Multi-Backend Zonal Statistics Execution with Raven

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

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

Laert Nuhu*
Deutsche Kreditbank AG
Germany
laertnuhu@gmail.com

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

Eleni Tzirita Zacharatou
IT University of Copenhagen
Denmark
elza@itu.dk

ABSTRACT

The recent explosion in the number and size of spatial remote sensing datasets from satellite missions creates new opportunities for data-driven approaches in domains such as climate change monitoring and disaster management. These approaches typically involve a feature engineering step that summarizes remote sensing pixel data located within zones of interest defined by another spatial dataset, an operation called zonal statistics. Although several spatial systems support zonal statistics operations, they differ significantly in terms of interfaces, architectures, and algorithms, making it hard for users to select the best system for a specific workload. To address this limitation, we propose *Raven*, a zonal statistics framework that provides users with a unified interface across multiple execution backends, while facilitating easy benchmarking and comparisons across systems. This demonstration showcases *Raven*'s multi-backend execution environment, domain-specific declarative language, optimization techniques, and benchmarking capabilities.

CCS CONCEPTS

• **Information systems** → **Spatial-temporal systems**; • **Applied computing** → *Earth and atmospheric sciences*.

KEYWORDS

unified spatial data analytics; zonal statistics; parcel-based classification; spatial join; satellite imagery; big spatial data

ACM Reference Format:
Gereon Dusella, Haralampos Gavriilidis, Laert Nuhu, Volker Markl, and Eleni Tzirita Zacharatou. 2024. Multi-Backend Zonal Statistics Execution with Raven. In *Companion of the 2024 International Conference on Management of Data (SIGMOD-Companion '24), June 9–15, 2024, Santiago, AA, Chile*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3626246.3654730

1 INTRODUCTION

Over the past decade, the launch of an ever-increasing number of satellites has led to the accumulation of unprecedented volumes of Earth Observation (EO) data [1, 6, 8]. For example, the Sentinel archive alone contains Earth images captured by eight satellites, amounting to 6.64 petabytes [6]. The efficient processing of EO data offers an opportunity to substantially improve our understanding of our planet's state and the changes that occur on it [2, 11, 15, 17].

To extract meaningful features from EO imagery that can be used to train ML models, it is often necessary to compute aggregate information for image pixels within specific zones of interest defined by another spatial dataset [7], a process commonly known as zonal statistics (ZS). Remote sensing images are available in raster format, a multidimensional array representation where each pixel corresponds to a geographical region, while the pixel value reflects some characteristics of that region. However, spatial datasets defining zones of interest, like city boundaries from OpenStreetMap [14], are often in vector format, representing geographical features with points, lines, and polygons. As a result, computing zonal statistics requires combining heterogeneous raster and vector datasets. For example, to train an ML model for monitoring and predicting changes in vegetation health in different land plots over time, one needs to generate aggregated (e.g., mean and median) Normalized Difference Vegetation Index (NDVI) statistics as features for each land plot [10]. This feature engineering process requires joining remote sensing imagery data (raster) with land plot data (vector).

Current spatial systems for zonal statistics confront users with a jungle of interfaces, capabilities, and requirements. This plethora of different systems poses challenges in selecting the best system for a given workload. First, for systems lacking support for both raster and vector data, users need to perform additional pre-processing steps, i.e., rasterizing vector datasets or vectorizing raster datasets. Furthermore, they might need to perform file format conversions, given that most systems support only a limited number of file formats. The diversity of interfaces across spatial systems poses further challenges. First, it locks users into their initial system choice due to the significant effort required to rewrite applications. Second, it introduces a substantial barrier when testing different systems for optimal performance. For example, while both Beast [9] and PostGIS [16] support joining raster and vector data, Beast employs a map-reduce-like API, while PostGIS offers an SQL-like API. To provide a good user experience, avoid vendor lock-in, and optimize performance, there is a need to *abstract* ZS operations and enable their *unified execution across multiple spatial systems*.

