An I/O-Efficient Approach for Concurrent Spatio-Temporal Range Retrievals over Large-Scale Remote Sensing Image Data

Ao Long, Wei Lin, and Ze Deng†

***Abstract*—High-performance remote sensing analytics workflows require ingesting and retrieving massive image archives to support real-time spatio-temporal applications. While modern systems utilize window-based I/O reading to reduce data transfer, they face a dual bottleneck: (1) the prohibitive overhead of runtime geospatial computations caused by the decoupling of logical indexing from physical storage, and (2) severe storage-level I/O contention triggered by uncoordinated concurrent reads. To address these limitations, we present a comprehensive I/O-aware retrieval approach based on a novel "Index-as-an-Execution-Plan" paradigm. We introduce a dual-layer inverted index that serves as an I/O planner, pre-materializing grid-to-pixel mappings to completely eliminate runtime geometric calculations. Furthermore, we design a hybrid concurrency-aware I/O coordination protocol that adaptively integrates Calvin-style deterministic ordering with optimistic execution, effectively converting I/O contention into request merging opportunities. To handle fluctuating workloads, we incorporate a Surrogate-Assisted Genetic Multi-Armed Bandit (SA-GMAB) for automatic parameter tuning. Evaluated on a distributed cluster with Martian datasets, the experimental results indicate that: (1) I/O-aware indexing reduces retrieval latency by an order of magnitude; (2) hybrid concurrency-aware I/O coordination achieves a 54x speedup under high contention through request merging and automates optimal mode switching; and (3) SA-GMAB has the fastest convergence speed and recovers from workload shifts $2\times$ faster than TunIO.**

***Index Terms*—Remote sensing data management, Spatio-temporal range retrievals, I/O-aware indexing, Concurrency control, I/O tuning.**

I. Introduction

A Massive amount of remote sensing (RS) data, characterized by high spatial, temporal, and spectral resolutions, is being generated at an unprecedented speed due to the rapid advancement of Earth observation missions [1]. For instance, NASA's AVIRIS-NG acquires nearly 9 GB of data per hour, while the EO-1 Hyperion sensor generates over 1.6 TB daily [2]. Beyond the sheer volume of data, these datasets are increasingly subjected to intensive concurrent access from global research communities and real-time emergency response systems (e.g., multi-departmental coordination during natural disasters). Consequently, modern RS platforms are required to provide not only massive storage capacity but also high-throughput retrieval capabilities to satisfy the simultaneous demands of numerous spatio-temporal analysis tasks.

Existing RS data management systems [3]–[5] typically decompose a spatio-temporal range retrieval into a decoupled two-phase execution model. The first phase is the metadata filtering phase, which utilizes spatio-temporal metadata (e.g., footprints, timestamps) to identify candidate image files that intersect the retrieval predicate. Recent advancements have transitioned from traditional tree-based indexes [6], [7] to scalable distributed schemes based on grid encodings and space-filling curves, such as GeoHash [8], GeoSOT [4], and GeoMesa [9], [10]. By leveraging these high-dimensional indexing structures, the search complexity of the first phase has been effectively reduced to $O(\log N)$ or even $O(1)$, making metadata discovery extremely efficient even for billion-scale datasets.

The second phase is the data extraction phase, where the system reads the actual pixel data from the identified raw image files stored in distributed file systems or object stores. A critical observation in modern high-performance RS analytics is that the primary system bottleneck has fundamentally shifted from the first phase to the second. While the metadata search completes in milliseconds, the end-to-end retrieval latency is now dominated by the massive I/O overhead required to fetch, decompress, and process large-scale raw images. Traditional systems attempted to reduce I/O overhead by pre-slicing tiles and building pyramids (e.g., approaches used in Google Earth Engine [11] that store metadata in HBase and serve pre-tiled image pyramids), but aggressive tiling increases management complexity and produces many small files. More recent Cloud-Optimized GeoTIFF (COG) formats and COG-aware frameworks [3], [12] exploit internal overviews and window-based I/O to read only the portions of files that spatially intersect a retrieval.

While window-based I/O effectively reduces raw data transfer, it introduces a new computational burden due to the decoupling of logical indexing from physical storage. Current systems operate on a "Search-then-Compute-then-Read" model: after identifying candidate files, they must perform fine-grained, per-image geospatial computations at runtime to map retrieval coordinates to precise file offsets and clip boundaries. This runtime geometric resolution becomes computationally prohibitive when processing a large volume of candidate images, often negating the benefits of I/O reduction. Moreover,

A. Long, W. Lin and Z. Deng, (Corresponding author, dengze@cug.edu.cn) are with the School of Computer Science, China University of Geosciences, Wuhan, 430078, P. R. China.

Z. Deng is also with Hubei Key Laboratory of Intelligent Geo-Information Processing, China University of Geosciences, Wuhan 430074, China.

under concurrent workloads, the lack of coordination among these independent read requests leads to severe I/O contention and storage thrashing, rendering traditional indexing-centric optimizations insufficient for real-time applications.

To address the aforementioned problems, we propose a novel "Index-as-an-Execution-Plan" paradigm to bound the retrieval latency. Unlike conventional approaches that treat indexing and I/O execution as separate stages, our approach integrates fine-grained partial retrieval directly into the indexing structure. By pre-materializing the mapping between logical spatial grids and physical pixel windows, our system enables deterministic I/O planning without runtime geometric computation. To further ensure scalability, we introduce a concurrency control protocol tailored for spatio-temporal range retrievals and an automatic I/O tuning mechanism. The principal contributions of this paper are summarized as follows:

1) We propose an I/O-aware index schema. Instead of merely returning candidate image identifiers, our index directly translates high-level spatio-temporal predicates into concrete, byte-level windowed read plans. This design bridges the semantic gap between logical retrievals and physical storage, eliminating expensive runtime geospatial computations and ensuring that I/O cost is strictly proportional to the retrieval footprint.
2) We propose a hybrid concurrency-aware I/O coordination protocol. This protocol adapts transaction processing principles by integrating Calvin-style deterministic ordering [13] with optimistic execution [14]. It shifts the focus from protecting database rows to coordinating shared I/O flows. This protocol dynamically switches strategies based on spatial contention, effectively converting I/O contention into request merging opportunities.
3) We propose an automatic I/O tuning method to improve the I/O performance of spatio-temporal range retrievals over RS data. The method extends an existing AI-powered I/O tuning framework [15] based on a surrogate-assisted genetic multi-armed bandit algorithm [16].

The remainder of this paper is organized as follows: Section II presents the related work. Section III formulates the spatio-temporal range retrieval problem and establishes the cost models. Section IV provides an overview of the proposed framework and describes how the three modules are integrated. Section V presents the I/O-aware indexing structure. Section VI proposes the hybrid concurrency-aware I/O coordination protocol. Section VII presents the GMAB-based online I/O stack tuning method. Section VIII presents the experiments and results. Section IX concludes this paper with a summary.

II. Related Work

This section describes the most salient studies of I/O-efficient spatio-temporal retrieval processing, concurrency control, and I/O performance tuning.

A. I/O-Efficient Spatio-Temporal Retrieval Processing

Efficient spatio-temporal retrieval for RS data has been extensively studied, with early efforts primarily focusing on metadata organization and index-level pruning in relational database systems. Traditional approaches typically extend tree-based spatial indexes, such as R-tree [6], quadtree [17], and their spatio-temporal variants [7], to organize image footprints together with temporal attributes, and are commonly implemented on relational backends (e.g., MySQL and PostgreSQL). These methods provide efficient range filtering for moderate-scale datasets, but their reliance on balanced tree structures often leads to high maintenance overhead and limited scalability as the volume of remote sensing metadata grows rapidly. With the continuous increase in data volume and ingestion rate, recent systems have gradually shifted toward grid-based spatio-temporal indexing schemes deployed on distributed NoSQL stores. By encoding spatial footprints into uniform spatial grids [4], [8] or space-filling curves [5], [18] and combining them with temporal identifiers, these approaches enable lightweight index construction and better horizontal scalability on backends such as HBase and Elasticsearch. Such grid-based indexes can effectively reduce the candidate search space through coarse-grained pruning and are more suitable for large-scale, continuously growing remote sensing archives.

However, index pruning alone is insufficient to guarantee end-to-end retrieval efficiency for remote sensing workloads, where individual images are usually large and retrieval results require further pixel-level processing. To reduce the amount of raw I/O, Google Earth Engine [11] relies on tiling and multi-resolution pyramids that physically split images into small blocks. While more recent solutions leverage COG and window-based I/O to enable partial reads from monolithic image files, frameworks such as OpenDataCube [3] exploit these features to read only the image regions intersecting a retrieval window, thereby reducing unnecessary data transfer. Nevertheless, after candidate images are identified, most systems still perform fine-grained geospatial computations for each image, including coordinate transformations and precise pixel-window derivation, which may incur substantial overhead when many images are involved. In this paper, we propose an I/O-aware index that pre-materializes grid-to-pixel mappings to eliminate runtime geometric calculations, enabling direct translation of spatio-temporal predicates into byte-level windowed read plans.

B. Concurrency Control

Concurrency control has long been studied to provide correctness and high throughput in multi-user database and storage systems, with two broad paradigms dominating the literature: deterministic scheduling [13], [19] and non-deterministic schemes [20], [21]. Hybrid approaches [22], [23] that adaptively combine these paradigms seek to exploit the low-conflict efficiency of deterministic execution while retaining the flexibility of optimistic techniques. More recent proposals, such as OOCC [24], target read-heavy, disaggregated settings

by reducing validation and round-trips for read-only transactions, achieving low latency under OLTP-like workloads. These methods are primarily optimized for record- or key-level access patterns: their metrics and designs emphasize transaction latency, abort rates, and throughput under workloads with small, well-defined read/write sets.

Overall, existing concurrency control mechanisms are largely designed around transaction-level correctness and throughput, assuming record- or key-based access patterns and treating storage I/O as a black box. Their optimization objectives rarely account for I/O amplification or fine-grained storage contention induced by concurrent range retrievals. Consequently, these approaches are ill-suited for data-intensive spatio-temporal workloads, where coordinating overlapping window reads and mitigating storage-level interference are critical to achieving scalable performance under multi-user access. Unlike prior transaction-level concurrency control mechanisms, we adapt the hybrid deterministic-optimistic concept from HDCC [23] to the thread level, targeting I/O contention resolution for concurrent spatio-temporal range retrievals.

C. I/O Performance Tuning in Storage Systems

I/O performance tuning has been extensively studied in the context of HPC and data-intensive storage systems, where complex multi-layer I/O stacks expose a large number of tunable parameters. These parameters span different layers, including application-level I/O libraries, middleware, and underlying storage systems, and their interactions often lead to highly non-linear performance behaviors. As a result, manual tuning is time-consuming and error-prone, motivating a wide range of auto-tuning approaches [25].

Several studies focus on improving the efficiency of the tuning pipeline itself by reformulating the search space or optimization objectives. Chen et al. [26] proposed a meta multi-objectivization model that introduces auxiliary performance objectives to mitigate premature convergence to local optima. While such techniques can improve optimization robustness, they are largely domain-agnostic and do not explicitly account for the characteristics of I/O-intensive workloads. Other works, such as the contextual bandit-based approach by Bez et al. [27], optimize specific layers of the I/O stack (e.g., I/O forwarding) by exploiting observed access patterns. However, these methods are primarily designed for administrator-level tuning and target isolated components rather than end-to-end application I/O behavior [28].

