Benchmarking Spatial Operations over Heterogeneous Data: The Case of Zonal Statistics

Gereon Dusella
BIFOLD, Technische Universität Berlin
Germany
gereon.dusella@tu-berlin.de

Haralampos Gavriilidis
BIFOLD, Technische Universität Berlin
Germany
gavriilidis@tu-berlin.de

Volker Markl
BIFOLD, Technische Universität Berlin, DFKI
Germany
volker.markl@tu-berlin.de

Eleni Tzirita Zacharatou
Hasso Plattner Institute, University of Potsdam
Germany
eleni.tziritazacharatou@hpi.de

Abstract

Zonal Statistics (ZS) is a fundamental operation in Earth Observation workflows. It aggregates raster pixel values within regions defined by vector geometries, such as computing average vegetation indices across farmland parcels. Unlike traditional database operations, which operate within a single data model, ZS requires joining two fundamentally different spatial data models: raster and vector. This data heterogeneity introduces unique benchmarking challenges that existing spatial benchmarks, which focus on one data model in isolation, do not address. In this paper, we present an experimental study of spatial operations over raster and vector data, using ZS as a representative case. Evaluating three architecturally diverse systems—PostGIS (relational), Beast (dataflow), and RasDaMan (array-based)—across 25 queries over 17 real-world datasets, we find that the interaction between data characteristics (geometry type, raster-to-vector size ratio, coordinate reference systems) and system internals causes performance differences of up to 22× between competitive systems, while parameter tuning alone can yield over 55× speedup within a single system.

CCS Concepts

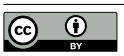
• **Information systems** → **Spatial-temporal systems**; **Database performance evaluation**.

Keywords

zonal statistics; raster-vector; spatial benchmark; remote sensing

1 Introduction

The rapid growth of satellite imagery and open geospatial datasets [6, 11] has made Zonal Statistics (ZS) a critical operation in Earth Observation (EO) pipelines [7]. ZS combines raster data (e.g., satellite imagery) with vector data (e.g., administrative boundaries) by aggregating pixel values within vector-defined zones—for example, computing average vegetation indices for farmland parcels [25] or tracking wildfire spread across defined regions [30].



DBTest '26, Bengaluru, India

ACM ISBN 979-8-4007-2701-6/26/05
https://doi.org/10.1145/3810991.3811631

What makes ZS particularly interesting for benchmarking is that it operates on *heterogeneous* spatial data. Unlike traditional databases that work only within the relational model, ZS requires combining two fundamentally different data models: raster data (regular grids of pixel values) and vector data (geometric features on a continuous coordinate space). This heterogeneity introduces challenges that have no direct parallel in standard database benchmarks: coordinate reference system alignment across datasets, model-specific preprocessing requirements, and tuning parameters sensitive to interactions between the data characteristics of both models.

Several spatial data systems support ZS, including PostGIS [1], Beast [16], RasDaMan [3], and Apache Sedona [2]. However, existing spatial benchmarks either focus on one data model in isolation [22, 28], or, in the case of combined workloads [32], evaluate systems only as-is, leaving out tuning parameters, and cross-format interactions that, as we show, dominate performance.

In this paper, we present an experimental study of spatial operations over heterogeneous data, using ZS as a representative and practically important case. We leverage Raven [15], our open-source ZS middleware, to ensure that each system executes the same logical query with consistent preprocessing. Our contributions are:

(1) We highlight and analyze several **challenges** that arise when benchmarking spatial operations over heterogeneous raster and vector data, which are not captured by existing single-data-model spatial benchmarks (Section 3).
(2) We present **experimental results** across relational, dataflow, and array-based spatial systems using 17 real-world datasets and 25 queries. Our results reveal performance gaps of up to 22× between competitive systems, and over 100× for less competitive ones, driven by workload characteristics. Notably, the often overlooked CRS alignment strategy is the most impactful tuning parameter for ZS workloads (Section 4).

2 Background

This section introduces the raster and vector data models, Zonal Statistics, spatial data alignment, and the systems under test.

2.1 Raster and Vector Data

Geospatial data comes in two fundamentally different data models. *Raster data* represents spatial information as a regular grid of cells (pixels), where each cell corresponds to a fixed geographic area and stores a numeric value such as temperature, elevation,

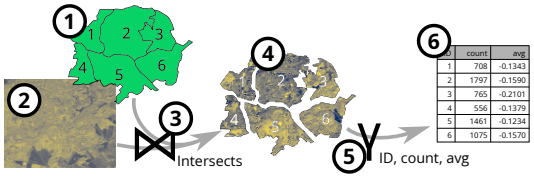


Figure 1: Zonal Statistics: vector features ① define zones over a raster image ②. After a spatial join ③④, pixel values are aggregated per zone ⑤ to produce a statistics table ⑥.

or a vegetation index. Satellite images, digital elevation models, and climate datasets are typically captured in raster format (e.g., GeoTIFF files). *Vector data* represents geographic features using geometric primitives—points, LineStrings, and polygons—along with associated attributes. Examples include administrative boundaries, coastlines, and wildfire perimeters.

