scale well and keep the average retrieval time stable within a small range when the time span or spatial region increases. Finally, the comparative experiment on data retrieval with and without data filtering proves that the QF data filtering strategy can help us filter out a small amount of high-quality data covering the target area from a large amount of data at a small time overhead.

In short, the data discovery, retrieval, and filtering method proposed in this article facilitates unified discovery, retrieval, and

access to RS data from multiple cloud platforms, distributed file systems, and local systems. It demonstrates good performance and scalability, thereby enabling users to improve the efficiency of data discovery and acquisition, and focus on data analysis and application. However, network speed and stability can impact the performance of data discovery, due to the data being sourced from multiple public cloud storage. It is important to note that the LSA data discovery strategy may not be suitable for all data discovery tasks.

VII. Conclusion

The amount of global archived EO data has increased significantly, reaching exabytes. The innovative paradigm of the EODC has transformed the traditional way of EO data acquisition, storage, processing, and sharing. However, the diverse data cube solutions have led to limited interoperability among existing data cube infrastructures, hindering data sharing and joint use across them. To tackle these issues, we proposed a method of in-memory distributed data cube autodiscovery and retrieval from RS big data across clouds. We constructed distributed in-memory data orchestration to provide users with a unified in-memory access view and operation interface of different cloud platforms. We proposed an LSA data discovery strategy across clouds and a QF data filtering strategy to support efficient data discovery and retrieval. These efforts overcome the challenge of data cube joint use and enable fast and accurate data discovery and retrieval across clouds. However, network speed and stability could impact the performance of data discovery, due to the experimental data being sourced from multiple public cloud storage. In the future, we will take the multidimensional and geometric characteristics of spatio-temporal RS data into account, as well as the spatial direction of data access in the analysis of RS data, to design access pattern-aware spatio-temporal RS data cache prefetching and replacement strategy, so as to further improve the access performance of spatio-temporal RS data.

Overall, our proposed method allows users to focus on data analysis and processing without the tedious and time-consuming data acquisition, such as searching and downloading data from multiple cloud platforms. This method is beneficial to promote the full potential of spatio-temporal RS data information, facilitating large-scale scientific research on global environmental change and sustainable development.

References

[1] J. T. Overpeck, G. A. Meehl, S. Bony, and D. R. Easterling, "Climate data challenges in the 21st century," *Science*, vol. 331, no. 6018, pp. 700–702, 2011.

[2] G. Giuliani, G. Camara, B. Killough, and S. Minchin, "Earth observation open science: Enhancing reproducible science using data cubes," vol. 4, no. 4, 2019, Art. no. 147.

[3] V. C. Gomes, G. R. Queiroz, and K. R. Ferreira, "An overview of platforms for big Earth observation data management and analysis," *Remote Sens.*, vol. 12, no. 8, 2020, Art. no. 1253.

[4] P. K. Hargreaves and G. R. Watmough, "Satellite Earth observation to support sustainable rural development," *Int. J. Appl. Earth Observ. Geoinf.*, vol. 103, 2021, Art. no. 102466.

[5] M. D. Mahecha et al., "Earth system data cubes unravel global multivariate dynamics," *Earth Syst. Dyn. Discuss.*, vol. 11, no. 1, pp. 201–234, 2020.

[6] G. Boulton, "The challenges of a Big Data Earth," *Big Earth Data*, vol. 2, no. 1, pp. 1–7, 2018.

[7] N. Dey, C. Bhatt, and A. S. Ashour, *Big Data for Remote Sensing: Visualization, Analysis and Interpretation*. Cham, Switzerland: Springer, 2018.

[8] Z. Wu et al., "User needs for future Landsat missions," *Remote Sens. Environ.*, vol. 231, 2019, Art. no. 111214.

[9] P. Kempeneers and P. Soille, "Optimizing Sentinel-2 image selection in a Big Data context," *Big Earth Data*, vol. 1, no. 1/2, pp. 145–158, 2017.

[10] M. Sudmanns et al., "Big Earth data: Disruptive changes in Earth observation data management and analysis?," *Int. J. Digit. Earth*, vol. 13, no. 7, pp. 832–850, 2020.

[11] B. D. Killough, "Satellite analysis ready data for the sustainable development goals," *Earth Observation Applications and Global Policy Frameworks*. Hoboken, NJ, USA: Wiley, 2022, pp. 133–143.

[12] Y. Ma et al., "Remote sensing Big Data computing: Challenges and opportunities," *Future Gener. Comput. Syst.*, vol. 51, pp. 47–60, 2015.