User-level I/O tuning has also been explored, most notably by H5Tuner [29], which employs genetic algorithms to optimize the configuration of the HDF5 I/O library. Although effective for single-layer tuning, H5Tuner does not consider cross-layer interactions and lacks mechanisms for reducing tuning cost, such as configuration prioritization or early stopping.

More recently, TunIO [15] proposed an AI-powered I/O tuning framework that explicitly targets the growing configuration spaces of modern I/O stacks. TunIO integrates several advanced techniques, including I/O kernel extraction, smart selection of high-impact parameters, and reinforcement learning–driven early stopping, to balance tuning cost and performance gain across multiple layers. Despite its effectiveness, TunIO and related frameworks primarily focus on single-application or isolated workloads, assuming stable access patterns during tuning. Retrieval-level I/O behaviors, such as fine-grained window access induced by spatio-temporal range retrievals, as well as interference among concurrent users, are generally outside the scope of existing I/O tuning approaches [30]. In contrast, we employ the GMAB algorithm [16], which introduces a memory mechanism into evolutionary algorithms to permanently preserve historical observations, thereby achieving fast convergence and rapid adaptation to dynamic workload shifts.

III. Problem Formulation

This section formalizes the spatio-temporal range retrieval problem and establishes the cost models for retrieval execution. We assume a distributed storage environment where large-scale remote sensing images are stored as objects or files.

Definition 1 (Spatio-temporal Remote Sensing Image). A remote sensing image R is defined as a tuple:

$$R = \langle id, \Omega, \mathcal{D}, t \rangle, \tag{1}$$

where id is the unique identifier; $\Omega = [0, W] \times [0, H]$ denotes the pixel coordinate space; $\mathcal{D}$ represents the raw pixel data; and t is the temporal validity interval. The image is associated with a spatial footprint $MBR(R)$ in the global coordinate reference system.

Definition 2 (Spatio-temporal Range Retrieval). Given a dataset $\mathbb{R}$, a retrieval query Q is defined by a spatio-temporal predicate $Q = \langle S, T \rangle$, where S is the spatial bounding box and T is the time interval. The retrieval result set $\mathcal{R}_Q$ is defined as:

$$\mathcal{R}_Q = R \in \mathbb{R} \mid MBR\,(R) \cap S \neq \emptyset \wedge R.t \cap T \neq \emptyset. \tag{2}$$

For each $R \in \mathcal{R}_Q$, the system must return the pixel matrix corresponding to the intersection region $MBR(R) \cap S$.

Definition 3 (Retrieval Execution Cost Model). The execution latency of a retrieval Q, denoted as $Cost(Q)$, is composed of two phases: metadata filtering and data extraction.

$$Cost\,(Q) = C_{meta}\,(Q) + \sum_{R \in \mathcal{R}_Q} \left(C_{geo}\,(R, Q) + C_{io}\,(R, Q)\right). \tag{3}$$

Here, $C_{meta}(Q)$ is the cost of identifying candidate images $\mathcal{R}_Q$ using indices. The data extraction cost for each image consists of two components: geospatial computation cost (C_{geo}) and I/O access cost (C_{io}). C_{geo} is the CPU time required to calculate the pixel-to-geographic mapping, determine the exact read windows (offsets and lengths), and handle boundary clipping. In window-based partial reading schemes, this cost is non-negligible due to the complexity of coordinate transformations. C_{io} is the latency to fetch the actual binary data from storage.

Definition 4 (Concurrent Spatio-temporal Retrievals). Let $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_N\}$ denote a set of spatio-temporal range retrievals issued concurrently by multiple users. Each retrieval Q_i independently specifies a spatio-temporal window $\langle S_i, T_i \rangle$

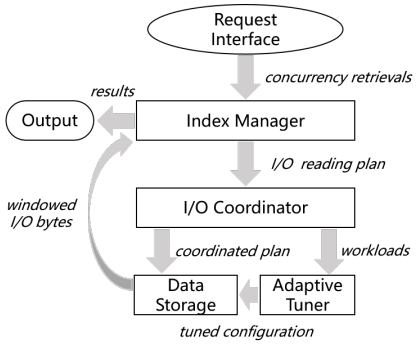


Fig. 1. The workflow for processing concurrent spatio-temporal range retrievals in the system

and may overlap with others in both spatial and temporal dimensions. Concurrent execution of $\mathcal{Q}$ may induce overlapping partial reads over the same images or image regions, leading to redundant I/O and storage-level contention if retrievals are processed independently.

Problem Statement (Latency-Optimized Concurrent Retrieval Processing). Given a dataset $\mathbb{R}$ and a concurrent workload $\mathcal{Q}$, the objective is to minimize the total execution latency:

$$\min \sum_{Q_i \in \mathcal{Q}} \big(C_{\text{meta}}(Q_i) + \sum_{R \in \mathcal{R}_{Q_i}} \big(C_{\text{geo}}(R, Q_i) + C_{\text{io}}(R, Q_i)\big)\big), \tag{4}$$

subject to:

1) *Correctness:* The returned data must strictly match the spatio-temporal predicate defined in Eq. (2).
2) *Isolation:* Concurrent reads must effectively share I/O bandwidth without causing starvation or excessive thrashing.

IV. System Overview

To address the challenges of storage-level I/O contention and expensive runtime computations, we propose a layered distributed retrieval framework. As illustrated in Fig. 1, the system architecture is composed of four primary processing components: (1) *requst interface*, (2) *index manager*, (3) *I/O coordinator*, (4) *parallel executors*, and (5) *adaptive tuner*.

The *requst interface* serves as the system entry point. It is responsible for accepting concurrent spatio-temporal retrievals. The *index manager* acts as the planner of the system, interacting with the metadata storage. It translates logical spatio-temporal predicates into physical storage locations using a dual-layer inverted index. The *I/O coordinator* serves as the traffic control layer. It detects spatial overlaps among concurrent reading plans to identify potential I/O conflicts and applies the hybrid concurrency-aware protocol to reorder or merge conflicting requests. Finally, the *parallel executors* interface with the distributed file system or object store to read the pixel data. What's more, *adaptive tuner* optimizes the execution parameters in the background.

V. I/O-aware Indexing Structure

This section introduces the details of the indexing structure for spatio-temporal range retrieval over RS data.

A. Index schema design

To enable I/O-efficient spatio-temporal query processing, we first decompose the global spatial domain into a uniform grid that serves as the basic unit for query pruning and data access coordination. Specifically, we adopt a fixed-resolution global tiling scheme based on the Web Mercator (or EPSG:4326) coordinate system, using zoom level 14 to partition the Earth's surface into fine-grained grid cells (experiments show that the 14-level grid has the highest indexing efficiency, as discussed in Section VIII-B3). This resolution strikes a practical balance between spatial selectivity and index size: finer levels would significantly increase metadata volume and maintenance cost, while coarser levels would reduce pruning effectiveness and lead to unnecessary image I/O. At this scale, each grid cell typically corresponds to a spatial extent comparable to common query footprints and to the internal tiling granularity used by modern raster formats, making it well suited for partial data access.

Grid-to-Image Mapping (G2I). Based on the grid decomposition, we construct a grid-centric inverted index to associate spatial units with covering images. In our system, each grid cell is assigned a unique *GridKey*, encoded as a 64-bit Z-order value to preserve spatial locality and enable efficient range scans in key-value stores such as HBase. The G2I table stores one row per grid cell, where the row key is the GridKey and the value contains the list of image identifiers (ImageKeys) whose spatial footprints intersect the corresponding cell, as illustrated in Fig. 2(a).

This grid-to-image mapping allows retrieval processing to begin with a lightweight enumeration of grid cells covered by a retrieval region, followed by direct lookups of candidate images via exact GridKey matches. By treating each grid cell as an independent spatial bucket, the G2I table provides efficient metadata-level pruning and avoids costly geometric intersection tests over large image footprints.

However, the G2I table alone is insufficient for I/O-efficient retrieval execution. While it identifies which images are relevant to a given grid cell, it does not capture how the grid cell maps to pixel regions within each image. As a result, a grid-only representation cannot directly guide partial reads and would still require per-image geospatial computations at retrieval time. Therefore, the G2I table functions as a coarse spatial filter and must be complemented by an image-centric structure that materializes the correspondence between grid cells and pixel windows, enabling fine-grained, window-based I/O.

Image-to-Grid Mapping (I2G). To complement the grid-centric G2I table and enable fine-grained, I/O-efficient data access, we introduce an image-centric inverted structure, referred to as the Image-to-Grid mapping (I2G). In contrast to G2I, which organizes metadata by spatial grids, the I2G table stores all grid-level access information of a remote sensing image in a single row. Each image therefore occupies exactly one row in the table, significantly improving locality during retrieval execution.

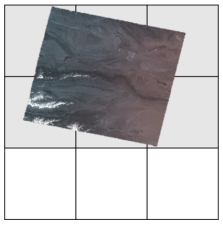
As illustrated in Fig. 2(b), the row key of the I2G table is the *ImageKey*, i.e., the unique identifier of a remote sensing

Key	Value
Grid Key	Remote Sening Image List
$grid_1$	img_2, img_{30}
$grid_2$	img_1, img_2, img_5
...	...
$grid_n$	img_9, img_{60}, img_7

(a) Grid to Image Mapping

Key	Value		
Image Key	Grid–Window Mapping	Temporal Properties	Storage Path
img_1	$< grid_2, W_{1\_2} >, < grid_5, W_{1\_5} >$	t_1	p_1
img_2	$< grid_2, W_{2\_2} >$	t_2	p_2
...	...	...	...
img_n	$< grid_{30}, W_{n\_30} >, < grid_8, W_{n\_8} >$	t_n	p_n

(b) Image to Grid Mapping



(c) Image Signature

Fig. 2. Index schema design

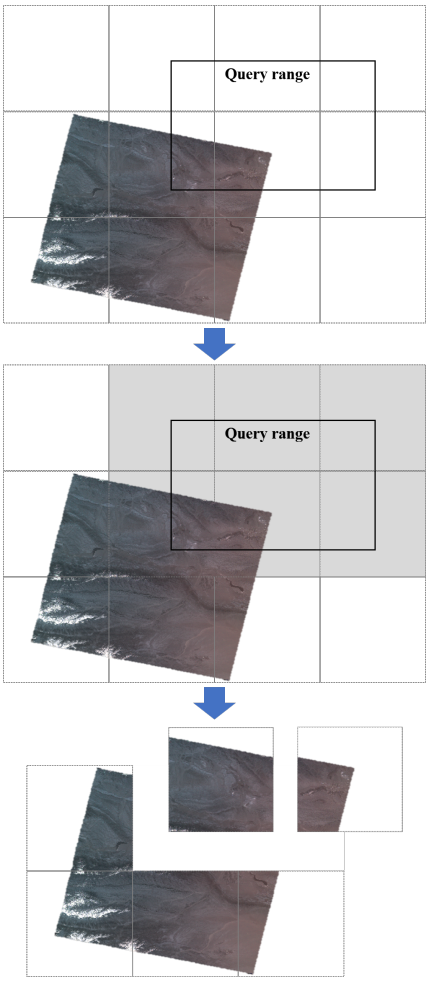


Fig. 3. Retrieval-time Execution

image. The row value is organized into three column families, each serving a distinct role in retrieval-time pruning and I/O coordination:

Grid–Window Mapping. This column family records the list of grid cells intersected by the image together with their corresponding pixel windows in the image coordinate space. Each entry has the form:

$$\langle GridKey, W_{ImageKey\_GridKey} \rangle, \tag{5}$$

where *GridKey* identifies a grid cell at the chosen global resolution, and $W_{ImageKey\_GridKey}$ denotes the minimal pixel bounding rectangle within the image that exactly covers that grid cell.