To address the challenges in processing zonal statistics over large-scale heterogeneous datasets, we developed *Raven*, a zonal statistics framework that offers users a unified interface across multiple

*Work performed while at Technische Universität Berlin

SIGMOD-Companion '24, June 9–15, 2024, Santiago, AA, Chile

ACM ISBN 979-8-4007-0422-2/24/06
https://doi.org/10.1145/3626246.3654730

spatial systems serving as execution backends.[1] Raven exposes a DSL tailored for zonal statistics, abstracts system-specific details, and optimizes execution. Furthermore, it supports effortless system benchmarking, assisting users in selecting the most efficient system for their workload. To the best of our knowledge, Raven is the first system for *unified spatial analytics*. Previous efforts to unify data analytics focus on integrating structured and semi-structured data with SQL [5, 12, 19] and map-reduce-like interfaces [4]; however, these efforts do not provide support for spatial operations.

In this demonstration, we first aim to illustrate the complexity of implementing zonal statistics in different spatial systems. We let the audience interact with the systems and showcase their diversity in terms of interfaces, capabilities, and requirements. We then dive into Raven's internals, enabling the audience to implement a zonal statistics task within our tool. We discuss how Raven translates this task into different system APIs, performs necessary pre-processing, and manages the execution lifecycle. Furthermore, we highlight how Raven guides users in selecting the best system for their task by executing this task on multiple state-of-the-art spatial systems and generating performance metrics that offer insights into performance variations among these systems. Finally, we highlight the benefits of Raven by implementing an exemplary application and letting users manipulate different parameters and datasets interactively.

2 RAVEN OVERVIEW

Today's data scientists face multiple challenges when implementing zonal statistics, due to the varying interfaces and configuration parameters exposed by today's spatial systems, the varying pre-processing steps that these systems require, and their divergent runtime performance capabilities. In response to these challenges, Raven aims to 1) offer an easy-to-use zonal statistics interface and 2) highlight performance differences in spatial systems. To achieve this, Raven exposes a declarative zonal statistics interface based on a DSL that we developed. Using this DSL, Raven can transparently optimize and execute a given zonal statistics task on multiple spatial systems. As a result, Raven provides system independence, thereby helping users avoid vendor lock-ins. Furthermore, by automating execution and providing detailed performance results, Raven simplifies selecting the most efficient system for a given workload. In the following, we give a brief overview of Raven's components.

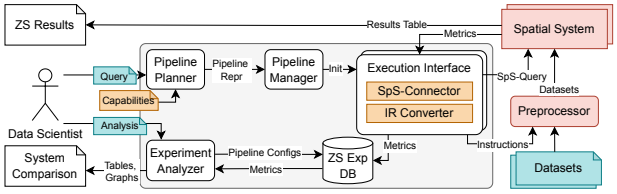

Figure 1: Raven Architecture

2.1 Architecture Overview

Figure 1 presents Raven's architecture. Raven takes as input a zonal statistics task expressed in its DSL (the query) and relies on its `Pipeline Planner` for optimization. Additionally, the Pipeline Planner takes as input a "System Capabilities" file, specifying the operations supported by the execution backends. Based on this information, it determines the need for pre-processing steps, such as format or Coordinate Reference System (CRS) conversions, and selects the appropriate join type. The Pipeline Planner outputs an Abstract Syntax Tree (AST) including all required operations, from pre-processing to joining and aggregation. Then, the `Pipeline Manager` is responsible for assembling and executing the pipeline. Here, Raven relies on (system-developer-provided) implementations of the `Execution Interface`, which includes a `IR` (Internal Representation) `Converter` and a `SpS` (Spatial System) `Connector`. The IR Converter translates Raven's AST into system-specific code using parameterized templates, and the SpS-Connector enables execution on the underlying systems and retrieving the results. Raven stores execution metrics, e.g., runtime and resource consumption for each step, in its experiment database, which the `Experiment Analyzer` uses to gain insights into the execution.

```
1  # Datasets definition
2  zs_result = ZSGen.build(
3          raster="/data/sentinel2a_mol_band9",
4          vector="/data/ALKIS_bezirk_MOL")
5  # Aggregation operations
6    .group("oid")
7    .summarize({"max": ZSGen.MAX, "avg": ZSGen.AVG})
8    .join_using(ZSGen.INTERSECT)
9  # Systems
10   .system([ZSSystem.PostGIS(params),...])
```

