Received: 2 May 2021 | Revised: 13 September 2022 | Accepted: 5 October 2022
DOI: 10.1002/cpe.7484

RESEARCH ARTICLE

WILEY

RADIX+: High-throughput georeferencing and data ingestion over voluminous and fast-evolving phenotyping sensor data

Saptashwa Mitra[1] | **Maxwell Roselius**[2] | **Pedro Andrade-Sanchez**[3] | **John K. McKay**[4] | **Sangmi Lee Pallickara**[1]

[1]Department of Computer Science, Colorado State University, Fort Collins, Colorado, USA
[2]Spectra Logic, Boulder, Colorado, USA
[3]Department of Biosystems Engineering, University of Arizona, Tucson, Arizona, USA
[4]College of Agriculture, Colorado State University, Fort Collins, Colorado, USA

Correspondence
Saptashwa Mitra, Department of Computer Science, Colorado State University, Fort Collins, CO, USA.
Email: sapmitra@cs.colostate.edu

Funding information
Advanced Research Projects Agency - Energy; Cochran Family Professorship; National Institute of Food and Agriculture, Grant/Award Number: COL0-FACT-2019; National Science Foundation, Grant/Award Numbers: ACI-1553685, OAC-1931363

Summary
Remote sensing of plant traits and their environment facilitates non-invasive, high-throughput monitoring of the plant's physiological characteristics. However, voluminous observational data generated by such autonomous sensor networks overwhelms scientific users when they have to analyze the data. In order to provide a scalable and effective analysis environment, there is a need for storage and analytics that support high-throughput data ingestion while preserving spatiotemporal and sensor-specific characteristics. Also, the framework should enable modelers and scientists to run their analytics while coping with the fast and continuously evolving nature of the dataset. In this paper, we present RADIX+, a high-throughput distributed data storage system for supporting scalable georeferencing, and interactive query-based spatiotemporal analytics with trackable data integrity. We include empirical evaluations performed on a commodity machine cluster with up to 1 TB of data. Our benchmarks demonstrate subsecond latency for majority of our evaluated queries and ∼ 8× improvement in data ingestion rate over systems such as Geomesa.

KEYWORDS
high-throughput phenotyping, georeferencing, distributed analytics, sensor, data ingestion, visualization

1 | INTRODUCTION

The proliferation of satellite, UAV, and ground based remote sensing and high-resolution automated observational equipment allows scientists to monitor dynamic, spatiotemporal phenomena. Recently, there has been a demand for data-intensive *high throughput phenotyping research platforms* within regional and national phenotyping facilities.

Field-based plant phenotyping allows the study of how variation in genomes leads to variation in plant ecophysiology and how plants respond to changes in the environment and human management. The phenotype (trait value) of a given plant is formed based on the dynamic interaction between the genotype, the abiotic and biotic surroundings in which plant grows and develops (environment) and production practices(management). These interactions (Genotype × Environment × Management) determine the phenotypes of plant performance and productivity as measured by such traits as accumulated biomass, grain yield on an area basis and resource utilization efficiency. To observe and track these interactions, phenotyping platforms have actively adopted frequent deployment of fast response electronic monitoring methodologies such as environmental, displacement, and plant sensors; coupled with robust electronic data acquisition and processing systems, GNSS (global navigation satellite

Saptashwa Mitra, Maxwell Roselius, Pedro Andrade-Sanchez, John K. McKay, and Sangmi Lee Pallickara contributed equally to this article.

system) positioning, LIDAR (light detection and ranging), and aerial imagery captured by UAVs over a massive number of experimental plots of crop plants.[1]

High-throughput phenotyping (HTP) with advanced electronic sensing generates large volumes of extensively heterogeneous datasets. Completing data collection involves months of observations using various mobile and stationary sensors. To track the genotype with the pertinent environment, data is often organized based on the pre-planned geospatial plots and phenotype treatments throughout the duration of observation. There have been several prototypes developed for plant phenotyping purposes at academic institutions and private firms.[2-7] These research works individually present methods for the efficient storage, monitoring, analytics or modeling over plant attributes for phenotyping. Although these systems introduce novel approaches to plant phenotyping, they do not provide a scalable solution for an *integrated* data analysis pipeline that enables pre-processing, ingestion, storage, access and integration with publicly available analysis frameworks.

One of the primary concerns for the phenotyping data analysis is to access data based on spatiotemporal attributes along with their metadata effectively while the dataset is evolving fast. Voluminous data must be ingested in a near real-time fashion after comprehensive data preprocessing, such as, georeferencing and indexing across the heterogeneous types of observations. Also, to visualize and analyze continuously ingested datasets, users or applications should be able to validate that their query output is up-to-date without reissuing the query. Our goal is to enable high-throughput data ingestion while supporting scalable data preprocessing and trackable data query evaluation that provides data integrity between the dataset and query results.