These precomputed window offsets allow the retrieval executor to directly issue windowed reads on large raster files without loading entire images into memory or recomputing geographic-to-pixel transformations at retrieval time. As a result, grid cells become the smallest unit of coordinated I/O, enabling precise partial reads and effective elimination of redundant disk accesses.

Temporal Metadata. To support spatio-temporal range retrievals, each image row includes a lightweight temporal column family that stores its acquisition time information, such as the sensing timestamp or time interval. This metadata enables efficient temporal filtering to be performed jointly with spatial grid matching, without consulting external catalogs or secondary indexes.

Storage Pointer. This column family contains the information required to retrieve image data from the underlying storage system. It stores a stable file identifier, such as an object key in an object store (e.g., MinIO/S3) or an absolute path in a POSIX-compatible file system. By decoupling logical image identifiers from physical storage locations, this design supports flexible deployment across heterogeneous storage backends while allowing the retrieval engine to directly access image files once relevant pixel windows have been identified.

The I2G table offers several advantages. First, all grid-level access information for the same image is co-located in a single row, avoiding repeated random lookups and improving cache locality during retrieval execution. Second, by materializing grid-to-window correspondences at ingestion time, the system completely avoids expensive per-retrieval geometric computations and directly translates spatial overlap into byte-range I/O requests. Third, the number of rows in the I2G table scales with the number of images rather than the number of grid cells, substantially reducing metadata volume and maintenance overhead.

During data ingestion, the grid–window mappings are generated by projecting grid boundaries into the image coordinate system using the image's georeferencing parameters. This process requires only lightweight affine or RPC transformations and does not involve storing explicit geometries or performing polygon clipping. As a result, the I2G structure enables efficient partial reads while keeping metadata compact and ingestion costs manageable.

B. Retrieval-time Execution

The I/O-aware index enables efficient spatio-temporal range retrievals by directly translating retrieval predicates into windowed read plans, while avoiding both full-image loading and expensive geometric computations. Given a user-specified spatio-temporal retrieval $q = \langle [x_{\min}, y_{\min}, x_{\max}, y_{\max}], [t_s, t_e] \rangle$, the system resolves the retrieval through three consecutive stages: *Grid Enumeration*, *Candidate Image Retrieval with Temporal Pruning*, and textitWindowed Read Plan Generation. As illustrated in Fig. 3, this execution pipeline bridges high-level retrieval predicates and low-level I/O operations in a fully deterministic manner.

Grid Enumeration. As shown in Step 1 and Step 2 of Fig. 3, the retrieval execution starts by rasterizing the spatial footprint of q into the fixed global grid at zoom level 14. Instead of performing recursive space decomposition as in quadtrees or hierarchical spatial indexes, our system enumerates the minimal set of grid cells $\{g_1, \ldots, g_k\}$ whose footprints intersect the retrieval bounding box.

Each grid cell corresponds to a unique 64-bit GridKey, which directly matches the primary key of the G2I table. This design has important implications: grid enumeration has constant depth and low computational cost, and the resulting GridKeys can be directly used as lookup keys without any geometric refinement. Consequently, spatial key generation is reduced to simple arithmetic operations on integer grid coordinates.

Candidate Image Retrieval with Temporal Pruning. Given the enumerated grid set $\{g_1, \ldots, g_k\}$, the retrieval processor performs a batched multi-get on the G2I table. Each G2I row corresponds to a single grid cell and stores the identifiers of all images whose spatial footprints intersect that cell. For each grid g_i, the lookup returns:

$$G2I[g_i] = \{imgKey_1, \ldots, imgKey_m\}. \tag{6}$$

All retrieved image identifiers are unioned to form the spatial candidate set $C_s = \bigcup_{i=1}^{k} G2I[g_i]$. This step eliminates the need for per-image polygon intersection tests that are commonly required in spatial databases and data cube systems.

To incorporate the temporal constraint $[t_s, t_e]$, each candidate image in C_s is further filtered using the temporal column family of the Image-to-Grid (I2G) table. Images whose acquisition time does not intersect the retrieval interval are discarded early, yielding the final candidate set C. This lightweight temporal pruning is performed without accessing any image data and introduces negligible overhead.

Windowed Read Plan Generation. As shown in Step 3 of Fig. 3, the final stage translates the candidate image set into a concrete I/O plan. For each image $I \in C$, the retrieval executor issues a selective range-get on the I2G table to retrieve only the grid–window mappings relevant to the retrieval grids:

$$I2G\left[I, \{g_1, ..., g_k\}\right] = \{W_{I\_g_i} \mid g_i \cap I \neq \emptyset\}. \tag{7}$$

Each $W_{I\_g_i}$ specifies the exact pixel window in the original raster file that corresponds to grid cell g_i. Since these window offsets are precomputed during ingestion, retrieval execution requires only key-based lookups and arithmetic filtering. No geographic coordinate transformation, polygon clipping, or raster–vector intersection is performed at retrieval time.

The resulting collection of pixel windows constitutes a windowed read plan, which can be directly translated into byte-range I/O requests against the storage backend. This approach avoids loading entire scenes and ensures that the total I/O volume is proportional to the retrieved spatial extent rather than the image size.

C. Why I/O-aware

The key reason our indexing design is I/O-aware lies in the fact that the index lookup results are not merely candidate identifiers, but constitute a concrete I/O access plan. Unlike traditional spatial indexes, where retrieval processing yields a set of objects that must still be fetched through opaque storage accesses, our Grid-to-Image and Image-to-Grid lookups deterministically produce the exact pixel windows to be read from disk. As a result, the logical retrieval plan and the physical I/O plan are tightly coupled: resolving a spatio-temporal predicate directly specifies which byte ranges should be accessed and which can be skipped.

This tight coupling fundamentally changes the optimization objective. Instead of minimizing index traversal cost or result-set size, the system explicitly minimizes data movement by ensuring that disk I/O is proportional to the retrieval's spatio-temporal footprint. Consequently, the index serves as an execution-aware abstraction that bridges retrieval semantics and storage behavior, enabling predictable, bounded I/O under both single-retrieval and concurrent workloads.

Theoretical Cost Analysis. To rigorously quantify the performance advantage, we revisit the retrieval cost model defined in Eq. (3). In traditional full-image reading systems, although the geospatial computation cost is negligible ($C_{geo} = 0$) as no clipping is performed, the I/O cost C_{io} is determined by the full file size. Consequently, the total latency is entirely dominated by massive I/O overhead, rendering C_{meta} (typically milliseconds) irrelevant.

Existing window-based I/O systems successfully reduce the I/O cost to the size of the requested window. However, this reduction comes at the expense of a significant surge in C_{geo}. For every candidate image, the system must perform on-the-fly coordinate transformations and polygon clipping to calculate read offsets. When a retrieval involves thousands of images, the accumulated CPU time ($\sum C_{geo}$) becomes a new bottleneck (e.g., hundreds of milliseconds to seconds), often negating the benefits of I/O reduction (detailed quantitative comparisons are provided in Sec. VIII-B2).

In contrast, our I/O-aware indexing approach fundamentally alters this trade-off. By materializing the grid-to-pixel mapping in the I2G table, we effectively shift the computational burden from retrieval time to ingestion time. Although the two-phase lookup (G2I and I2G) introduces a slight overhead compared to simple tree traversals, C_{meta} remains in the order of milliseconds—orders of magnitude smaller than disk I/O latency. Since the precise pixel windows are pre-calculated and stored, the runtime geospatial computation is effectively eliminated, i.e., $C_{geo} = 0$. The system retains the minimal I/O

cost characteristic of window-based approaches, fetching only relevant byte ranges. Therefore, our design achieves the theoretical minimum for both computation and I/O components within the retrieval execution critical path.

VI. Hybrid Concurrency-Aware I/O Coordination

In this section, we propose a hybrid coordination mechanism that adaptively employs either lock-free non-deterministic execution or deterministic coordinated scheduling based on the real-time contention level of spatio-temporal workloads.

A. Retrieval Admission and I/O Plan Generation

When a spatio-temporal range retrieval Q arrives, the system first performs index-driven plan generation. The retrieval footprint is rasterized into the global grid to enumerate the intersecting grid cells. The G2I table is then consulted to retrieve the set of candidate images, followed by selective lookups in the I2G table to obtain the corresponding pixel windows.

As a result, each retrieval is translated into an explicit *I/O access plan* consisting of image–window pairs:

$$Plan\left(Q\right)=\left\{\left(img_1,w_1\right),\left(img_1,w_2\right),\left(img_3,w_5\right),...\right\}, \quad (8)$$

where each window w denotes a concrete pixel range to be accessed via byte-range I/O. Upon admission, the system assigns each retrieval a unique *RetrievalID* and records its arrival timestamp.

B. Contention Estimation and Path Selection

To minimize the overhead of global ordering in low-contention scenarios, the system introduces a Contention-Aware Switch. Upon the arrival of a retrieval batch $\mathcal{Q} = \{Q_1, Q_2, ..., Q_n\}$, the system first estimates the Spatial Overlap Ratio (σ) among their generated I/O plans.

Let $A(Plan(Q_i))$ be the aggregate spatial area of all pixel windows in the I/O plan of retrieval Q_i. The overlap ratio σ for a batch is defined as:

$$\sigma = 1 - \frac{\mathrm{A}(\bigcup_{i=1}^{n} Plan(Q_i))}{\sum_{i=1}^{n} \mathrm{A}(Plan(Q_i))}, \quad (9)$$

where $\sigma \in [0, 1]$. A high σ indicates that multiple retrievals are competing for the same image regions, leading to high I/O amplification if executed independently.

The system utilizes a rule-based assignment mechanism similar to HDCC [23] to select the execution path:

1) Path A (Non-deterministic/OCC-style): If $\sigma < \tau$ (where τ is a configurable threshold), retrievals proceed directly to execution to maximize concurrency.
2) Path B (Deterministic/Calvin-style): If $\sigma \geq \tau$, retrievals are routed to the Global I/O Plan Queue for coordinated merging.

C. Deterministic Coordinated and Non-deterministic Execution

When $\sigma \geq \tau$, the system switches to a deterministic path to mitigate storage-level contention and I/O amplification, as shown in Fig. 4. To coordinate concurrent access to shared storage resources, we introduce a Global I/O Plan Queue that enforces a deterministic ordering over all admitted I/O plans. Each windowed access (img, w) derived from incoming retrievals is inserted into this queue according to a predefined policy, such as FIFO based on arrival time or lexicographic ordering by $(timestamp, RetrievalID)$.

This design is inspired by deterministic scheduling in systems such as Calvin, but differs fundamentally in its scope: the ordering is imposed on window-level I/O operations rather than on transactions. As a result, accesses to the same image region across different retrievals follow a globally consistent order, preventing uncontrolled interleaving of reads and reducing contention at the storage layer. The deterministic ordering also provides a stable foundation for subsequent I/O coordination and sharing.