Listing 1: A simple ZS Task in Raven's DSL

2.2 Raven's Domain-Specific Language

Performing zonal statistics on raw raster and vector data involves multiple steps. First, data require pre-processing to handle variations in format, Coordinate Reference Systems (CRSs), and standards governing geometry representation and interpretation. Second, data might require filtering based on specified conditions. The next processing stage involves joining and aggregating the data. In this stage, one can apply various interpretations for the join condition and implement optimizations, such as tuning tile sizes.[2] To abstract these steps, we designed a simple DSL for Raven, allowing users to easily express zonal statistics on both raster and vector datasets. The DSL exposes primitives for zonal statistics computation, such as defining transformations, filter predicates, and join conditions. Listing 1 shows an example. Here, the user loads a raster and a vector dataset (Lines 2–4), selects a grouping key, two aggregate functions, and a join method (Lines 6–8), and chooses PostGIS for execution (Line 10). Note that, for brevity, we do not show DSL primitives related to other parameters and spatial systems.

Subsequently, Raven converts programs expressed in its DSL to system-specific implementations, optimizes them, and executes them across the user-specified spatial systems as described next.

2.3 Zonal Statistics Pipelines

The AST generated by Raven's `Pipeline Planner` (cf. Figure 1) encapsulates the end-to-end processing of a zonal statistics task. This

[1] Raven is open-source, available at: https://github.com/polydbms/RaVeN

[2] Tiles are used to divide raster data into smaller chunks.

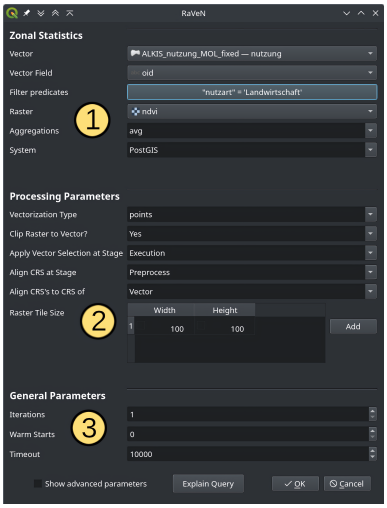

Figure 2: Raven's Config. Panel

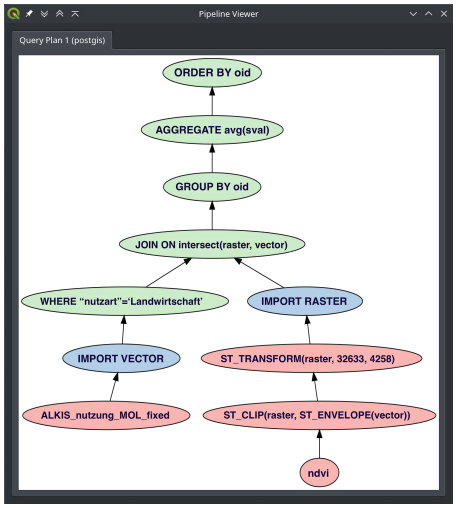

Figure 3: Zonal Statistics Pipeline Viewer

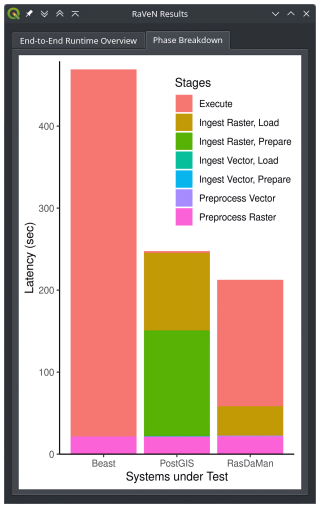

Figure 4: Benchmark Results

includes pre-processing operations, such as changing format to support loading into the given system, aligning CRSs, and filtering the datasets, as well as performing the join and aggregation. Raven currently optimizes the execution plan using simple heuristics, such as reducing redundant data loading by filter pushdown.