The two data models have different strengths: raster data efficiently represents continuous spatial phenomena at uniform resolution, while vector data precisely describes discrete geographic features with arbitrary shapes and boundaries. Many Earth Observation applications require combining both data models, which is the core challenge underlying Zonal Statistics.

2.2 Zonal Statistics

A Zonal Statistics (ZS) operation computes aggregate statistics over raster pixel values within zones defined by vector geometries [7]. Figure 1 illustrates the process: given a set of vector features (e.g., land parcels) and a raster image (e.g., a vegetation index), the operation first joins the two datasets by determining which pixels fall within each vector feature. The join produces an intermediate result where each vector feature is associated with a set of pixel values. Finally, aggregation functions (e.g., count, mean, min, max) are applied per feature, producing a statistics table that can serve as input to a variety of downstream analysis use cases.

From a database perspective, ZS can be understood as a spatial join between a raster and a vector relation, followed by a group-by aggregation. However, unlike standard relational joins, the join predicate operates across heterogeneous data models: pixels occupy fixed grid cells while vector features are defined on a continuous coordinate space. This mismatch introduces ambiguity in the semantic meaning of pixels intersecting a feature and creates preprocessing requirements that do not arise in traditional benchmarks.

2.3 Spatial Data Alignment

Two technical concepts significantly affect ZS performance and are central to the benchmarking challenges we identify.

Coordinate Reference Systems (CRS). Every geospatial dataset is defined relative to a CRS, which specifies how geographic coordinates map to locations on Earth. Different datasets may use different CRSs (e.g., WGS 84 for global data, UTM zones for regional data). Prior to a spatial join, both datasets must exist in the same CRS. Otherwise, one dataset has to be *reprojected*, which is computationally expensive [23] and can introduce numerical inaccuracies, making the choice of *which* dataset to reproject non-trivial.

Clipping. Every spatial dataset has a spatial extent: the axis-aligned bounding box of its geographic coverage. Since ZS operates only within the overlapping area of both datasets, irrelevant data can be discarded early. For vector data, this involves a spatial filter that discards features outside the common extent. For raster data, the equivalent operation is clipping, which extracts the overlapping portion of the grid into a smaller raster. Clipping is more expensive than vector filtering because it usually requires decoding and rewriting pixel data, making the decision of when to apply it—during preprocessing or query time—crucial.

2.4 Spatial Systems Under Test

We select systems that represent three fundamentally different architectural approaches to resolving raster-vector heterogeneity: relational spatial databases (PostGIS), distributed dataflow systems (Beast), and array databases (RasDaMan). We also evaluated Apache Sedona [2] as an additional dataflow system, but excluded it from our analysis because its ZS support produced inconsistent results across configurations and was outperformed by the other systems.

PostGIS [1] is a spatial extension for PostgreSQL. It models both vector and raster data in relational tables and exposes an SQL-based query interface. Data must be ingested into internal tables before querying, incurring upfront cost but enabling efficient repeated queries through indexing. Raster operations in PostGIS are handled by GDAL [21], which is accessed via C-language UDFs.

Beast [16] is built on Apache Spark and represents the class of distributed spatial analytics systems that execute spatial workloads in dataflow engines. It implements the RaptorJoin algorithm [33] for native raster-vector joins without format conversion. Beast reads data directly from source files at query time using a Scala-based API, avoiding ingestion overhead. Internally, Beast uses the Java Topology Suite (JTS) library for spatial operations.

RasDaMan [3] is an array-based database optimized for multi-dimensional raster data. It provides a Web Coverage Processing Service (WCPS) query interface, requires vector data to be rasterized or inlined as Well-Known Text (WKT) coordinates before ZS queries can execute, and processes one vector feature per request. We include RasDaMan to represent the array-database paradigm, which is widely used for large raster archives in Earth Observation infrastructures. However, as we show in Section 4, its sequential per-feature processing model makes it unsuitable for ZS workloads with large vector datasets.

3 Challenges and Experimental Methodology

We identify four key challenges that arise when benchmarking operations across heterogeneous spatial data, where inputs do not share a common data model.

C1: API and Data Model Heterogeneity. The systems under test do not merely differ in query syntax. They adopt fundamentally different strategies for bridging the raster-vector divide. PostGIS normalizes both formats into relational tables, Beast preserves native formats and joins across them using its RaptorJoin algorithm [33], and RasDaMan requires vector data to be rasterized or inlined as WKT coordinates before querying. These are not interchangeable implementations of the same operation; they represent

distinct resolution strategies for the underlying data heterogeneity, each with different data preparation steps. Even the definition of the spatial join predicate can differ across systems if not explicitly controlled. For example, a pixel may belong to a zone if it intersects the boundary, or only if its center lies within it. A fair benchmark must therefore ensure that all systems execute semantically equivalent operations over consistently prepared, isolating system performance from modeling and implementation choices.