The core of our approach lies in coordinating concurrent windowed reads at the image level. Windows originating from different retrievals may overlap spatially, be adjacent, or even be identical. Executing these requests independently would lead to redundant reads and excessive I/O amplification.

To address this, the system performs three coordination steps within each scheduling interval. Stage 1: Global De-duplication. The system first extracts all windowed access pairs (img, w) from the admitted retrievals and inserts them into a global window set ($\mathcal{W}_{total}$). If multiple retrievals $Q_1, Q_2, ..., Q_n$ request the same pixel window w from image img, the system retains only one unique entry in $\mathcal{W}_{total}$. This stage ensures that any specific byte range is identified as a single logical requirement, effectively preventing the redundant retrieval of overlapping spatial grids. Stage 2: Range Merging. After de-duplication, the system analyzes the physical disk offsets of all unique windows in $\mathcal{W}_{total}$. Following the principle of improving access locality, windows that are physically contiguous or separated by a gap smaller than a threshold θ are merged into a single read. Stage 3: Dispatching. This stage maintains a mapping between the physical byte-offsets in the buffer and the logical window requirements of each active retrieval. Each retrieval Q_i receives only the exact pixel windows $w \in Plan(Q_i)$ it originally requested. This is achieved via zero-copy memory mapping where possible, or by slicing the shared system buffer into local thread-wise structures. This ensures that while the physical I/O is shared to reduce amplification, the logical execution of each retrieval remains independent and free from irrelevant data interference.

For example, when Q_1 requests grids $\{1, 2\}$ and Q_2 requests grids $\{2, 3\}$, Stage 1 identifies the unique requirement set $\{1, 2, 3\}$. Stage 2 then merges these into a single contiguous I/O operation covering the entire range $[1, 3]$. In Stage 3, the dispatcher identifies memory offsets corresponding to grids 1 and 2 within the buffer and maps these slices to the private cache of Q_1. For Q_2, similarly, the dispatcher extracts and delivers slices for grids 2 and 3 to Q_2.

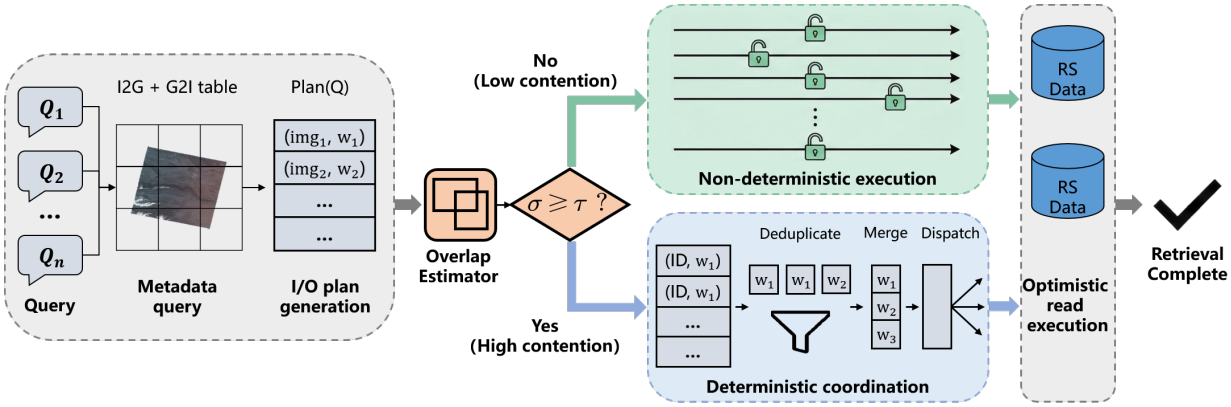


Fig. 4. Hybrid Concurrency-Aware I/O Coordination.

Through these mechanisms, concurrent retrievals collaboratively share I/O, and the execution unit becomes a coordinated window read rather than an isolated request. Importantly, this coordination operates entirely at the I/O planning level and does not require any form of locking or transaction-level synchronization.

When contention remains below the threshold ($\sigma < \tau$), the system prioritizes low latency over merging efficiency by adopting an optimistic dispatch mechanism, as shown in Fig. 4. Instead of undergoing heavy-weight sorting, I/O plans are immediately offloaded to the execution engine. By utilizing thread-local sublists, each thread independently handles its byte-range requests.

D. Optimistic Read Execution and Completion

Once a coordinated window read is scheduled, the system issues the corresponding byte-range I/O request immediately. Read execution is fully optimistic: there is no validation phase, no abort, and no rollback. This is enabled by the immutability of remote-sensing imagery and by the deterministic ordering of I/O plans, which together ensure consistent and repeatable read behavior.

A retrieval is considered complete when all windows in its I/O plan have been served and the associated local processing (e.g., reprojection or mosaicking) has finished. By eliminating validation overhead and allowing read execution to proceed independently once scheduled, the system achieves low-latency retrieval completion while maintaining predictable I/O behavior under concurrency.

Overall, this concurrency-aware I/O coordination mechanism reinterprets concurrency control as a problem of coordinating shared I/O flows. By operating at the granularity of windowed reads and leveraging deterministic ordering and optimistic execution, it effectively reduces redundant I/O and improves scalability for multi-user spatio-temporal retrieval workloads.

VII. I/O Stack Tuning

We first describe the I/O stack tuning problem and then propose the surrogate-assisted GMAB algorithm to solve the problem.

A. Formulation of Online I/O Tuning

We study a concurrent spatio-temporal retrieval engine that processes many range retrievals simultaneously. The system operates on large remote sensing images stored in shared storage. Unlike traditional HPC jobs or single-application I/O workloads, the system does not run one fixed job. Instead, it continuously receives a stream of user retrievals. Each retrieval is turned into many small I/O operations that often touch overlapping regions in large raster files.

Let $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_N\}$ denote a stream of spatio-temporal range retrievals submitted by multiple users. Each retrieval q is decomposed by the I/O-aware index into a set of grid-aligned spatial windows based on a predefined global grid system. These windows are further mapped to sub-regions of one or more large remote sensing images. In this way, every retrieval produces an I/O execution context $c = \langle W, M, S \rangle$, where W describes the set of image windows to be accessed, including their sizes, spatial overlap, and distribution across images; M captures window-level coordination opportunities, such as window merging, deduplication, or shared reads across concurrent retrievals; and S represents system-level execution decisions, including batching strategies, I/O scheduling order, and concurrency limits. Importantly, the I/O behavior of the system is not determined solely by static application code, but emerges dynamically from the interaction between retrieval workloads, execution plans, and system policies.

The goal of I/O tuning in this system is to optimize the performance of retrieval-induced I/O execution under continuous, concurrent workloads. We focus on minimizing the observed I/O cost per retrieval, which may be measured by metrics such as average retrieval latency, effective I/O throughput, or amortized disk read time. Let $\theta \in \Theta$ denote a tuning configuration, where each configuration specifies a combination of system-level I/O control parameters, including window batching size, merge thresholds, queue depth, concurrency limits, and selected storage-level parameters exposed to the engine. Unlike traditional I/O tuning frameworks, the decision variables θ are applied at the retrieval execution level, rather than at application startup or compilation time.

For a given tuning configuration θ and execution context c, the observed I/O performance is inherently stochastic due

to: interference among concurrent retrievals; shared storage contention; variability in window overlap and access locality. We model the observed performance outcome as a random variable:

$$Y(\theta, c) = f(\theta, c) + \epsilon, \tag{10}$$

where $f(\cdot)$ is an unknown performance function and ϵ captures stochastic noise. Moreover, as retrieval workloads evolve over time, the distribution of execution contexts c may change, making the tuning problem non-stationary.

Given a stream of retrievals $\mathcal{Q}$ and the resulting sequence of execution contexts $\{c_t\}$, the problem is to design an online tuning strategy that adaptively selects tuning configurations θ_t for retrieval execution, so as to minimize the long-term expected I/O cost:

$$\min_{\{\theta_t\}} \mathbb{E}\left[\sum_{t=1}^{T} Y(\theta_t, c_t)\right], \tag{11}$$

subject to practical constraints on tuning overhead and system stability.

B. Surrogate-Assisted GMAB for Online I/O Tuning

To address the online I/O tuning problem, we use a Surrogate-Assisted Genetic Multi-Armed Bandit (SA-GMAB) framework. It combines genetic search, bandit-style exploration, and a simple performance model. The goal is to handle workloads where behavior changes over time, where results are random, and where retrievals may affect each other. The main steps of this framework are shown in Algorithm 1.

We first initialize the memory table and the surrogate model, and then generate an initial population of configurations (lines 1–4). In our system, each arm is an I/O tuning configuration $\theta \in \Theta$. A configuration is a group of I/O control parameters, such as merge thresholds, batch size, queue depth, and limits on parallel requests. The space of possible configurations is large and discrete. It is not possible to list or test all of them. Therefore, we do not fix all arms in advance. Instead, new configurations are created dynamically by genetic operators during candidate generation (line 6). Each configuration acts as a policy that tells the system how to run I/O plans during a scheduling period.

When a retrieval q_t with context c_t arrives, the framework enters the online tuning loop (line 5). For this retrieval, a set of candidate configurations is created through selection, crossover, and mutation (line 6). For every candidate configuration, the surrogate model predicts its reward under the current context (lines 7–9). These predicted rewards are then used to filter and keep only the top promising configurations, or those with high uncertainty (line 10).

When a configuration θ is used to process a retrieval q_t with context c_t, the system observes a random performance result $Y_t = Y(\theta, c_t)$. We define the reward as a simple transformation of I/O cost so that a higher reward means better performance. A common form is the negative latency of the retrieval, or the negative I/O time per unit work. Because other retrievals run at the same time, the reward may change even for the same configuration. Thus, many samples are needed to estimate the expected reward.

Algorithm 1: Surrogate-Assisted Genetic Multi-Armed Bandit (SA-GMAB)

Input : Configuration space Θ, Initial population size P, Exploration parameter α, Surrogate update interval Δ
Output: Online selection of I/O coordination configuration θ_t

```
// Initialization
1  Initialize memory table M = ∅;
2  Initialize surrogate model f̃ with empty training data;
3  Generate an initial population P_0 ⊂ Θ;
4  Set tuning step counter t ← 0;
// Online Tuning Loop
5  while arrival of retrieval q_t with execution context c_t do
       // Candidate Generation
6      Apply genetic operators (selection, crossover, mutation) on current population to generate candidate set C_t ⊂ Θ;
       // Surrogate-based Pre-evaluation
7      foreach θ ∈ C_t do
8          r̂_θ ← f̃(θ, c_t);
9      end
       // Candidate Filtering
10     Select top-K configurations C'_t ⊂ C_t based on r̂_θ or uncertainty;
       // Bandit-based Selection
11     foreach θ ∈ C'_t do
12         Score(θ) = μ̂_θ + α sqrt(log(t+1) / (n_θ + 1));
13     end
14     Select configuration: θ_t = arg max_{θ∈C'_t} Score(θ);
       // Retrieval Execution & Reward Observation
15     Execute retrieval q_t using I/O coordination policy θ_t;
16     Measure performance outcome and compute reward r_t;
       // State Update
17     Update memory entry for θ_t: n_{θ_t} ← n_{θ_t} + 1;
18     μ̂_{θ_t} ← μ̂_{θ_t} + (r_t − μ̂_{θ_t}) / n_{θ_t};
19     Update population P by inserting θ_t (optionally evicting low-performing ones);
20     if t mod Δ = 0 then
21         Retrain surrogate model f̃ using observations in M;
22     end
23     t ← t + 1;
24 end
```

For the remaining candidates, the framework computes a bandit score using both historical average reward and exploration term (lines 11–13), and then selects the configuration with the highest score (line 14). In this way, the method prefers configurations that have performed well before, but it also tries

TABLE I
DATASET STATISTICS

Dataset	Resolution	Time Span	Total Volume	Number
MoRIC	76m	2021–2022	1.1 TB	12060
CTX	5m	2007–2024	3.1 TB	3960
THEMIS	100m	2003–2024	6.5 TB	669641
HiRISE	0.5m	2007–2024	41.2 TB	96693

TABLE II
CLUSTER CONFIGURATIONS

Hardware Configuration (Per Node)	
CPU	Intel Xeon Gold 6248 (20 cores, 2.50GHz)
Memory	128GB
Storage	18TB NVMe SSD (Data) + 500GB SSD (OS)
Network	10 Gigabit Ethernet (10GbE)
Software Stack	
OS	Ubuntu 22.04 LTS
Storage	Hadoop 3.2.1, HBase 2.4.5, Lustre
Framework	OpenJDK 11

configurations that have been used only a few times.