[13] M. Chi, A. Plaza, J. A. Benediktsson, Z. Sun, J. Shen, and Y. Zhu, "Big data for remote sensing: Challenges and opportunities," *Proc. IEEE*, vol. 104, no. 11, pp. 2207–2219, Nov. 2016.

[14] K. A. Moser et al., "Mountain lakes: Eyes on global environmental change," *Glob. Planet. Change*, vol. 178, pp. 77–95, 2019.

[15] P. Liu, J. Li, L. Wang, and G. He, "Remote sensing data fusion with generative adversarial networks: State-of-the-art methods and future research directions," *IEEE Geosci. Remote Sens. Mag.*, vol. 10, no. 2, pp. 295–328, Jun. 2022.

[16] L. Mu, L. Wang, Y. Wang, X. Chen, and W. Han, "Urban land use and land cover change prediction via self-adaptive cellular based deep learning with multisourced data," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 12, no. 12, pp. 5233–5247, Dec. 2019.

[17] W. Han et al., "Sample generation based on a supervised Wasserstein generative adversarial network for high-resolution remote-sensing scene classification," *Inf. Sci.*, vol. 539, pp. 177–194, 2020.

[18] L. Zhang, P. Liu, L. Zhao, G. Wang, W. Zhang, and J. Liu, "Air quality predictions with a semi-supervised bidirectional LSTM neural network," *Atmospheric Pollut. Res.*, vol. 12, no. 1, pp. 328–339, 2021.

[19] Y. Wang, L. Wang, X. Chen, and D. Liang, "Offshore petroleum leaking source detection method from remote sensing data via deep reinforcement learning with knowledge transfer," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 15, pp. 5826–5840, Jul. 2022.

[20] G. Giuliani et al., "Live monitoring of Earth surface (limes): A framework for monitoring environmental changes from Earth observations," *Remote Sens. Environ.*, vol. 202, pp. 222–233, 2017.

[21] A. Shelestov, M. Lavreniuk, N. Kussul, A. Novikov, and S. Skakun, "Large scale crop classification using Google Earth engine platform," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, 2017, pp. 3696–3699.

[22] N. Sánchez, Á. González-Zamora, J. Martínez-Fernández, M. Piles, and M. Pablos, "Integrated remote sensing approach to global agricultural drought monitoring," *Agricultural Forest Meteorol.*, vol. 259, pp. 141–153, 2018.

[23] P. Potapov et al., "Mapping global forest canopy height through integration of Gedi and Landsat data," *Remote Sens. Environ.*, vol. 253, 2021, Art. no. 112165.

[24] S. Nativi, P. Mazzetti, and M. Craglia, "A view-based model of data-cube to support Big Earth Data systems interoperability," *Big Earth Data*, vol. 1, no. 1/2, pp. 75–99, 2017.

[25] Y. Zhang and J. Cheng, "Spatio-temporal analysis of urban heat island using multisource remote sensing data: A case study in Hangzhou, China," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 12, no. 9, pp. 3317–3326, Sep. 2019.

[26] F. Zellweger, P. De Frenne, J. Lenoir, D. Rocchini, and D. Coomes, "Advances in microclimate ecology arising from remote sensing," *Trends Ecol. Evol.*, vol. 34, no. 4, pp. 327–341, 2019.

[27] J. Li, Y. Pei, S. Zhao, R. Xiao, X. Sang, and C. Zhang, "A review of remote sensing for environmental monitoring in China," *Remote Sens.*, vol. 12, no. 7, 2020, Art. no. 1130.

[28] C. A. Lee, S. D. Gasster, A. Plaza, C.-I. Chang, and B. Huang, "Recent developments in high performance computing for remote sensing: A review," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 508–527, Sep. 2011.

[29] Y. Ma, L. Wang, A. Y. Zomaya, D. Chen, and R. Ranjan, "Task-tree based large-scale Mosaicking for massive remote sensed imageries with dynamic DAG scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 8, pp. 2126–2137, Aug. 2014.

[30] A. Lewis et al., "Rapid, high-resolution detection of environmental change over continental scales from satellite data–the Earth observation data cube," *Int. J. Digit. Earth*, vol. 9, no. 1, pp. 106–111, 2016.

[31] S. Kopp, P. Becker, A. Doshi, D. J. Wright, K. Zhang, and H. Xu, "Achieving the full vision of Earth observation data cubes," *Data*, vol. 4, no. 3, 2019, Art. no. 94.

[32] G. Giuliani, B. Chatenoux, T. Piller, F. Moser, and P. Lacroix, "Data cube on demand (DCOD): Generating an Earth observation data cube anywhere in the world," *Int. J. Appl. Earth Observ. Geoinf.*, vol. 87, 2020, Art. no. 102035.