The `Pipeline Manager` (cf. Figure 1) transforms the AST into system-specific code. To achieve this, it employs parameterized templates in the APIs of the supported backend systems through the `IR Converter`. System developers need to provide these templates when integrating a system into Raven. Finally, Raven executes the system-specific code through the `SpS-Connector`. In our reference implementation, we support pre-processing using GDAL and zonal statistics execution on Beast [9], PostGIS [16], RasDaMan [3], heavy.ai [13], and Sedona [18]. The Pipeline Manager composes the pipeline by filling the templates with information from the AST, and coordinates the execution.

2.4 Benchmarking Mode

The performance of zonal statistics tasks in different spatial systems can vary significantly depending on data and workload. In addition to facilitating the seamless execution of zonal statistics across multiple systems with diverse configurations, Raven also allows users to benchmark these systems. To facilitate benchmarking, Raven features a dedicated benchmarking mode. This mode allows users to execute multiple pipelines and produce detailed performance plots, e.g., breakdown performance of different pipeline stages. These plots enable Raven's users to compare different systems and parameter combinations. As a result, users can gain insights into potential bottlenecks and enhance system performance by fine-tuning available parameters. Overall, Raven's integrated benchmarking component provides valuable tools for optimizing zonal statistics tasks across diverse spatial systems.

2.5 Integration with QGIS

To enhance the interactivity of our demonstration, we seamlessly integrated Raven into QGIS [20]. This integration enables users to visually formulate a query through a user-friendly UI, which is then translated into Raven's IR AST. The UI automatically pulls information about the loaded data (or layers) from QGIS. Furthermore, users can specify other processing parameters, such as the type of vectorization applied. When a user selects multiple systems or multiple conflicting processing parameters (e.g., different tile sizes), Raven automatically switches to benchmarking mode. The benchmarking results are displayed in the main QGIS UI after Raven concludes its benchmark run.

In addition to the UI, we integrated Raven into the Processing API of QGIS. This API empowers users to build complex pipelines with multiple inputs, outputs, and parameters. Consequently, users can harness Raven through QGIS to tackle more complex problems and execute recurring tasks effortlessly.

3 DEMONSTRATION PLAN

The goal of this demonstration is twofold. First, it aims to show the intricacy of implementing zonal statistics using state-of-the-art spatial data management systems. Second, it aims to showcase the capabilities of Raven. Specifically, we show its ease of use for expressing zonal statistics and its practical utility for spatial applications. Furthermore, we show how Raven assists data scientists in selecting the most efficient spatial system for a given task, leveraging its benchmarking mode. Attendees can experience a live demo of Raven using the QGIS UI. The UI runs on a local laptop, where Raven can be run directly. For scenarios involving large datasets, the laptop connects to a remote server hosting Raven. In the following, we describe our demonstration scenarios.