The selected configuration is then applied to execute the retrieval (line 15). After execution, the system observes the performance result and converts it into a reward value (line 16). For each configuration θ, the system keeps a memory entry that records how many times it has been used and its average reward. These values are updated after each execution (lines 17–18). This keeps all historical observations instead of discarding older ones, so estimates become more accurate over time, and poor configurations are not repeatedly tried.

The selected configuration may also be added into the population, while poor ones may be removed (line 19). The surrogate model is retrained periodically using data stored in memory (lines 20–22), so that its predictions follow the most recent workload. The tuning step counter is then increased (line 23), and the framework continues with the next retrieval (line 24).

VIII. PERFORMANCE EVALUATION

First, we introduce the experimental setup, covering the dataset characteristics, retrieval workload generation, and the distributed cluster environment. Then, we present the experimental results evaluating the proposed I/O-aware indexing structure, the hybrid concurrency-aware I/O coordination mechanism, and the online I/O tuning framework, respectively.

A. Experimental Setup

1) Dataset: We employed a large-scale, multi-source planetary remote sensing dataset derived from major Martian exploration missions to evaluate the system under heterogeneous workloads. Specifically, the dataset integrates imagery from the Tianwen-1 Medium Resolution Imaging Camera (MoRIC), the Context Camera (CTX), the Thermal Emission Imaging System (THEMIS), and the High Resolution Imaging Science Experiment (HiRISE). These datasets encompass a diverse range of spatial resolutions and spectral bands, covering extensive global Martian surface features. To facilitate efficient window-based I/O and random access, all raw planetary data products were pre-processed and converted into the Cloud-Optimized GeoTIFF (COG) format. The detailed specifications and statistics of the dataset are summarized in Table I.

2) Retrieval Workload: To evaluate the system performance under diverse scenarios, we developed a synthetic workload generator that simulates concurrent spatio-temporal range retrievals. The retrieval parameters are configured as follows:

- Spatial Extent: The spatial range of retrievals follows a log-uniform distribution, ranging from small tile-level access (0.0001% of the scene) to large-scale regional mosaics (1% to 100% of the scene).
- Temporal Range: Each retrieval specifies a time interval randomly chosen between 1 day and 1 month.
- Concurrency & Contention: The number of concurrent clients N varies from 1 to 64. To test the coordination mechanism, we control the Spatial Overlap Ratio $\sigma \in [0, 0.9]$ to simulate workloads ranging from disjoint access to highly concentrated hotspots.

It is worth noting that, given the data-intensive nature of retrievals where a single request triggers GB-scale I/O and complex decoding, 64 concurrent streams are sufficient to fully saturate the aggregate I/O bandwidth and CPU resources of our experimental cluster. With 8 worker nodes connected via 10GbE, a concurrency of 64 implies an average of 8 heavy I/O threads per node. Previous characterization studies on Lustre-based supercomputers [31] have revealed that client-side flow control typically limits in-flight RPCs to 8 concurrent requests and that exceeding this parallelism level exacerbates resource contention and straggler effects. Therefore, this setting represents a realistic heavy-load scenario where I/O interference significantly impacts performance.

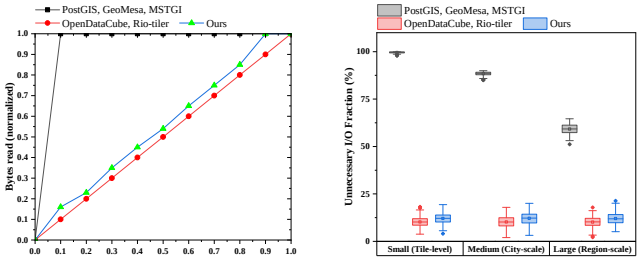
3) Experimental Environment: All experiments are conducted on a cluster with 9 homogenous nodes (1 master node and 8 worker nodes). The cluster is connected via a 10Gbps high-speed Ethernet to ensure that network bandwidth is not the primary bottleneck compared to storage I/O. Table II lists the detailed hardware and software configurations. The I/O-aware index (G2I/I2G) is deployed on HBase, while the raw image data is served by the Lustre parallel file system.

B. Evaluating the Data Indexing Structure

To comprehensively evaluate the effectiveness of the proposed I/O-aware indexing structure, we conducted experiments on a single cluster node, as each node independently performs indexing for spatial retrieval in the distributed setting. We compare our approach against five representative baseline systems that span traditional database indexes, distributed NoSQL-based schemes, and state-of-the-art windowed I/O frameworks.

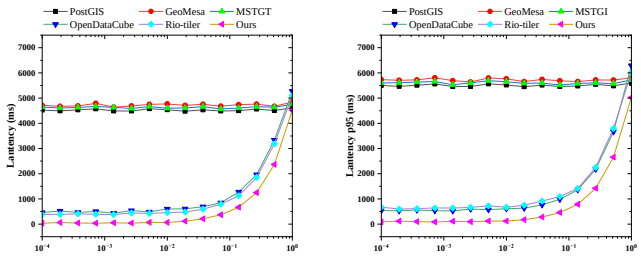
The comparative methods are categorized as follows:

1) **PostGIS (Full-file Retrieval):** A traditional relational database approach that employs R-tree spatial indexes for metadata filtering. While it efficiently identifies candidate images through spatial intersection tests, it



(a) Query footprint ratios (b) Query spatial extents

Fig. 5. The efficiency of I/O selectivity



(a) Query footprint ratios (b) Query footprint ratios

Fig. 6. End-to-End retrieval latency

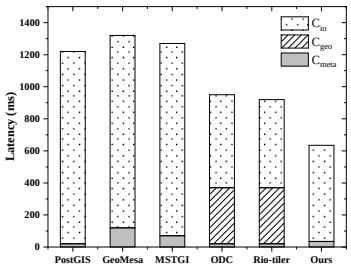


Fig. 7. Latency breakdown

retrieves entire image files during data extraction, incurring substantial I/O overhead even for small spatial queries.

2) **GeoMesa (Full-file Retrieval):** A distributed spatio-temporal index built on Hbase, which encodes spatial footprints using Z-order space-filling curves for scalable metadata discovery. Despite its superior indexing performance for billion-scale datasets, it still relies on full-file data loading.
3) **MSTGI (Full-file Retrieval):** A recently proposed multi-scale spatio-temporal grid index model [5] that enhances GeoMesa through hierarchical time granularity (year/month/day) and Hilbert curve-based linearization. It inherits the full-file retrieval limitation, where data extraction cost remains decoupled from index-level optimization.
4) **OpenDataCube (Window-based I/O):** A state-of-the-art data cube system that couples PostGIS indexes with windowed I/O via rasterio, enabling partial reads from monolithic image files. By leveraging GeoBox-based ROI computation and automatic overview selection, OpenDataCube represents the theoretical optimum for I/O selectivity but incurs runtime geospatial computation overhead to resolve pixel-to-geographic mappings.
5) **rio-tiler (Window-based I/O):** A lightweight raster reading engine optimized for dynamic tile generation. Similar to OpenDataCube, it employs PostGIS for spatial indexing and windowed I/O for partial data access, but features a streamlined execution path with minimal abstraction layers, resulting in lower per-query overhead. rio-tiler serves as a high-performance baseline for windowed reading without the complexity of full data cube management.
6) **Ours (I/O-aware Indexing):** The proposed approach leverages a dual-layer inverted index structure comprising Grid-to-Image (G2I) and Image-to-Grid (I2G) mappings. By pre-materializing grid-to-pixel correspondences at ingestion time, our method translates spatio-temporal predicates directly into byte-level read plans, completely eliminating runtime geometric computations while preserving minimal I/O volume through precise windowed access.

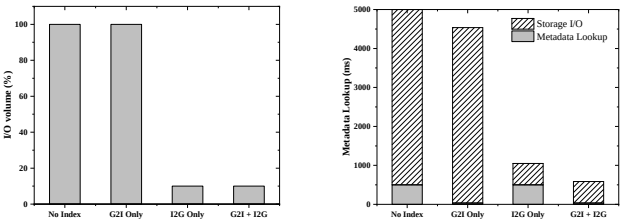
1) I/O Selectivity Analysis: First, we evaluated the effectiveness of data reduction by measuring the I/O selectivity, defined as the ratio of the retrieved data volume to the total file size. Fig. 5 compares our method against baselines. As illustrated in Fig. 5(a), systems such as PostGIS, GeoMesa, and MSTGI, which rely on full file loading, exhibit consistent performance. They always reads the entire image regardless of the proportion of the intersection between the query range and the image. In contrast, OpenDataCube, Rio-tiler, and ours significantly reduce I/O traffic by enabling partial reads. It is worth noting that our method incurs slightly higher I/O volume compared to the theoretically optimal baseline (OpenDataCube and Rio-tiler). This marginal data redundancy is attributed to the grid alignment effect: our index retrieves pixel blocks based on fixed grid boundaries, whereas OpenDataCube and Rio-tiler perform precise geospatial clipping. Fig. 5(b) further presents the distribution of unnecessary data fraction. While our method introduces a small amount of over-reading due to grid padding, it successfully avoids the massive data waste observed in the full-file retrieval systems. As we will demonstrate in the next section, this slight compromise in I/O precision is a strategic trade-off that eliminates expensive runtime computations.

2) End-to-End Retrieval Latency: We next measured the end-to-end retrieval latency to verify whether I/O reduction effectively translates into time efficiency across different indexing and retrieval strategies. Fig. 6(a) reports the mean latency, and Fig. 6(b) shows the 95th percentile (p95) across varying retrieval footprint ratios. The results reveal three distinct performance categories, each corresponding to a fundamental architectural design choice.