C2: Asymmetries Across Evaluation Phases. Operations over heterogeneous data introduce asymmetries across three distinct evaluation phases: *preprocessing* (e.g., format conversion, CRS alignment, and clipping), *ingestion* (loading data into the system's internal storage), and *query execution* (running the ZS query). Not all systems require all phases: PostGIS incurs upfront ingestion costs exceeding 50 minutes for large raster datasets, while Beast reads directly from source files and avoids ingestion entirely. Moreover, the costs of these phases are not independent: whether CRS reprojection is applied during preprocessing or deferred to execution time affects both preprocessing cost and query latency (cf. C3). A benchmark must therefore measure all three phases separately, so that users can interpret results in light of their use case.

C3: Tuning Parameter Sensitivity. ZS performance is highly sensitive to parameters that are often overlooked in benchmarks. We identify two critical tuning dimensions: *(i) CRS alignment strategy*: the choice of which dataset to reproject affects both cost and accuracy; *(ii) early filtering*: reducing the data volume before the join, either spatially (e.g., clipping the raster or discarding vector features outside the common extent, cf. Section 2) or by applying attribute-based predicates to the vector dataset. A benchmark that uses default configurations for all systems may inadvertently penalize systems whose defaults are suboptimal for the workload.

C4: Workload Sensitivity. Unlike traditional database benchmarks, where workload dimensions (e.g., table size, selectivity) belong to a single data model, ZS workloads vary across dimensions from two different data models. For example, geometry type and feature count are vector properties, while resolution and pixel count are raster properties. Yet, their *interaction* governs performance: as we show in Section 4, geometry type can reverse which system performs best because it triggers fundamentally different indexing behavior depending on the system's join strategy. This cross-model interaction means that varying one dimension at a time, as single-model benchmarks typically do, fails to capture the workload configurations that cause the largest performance gaps. A benchmark over heterogeneous data must therefore vary dimensions from both models jointly and evaluate enough combinations to expose these interactions.

We adopt the following methodology to address these challenges:

Unified Query Specification. To address C1, we use Raven [15], an open-source middleware that translates a single declarative ZS query specification into native code for each execution platform. Importantly, Raven operates at query translation time: it generates native code for each system (SQL for PostGIS, Scala for Beast, WCPS for RasDaMan), so it introduces no overhead during preprocessing, ingestion, or query execution. Raven controls for join semantics and ensures consistent data preparation across systems, so that observed performance differences reflect system capabilities rather than differences in how each system bridges the raster-vector divide. Both Raven [4] and all experiment scripts are publicly available [5], enabling full reproducibility of our experiments.

Separating Evaluation Phases. To address C2, we measure preprocessing, ingestion, and query execution separately. End-to-end measurements would obscure the cost of bridging heterogeneous data models, making it impossible to distinguish whether a system is slow because of expensive preprocessing, high ingestion overhead, or inefficient query execution. Reporting each phase independently allows users to interpret the results in light of their use case—for example, whether ingestion costs can be amortized across many queries.

Systematic Tuning Exploration. To address C3, we evaluate each system across multiple tuning configurations, including different CRS alignment strategies (e.g., reprojecting vector data to the raster's CRS versus the reverse) and filtering approaches (spatial clipping and attribute-based predicates). We report both default and tuned performance to quantify the gap between out-of-the-box and optimized configurations.

Diverse Workloads. To address C4, we design 25 queries (Table 2) over 10 raster and 7 vector datasets (Table 1). The queries vary dimensions from both data models jointly: geometry types and feature counts on the vector side, resolutions on the raster side, and spatial selectivity (the ratio of overlapping to total extent) across the two. Crucially, we include query pairs that isolate cross-model interactions—e.g., the same raster dataset joined with both Polygon and LineString geometries—so that the performance effects of these interactions are directly observable.

4 Experimental Results

We organize our evaluation around three key questions that arise from the challenges identified in Section 3: **RQ1:** How does system choice affect performance? **RQ2:** How sensitive is performance to tuning? **RQ3:** What workload characteristics drive performance?

In this section, we first describe our experimental setup, present the findings for each question, and conclude with implications for benchmark design.

Experimental Setup. All experiments were conducted on a single Intel Xeon E-2278G server with 128 GiB RAM and 2 TB SSD. This reflects common practice among domain scientists, who typically develop and test their analyses in single-node environments before scaling up. Furthermore, the single-node evaluation allows us to isolate architectural differences from the effects of the cluster configuration. Distributed systems like Beast may benefit from multi-node deployments that amortize Spark's startup overhead. However, in preliminary tests with up to 4 nodes, only a few queries saw gains while all incurred higher startup overhead. We evaluate PostGIS v3.3, Beast v0.9.5, and RasDaMan v10.0.5 in isolated Docker containers with default system-level settings. Each query is executed once cold, then we report the average of three warm runs. Both Raven and all experiment scripts, queries, and datasets are publicly available [4, 5].