3.1 Exploring State-of-the-Art Systems

First, we introduce the audience to the fundamental characteristics of geospatial data, emphasizing the differences between raster and vector data. Then, we describe the task of zonal statistics and explain how raster data can be combined with vector data. Finally, we show how users implement zonal statistics tasks in state-of-the-art

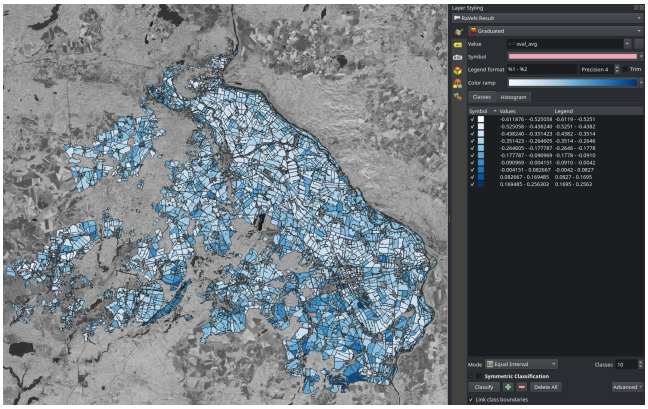

Figure 5: Zonal Statistics Output

systems by presenting an example task, e.g., computing NDVI statistics, and demonstrating its implementation using the interfaces of various systems, e.g., Beast, PostGIS, and RasDaMan.

3.2 Implementing Zonal Statistics with Raven

Our second demonstration scenario aims to familiarize the audience with Raven. For this purpose, we guide the audience through Raven's DSL and show its ease of use for implementing zonal statistics tasks. Leveraging the integration of Raven with QGIS (cf. Section 2.5), users can seamlessly compose zonal statistics tasks in Raven's DSL through a GUI, as shown in Figure 2. In (1), users can choose among available raster and vector datasets and apply filter predicates. In (2), users can specify values for different processing and optimization parameters. Then, in (3), users can define evaluation parameters mostly relevant for Raven's benchmarking mode. In this scenario, we also discuss how Raven abstracts the APIs of different spatial systems, translates zonal statistics tasks into these APIs, performs pre-processing of raw raster and vector data, and efficiently orchestrates the execution lifecycle. To that end, we use the `ZS Pipeline Viewer` (c.f. Figure 3) that visualizes the generated query plan for each system based on Raven's IR. The visualization uses different colors to highlight different types of operations, i.e., pre-processing, ingestion, and actual execution.

3.3 Benchmarking Zonal Statistics with Raven

In our third demonstration scenario, we aim to illustrate the performance discrepancies among state-of-the-art spatial data management systems when executing a zonal statistics task. Therefore, we leverage Raven's benchmarking mode to execute a given task across multiple systems and explore different configurations. Throughout the entire execution lifecycle, Raven gathers statistics that provide valuable information on each processing stage: from format conversions to CRS alignment, data filtering, and finally joining raster and vector data. To show the performance characteristics of different systems, we run pre-defined zonal statistics tasks and discuss the performance breakdown graphs generated by Raven (cf. Figure 4). Additionally, users can define their own zonal statistics tasks and benchmarking parameters using Raven's configuration panel, thereby gaining a better understanding of how different settings influence the performance of different systems. Overall, in this scenario, we demonstrate Raven's versatility in benchmarking spatial data management systems and emphasize its utility in guiding users to select the right system for their needs.

3.4 Raven as a Spatial Application Backend

In our fourth demonstration scenario, we want to show how one can integrate Raven with current spatial applications. For this purpose, we describe our integration with QGIS. This integration allows computing zonal statistics on any available underlying spatial system and then returning the result to QGIS to produce different data visualizations. In this demonstration scenario, attendees can actively interact with Raven through the GUI and visually explore raster and vector data, along with the produced zonal statistics (cf. Figure 5). We will supply raster and vector datasets, but we also encourage attendees to bring their datasets that we can explore with Raven.

ACKNOWLEDGEMENTS

We gratefully acknowledge funding from the German Federal Ministry of Education and Research under the grants BIFOLD24B and 01IS17052 (as part of the Software Campus project PolyDB).