1.1 Research questions

Research questions that we explore in this study include the following:

RQ1: How can a system perform high-throughput georeferencing and spatiotemporal indexing over sensor data (that are voluminous and heterogeneous) to make the data available immediately to users?

RQ2: How can we ensure data integrity between the fast-evolving sensor dataset and output of the user's query? Block-level data integrity should be preserved even in a query where results are composed of multiple query predicates.

RQ3: Given a voluminous high-throughput phenotyping dataset, how can we provide an integrated storage framework that provides real-time access for interactive analytics and reliable long-term data archival along with seamless interaction with back-end data analytics frameworks?

1.2 Overview of approach

In this article, we present a distributed data framework that enables high throughput data ingestion, a trackable data retrieval scheme with block-level integrity, and flexible and integrated data analysis for highly heterogeneous sensor datasets. These datasets are ingested after comprehensive data preprocessing including georeferencing, spatiotemporal indexing, and metadata extraction. We propose a nested hash grid that is a distributed bitmap indexing scheme to encode observations and the targeted geospatial shape. A nested hash grid is a hierarchical bitmap hash that extends to multiple resolutions of geospatial blocks overlapping over a region. Based on user-specified plot shapes, our system determines resolution levels for observations. Small areas or complicated shapes often require higher resolutions of the nested hash grid to achieve better georeferencing accuracy. Different types of sensor datasets are aligned using the nested hash grid and indexed based on application-specific indexing attributes such as spatial polygons, temporal intervals, and metadata.

Integrity of the data retrieved through queries is tracked by using a distributed, memory-resident Hierarchical Integrity Graph (RHIG). Each vertex in RHIG maintains a label, which is a tuple of the id of the directory/data block and a 32-bit Adler checksum[8] that represents the hash-value for the block's contents. We leverage a Merkle Tree[9] to structure and manage hierarchical checksums. The set of directory/data blocks that is the result of the query evaluation creates a local RHIG with the block-level data checksums. Evaluating data integrity between the retrieved data and the most up-to-date dataset is performed by generating and tracking the relevant paths. This tracking scheme enhances the traditional data query to identify data blocks that should be retrieved to achieve consistency between the query output that is already retrieved at the client and the blocks that have been updated or recently added at the server side. Our approach achieves a significant reduction in the amount of data transferred for data queries over frequently updated datasets.

Finally, we interoperate with a distributed storage framework that interacts with near real-time monitoring and explorative analytics environments, as well as, pipeline the dataset to a cloud computing environment for subsequent customized analysis.

1.3 Paper contributions

Our system enables ingestion of voluminous observation data while supporting high-throughput georeferencing, metadata extraction, indexing, and staging. Users are allowed to incrementally retrieve continually arriving observations with the trackable data integrity. Specific contributions of this research include (1) high-throughput and customizable georeferencing, (2) a streamlined metadata extraction and indexing scheme over heterogeneous sensor data, (3) trackable data integrity for consistency of query outputs over fast-evolving dataset, and (4) an integrated data pipeline from voluminous noisy sensor data to interactive analytics and cloud-based advanced analysis.

1.4 Paper organization

The remainder of this paper is organized as follows. In Section 3, we discuss background information regarding the scope of this work, including phenotyping, georeferencing, and various sensor data. Section 4 describes our data model, followed by our methodology in Section 5. Section 6 describes the system architecture and design. In Section 7 we present our analysis of system experiments. Section 2 discusses related works, and Section 8 presents our conclusions, followed by Acknowledgment, Data Availability Statement, and Conflict of Interest Statement.

1.5 Paper extensions

Since the publication of our previous work, *Radix: Enabling High-throughput Georeferencing for Phenotype Monitoring over Voluminous Observational Data*,[10] we have extended our system to track data integrity at the block level. This allows users to validate the consistency of query outputs compared to the fast-evolving dataset stored in the backend storage framework (Section 6.3). This capability enables the system to retrieve data incrementally for reducing the amount of data transfer (Section 5.3). Finally, we have extended and integrated our methodology into the greater data and analytics frameworks that include optimizations for a faster geo-referencing scheme, handling of heterogeneous and diverse input data-types with pipelined preprocessing capability and inclusion of an efficient data backup and data-block retrieval and update framework (Section 6.4). In total, these extensions represent 50% of new material reflecting new scientific contributions.

2 RELATED WORKS

Field-based high-throughput phenotyping has been an active area of interest and made possible through platforms that help monitor plant attributes using high-throughput phenotyping.[11,12] Efficient online database for phenotypic data for retrieval and analytics has been explored in References 2-4, allowing users an accumulated repository for scientific data.