Datasets. We use 10 raster and 7 vector real-world datasets (Table 1) spanning wildfire analysis, vegetation monitoring, elevation modeling, land use classification, and coastal hazard assessment. Vector

Table 1: Datasets used in the evaluation.

Name	Domain	Size	Features / Pixels	Geom / Resolution	CRS (EPSG)	Source
Vector Datasets (7)						
Land Usage (County)	Land use	94.8 MiB	78,819 feat.	Polygon	25833	[26]
Land Usage (State)	Land use	1.44 GiB	1,245,678 feat.	Polygon	25833	[26]
Wildfires EU	Wildfire	36.0 MiB	12,244 feat.	Polygon	3035	[18]
Wildfires USA	Wildfire	514.7 MiB	29,583 feat.	Polygon	4269	[35]
Coastline	Coastal hazard	237.6 MiB	130,686 feat.	LineString	3857	[38]
Train Routes US	Transportation	7.41 MiB	49 feat.	LineString	4326	[27]
Landslides	Geology	11.7 MiB	49,103 feat.	Polygon	32632	[20]
Raster Datasets (10)						
NDVI (coarse, sm)	Vegetation	115 MiB	5,490×5,490	∼1 km	32633	[19]
NDVI (coarse, lg)	Vegetation	432.7 MiB	19,411×11,686	∼1 km	4326	[19]
NDVI (fine, lg)	Vegetation	1.69 GiB	38,822×23,371	∼500 m	4326	[19]
MERIS LC	Land cover	315.5 MiB	129,600×64,800	∼300 m	4326	[17]
Air Density	Climate	6.99 GiB	144,263×55,933	∼1 km	4326	[10]
Wind Power Density	Climate	13.8 GiB	144,263×55,933	∼1 km	4326	[10]
Wind Speed	Climate	13.2 GiB	144,263×55,933	∼1 km	4326	[10]
US DEM	Elevation	4.63 GiB	53,101×23,401	1/3 arc-sec	4269	[34]
LU/LC North Italy	Land cover	193.1 MiB	72,000×36,000	10 m	4326	[37]
NLCD	Land cover	18.2 GiB	204,254×95,543	30 m	4326	[12]

Table 2: Query workload. Selectivity refers to the fraction of vector features retained after filtering (spatial or attribute-based).

Q	Vector Dataset	Raster Dataset	Geom	Selectivity	Remarks
1	Land Usage (County)	NDVI (coarse, sm)	Poly	0.15 (attr.)	Attribute filtering baseline
2	Land Usage (State)	NDVI (coarse, lg)	Poly	0.15 (attr.)	Effect of scaling vector size (cf. Q1)
3	Land Usage (State)	NDVI (fine, lg)	Poly	0.15 (attr.)	Effect of doubling raster resolution (cf. Q2)
4–7	Wildfires EU	MERIS LC	Poly	1.0	Isolates effect of aggregation group size
8–9	Wildfires USA	Air Density	Poly	0.04 / 1.0	Spatial selectivity pair
10–11	Wildfires USA	Wind Power Density	Poly	0.04 / 1.0	Spatial selectivity pair; larger raster (cf. Q8–9)
12–13	Wildfires USA	Wind Speed	Poly	0.04 / 1.0	Spatial selectivity pair; same raster scale as Q10–11
14–16	Wildfires EU	Wind Speed / Wind Power / Air Density	Poly	1.0	Effect of vector scale on same rasters (cf. Q8–13)
17	Train Routes US	US DEM	LS	1.0	Sparse and long LineStrings (49 features)
18	Landslides	LU/LC North Italy	Poly	1.0	Regional-scale raster–vector overlap
19	Wildfires USA	MERIS LC	Poly	1.0	Cross-reference with Q4–7 (same raster, different vector)
20	Coastline	Wind Power Density	LS	1.0	Dense LineStrings (130K features); flash index stress test
21	Coastline	Wind Speed	LS	1.0	Same vector as Q20, different raster
22	Train Routes US	NLCD	LS	1.0	Sparse LineStrings against largest raster
23	Coastline	Air Density	LS	1.0	Same vector as Q20–21, different raster
24	Wildfires USA	NLCD	Poly	1.0	Largest raster–vector combination (Polygon)
25	Wildfires USA	US DEM	Poly	1.0	CRS alignment sensitivity target (cf. Section 4.2)

datasets range from 49 features (train routes) to over 1.2 million features (land use parcels), cover both Polygon and LineString geometry types, and are sourced from OpenStreetMap [27], governmental cadastral data [26], and domain-specific archives [18, 20, 35, 38]. Raster datasets span from 115 MiB regional images to 18.2 GiB global-scale grids and are drawn from satellite programs and global climate models [10, 12, 17, 19, 34, 37]. This diversity in size, geometry type, resolution, and spatial coverage ensures that the workload is representative of real-world ZS scenarios.