As shown in Fig. 6(a), all three full-file baselines (PostGIS, GeoMesa, MSTGI) exhibit high and flat latency curves ranging from approximately 4,480 to 4,850 ms, nearly independent of the query footprint ratio. This performance ceiling is

imposed by the necessity of transferring complete image files regardless of the spatial extent requested. Among these methods, PostGIS achieves the lowest latency ($\approx 4,500$ ms) due to the efficiency of R-tree traversal for metadata filtering. GeoMesa incurs slightly higher latency ($\approx 4,700$–4,850 ms) as a result of distributed query coordination overhead in its Hbase-backed architecture. MSTGI positions between these two ($\approx 4,600$–4,750 ms), reflecting its multi-scale indexing optimization that partially compensates for the underlying GeoMesa framework. Critically, all three methods are dominated by massive I/O transfer that entirely masks computational overheads, rendering index-level optimizations ineffective for end-to-end latency. In contrast, both OpenDataCube and Rio-tiler demonstrate strong correlation between retrieval latency and query footprint, confirming that partial reading successfully eliminates unnecessary data transfer. For small tile-level retrievals (footprint ratio $\leq 10^{-3}$), rio-tiler achieves latencies of 34–41 ms, while OpenDataCube ranges from 43–52 ms. This 15–20% advantage of rio-tiler stems from its streamlined execution path with minimal abstraction layers. However, as the query footprint grows beyond 10^{-2}, both methods exhibit linear latency growth, reaching 5,090 ms (rio-tiler) and 5,270 ms (OpenDataCube) at full-image queries. The floor latency of approximately 380 ms for small retrievals indicates a fixed computational cost component that is not attributable to I/O. Our method achieves the lowest latency across all query scales. For typical tile-level retrievals (footprint ratio 10^{-4} to 10^{-2}), latency ranges from 32 to 114 ms, representing a 3.8–12× speedup over rio-tiler and a 4.2–13× improvement over OpenDataCube. Crucially, our method preserves near-constant latency for small-to-medium queries and only exhibits noticeable growth when the footprint ratio exceeds 0.1. At the extreme of full-image queries, our approach reaches 4,525 ms, comparable to full-file methods, as the grid-alignment overhead becomes negligible relative to complete data transfer.

To empirically validate the cost model in Eq. 3, we decomposed the retrieval latency into three components: metadata lookup (C_{meta}), geospatial computation (C_{geo}), and I/O access (C_{io}). Fig. 7 presents the breakdown for a representative medium-scale retrieval involving approximately 50 image tiles. PostGIS, GeoMesa, and MSTGI all exhibit C_{io}-dominated profiles, with I/O access consuming approximately 1,200 ms and accounting for over 98% of total latency. PostGIS maintains the lowest C_{meta} (20 ms) due to its single-node R-tree structure. GeoMesa incurs higher C_{meta} (120 ms) from distributed metadata coordination. MSTGI achieves intermediate C_{meta} (70 ms) by optimizing the query path through its hierarchical time-granularity design. All three methods maintain $C_{geo} \approx 0$ since no geometric computation is performed—entire images are returned directly. Both OpenDataCube and rio-tiler successfully reduce C_{io} to approximately 550–580 ms by reading only intersecting windows. However, this I/O advantage is partially offset by substantial C_{geo} overhead (350 ms, representing 38–40% of total latency), incurred by runtime coordinate transformations and precise clipping computations. The nearly identical C_{meta} (20 ms) reflects their shared reliance on PostGIS for spatial indexing. Our method achieves a balanced profile with $C_{meta} = 35$ ms



(a) I/O reduction analysis (b) Latency breakdown

Fig. 8. Ablation analysis

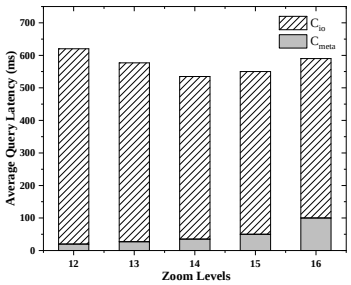


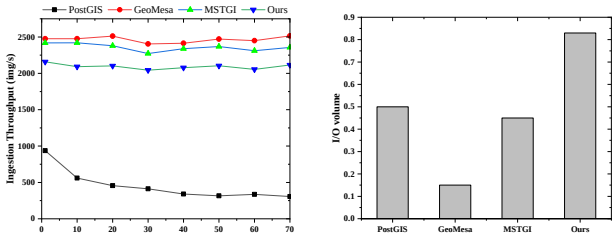
Fig. 9. Impact of grid resolution on query latency

(slightly higher than windowed baselines due to the two-phase G2I+I2G lookup) and $C_{io} = 600$ ms (comparable to windowed I/O methods). Critically, C_{geo} is completely eliminated by pre-materialized grid-to-pixel mappings, resulting in a total latency of 635 ms. This represents a 1.7× speedup over OpenDataCube (1,070 ms) and a 1.6× improvement over rio-tiler (970 ms). The decomposition validates our core thesis: by shifting computational burden from retrieval time to ingestion time, I/O-aware indexing achieves near-optimal I/O cost while avoiding the computational bottleneck that plagues existing windowed I/O systems.

3) Ablation Study: To quantify the individual contributions of the G2I (coarse filtering) and I2G (fine-grained access) components, we decomposed the system into four variants. Fig. 8 breaks down the performance in terms of I/O volume and latency components. Fig. 8(a) confirms that removing either component leads to suboptimal I/O behavior. The *No Index* and *G2I Only* variants result in 100% I/O volume, as they lack the window information required for partial access. Conversely, *I2G Only* and *G2I+I2G* achieve minimal I/O volume ($\approx 10\%$).

Fig. 8(b) reveals the latency breakdown. *No Index* suffers from both high full table scanning cost and high storage I/O cost. *G2I Only* efficiently reduces metadata lookup time ($\approx$ 50 ms) but fails to reduce storage I/O ($\approx$ 8000 ms). Although *I2G Only* minimizes storage I/O ($\approx$ 100 ms), it incurs prohibitive metadata lookup overhead ($\approx$ 1500 ms) because the system must scan the entire I2G table to identify relevant images without spatial pruning. *G2I+I2G* achieves the best performance, maintaining low metadata latency ($\approx$ 60 ms) via G2I pruning while ensuring minimal storage I/O ($\approx$ 100 ms) via I2G windowing.

Moreover, the choice of grid resolution (Zoom Level) is a critical parameter that dictates the trade-off between metadata



(a) Ingested images (10^4) (b) Various index types

Fig. 10. Index construction and storage overhead

management overhead and I/O precision. To justify our selection of Zoom Level 14, we conducted a sensitivity analysis by varying the grid resolution from Level 12 to Level 16 under a fixed workload of medium-scale range queries. Fig. 9 illustrates the latency breakdown across different resolutions. The results reveal a clear convex trajectory in total query latency, driven by two opposing forces. For coarse-grained grids (Level ≤ 13), while metadata lookup is extremely fast ($C_{meta} < 30$ ms) due to the small number of grid keys, the I/O cost is prohibitively high. Large grid cells force the system to read significant amounts of irrelevant pixel data outside the actual query boundary, serving as the dominant bottleneck. Conversely, finer grids (Level 15, 16) maximize I/O precision, reducing C_{io} to its theoretical minimum. However, this comes at the expense of increased metadata volume. A single query may intersect thousands of Level 16 micro-grids, causing C_{meta} to surge drastically (> 100 ms) due to the overhead of scanning and processing massive key lists in the G2I/I2G tables. As evidenced by the trough in the total latency curve, Zoom Level 14 represents the optimal spot for our dataset. At this resolution, the grid cell size roughly matches the typical internal tile size of remote sensing images, keeping I/O waste low while maintaining a manageable number of index keys. Consequently, our system adopts Level 14 as the default global configuration.

4) Index Construction and Storage Overhead: Finally, we evaluated the scalability and cost of maintaining the index. Fig. 10 compares our method against PostGIS (R-tree), GeoMesa (Z-order), and MSTGI during the ingestion of 7×10^5 images. Fig. 10(a) illustrates the ingestion throughput. PostGIS exhibits a degrading trend as the dataset grows, bottlenecked by the logarithmic cost of R-tree rebalancing. In contrast, GeoMesa maintains a stable and high throughput (≈ 2500 img/s) owing to its append-only write pattern in HBase. MSTGI achieves a throughput of $\approx$ 2350 img/s—slightly lower than GeoMesa due to the additional overhead of maintaining multi-scale temporal indexes. Our method demonstrates stable throughput at ≈ 2100 img/s, lower than both GeoMesa and MSTGI due to the dual-table write overhead (G2I + I2G), yet still exhibits linear scalability suitable for high-velocity streaming data. Regarding storage cost (Fig. 10(b)), our index occupies approximately 0.83% of the raw data size. GeoMesa maintains the lowest storage footprint (0.15%) by encoding only spatial footprints via Z-order curves. MSTGI incurs moderately higher storage cost as it must maintain multi-scale temporal partitions alongside spatial indexes. While our method's storage overhead is higher than both GeoMesa and PostGIS (0.51%) due to the storage of pre-materialized grid-window mappings, it remains strictly below the 1% threshold. This result validates that the proposed method achieves significant performance gains with a negligible storage penalty.

C. Evaluating the Concurrency Control

In this section, we evaluate the proposed hybrid coordination mechanism on a distributed storage cluster to assess its scalability, robustness under contention, and internal storage efficiency.

To systematically control the workload characteristics, we developed a synthetic workload generator. We define the Spatial Overlap Ratio (σ) to quantify the extent of shared data regions among concurrent queries, ranging from $\sigma = 0$ (disjoint) to $\sigma = 0.9$ (highly concentrated hotspots). The number of concurrent clients varies from $N = 1$ to $N = 64$.

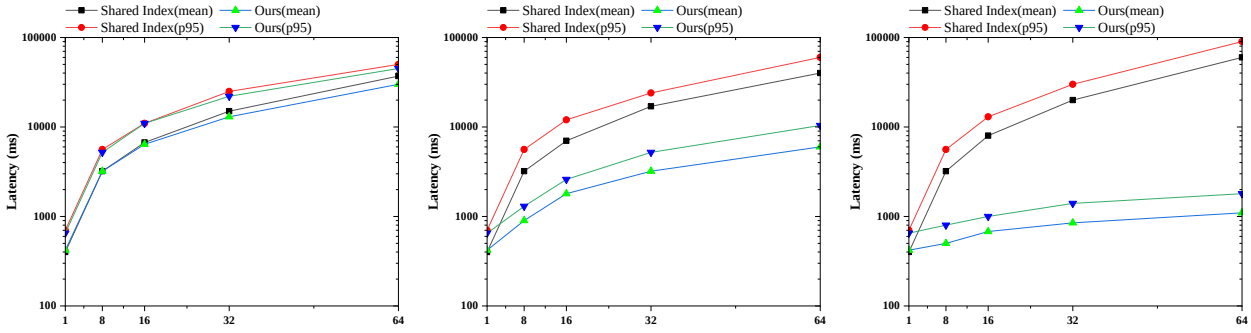
For comparison, we evaluate the following execution schemes:

1) **Baseline (Shared Index):** Metadata access is shared, but data retrieval remains uncoordinated, representing the state-of-the-art systems like OpenDataCube.
2) **Ours:** The proposed mechanism featuring contention-aware switching, global I/O plan ordering, and window merging.

1) Concurrency Scalability: To evaluate the system's robustness under different workload characteristics, we conducted a sensitivity analysis by manipulating the Spatial Overlap Ratio (σ). We examined three distinct scenarios: low overlap ($\sigma = 0.4$, simulating dispersed random queries), medium overlap ($\sigma = 0.6$), and high overlap ($\sigma = 0.8$, simulating hotspot analysis). Note that $\sigma = 0.4$ is defined as a low overlap scenario because: when $\sigma \leq 0.35$, the performance of the deterministic scheduling mode is even lower than that of the optimistic mode (See Sec. VIII-C3). So, the performance of our method is the same as the Baseline when $\sigma \leq 0.35$. Fig. 11 illustrates the query latency trends as the number of concurrent clients increases from 1 to 64.