Automated approaches to high-throughput phenotyping, such as data extraction through image analysis or through profiling and prediction with the help of deep-learning models, have also been explored in several research works.[13,14] Automated image analysis techniques helps overcome the limitations of the human eye and help determine high-resolution structural features of cells available at a microscopic level. For instance,[15] implements performs image analysis over microscopic images to identify important cellular traits to perform automated phenotyping. There has also been multiple research into novel approaches to performing automated analytics like building predictive models over the phenotypic datasets.[6,7] Various model training frameworks, such as PyTorch,[16] TensorFlow,[17] and Apache Submarine[18] have been utilized for model training over large-scale geospatial data. However, these approaches do not scale when it comes to interactive query processing over voluminous data with high ingestion rates.

Multiple GIS tools and geospatial database management systems support georeferencing and analytics.[19,20] PostgreSQL,[21] MySQL,[22] and MongoDB[23] include support for geospatial relationship queries to evaluate whether a geospatial object (e.g., point or polygon) lies inside the spatial bounds of a given query. ArcGIS[24] is a geographic information system designed for working with geospatialspatial data and imagery and managing geographic information in a database, providing a wide range of geospatial analytics and mapping capabilities for geospatial data but it does not scale well for large-scale raw sensor data. However, since these systems are not specifically designed for large-scale datasets, they do not scale in scenarios that require efficient query evaluations over a large data store with frequent data staging and ingestions.[25] Ingesting data for these systems requires an extra step of indexing the incoming data in order to facilitate the georeferencing query, which negatively affects the ingestion time for voluminous incoming data.

There are geospatial analytics frameworks,[26,27] that are based on distributed, open-source frameworks such as Apache Hadoop[28] or Spark.[29,30] These software frameworks leverage additional indexing schemes over HDFS to avoid disk I/O, resulting in notable overheads in processing in addition to ineffective memory utilization for the continuously arriving voluminous data.

Systems like Synopsis,[31] Stash,[32] and Glance[33] construct sketches of voluminous spatiotemporal data simultaneously and stage them in-memory during ingestion. Since these in-memory data structures can be queried directly, avoiding disk I/O, we have leveraged this concept in our in-memory georeferencing during the ingestion phase. Similarly, we have based our querying over in-memory phenotyped summary-data on the metadata extraction schemes such as.[34] Geomesa[27] creates indices on the geospatial features of datasets by creating a space-filling curve atop their Geohash index which are used to determine intersecting data-records with query polygons. However, unlike Radix+, it does not support user-defined granularity in the form of geohash length that would allow large amounts of data concentrated in a small geospatial area.

3 BACKGROUND

3.1 Subterra: High-throughput phenotyping and large-scale analytics

Phenotyping of plant traits is used to facilitate the development of improved varieties by exploring characteristics of the genetic variation with the target parameters. In order to breed the best possible genetic traits into a crop variety, the more crosses and environments used for selection, the greater the probability of identifying superior variation. Hence, the idea is to phenotype very large numbers of lines of crops rapidly and accurately and dissect the best genetics of desired quantitative traits.

Our system, Radix+ has been developed and evaluated within the context of high-throughput phenotyping (HTP) project, Subterra.[35] Subterra investigates plant(corn) traits, specifically looking at genetic traits related to drought tolerance targeting drought-prone/arid areas and carbon sequestration[36] that preserves the soil carbon longer leading to less reliance on synthetic fertilizer and drought resistance.

Our scientists monitor the development and trait values of hundreds of genotypes of maize in replicated plots spanning environmental gradients. Differences among plots are measured using data collection methods that include remote sensing of reflectance data over the fields using multispectral UAV photography to compute, among others, vegetation indices from the observed wavelengths and through various types of autonomous sensors, such as sonar, thermal, and LIDAR.

3.2 Data acquisition and formats

In Subterra, we deal with two main types of raw data: (1) spectral images scanned using a UAV over the field and (2) various autonomous sensor measurements recorded using a ground platform with a front-mounted instrumented frame that traverses the field in a serpentine fashion. Figure 1A shows the composite mosaic of all the multiband aerial images taken over the field and Figure 1B shows the setup of the sensor array on our ground-platform.

3.2.1 Preprocessing

The raw data received from the fields are not suitable for ingestion, either because they do not have explicit spatiotemporal information or attributes of interest are not directly available and have to be interpolated through related data available in the raw records. Pre-processing of incoming voluminous data in a standalone fashion can be time-consuming and compute-intensive.