Query Workload. We design 25 queries (Table 2) to systematically vary dimensions across both data models. On the vector side, queries vary in geometry type (Polygon vs. LineString), feature count, and selectivity (both spatial and attribute-based). On the raster side, queries vary in resolution and spatial coverage. Queries 4–7 incrementally increase aggregation group sizes over the same datasets to isolate the effect of aggregation complexity. Queries 8–13 form three selectivity pairs, joining the same large Wildfire USA polygon dataset with different global rasters at 0.04 and 1.0 spatial selectivity, isolating the effect of early spatial filtering. Queries 14–16 join the same three global rasters with a smaller EU wildfire vector dataset, enabling a direct comparison of vector-scale effects. The LineString queries (17, 20–23) pair both sparse (49 features) and dense (130K features) LineString datasets against large rasters, directly exposing the cross-model interaction between geometry type and system join strategy. Due to space constraints, we present a representative subset of our full experimental evaluation.

4.1 How Does System Choice Affect Performance?

Figure 2 shows execution times for all 25 queries. PostGIS and Beast trade the lead depending on workload, with performance gaps of 6–22×, while RasDaMan's sequential per-feature processing causes most queries to exceed the 1000-second timeout. PostGIS is generally faster for smaller datasets and moderate-complexity geometries, but Beast outperforms it for large polygon-based vector datasets (e.g., wildfire polygons with 29K features), where its RaptorJoin algorithm efficiently handles raster-vector spatial joins. In contrast,

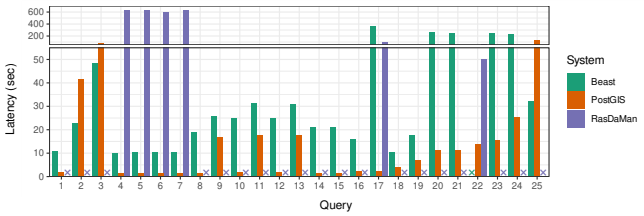


Figure 2: ZS execution latency across 25 queries. RasDaMan exceeds the 1000s timeout on multiple queries (✗).

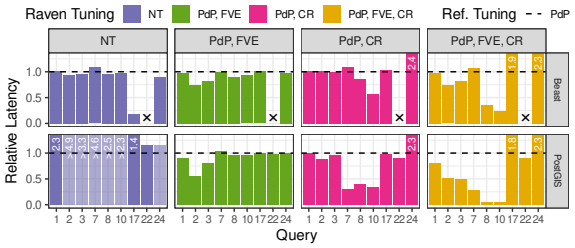


Figure 3: Cumulative benefit of tuning strategies for a representative query subset. NT: No Tuning, PdP: Projection Pushdown, FVE: Filter Vector Early, CR: Clip Raster. Baseline: PdP. Transparent: lower bounds on timeouts (>1000 sec).

Beast struggles with LineString geometries, exhibiting significantly higher latency, which we analyze further in Section 4.3.

When ingestion costs are included, the picture changes. PostGIS's ingestion step can exceed 50 minutes for large raster datasets, whereas Beast avoids ingestion entirely, making Beast faster for single executions. These loading overheads are consistent with prior analyses of geospatial data loading, which show that the choice of file format and system materially affects end-to-end performance [36]. However, PostGIS's per-query execution cost is lower for most queries, so it becomes preferred after enough repeated executions. The crossover point is workload-dependent: for most polygon-based queries, PostGIS needs 20–40 executions to amortize its ingestion cost, while for LineString queries, where Beast's per-query cost is disproportionately high (cf. Section 4.3), PostGIS outperforms it after just 1–3 runs.

Finding 1: *Whether a system with upfront ingestion cost outperforms one that reads directly from files at query time depends on both data characteristics and expected query frequency. The amortization crossover point is itself workload-dependent: geometry type alone shifts it by an order of magnitude (1–3 vs. 20–40 repeated executions).*

4.2 How Sensitive Is Performance to Tuning?

We evaluate three tuning strategies: (1) projection pushdown (PdP), which applies CRS reprojection during preprocessing rather than at execution time, (2) vector filter pushdown (FVE), which eliminates irrelevant vector features before the join, and (3) raster clipping (CR), which trims the raster to the vector dataset's spatial extent. Given RasDaMan's exclusion from most queries due to timeouts (cf. Section 4.1), this analysis focuses on PostGIS and Beast.

Tuning strategies compound non-linearly. Figure 3 illustrates the cumulative benefit of combining tuning strategies for a representative subset of queries. Applying all three strategies together in PostGIS yields speedups of 3-16×, in some cases over 55×, far more than the sum of the individual improvements would suggest. For example, PostGIS times out on Query 10 under no tuning (800 seconds preprocess + ingestion, over 1000 seconds execution) but completes in 32.5 seconds with all three strategies combined, with the largest single gain already visible when CRS reprojection is moved from execution time to preprocessing. This is because the strategies address different bottlenecks that interact: CRS pushdown reduces per-tile computation, while filtering and clipping reduce the number of features and tiles that need to be loaded. The interaction is strongest when the raster is much larger than the vector extent and the vector dataset has highly selective predicates.