REFERENCES

[1] European Space Agency. 2024. Copernicus Data Space Ecosystem. https://dataspace.copernicus.eu/
[2] Ahmet Kerem Aksoy, Pavel Dushev, Eleni Tzirita Zacharatou, Holmer Hemsen, Marcela Charfuelan, Jorge-Arnulfo Quiané-Ruiz, Begüm Demir, and Volker Markl. 2022. Satellite image search in AgoraEO. *PVLDB* 15, 12 (2022), 3646–3649.
[3] Peter Baumann, Andreas Dehmel, Paula Furtado, Roland Ritsch, and Norbert Widmann. 1998. The multidimensional database system RasDaMan. In *Proc. SIGMOD*. 575–577.
[4] Kaustubh Beedkar, Bertty Contreras-Rojas, Haralampos Gavriilidis, Zoi Kaoudi, Volker Markl, Rodrigo Pardo-Meza, and Jorge-Arnulfo Quiané-Ruiz. 2023. Apache Wayang: A Unified Data Analytics Framework. *SIGMOD Rec.* 52, 3 (2023).
[5] Michael J Carey et al. 1995. Towards heterogeneous multimedia information systems: The Garlic approach. In *Proc. RIDE-DOM*.
[6] Adriana Grazia Castriotta. 2023. *Copernicus Sentinel Data Access Annual Report Y2022*. Technical Report 1. European Commission. 119 pages.
[7] Russell G. Congalton and Kate Green. 2020. *Assessing the Accuracy of Remotely Sensed Data: Principles and Practices*. CRC Press, Taylor & Francis Group.
[8] Arne de Wall, Björn Deiseroth, Eleni Tzirita Zacharatou, Jorge-Arnulfo Quiané-Ruiz, Begüm Demir, and Volker Markl. 2021. Agora-EO: A Unified Ecosystem for Earth Observation – A Vision for Boosting EO Data Literacy –. In *Proc. Big Data from Space (BiDS)*.
[9] Ahmed Eldawy, Vagelis Hristidis, Saheli Ghosh, Majid Saeedan, Akil Sevim, A.B. Siddique, Samriddhi Singla, Ganesh Sivaram, Tin Vu, and Yaming Zhang. 2021. Beast: Scalable Exploratory Analytics on Spatio-temporal Data. In *Proc. CIKM*.
[10] Lukas Kondmann et al. 2021. DENETHOR: The DynamicEarthNET dataset for Harmonized, inter-Operable, analysis-Ready, daily crop monitoring from space. In *Proc. NeurIPS Datasets and Benchmarks*.
[11] Stefanie Holzwarth et al. 2020. Earth Observation Based Monitoring of Forests in Germany: A Review. *Remote Sensing* 12, 21 (2020), 3570.
[12] Haralampos Gavriilidis, Kaustubh Beedkar, Jorge-Arnulfo Quiané-Ruiz, and Volker Markl. 2023. In-situ cross-database query processing. In *ICDE*. 2794–2807.
[13] HEAVY.AI. 2024. https://heavy.ai/
[14] OpenStreetMap. 2024. https://www.openstreetmap.org.
[15] Paul J. Pinter, Jr., Jerry L. Hatfield, James S. Schepers, Edward M. Barnes, M. Susan Moran, Craig S.T. Daughtry, and Dan R. Upchurch. 2003. Remote Sensing for Crop Management. *Photogrammetric Engineering & Remote Sensing* 69, 6 (2003).
[16] PostGIS. 2023. https://postgis.net/
[17] Jerry C. Ritchie, Paul V. Zimba, and James H. Everitt. 2003. Remote Sensing Techniques to Assess Water Quality. *Photogrammetric Engineering & Remote Sensing* 69, 6 (2003), 695–704.
[18] Apache Sedona. 2024. https://sedona.apache.org/
[19] Raghav Sethi, Martin Traverso, Dain Sundstrom, David Phillips, Wenlei Xie, Yutian Sun, Nezih Yegitbasi, Haozhun Jin, Eric Hwang, Nileema Shingte, et al. 2019. Presto: SQL on everything. In *Proc. ICDE*. 1802–1813.
[20] QGIS Geographic Information System. 2024. http://qgis.org