The UAV images come as multi-page .tif ortho-mosaic (created by collaging several geotagged snapshots taken by a UAV), each page representing the spectrums near-infrared, blue, green, yellow, red, and red-edge, that can be converted to their individual band intensity tagged

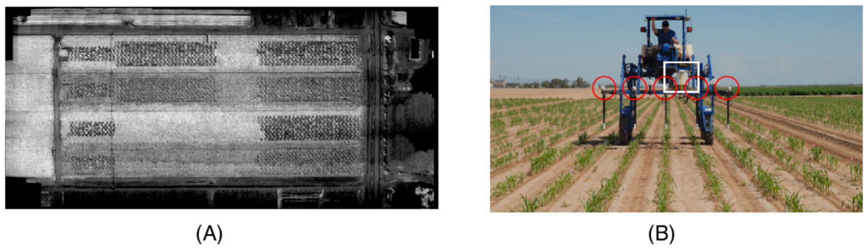

FIGURE 1 Heterogeneous data collection. (A) TIFF multispectral raster file from UAV. (B) Autonomous sensors mounted on ground platform.

to the specific lat-long coordinates using GIS APIs or software such as ArcGIS. The pre-processing for this data involves creating an interpolation formula between intensity value for each spectrum to their reflectance values by referencing some pre-configured reference tiles placed in known positions on the field. Using the calculated reflectance values for the relevant spectrums for the pixels in the mosaic, we can compute various vegetation indices such as NDVI and OSAVI. For instance, the Normalized difference vegetation index(NDVI) can be calculated as:

$$NDVI = \frac{R_{860\ nm} - R_{680\ nm}}{R_{860\ nm} + R_{680\ nm}}.$$

The sensor data is collected over the SUBTERRA experimental fields using a ground platform as shown in Figure 1B that goes over the experimental plots and periodically triggers measurements simultaneously from the various sensors such as IR surface(plant canopy) temperature, NDVI, LIDAR, as marked by circles in the figure. The coordinate recorded against each measurement is the coordinate of the GNSS sensor marked by the white square. Evidently, the coordinate of the actual sensor measurements must be offset by pre-configured relative distances of the equipment from the GNSS antenna using geodesic calculations. In addition to this, in case of the LIDAR sensors which tracks height and width of plants by sweeping at an angle of −45° to 45°, an extra geodesic displacement is required based on the current angle of sweep with respect to the LIDAR sensor to get the actual coordinate of the measurement. Also, LIDAR sweeps that fall beyond the plots in a buffer region can be rejected during pre-processing itself.[37]

These heterogeneous raw datasets collected from the phenotyping plots are not ready to be ingested into storage frameworks; they do not have explicit spatiotemporal information or attributes of interest are not directly available and have to be interpolated through related data available in the raw records. A scalable scheme to preprocess and index high throughput sensor data while coping with the heterogeneous sensing methodologies, geo-referencing,[38,39] and aligning related attributes is critical to enable effective data retrieval for subsequent analytics.

4 | DATA MODEL

RADIX+ is a spatiotemporally indexed distributed multi-dimensional map with distributed data preprocessing features. The data points are organized using chronological measures and geospatial coordinates. Data are also grouped as plot-level data based on the matching phenotyping plots and temporal durations.

4.1 | Query structure

RADIX+ supports 2 types of queries—spatiotemporal query and plot-wise query. The spatiotemporal query consists of the analytics query and the block query. Analytics queries retrieve simple statistics of attributes for the target spatiotemporal ranges. The following demonstrates the structure of the analytics query that returns a requested set of attributes aggregated over each plot that overlaps with the spatial_coverage specified in terms of a spatial polygon on the earth's surface, grouped by the temporal_coverage specified as a range in terms of a starting and ending timestamp.

```
SELECT max(temp), avg(ndvi), std(elevation)
FROM Roots_Data
WHERE coordinates in spatial_coverage AND timestamp IN temporal_range
```

Listing 1

A block query, as shown in Listing 2, is a query for the location of the data elements that satisfy the query attributes. It is particularly useful to pipeline data to external computing components and has a syntax similar to the one depicted in Listing 1, but the response, in this case, is a mapping between the plotID, date, sensorType to the locations of their corresponding raw data blocks'; these locations can either be a link to the block on IRODS (see Section 6.3) or RADIX+, depending on the system's configuration.

```
SELECT block_key, block_path
FROM Roots_Data
WHERE plotID IN plot_List AND timestamp IN temporal_range GROUP BY block_key
```

Listing 2

Finally, a plot-wise query, depicted in Listing 3 is evaluated over the geospatially grouped plot data. Therefore, the location of the record is represented using the corresponding plot identifier. Since plots are organized based on experimental metadata such as phenotyping treatments and genotype, this type of query is essential to trace the plants' development. The following depicts the syntax of a query that retrieves summary statistics for plots in the specified plot list over the particular temporal range.

```
SELECT max(temp), avg(ndvi), std(elevation)
FROM Roots_Data
WHERE plotID in plot_list AND timestamp IN temporal_range
```

Listing 3

4.2 | Distributed spatiotemporal data storage system

The basic unit of data storage is Radix+ is a block, which represents the data collected for a given experimental plot. Within each node, data is stored in the form of a collection of blocks, partitioned in a hierarchical directory structure that also reflects the metadata of the blocks. Each block contains data in a multi-dimensional array that is stored on the disk as a stream of bytes. Figure 2 shows the directory structure in Radix+ for multiple blocks in-memory. For instance, data belonging to plot 22,431, date 2020-06-28, and from the IR sensor would be stored in the directory path denoted by the leftmost depth-first tree traversal in Figure 2. The leaf of the directory structure contains the actual block representing the data belonging to plot 22,431.