CRS alignment is the dominant tuning factor for PostGIS. This is perhaps our most practically relevant finding. As the Query 10 example above illustrates, the largest single gain for PostGIS comes from pushing CRS reprojection into a preprocessing step rather than deferring it to execution time. This has a far larger impact on performance than filtering alone, yet it is the parameter least discussed in existing spatial benchmarking work. The reason is that PostGIS's internal raster data structure does not support on-the-fly CRS transformations efficiently: when reprojection is deferred to execution time, each tile must be individually reprojected during the join, which is done by first deserializing, then calculating reprojection parameters and transforming with a call to GDAL, and finally serializing the tile again. Preprocessing avoids this overhead by aligning the CRS once before ingestion, reducing the transformation calls to one per file instead of thousands of calls for each tile. Beast, by contrast, shows negligible sensitivity to CRS strategy, likely because its architecture and Spark's distributed execution model absorb the reprojection cost more gracefully.

Finding 2: *Tuning sensitivity is system-specific. PostGIS benefits dramatically from tuning (3-16×, in some cases over 55×). Beast is relatively robust. Reporting only default-configuration results would significantly understate PostGIS performance.*

Finding 3: *CRS alignment strategy has the largest impact on PostGIS performance, yet it is a dimension that existing benchmarks overlook.*

4.3 What Workload Characteristics Drive Performance?

Geometry type causes order-of-magnitude differences within the same system. Beast exhibits significantly higher latency when the vector geometry is of type LineString compared to Polygon, even at similar data volume. This is an architectural effect: Beast's RaptorJoin uses a flash index that creates entries for the start and end pixels of each geometry edge. LineStrings, being open-ended, generate proportionally more index entries per unit of spatial extent than closed Polygons, leading to a larger flash index and more expensive spatial intersection checks.

We apply the Douglas-Peucker simplification algorithm [13] to progressively reduce the vertex count and analyze its effect on latency. As shown in Figure 4, Beast's behavior depends on both geometry type and vertex count. For dense LineString datasets

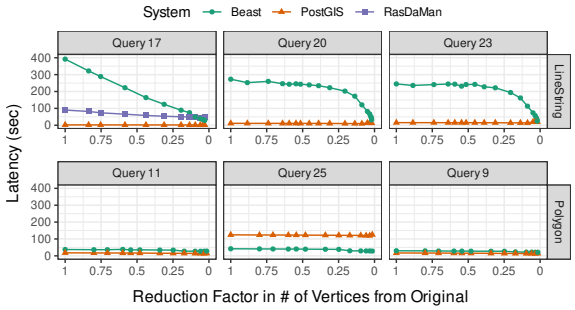


Figure 4: Latency as vertex count is reduced via simplification. Top: LineString queries; Bottom: Polygon queries. Beast's latency is highly sensitive to LineString complexity, while PostGIS scales consistently.

(Queries 20 and 23, with over 14M vertices), Beast's latency remains nearly constant until a reduction factor of 0.33, because Douglas-Peucker initially removes only vertices whose removal preserves the edge structure. Beyond this point, edges are eliminated, Beast's flash index shrinks, and latency drops sharply. For sparse LineStrings (Query 17, with 456K vertices), the effect is less pronounced, as reducing the vertex count has a more linear effect on the flash index size. PostGIS, in contrast, shows near-constant latency across geometry types, indicating that this sensitivity is specific to Beast's indexing strategy. RasDaMan completes only for Query 17, where it exhibits similar near-constant latency.

Reducing the vector resolution has little impact on the result values themselves, as due to spatial autocorrelation [8] we expect nearby pixels to be similar to each other. Additionally, the simplification step tries to preserve the original shape of the geometry as best as possible. However, simplification causes the number of pixels covered to decrease on average to 25 % with LineStrings and 98 % with Polygons, increasing the risk of missing or including outliers, which may be most present at feature boundaries.

The raster-to-vector size ratio influences which tuning strategy is most effective. When the raster extent far exceeds the vector extent, raster clipping provides the largest gain by reducing the raster size before the join (e.g., Queries 8 and 10, where global rasters are joined with regional wildfire polygons). When selective predicates can reduce the vector dataset, vector filter pushdown is most effective (e.g., Queries 1–3, where filtering retains only 15% of features). When neither side can be significantly reduced (e.g., spatial extents are similar and no selective predicates apply), the CRS alignment strategy becomes a key tuning factor (e.g., Query 25).