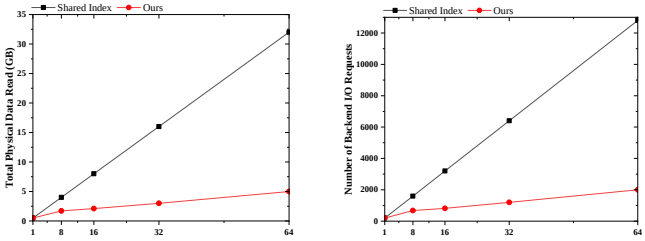
The results reveal a fundamental divergence in how the two systems respond to data contention. As shown in Fig. 11(a), when query footprints are spatially dispersed, the opportunities for physical I/O merging are minimal. Consequently, the performance of both systems is primarily constrained by the aggregate physical bandwidth of the storage cluster. Both approaches exhibit linear latency growth with respect to the client count. At $N = 64$, the Baseline reaches a mean latency of approx. 37,000 ms, while our method records approx. 30,000 ms. Although our method maintains a slight performance edge due to the reduced read amplification provided by the I/O-aware index, it inevitably degrades to a linear query processing mode similar to the Baseline. This confirms that without spatial locality to leverage request collapsing, the system is bound by the hardware's I/O throughput limits.

A sharp performance divergence is observed as the overlap ratio increases to $\sigma = 0.8$ (Fig. 11(b)). The Baseline suffers from severe performance degradation, with latency spiking



(a) $\sigma = 0.4$ (b) $\sigma = 0.6$ (c) $\sigma = 0.8$

Fig. 11. Concurrency scalability analysis under varying spatial overlap ratios (σ).



(a) The number of clients (b) The number of clients

Fig. 12. The data volume reduction and request collapse

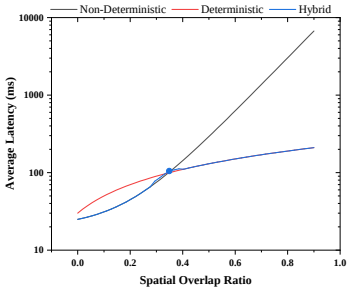


Fig. 13. Mode Switching

from 37,000 ms (at $\sigma = 0.2$) to 60,000 ms (at $\sigma = 0.8$) under peak load. This deterioration is attributed to the I/O blender effect and lock convoys: highly concurrent requests competing for the same index pages and disk blocks cause excessive disk seek thrashing and thread blocking, significantly reducing effective throughput. Conversely, our method demonstrates sub-linear scalability in this scenario. The latency at $N = 64$ drops significantly to 1,100 ms—a $54\times$ speedup over the Baseline. This result validates the efficacy of the *Request Collapse* mechanism. As σ increases, the probability of multiple logical queries targeting the same physical byte ranges rises, allowing the scheduler to merge N concurrent requests into a single physical I/O operation.

The medium-overlap scenario (Fig. 11(c)) serves as a transition point, where our method achieves a mean latency of approx. 6,000 ms at peak load, compared to 40,000 ms for the Baseline. This indicates that the system's efficiency scales dynamically with the degree of data contention. The experimental results demonstrate the workload-adaptive nature of the proposed architecture. While the system performs comparably to traditional approaches under dispersed workloads (limited by physical bandwidth), its advantages become order-of-magnitude significant in data-intensive, high-contention scenarios, effectively turning I/O contention into an opportunity for optimization.

2) Storage-Level Effects and Request Collapse: To uncover the cause of the significant latency reduction observed in high-contention scenarios ($\sigma = 0.8$), we further analyzed the internal I/O behavior of the system. Specifically, we measured the total volume of physical data transferred from disk and the number of backend I/O requests issued to the storage system. Fig. 12 compares the physical storage pressure between the shared index baseline and our method.

Fig. 12(a) plots the total physical data read size. The baseline exhibits a strict linear increase in data volume. At $N = 64$, the system is forced to fetch 32 GB of data. This confirms that without coordination, logically overlapping queries translate into redundant physical reads, leading to severe bandwidth saturation. In contrast, our approach effectively decouples logical demand from physical execution. Although 64 clients logically request 32 GB of data, the request collapse mechanism merges these overlapping windows, resulting in only 5 GB of actual disk traffic. This 84% reduction in data volume explains why our system avoids the bandwidth bottleneck.

Fig. 12(b) illustrates the number of backend I/O requests (IOPS). The baseline generates distinct I/O requests for every client, reaching 12,800 requests at $N = 64$. This massive influx of small random reads overwhelms the storage scheduler, causing excessive disk head seek latency. Our system keeps the request count low and stable. Even at peak load ($N = 64$), the number of physical I/O requests is suppressed to 2,000.

Fig. 12(a) and Fig. 12(b) demonstrate the Request Collapse effect. While 64 concurrent clients generate 12,800 IOPS in the baseline, our system collapses them into fewer than 600 physical operations.

3) Deterministic and Non-Deterministic Modes: To validate the necessity of the proposed hybrid switching mechanism, we compared our adaptive approach against two static execution policies: non-deterministic mode and deterministic mode. We swept the spatial overlap ratio from 0 to 0.9

to capture the performance crossover zone. The workload concurrency was fixed at a moderate level ($N = 16$) to highlight the sensitivity to spatial contention.

Fig. 13 depicts the average latency curves for the three strategies. The results demonstrate that neither static policy is universally optimal, validating the design of our contention-aware switch. In the low-overlap condition, the non-deterministic mode achieves the lower latency. In contrast, the deterministic mode incurs a higher baseline latency. This performance gap is attributed to the inherent coordination overhead. Even when no conflicts exist, the deterministic scheduler must still perform time-window batching and plan sorting. Consequently, enforcing global ordering for disjoint queries introduces unnecessary serialization latency.

As σ increases, the non-deterministic approach suffers from rapid performance deterioration, exhibiting an exponential growth trend. At $\sigma = 0.4$, its latency spikes to 146 ms. This degradation is caused by severe lock and I/O contention at the storage layer, where uncoordinated threads compete for the same disk blocks. Conversely, the deterministic mode exhibits stable, linear scalability thanks to its request merging capability, maintaining a latency of 110 ms at $\sigma = 0.4$.

Our hybrid approach successfully combines the benefits of both worlds. As shown in Fig. 13, the hybrid curve (blue line) tightly tracks the lower performance envelope of the two static policies. The system identifies the intersection point at $\sigma \approx 0.34$. When contention is below this threshold, it operates optimistically to minimize latency. Once contention exceeds it, the system automatically engages the deterministic scheduler to prevent thrashing.

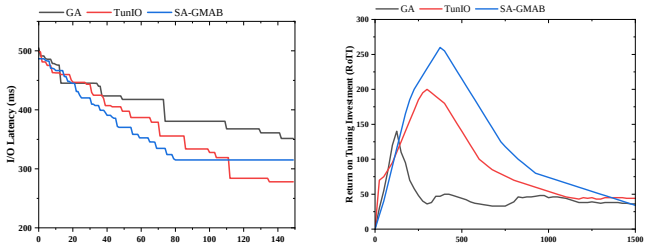
D. Evaluating the I/O Tuning

In this section, we evaluate the effectiveness of the proposed SA-GMAB tuning framework. The experiments are designed to verify four key properties: fast convergence speed, robustness against stochastic noise, adaptability to workload shifts, and tangible end-to-end performance gains.

For comparison, we benchmark against three representative tuning strategies:

1) **Genetic algorithm (GA):** The standard genetic algorithm to explore the configuration space, serving as the basic algorithm in the TunIO.
2) **TunIO:** A state-of-the-art framework that integrates high-impact parameter selection and Reinforcement Learning (RL)-driven early stopping to balance tuning cost and performance in complex HPC I/O stacks.
3) **SA-GMAB (Ours):** The proposed framework combining surrogate modeling with a Genetic Multi-Armed Bandit strategy, explicitly designed to accelerate convergence and handle the stochastic performance fluctuations of concurrent workloads.

1) Convergence Speed and Tuning Cost: We first initiated a cold-start tuning session to evaluate how efficiently each method identifies high-quality configurations starting from a default, unoptimized state. Fig. 14(a) reports the convergence trajectory of the best-observed latency over tuning steps.



(a) Tuning steps (b) Time (mins)

Fig. 14. Efficiency analysis of the tuning framework.

As illustrated in Fig. 14(a), the three methods exhibit distinct search behaviors. The GA baseline demonstrates the slowest convergence. It exhibits a staircase-like descent with prolonged plateaus, requiring over 100 steps to reduce latency significantly. This sluggishness is attributed to its mutation mechanism, which lacks historical memory and repeatedly explores ineffective parameter spaces. The RL-based TunIO outperforms GA but still suffers from a slow start. While it eventually reaches a competitive latency ($\approx$ 277 ms at step 140), its exploration phase is costly. The reinforcement learning agent requires a substantial number of interaction samples to learn the complex mapping between I/O parameters and reward signals. Our method achieves the fastest latency drop, rapidly decreasing from 500 ms to a near-optimal zone ($\approx$ 315 ms) within a short window. Unlike GA and TunIO, SA-GMAB leverages the surrogate model to pre-screen candidates. By effectively pruning unpromising configurations before they incur actual execution costs, SA-GMAB maximizes the information gain per step, making it particularly suitable for online scenarios where tuning overhead must be minimized.

To strictly quantify the cost-effectiveness of the tuning process, we adopt the *Return on Tuning Investment* (RoTI) metric proposed in TunIO [15]. We define the application performance $\mathcal{P}$ as the reciprocal of the query latency (i.e., $\mathcal{P} \propto 1/\mathcal{L}$). The RoTI metric is formalized as follows:

$$RoTI(t) = \frac{\mathcal{P}_{achieved}(t) - \mathcal{P}_{initial}}{t}, \tag{12}$$

where t denotes the cumulative tuning time (overhead). $\mathcal{P}_{initial} = 1/\mathcal{L}_0$ represents the baseline performance derived from the default configuration, and $\mathcal{P}_{achieved}(t) = 1/\mathcal{L}_t$ represents the maximum performance achieved up to time t. Functionally, this metric represents the performance gain purchased per unit of tuning time. A higher RoTI value signifies that the optimizer rapidly identifies low-latency configurations with minimal computational overhead.

Fig. 14(b) plots the RoTI curves over time. Our method (SA-GMAB) reaches a remarkable RoTI peak ($\approx$ 100) at the early stage ($t = 825$). This indicates that SA-GMAB yields the highest immediate return on investment, successfully locating high-quality configurations when the tuning budget is strictly limited. In contrast, TunIO peaks at a significantly lower value ($\approx$ 68), while GA remains flat and inefficient ($\approx$ 46). This confirms that the surrogate-assisted mechanism effectively amplifies the value of each exploration step. All

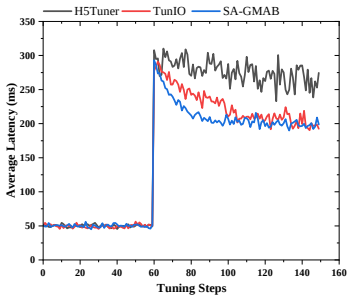


Fig. 15. Mode Switching

curves exhibit a decaying trend as time progresses ($t \to \infty$). This is expected behavior: as the system converges to the global optimum, the marginal performance gain ($\Delta\mathcal{P}$) saturates while the accumulated time t continues to grow. Notably, SA-GMAB's RoTI decays faster in the late stages simply because it has already exhausted the potential for improvement much earlier than the baselines.