The feature graph illustrated in Figure 2 a distributed in-memory **Metadata Graph** that incorporates in it the metadata associated with every Radix+ block and assists in the identification of relevant data blocks against users' spatiotemporal queries over the Radix+ system and preservation of integrity for each data blocks.

Radix+ extends Galileo,[40] a distributed zero-hop DHT storage framework with spatiotemporal data partitioning among the nodes in a cluster. One of the key capabilities inherited from Galileo is the linking of blocks with their associated metadata in a hierarchical tree. The nodes traversed in every depth-first traversal of this balanced tree represents the metadata related to a block pointed to by the leaf node. Figure 2 shows an example Metadata Graph housing 4 data blocks (B1, B2, B3, B4), belonging to 2 plots with id's 22,431 and 22,432 for dates: 2020-06-28 and 2020-07-10. The features that form the nodes of the graph and their order in the hierarchical graph is configurable and represent the common attributes(metadata) among all records in a block. This graph can be evaluated against a given spatiotemporal query by traversing the tree in a top-down fashion going down only the paths whose nodes satisfy the query parameters, with the leaf node containing block-level metadata. We have extended the existing distributed Metadata Graph to maintain the following three information at each leaf node:

Block Path represents a path to the block's physical location on either on the local node and/or the CyVerse IRODS storage system(our historical data archive). For Radix+ data blocks that get removed due to data back-up into IRODS, their metadata(which has very low memory consumption) is still maintained in the Metadata Graph for analytical query support. Access to raw data blocks are provided by the RHIG service, as we will explain later.

SensorCell is a data structure that holds the aggregate of the data records in a block. We can think of the SensorCell as an aggregation over all the data from a particular sensor that falls within a 3-dimensional bound in the spatiotemporal space marked, in our case, by the coordinate bounds of the plot and the temporal extent of the day of recording, as evident in Figure 2. Hence SensorCell provides a one-to-one mapping between the plot and the summarized sensor statistics over that plot for a given day. The aggregates supported in SensorCells, for now, are average, minimum,

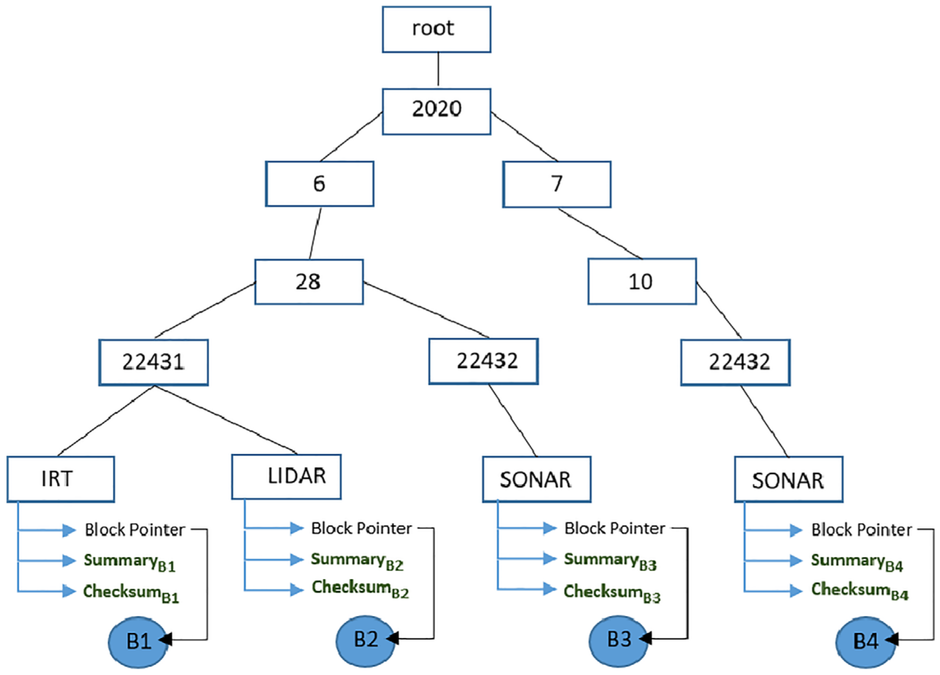

FIGURE 2 Structure of Radix+ Metadata Graph: this demonstrates the hierarchical nature of the graph with links to actual data blocks in the leaf nodes for fast query evaluation.

maximum, median, and standard deviation. Also, it is to be noted that since the leaf nodes partitioned by the date of recording, the plot-id and the sensor they represent, the only time these cells are rarely updated or overwritten.