[33] P. Baumann, "The datacube manifesto," 2018. Accessed: Jan. 1, 2023. [Online]. Available: http://www.earthserver.eu/tech/datacube-manifesto

[34] P. Baumann, D. Misev, V. Merticariu, and B. P. Huu, "Datacubes: Towards space/time analysis-ready data," in *Service-Oriented Mapping*. Berlin, Germany: Springer, 2019, pp. 269–299.

[35] G. Giuliani et al., "Building an Earth observations data cube: Lessons learned from the Swiss data cube (SDC) on generating analysis ready data (ARD)," *Big Earth Data*, vol. 1, no. 1/2, pp. 100–117, 2017.

[36] A. Lewis et al., "The Australian geoscience data cube–foundations and lessons learned," *Remote Sens. Environ.*, vol. 202, pp. 276–292, 2017.

[37] P. Merodio Gómez, A. Ramírez Santiago, O. J. Juárez Carrillo, and F. J. Jiménez Nava, "The potential contribution of Earth observation data cubes for the production of information for sustainable development in emerging countries," *Geomatics Environ. Eng.*, vol. 16, no. 3, pp. 131–155, 2022.

[38] C. Ariza-Porras et al., "CDCOL: A geoscience data cube that meets colombian needs," in *Proc. Colombian Conf. Comput.*, 2017, pp. 87–99.

[39] P. Baumann et al., "Big data analytics for earth sciences: The Earthserver approach," *Int. J. Digit. Earth*, vol. 9, no. 1, pp. 3–29, 2016.

[40] G. Camara et al., "The e-sensing architecture for big Earth observation data analysis," in *Proc. Conf. Big Data From Space*, 2017, pp. 28–30.

[41] M. Stonebraker, P. Brown, D. Zhang, and J. Becla, "SCIDB: A database management system for applications with complex analytics," *Comput. Sci. Eng.*, vol. 15, no. 3, pp. 54–62, 2013.

[42] N. Gorelick, M. Hancher, M. Dixon, S. Ilyushchenko, D. Thau, and R. Moore, "Google Earth engine: Planetary-scale geospatial analysis for everyone," *Remote Sens. Environ.*, vol. 202, pp. 18–27, 2017.

[43] Amazon, "EAWS-Earth on Amazon web services," 2023. Accessed: Jan. 10, 2023. [Online]. Available: https://aws.amazon.com/cn/earth

[44] P. Baumann et al., "Fostering cross-disciplinary Earth science through datacube analytics," in *Earth Observation Open Science and Innovation*. Cham, Switzerland: Springer, 2018, pp. 91–119.

[45] G. Giuliani, J. Masó, P. Mazzetti, S. Nativi, and A. Zabala, "Paving the way to increased interoperability of Earth observations data cubes," *Data*, vol. 4, no. 3, 2019, Art. no. 113.

[46] L. Wang, Y. Ma, A. Y. Zomaya, R. Ranjan, and D. Chen, "A parallel file system with application-aware data layout policies for massive remote sensing image processing in digital Earth," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 6, pp. 1497–1508, Jun. 2015.

[47] L. Wang, Y. Ma, J. Yan, V. Chang, and A. Y. Zomaya, "pipsCloud: High performance cloud computing for remote sensing Big Data management and processing," *Future Gener. Comput. Syst.*, vol. 78, pp. 353–368, 2018.

[48] Z. Sun, F. Chen, M. Chi, and Y. Zhu, "A spark-based Big Data platform for massive remote sensing data processing," in *Proc. Int. Conf. Data Sci.*, 2015, pp. 120–126.

[49] L. Li, W. Jing, and N. Wang, "An improved distributed storage model of remote sensing images based on the HDFs and pyramid structure," *Int. J. Comput. Appl. Technol.*, vol. 59, no. 2, pp. 142–151, 2019.

[50] R. D. Price, M. D. King, J. T. Dalton, K. S. Pedelty, P. E. Ardanuy, and M. K. Hobish, "Earth science data for all: EoS and the EoS data and information system," *Photogrammetric Eng. Remote Sens.*, vol. 60, no. 3, pp. 277–285, 1994.

[51] ESA, "Copernicus open access hub," 2023. Accessed: Jan. 16, 2023. [Online]. Available: https://scihub.copernicus.eu/

[52] FENGYUN Satellite Data Center, "FY satellite remote sensing data service web portal," 2023. Accessed: Jan. 16, 2023. [Online]. Available: http://satellite.nsmc.org.cn/portalsite

[53] Y. Shao, L. Di, Y. Bai, H. Wang, and C. Yang, "Federated catalogue for discovering Earth observation data," *Photogrammetrie-Fernerkundung-Geoinf.*, vol. 2013, no. 1, pp. 43–52, 2013.