2) Adaptation to Workload Shifts: We further investigated the system's resilience in non-stationary environments. We introduced a sudden workload shift at tuning step $t = 60$, drastically changing the query pattern, from sparse random access to dense sequential scans to invalidate the previously learned optimal parameters.

Fig. 15 illustrates the latency evolution before and after the shift. At $t = 60$, the workload transition causes an immediate performance collapse across all methods, with latency spiking from a stable ≈ 50 ms to > 300 ms. This confirms that the configuration optimal for the previous phase is detrimental in the new environment. The GA-based method fails to adapt effectively. Post-shift, its latency hovers around $290 - 300$ ms. Lacking a mechanism to quickly reset or guide exploration, the genetic algorithm remains trapped in the local optima of the previous workload, exhibiting almost zero recovery within the observation window. TunIO manages to reduce latency but at a slow pace. It takes 40 steps to lower the latency from 308 ms to 134 ms ($t = 100$). While the RL agent eventually learns the new reward function, the high sample complexity delays the recovery, leaving the system in a suboptimal state for a prolonged period. In contrast, SA-GMAB executes a decisive recovery. By leveraging the surrogate model to filter high-uncertainty candidates, it rapidly identifies the new optimal region. The latency drops to ≈ 88 ms at $t = 80$ and further stabilizes at ≈ 74 ms at $t = 100$.

IX. Conclusions

This paper presents an I/O-aware retrieval approach designed to bound retrieval latency and maximize throughput for large-scale spatio-temporal analytics. By introducing the "Index-as-an-Execution-Plan" paradigm, the dual-layer inverted index bridges the semantic gap between logical indexing and physical storage, effectively shifting the computational burden from retrieval time to ingestion time. To address the scalability challenges in concurrent environments, we developed a hybrid concurrency-aware I/O coordination protocol that adaptively switches between deterministic ordering and optimistic execution based on spatial contention. Furthermore, to handle the complexity of parameter configuration in fluctuating workloads, we integrated the SA-GMAB method for online automatic I/O tuning. The experimental results indicate that: (1) I/O-aware indexing achieves an order-of-magnitude latency reduction with negligible storage overhead; (2) the hybrid coordination protocol realizes a $54\times$ throughput improvement in high-overlap scenarios; and (3) the SA-GMAB method recovers from workload shifts $2\times$ faster than RL baselines while maximizing RoTI.

Acknowledgments

This work is supported by the National Key R&D Program of China "Intergovernmental International Science and Technology Innovation Cooperation" (Grant No.2025YFE0107100).

References

[1] "Remote sensing big data computing: Challenges and opportunities," *Future Gener. Comput. Syst.*, vol. 51, pp. 47–60, 2015, special Section: A Note on New Trends in Data-Aware Scheduling and Resource Provisioning in Modern HPC Systems.

[2] J. M. Haut, M. E. Paoletti, S. Moreno-Álvarez, J. Plaza, J. Rico-Gallego, and A. Plaza, "Distributed deep learning for remote sensing data interpretation," *Proc. IEEE*, vol. 109, no. 8, pp. 1320–1349, 2021.

[3] "The australian geoscience data cube — foundations and lessons learned," *Remote Sens. Environ.*, vol. 202, pp. 276–292, 2017, big Remotely Sensed Data: tools, applications and experiences.

[4] J. Yan, Y. Liu, L. Wang, Z. Wang, X. Huang, and H. Liu, "An efficient organization method for large-scale and long time-series remote sensing data in a cloud computing environment," *IEEE J. Sel. Top. Appl. Earth Obs. Remote. Sens.*, vol. 14, pp. 9350–9363, 2021.

[5] H. Liu, J. Yan, J. Wang, D. Zhang, J. Li, L. He, and X. Yu, "Mstgi: a multi-scale spatio-temporal grid index model for remote-sensing big data retrieval," *Remote Sensing Letters*, vol. 15, no. 1, pp. 44–54, 2024.

[6] C. Strobl, "Postgis," in *Encyclopedia of GIS*, S. Shekhar and H. Xiong, Eds. Springer, 2008, pp. 891–898.

[7] R. E. O. Simões, G. R. de Queiroz, K. R. Ferreira, L. Vinhas, and G. Câmara, "Postgis-t: towards a spatiotemporal postgresql database extension," in *XVII Brazilian Symposium on Geoinformatics - GEOINFO 2016, Campos do Jordão, SP, Brazil, November 27-30, 2016*, C. E. C. Campelo and L. M. Namikawa, Eds. MCTIC/INPE, 2016, pp. 252–262.

[8] I. S. Suwardi, D. Dharma, D. P. Satya, and D. P. Lestari, "Geohash index based spatial data model for corporate," in *2015 International Conference on Electrical Engineering and Informatics (ICEEI)*. IEEE, 2015, pp. 478–483.

[9] J. N. Hughes, A. Annex, C. N. Eichelberger, A. Fox, A. Hulbert, and M. Ronquest, "Geomesa: a distributed architecture for spatio-temporal fusion," in *Geospatial informatics, fusion, and motion video analytics V*, vol. 9473. SPIE, 2015, pp. 128–140.

[10] R. Li, H. He, R. Wang, S. Ruan, T. He, J. Bao, J. Zhang, L. Hong, and Y. Zheng, "Trajmesa: A distributed nosql-based trajectory data management system," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 1, pp. 1013–1027, 2023.

[11] 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.

[12] "rio-tiler: User friendly rasterio plugin to read raster datasets," 2025. [Online]. Available: https://github.com/cogeotiff/rio-tiler

[13] A. Thomson, T. Diamond, S. Weng, K. Ren, P. Shao, and D. J. Abadi, "Calvin: fast distributed transactions for partitioned database systems," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, K. S. Candan, Y. Chen, R. T. Snodgrass, L. Gravano, and A. Fuxman, Eds. ACM, 2012, pp. 1–12.

[14] H. Lim, M. Kaminsky, and D. G. Andersen, "Cicada: Dependably fast multi-core in-memory transactions," ser. SIGMOD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 21–35.

[15] N. Rajesh, K. Bateman, J. L. Bez, S. Byna, A. Kougkas, and X. Sun, "Tunio: An ai-powered framework for optimizing HPC I/O," in *IEEE International Parallel and Distributed Processing Symposium, IPDPS 2024, San Francisco, CA, USA, May 27-31, 2024*. IEEE, 2024, pp. 494–505.

[16] D. Preil and M. Krapp, "Genetic multi-armed bandits: A reinforcement learning inspired approach for simulation optimization," *IEEE Trans. Evol. Comput.*, vol. 29, no. 2, pp. 360–374, 2025.

[17] J. Tang, Z. Zhou, K. Ning, Y. Sun, and Q. Wang, "A novel spatial indexing mechanism leveraging dynamic quad-tree regional division," in *Proceedings of the 2012 International Conference on Information Technology and Software Engineering*, W. Lu, G. Cai, W. Liu, and W. Xing, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 909–917.

[18] X. Yang, X. Guan, Z. Pang, X. Kui, and H. Wu, "Gridmesa: A nosql-based big spatial data management system with an adaptive grid approximation model," *Future Gener. Comput. Syst.*, vol. 155, pp. 324–339, 2024.

[19] Y. Hong, H. Zhao, Y. Wang, X. Shi, W. Lu, S. Yang, and S. Du, "Deterministic concurrency control based multiwriter transaction processing over cloud-native databases." *International Journal of Software & Informatics*, vol. 15, no. 1, 2025.

[20] P. A. Bernstein and N. Goodman, "Concurrency control in distributed database systems," *ACM Comput. Surv.*, vol. 13, no. 2, pp. 185–221, 1981.

[21] H. T. Kung and J. T. Robinson, "On optimistic methods for concurrency control," *ACM Trans. Database Syst.*, vol. 6, no. 2, pp. 213–226, 1981.

[22] T. Wang and H. Kimura, "Mostly-optimistic concurrency control for highly contended dynamic workloads on a thousand cores," *Proc. VLDB Endow.*, vol. 10, no. 2, pp. 49–60, 2016.

[23] Y. Hong, H. Zhao, W. Lu, X. Du, Y. Chen, A. Pan, and L. Zheng, "A hybrid approach to integrating deterministic and non-deterministic concurrency control in database systems," *Proc. VLDB Endow.*, vol. 18, no. 5, pp. 1376–1389, 2025.

[24] H. Wu, M. Zhang, K. Chen, X. Liao, Y. Shan, and Y. Wu, "OOCC: one-round optimistic concurrency control for read-only disaggregated transactions," in *41st IEEE International Conference on Data Engineering, ICDE 2025, Hong Kong, May 19-23, 2025*. IEEE, 2025, pp. 238–250.

[25] J. Peng, L. Yang, H. Wu, W. Zhang, Z. Wu, W. Zhang, J. Li, Y. Dai, and Y. Dong, "A survey on machine learning-based hpc i/o analysis and optimization," *IEEE Trans. Parallel Distrib. Syst.*, vol. 37, no. 3, pp. 618–632, 2026.

[26] T. Chen and M. Li, "Multi-objectivizing software configuration tuning (for a single performance concern)," *CoRR*, vol. abs/2106.01331, 2021.

[27] J. L. Bez, F. Z. Boito, R. Nou, A. Miranda, T. Cortes, and P. O. A. Navaux, "Adaptive request scheduling for the I/O forwarding layer using reinforcement learning," *Future Gener. Comput. Syst.*, vol. 112, pp. 1156–1169, 2020.

[28] B. Yang, Y. Zou, W. Liu, and W. Xue, "An end-to-end and adaptive I/O optimization tool for modern HPC storage systems," in *2022 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2022, Lyon, France, May 30 - June 3, 2022*. IEEE, 2022, pp. 1294–1304.

[29] B. Behzad, J. Huchette, H. V. T. Luu, R. A. Aydt, S. Byna, Y. Yao, Q. Koziol, and Prabhat, "A framework for auto-tuning HDF5 applications," in *The 22nd International Symposium on High-Performance Parallel and Distributed Computing, HPDC'13, New York, NY, USA - June 17 - 21, 2013*, M. Parashar, J. B. Weissman, D. H. J. Epema, and R. J. O. Figueiredo, Eds. ACM, 2013, pp. 127–128.

[30] W. Zhang, H. Wang, Z. Shi, Y. Wu, M. Li, T. Li, F. Wang, and D. Feng, "Rethinking parameter tuning in distributed storage systems via knowledge graph query," *IEEE Trans. Parallel Distrib. Syst.*, vol. 37, no. 3, pp. 633–650, 2026.

[31] B. Xie, J. S. Chase, D. Dillow, O. Drokin, S. Klasky, S. Oral, and N. Podhorszki, "Characterizing output bottlenecks in a supercomputer," in *SC Conference on High Performance Computing Networking, Storage and Analysis, SC '12, Salt Lake City, UT, USA - November 11 - 15, 2012*, J. K. Hollingsworth, Ed. IEEE/ACM, 2012, p. 8.