Checksum: This is a 32-bit Adler checksum for the contents of the block to verify its integrity. It serves a couple of purposes. Firstly, it helps Radix+ monitor the success of the backup operation to the data archival system(IRODS). Secondly, it is later used to form a hierarchical integrity verification framework for end-users to verify their downloaded data-blocks or check for staleness in their local data repository without redundant requests to the IRODS server.

The Metadata Graph is persisted on disk at the end of every update and during shutdown of the cluster and can be unmarshalled from the serialized copy during startup.

5 | METHODOLOGY

5.1 | High-throughput georeferencing

Raw data collected from multiple sensors are not spatiotemporally well-aligned. Different data acquisition types (e.g., temperature collected from a sensor from ground platform vs. imagery from an UAV) or the positions within the sensor array result in misaligned spatiotemporal measurements. To trace and analyze plants' development while parameterizing corresponding treatments, we group raw observations geospatially at the plot-level.

Generally, georeferencing is the process of associating objects with a location(s) in physical space (e.g., a building, landmark, or data point). There are several variations of georeferencing. Raster to geolocation georeferencing[41] refers to the process of aligning raster images to a map coordinate system. Direct georeferencing[38] is similar but does not require the use of ground control points.[39] Finally, georeferencing may refer to the process of associating geographic coordinates to polygons on the earth's surface, for example, mapping points to a particular county in a state. For this study, we refer to georeferencing as the process of mapping a data point associated with a geographic coordinate to a predefined polygon representing a phenotyping plot. Providing this feature by checking membership of each data points against a large number of polygons representing the plots would be computationally intensive while drastically reducing the speed of data ingestion. The following subsections describe how we circumvent these inefficiencies.

5.2 | Distributed nested HashGrid

Our distributed hashgrid uses a 2-tier georeferencing scheme and gets initialized during the creation of the distributed filesystem. Figure 3 shows a generalized representation of the RADIX+'s hashgrid, where, the first level of the hashgrid is a bitmap formed by splitting the entire spatial area under the purview of the DHT node is split into a grid of N-character geohashes[42] (N = 11). The GeoJSON shapefile of the plots is then AND-ed with this bitmap to form a low-resolution bitmap(Figure 3) of the plots over the spatial coverage of this node—this represents the first tier of our georeferencing process. In case of grids (11 character geohashes) that hold more than 1 plot, as shown in Figure 3, a second level of mapping is maintained between these grids and the bounds of the plots that clash with this grid. For such plots, we also check for the rare condition where its spatial extent might be shared among multiple DHT nodes and if that is the case, we maintain a map of (plotId, Node) to keep track of such plots, which we call **SharedPlotMap**.

Hence, during the georeferencing process, given a data record, we use the 11 character geohash representation of its coordinate to find its position in the hashgrid and in the optional second phase, if that geohash contains a second-tier mapping, we try to resolve the collision by comparing the data coordinate to the bounds of the colliding plots happed to that grid. It is to be noted that the majority of the grids should lie in a single plot and the collisions are a more rare scenario and thus, we avoid having to compare polygonal bounds in a majority of the scenarios and even when we have to, we narrow down the scope of the candidate plots that might hold that particular record.

5.3 | Dynamic integrity tracking

RADIX+ provides a lightweight block query evaluation and integrity tracking scheme between the results of the queries and the original data stored in the backend storage systems through the **RHIG (RADIX+ memory-resident Hierarchical Integrity Graph)**. RHIG is a data structure that is maintained by all RADIX+ data stores that fetch plot data from IRODS using spatiotemporal block queries. Each data-store maintains 2 RHIGs - one that reflects its local contents (**Local RHIG**) and the other that reflects the perennial IRODS data(**global RHIG**). It is through comparison between these 2 RHIGs, that fast evaluations can be made regarding the staleness of the local data-store contents against any given spatiotemporal query.

RHIG resembles the MetadataGraph of the RADIX+ filesystem: it is a hierarchical feature tree constructed out of the relative paths of the physical location of the blocks and their checksums. The RHIG node labels reflect the directory in the traversal for the relative paths (the leaf node corresponding to the plot-data file/block) and hence also correspond to the nodes on a MetadataGraph, since the MetadataGraph's node hierarchy, too, reflects the directory structure. Figure 4A shows an example of a RHIG constructed from relative paths for three data blocks for plots with IDs 22431 and 22342.