[54] M. Appel and E. Pebesma, "On-demand processing of data cubes from satellite image collections with the gdalcubes library," *Data*, vol. 4, no. 3, 2019, Art. no. 92.

[55] M. Lu, E. Pebesma, A. Sanchez, and J. Verbesselt, "Spatio-temporal change detection from multidimensional arrays: Detecting deforestation from modis time series," *ISPRS J. Photogrammetry Remote Sens.*, vol. 117, pp. 227–236, 2016.

[56] P. Baumann, A. Dehmel, P. Furtado, R. Ritsch, and N. Widmann, "The multidimensional database system Rasdaman," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1998, pp. 575–577.

[57] G. Planthaber, M. Stonebraker, and J. Frew, "EarthDB: Scalable analysis of MODIS data using SciDB," in *Proc. 1st ACM SIGSPATIAL Int. Workshop Analytics Big Geospatial Data*, 2012, pp. 11–19.

[58] M. C. A. Picoli et al., "Big Earth observation time series analysis for monitoring Brazilian agriculture," *ISPRS J. Photogrammetry Remote Sens.*, vol. 145, pp. 328–339, 2018.

[59] M. Appel, F. Lahn, W. Buytaert, and E. Pebesma, "Open and scalable analytics of large Earth observation datasets: From scenes to multidimensional arrays using SciDB and GDAL," *ISPRS J. Photogrammetry Remote Sens.*, vol. 138, pp. 47–56, 2018.

[60] S. Papadopoulos, K. Datta, S. Madden, and T. Mattson, "The TileDB array data storage manager," *Proc. VLDB Endowment*, vol. 10, no. 4, pp. 349–360, 2016.

[61] K. Doan et al., "Evaluating the impact of data placement to spark and SciDB with an earth science use case," in *Proc. IEEE Int. Conf. Big Data*, 2016, pp. 341–346.

[62] M. D. Mahecha et al., "Earth system data cubes unravel global multivariate dynamics," *Earth Syst. Dyn.*, vol. 11, no. 1, pp. 201–234, 2020.

[63] B. Killough, "Overview of the open data cube initiative," in *Proc. IEEE Int. Geosc. Remote Sens. Symp.*, 2018, pp. 8629–8632.

[64] M. Villamizar et al., "Scaling the colombian data cube using a distributed architecture," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, 2018, pp. 441–444.

[65] K. Mubea, B. Killough, O. Seidu, J. Kimani, B. Mugambi, and S. Kamara, "Africa regional data cube (ARDC) is helping countries in africa report on the sustainable development goals (SDGs)," in *Proc. IEEE Int. Geosc. Remote Sens. Symp.*, 2020, pp. 3379–3382.

[66] P. Strobl et al., "The six faces of the data cube," in *Proc. Conf. Big Data Space*, 2017, pp. 28–30.

[67] H. Augustin, M. Sudmanns, D. Tiede, S. Lang, and A. Baraldi, "Semantic Earth observation data cubes," *Data*, vol. 4, no. 3, 2019, Art. no. 102.

[68] C. Xu et al., "Analyzing large-scale data cubes with user-defined algorithms: A cloud-native approach," *Int. J. Appl. Earth Observ. Geoinf.*, vol. 109, 2022, Art. no. 102784.

[69] F. Gao et al., "A multi-source spatio-temporal data cube for large-scale geospatial analysis," *Int. J. Geograph. Inf. Sci.*, vol. 36, no. 9, pp. 1853–1884, 2022.

[70] M. Sudmanns et al., "Think global, cube local: An Earth observation data cube's contribution to the digital Earth vision," *Big Earth Data*, vol. 16, pp. 1022–1072, 2022.

[71] R. Rew et al., "NetCDF-4: Software implementing an enhanced data model for the geosciences," in *Proc. 22nd Int. Conf. Interact. Inf. Process. Syst. Meteorol., Oceanogr., Hydrol.*, 2006.

[72] W. Huang, L. Meng, D. Zhang, and W. Zhang, "In-memory parallel processing of massive remotely sensed data using an apache spark on hadoop yarn model," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 10, no. 1, pp. 3–19, Jan. 2017.

[73] M. Dugré, V. Hayot-Sasson, and T. Glatard, "A performance comparison of dask and apache spark for data-intensive neuroimaging pipelines," in *Proc. IEEE/ACM Workflows Support Large-Scale Sci.*, 2019, pp. 40–49.