Finding 4: *Geometry type (Polygon vs. LineString) causes order-of-magnitude performance differences and determines which system performs best, making it a first-class workload dimension that benchmarks over heterogeneous data must account for.*

Finding 5: *The most effective tuning strategy depends on the ratio of raster to vector data participating in the join: raster clipping yields the largest gains when the raster side dominates, vector filter pushdown when selective predicates can reduce the vector side, and CRS alignment when neither side can be significantly reduced.*

4.4 Implications for Benchmark Design

Our results highlight several factors that should be addressed when designing benchmarks for spatial operations over heterogeneous data. First, benchmarks should report both default and tuned configurations, since tuning yields order-of-magnitude differences that are highly system-specific (Finding 2). Second, CRS alignment strategies must be treated as a tuning dimension, since reprojection decisions can dominate query performance yet are largely overlooked by existing spatial benchmarks (Finding 3). Third, workloads should include different geometry types (e.g., Polygon and LineString), as the geometry type alone can determine which system performs best (Finding 4). Fourth, workloads should vary the raster-to-vector extent ratio and the selectivity of vector predicates, as these factors influence which tuning strategy is most effective (Finding 5).

5 Related Work

The challenge of efficiently joining raster and vector data has been recognized since the 1980s [29], yet systematic evaluation of how spatial systems handle this cross-model operation remains limited. Raven [15] is an open-source middleware that translates a single declarative ZS query specification into native code for each execution platform, ensuring semantically equivalent query execution across systems. In this paper, we leverage Raven to control for differences in data preparation and join semantics, and focus our contribution on the experimental design and the benchmarking findings derived from it. In this regard, the closest related work is by Singla et al. [32], who benchmark ZS queries using Beast and RaptorJoin and compare seven systems under default configurations. Their evaluation does not include PostGIS, which we find competitive in single-node setups. More importantly, their study uses only Polygon geometries and does not vary tuning parameters or CRS alignment strategy, factors our experiments show can significantly impact performance. Other spatial benchmarks [9, 22, 24, 28, 31] focus on either raster or vector operations in isolation, leaving out cross-model operations. Complementary work has characterized the cost of geospatial data loading itself [36], which our results confirm is critical for interpreting end-to-end system performance.

6 Conclusion

We presented an experimental study of spatial operations over heterogeneous data, using Zonal Statistics as a representative case. Evaluating relational, dataflow, and array-based spatial systems across 25 queries and 17 real-world datasets, we find that PostGIS and Beast exhibit performance gaps of up to 22× driven by workload characteristics, while RasDaMan's sequential processing model makes it unsuitable for most ZS workloads. Tuning alone yields over 55× speedup within a single system. Our findings show that previously largely overlooked factors, namely geometry type, CRS alignment strategy, and the raster-to-vector size ratio, are among the strongest determinants of performance. In future work, we plan to leverage these insights to develop a standardized benchmark specification for spatial operations over heterogeneous data, building on the datasets, queries, and methodology presented in this paper. We also plan to investigate automated tuning and system selection based on workload characteristics, building on our ongoing work on agent-based automation of EO analytics pipelines [14].