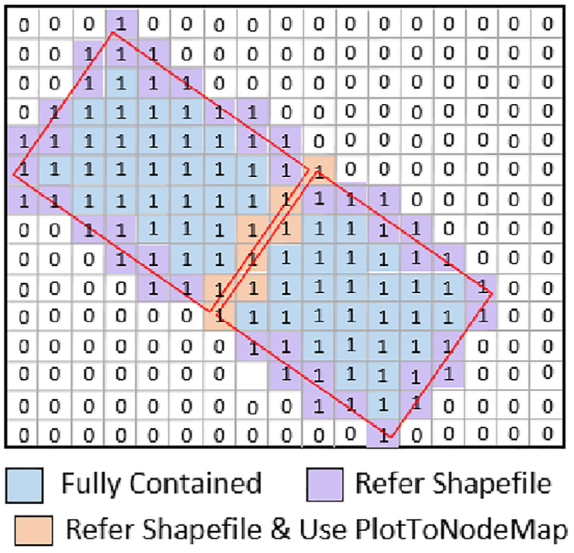

FIGURE 3 2-tier georeferencing scheme using the distributed RADIX+ HashGrid.

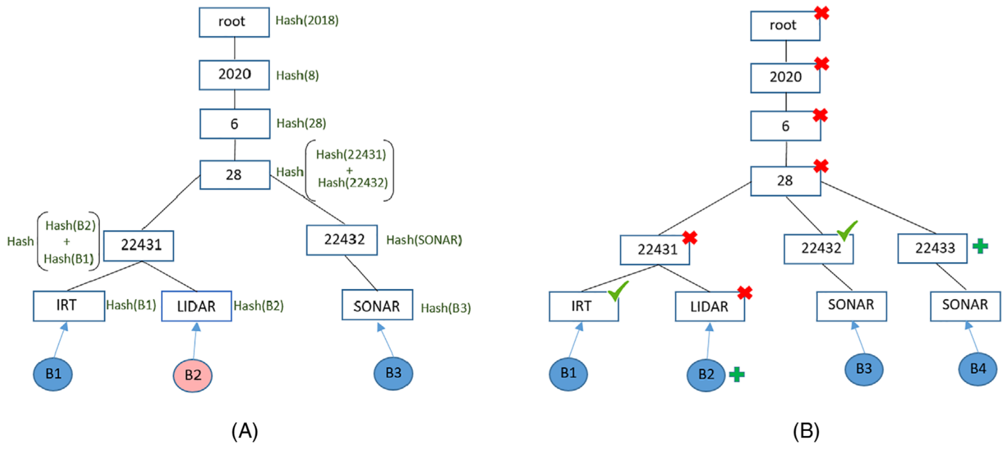

FIGURE 4 Integrity Tracking with RHIG: Comparison between the local and global RHIG to identify new data segments. RHIG being structured as a metadata graph helps in rapid evaluation. (A) Structure of a RHIG (local). (B) Global RHIG.

Unlike the MetadataGraph, the leaf node of a RHIG contains two elements: (1) the physical location of the block on the IRODS server and (2) a 32-bit Adler checksum that represents the hash-value for the contents the directory/data block (as computed in Section 4.2). All nonleaf RHIG nodes contain only a Merkle hash tree,[9] constructed out of the checksum of its immediate children. The nonleaf nodes are evaluated in a bottom-up fashion and are used to verify the integrity of the directory it represents and detect corruption in its immediate children node/directories, when the Local and Global RHIGs get compared (detailed in Section 5.3.1).

5.3.1 | Detecting stale/corrupt blocks

The structure of a RHIG makes it conducive to simultaneous query evaluation and integrity check. Since it is structurally a feature graph, it can easily be traversed in a top-down manner while evaluating a spatiotemporal query. The maintenance of a Merkle Tree on each non-leaf node, instead of maintaining one single Merkle Tree over the entirety of the data files, provides a scheme that preserves the spatiotemporal locality of the nodes and ensures only the nodes matching a query are checked for their integrity.

Algorithm 4 details our method of simultaneous query evaluation and integrity check using the local and global RHIGs (R_L and R_G, respectively). This enables a user to check for the freshness of their local data-backup using a query, that specifies the spatiotemporal region whose integrity is to be checked. Starting with the root node, for each RHIG, we recursively test the checksum of the nodes(which is the hash value of the root of their Merkle Tree). In case of a match, that particular directory is deemed up-to-date and no further evaluation needed along this path. If not, we compare the Merkle Tree of both nodes to find the children of the R_G nodes that are missing/stale and evaluate their validity against the spatiotemporal query and if true, their matching label is found in the node in R_L and the same process is repeated(in case of missing matching label, this directory is absent and needs download). L_D gives us a final set of directories and block, with their paths that need to be fetched from IRODS to get the local repository up-to-date.