[74] M. Amani et al., "Google Earth Engine cloud computing platform for remote sensing Big Data applications: A comprehensive review," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 13, pp. 5326–5350, Sep. 2020.

[75] "Open data cube," 2023. Accessed: Jan. 18, 2023. [Online]. Available: https://www.opendatacube.org/

[76] C. Jia and H. Li, "Virtual distributed file system: Alluxio," Ph.D. dissertation, Univ. California Berkeley, Berkeley, CA, USA, 2019.

[77] C. Castillo, M. Marin, A. Rodriguez, and R. Baeza-Yates, "Scheduling algorithms for web crawling," in *Proc. WebMedia and LA-Web*, 2004, pp. 10–17.

[78] R. Baeza-Yates, C. Castillo, M. Marin, and A. Rodriguez, "Crawling a country: Better strategies than breadth-first for web page ordering," in *Proc. Special Int. Tracks Posters 14th Int. Conf. World Wide Web*, 2005, pp. 864–872.

[79] L. Yu, Y. Li, Q. Zeng, Y. Sun, Y. Bian, and W. He, "Summary of web crawler technology research," in *Proc. J. Phys.: Conf. Ser.*, 2020, Art. no. 012036.

[80] M. Mahecha et al., "The emerging Earth system data cube: Idea, implementation, and first scientific case studies," in *Proc. EGU Gen. Assem. Conf. Abstr.*, 2017, Art. no. 11813.
[81] B. Killough, "The impact of analysis ready data in the Africa regional data cube," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, 2019, pp. 5646–5649.
[82] M. Flach et al., "Multivariate anomaly detection for Earth observations: A comparison of algorithms and feature extraction techniques," *Earth Syst. Dyn.*, vol. 8, no. 3, pp. 677–696, 2017.
[83] M. Sudmanns, H. Augustin, L. Van Der Meer, C. Werner, A. Baraldi, and D. Tiede, "One GUI to rule them all: Accessing multiple semantic EO data cubes in one graphical user interface," *GI_Forum*, vol. 1, pp. 53–59, 2021.
[84] K. R. Ferreira et al., "Earth observation data cubes for Brazil: Requirements, methodology and products," *Remote Sens.*, vol. 12, no. 24, 2020, Art. no. 4033.
[85] K. Ferreira et al., "Using remote sensing images and cloud services on AWS to improve land use and cover monitoring," in *Proc. IEEE Latin Amer. GRSS ISPRS Remote Sens. Conf.*, 2020, pp. 558–562.
[86] J. Wagemann, S. Siemen, B. Seeger, and J. Bendix, "Users of open Big Earth Data–An analysis of the current state," *Comput. Geosci.*, vol. 157, 2021, Art. no. 104916.
[87] "Crawlab," 2023. Accessed: Jan. 20, 2023. [Online]. Available: https://www.crawlab.cn/
[88] J. Carlson, *Redis in Action*. New York, NY, USA: Simon and Schuster, 2013.
[89] "Spatiotemporal asset catalog," 2023. Accessed: Jan. 16, 2023. [Online]. Available: https://stacspec.org/
[90] A. Vogt, A. Wytzisk-Arens, S. Drost, and S. Jirka, "Cloud based discovery and processing of geospatial data," in *Proc. Accepted Short Papers Posters 22nd AGILE Conf. Geo- Inf. Sci.*, 2019, pp. 17–20.

Jie Song received the bachelor's degree of management in information management and information system from Beijing Forestry University, Beijing, China, in 2020. She is currently working toward the master's degree in signal and information processing with the University of Chinese Academy of Sciences, Beijing.

Yan Ma (Member, IEEE) received the M.S. and Ph.D. degrees in signal and information processing from the University of Chinese Academy of Sciences, Beijing, China, in 2007 and 2013, respectively. She is currently an Associate Professor at the Aerospace Information Research Institute, Chinese Academy of Sciences.

Zhixin Zhang received the B.E. degree in network engineering from the China University of Geosciences, Wuhan, China, in 2021. She is currently working toward the master's degree in signal and information processing with the University of Chinese Academy of Sciences, Beijing, China.

Peng Liu (Member, IEEE) received the M.S. and Ph.D. degrees in signal processing from the Chinese Academic of Science, Beijing, China, in 2004 and 2009, respectively. He is currently an Associate Professor at the Aerospace Information Research Institute, Chinese Academy of Sciences. From May 2012 to May 2013, he was with the Department of Electrical and Computer Engineering, George Washington University, Washington, DC, USA, as a Visiting Scholar. His research is focused on big data, sparse representation, compressive sensing, deep learning and their applications to remote sensing data processing.