References

[1] 2023. PostGIS. https://postgis.net/.
[2] 2025. Apache Sedona. https://sedona.apache.org/.
[3] 2025. RasDaMan. https://rasdaman.org/.
[4] 2026. RaVeN. https://github.com/polydbms/RaVeN.
[5] 2026. RaVeN Recipes. https://github.com/polydbms/RaVeN-recipes.
[6] A. K. Aksoy, P. Dushev, E. Tzirita Zacharatou, H. Hemsen, M. Charfuelan, J.-A. Quiané-Ruiz, B. Demir, and V. Markl. 2022. Satellite Image Search in AgoraEO. *PVLDB* 15, 12 (2022), 3646–3649.
[7] Gilberto Camara, Luiz Fernando Assis, Gilberto Ribeiro, Karine Reis Ferreira, Eduardo Llapa, Lubia Vinhas, Victor Maus, Alber Sanchez, and Ricardo Cartaxo. 2016. Big earth observation data analytics: matching requirements to system architectures. In *BigSpatial*. 1–6.
[8] Andrew D. Cliff and Keith Ord. 1970. Spatial Autocorrelation: A Review of Existing and New Measures with Applications. *Economic Geography* 46 (1970), 269–292.
[9] Philippe Cudré-Mauroux, Hideaki Kimura, Kian-Tat Lim, Jennie Rogers, Samuel Madden, Michael Stonebraker, Stan Zdonik, and Paul Brown. 2010. SS-DB: A Standard Science DBMS Benchmark. In *XLDB*. 11.
[10] N. N. Davis et al. 2023. The Global Wind Atlas: A High-Resolution Dataset of Climatologies. *Bull. Amer. Meteor. Soc.* 104, 8 (2023).
[11] A. de Wall, B. Deiseroth, E. Tzirita Zacharatou, J.-A. Quiané-Ruiz, B. Demir, and V. Markl. 2021. Agora-EO: A Unified Ecosystem for Earth Observation – A Vision for Boosting EO Data Literacy. In *Proc. Big Data from Space (BiDS)*.
[12] J. A. Dewitz and U.S. Geological Survey. 2021. National Land Cover Database (NLCD) 2019 Products. https://doi.org/10.5066/P9KZCM54.
[13] D. H. Douglas and T. K. Peucker. 1973. Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature. *Cartographica* 10, 2 (1973).
[14] G. Dusella, H. Gavriilidis, B. Chen, B. Demir, V. Markl, and E. Tzirita Zacharatou. 2025. Automating Earth Observation Analytics Pipelines with Agent Raven. In *Proc. Big Data from Space (BiDS)*.
[15] G. Dusella, H. Gavriilidis, L. Nuhu, V. Markl, and E. Tzirita Zacharatou. 2024. Multi-Backend Zonal Statistics Execution with Raven. In *SIGMOD Companion*. 532–535.
[16] A. Eldawy et al. 2021. Beast: Scalable Exploratory Analytics on Spatio-temporal Data. In *CIKM*.
[17] ESA Climate Change Initiative. 2019. Climate Research Data Package. https://maps.elie.ucl.ac.be/CCI/viewer/.
[18] European Forest Fire Information System. 2024. Burnt Area Mapped using Sentinel-2/MODIS Images. https://forest-fire.emergency.copernicus.eu/.
[19] European Space Agency. 2024. Copernicus Data Space Ecosystem. https://dataspace.copernicus.eu/.
[20] M. F. Ferrario. 2024. Inventory of Landslides Triggered by Heavy Rainfall in the Emilia-Romagna Region (Italy) in May 2023. https://doi.org/10.5281/zenodo.13234762.
[21] GDAL/OGR contributors. 2026. *GDAL/OGR Geospatial Data Abstraction software Library* (3.12.3 ed.). Open Source Geospatial Foundation. https://doi.org/10.5281/zenodo.5884351
[22] D. Haynes, P. Mitchell, and E. Shook. 2020. Developing the Raster Big Data Benchmark: A Comparison of Raster Analysis on Big Data Platforms. *ISPRS Int. J. Geo-Inf.* 9, 11 (2020).
[23] V. Janssen. 2009. Understanding Coordinate Reference Systems, Datums and Transformations. *Int. J. of Geoinformatics* 5, 4 (2009).
[24] B. Kim, K. Koo, U. Enkhbat, S. Kim, J. Kim, and B. Moon. 2022. M2Bench: A Database Benchmark for Multi-Model Analytic Workloads. *PVLDB* 16, 4 (2022), 747–759.
[25] L. Kondmann, A. Toker, M. Rußwurm, A. Camero, D. Peressuti, G. Milcinski, P.-P. Mathieu, N. Longépé, T. Davis, and G. Marchisio. 2021. DENETHOR: The DynamicEarthNET Dataset for Harmonized, Inter-Operable, Analysis-Ready, Daily Crop Monitoring from Space. In *NeurIPS Track on Datasets and Benchmarks*.
[26] Landesvermessung und Geobasisinformation Brandenburg. 2024. ALKIS-Vektordaten. https://data.geobasis-bb.de/geobasis/daten/alkis/.
[27] OpenStreetMap contributors. 2024. https://www.openstreetmap.org.
[28] V. Pandey, A. Kipf, T. Neumann, and A. Kemper. 2018. How Good Are Modern Spatial Analytics Systems? *PVLDB* 11, 11 (2018).
[29] D. J. Peuquet. 1983. A Hybrid Structure for the Storage and Manipulation of Very Large Spatial Data Sets. *Computer Vision, Graphics, and Image Processing* 24, 1 (1983), 14–27.
[30] N. A. Povak, P. F. Hessburg, and R. B. Salter. 2018. Evidence for Scale-Dependent Topographic Controls on Wildfire Spread. *Ecosphere* 9, 10 (2018).
[31] S. Ray, B. Simion, and A. D. Brown. 2011. Jackpine: A Benchmark to Evaluate Spatial Database Performance. In *ICDE*. 1139–1150.
[32] S. Singla, A. Eldawy, T. Diao, A. Mukhopadhyay, and E. Scudiero. 2021. Experimental Study of Big Raster and Vector Database Systems. In *ICDE*. 2243–2248.
[33] S. Singla, A. Eldawy, T. Diao, A. Mukhopadhyay, and E. Scudiero. 2021. The Raptor Join Operator for Processing Big Raster + Vector Data. In *SIGSPATIAL*. 324–335.
[34] United States Geological Survey. 2023. 3D Elevation Program. https://www.usgs.gov/the-national-map-data-delivery.
[35] U.S. Department of Agriculture Forest Service. 2024. MTBS Burn Area Boundary. https://data.fs.usda.gov/geodata/edw/.
[36] A. Wachs and E. Tzirita Zacharatou. 2024. Analysis of Geospatial Data Loading. In *DBTest@SIGMOD*. 36–42.
[37] D. Zanaga et al. 2022. ESA WorldCover 10 m 2021 V200. https://doi.org/10.5281/zenodo.7254221.
[38] L. Zhang, J. Zuo, and B. Chen. 2024. GCL_FCS30: A Global Coastline Dataset with 30-m Resolution. *Int. Research Center of Big Data for Sustainable Development Goals* (2024).