Figure 4 shows an example of tracking stale entries in a repository through a comparison between the Local and Global RHIG. Ignoring the spatiotemporal query(Q_{ST}) let us assume that in Figure 4 the block marked in red is corrupted. Algorithm 4 would keep going down both the RHIGs till node 28, where through comparison of the Merkle Trees, we would find one matching node (22,432), one corrupt node (22,431) and one new node(22,433). The directory corresponding to 22,431, thus gets added to the list of paths to be downloaded(marked with + in Figure 4B), without any further evaluation, thus saving further computations and 22432 and all its children are ignored since they are deemed up-to-date. Only 22,431 is then further probed.

In a naive approach without a RHIG, we would have to evaluate a spatiotemporal query, find its corresponding blocks in the global repository, and compare their corresponding checksums in the RHIGDump (explained in Section 6.4) with the local blocks' checksums. Hence, ignoring the spatiotemporal query and considering N global data-blocks, M total and C corrupt local blocks, the naive scenario requires us to access $O(MN)$ nodes(RHIGDump entry) to detect corruption. In contrast, the RHIG structure, with F feature levels, accomplishes the same operation by visiting only $O(MF)$ nodes in the worst case($M, F << N$) and O(1) in a best case scenario.

Algorithm 1. Stale block(s) detection with RHIG

```
RHIG-Update ($RN_G, RN_L, Q_{ST}, L_D$)
    Input: $RN_G$ and $RN_D$ - matching RHIG nodes from global & local RHIG resp.; $Q_{ST}$ - query; $L_D$ - paths for download (initially empty)

if $RN_G$.checksum = $RN_L$.checksum then
  return; ▷ Directory/file is up-to-date
else if $RN_G$.children is NULL then
  $L_D$.append($RN_G$.path); ▷ Leaf node

else
  $N$=compare($RN_G$.merkle_tree,$RN_L$.merkle_tree)
  for $C$ in $N$ do
    if evaluate($C, Q_{ST}$) = True then
      if $RN_L$.children contains $C$ then
        $C_G$ ← matching child in $RN_G$
        RHIG-Update($C_G, C, Q_{ST}, L_D$)
      else
        $L_D$.append($RN_G$.path); ▷ New Entry
  end
```

6 | RADIX+ SYSTEM ARCHITECTURE

RADIX+ is a distributed framework that performs efficient and high-throughput storage of incoming spatiotemporal data and provides query-based data retrieval. The framework is designed to handle high-throughput periodic ingestion, preprocessing, and geo-tagging of voluminous heterogeneous data in a distributed manner. The system interacts with a data archival service, CyVerse's IRODS[43] to provide long-term data storage. If users desire to perform customized analyses, the selected data will be pipelined to the user's container from either short-term or long-term storage components. Figure 5 explains the activities and interactions among the various RADIX+ modules in order to provide the aforementioned functionalities.

6.1 | Data Ingestion

Data curation in RADIX+ is initiated when an external node(data-source) with unprocessed/raw data calls and activates the **RADIX+ Ingestion Manager (VIM)** module on any of the cluster nodes, which we call the **Ingestor Node**. The VIM module on the Ingestor processes the incoming data into chunks of size 250 MB and handles them over a threadpool of **ChunkProcessor** module whose job is to identify the source of the incoming data and pre-process them, if necessary, that is, if the records are missing correct spatiotemporal tags and then split the processed spatiotemporal data into groups of N (N = 2000) records each, called **Splits** and puts them in a **MessageQueue**.

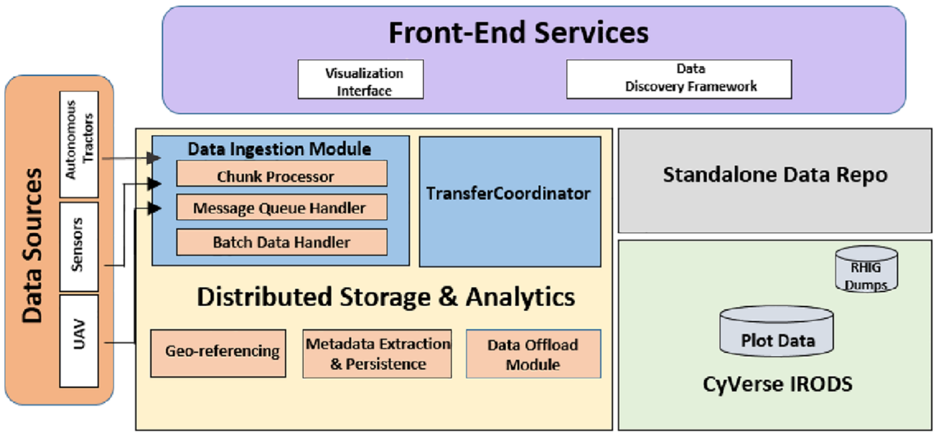

FIGURE 5 RADIX+ architecture.

A MessageQueueHandler picks off unprocessed Splits from the queue and assigns them to a thread-pool of **MessageRouters** that perform two actions: they **partially geo-reference** each Split by using the cluster's spatiotemporal partitioning algorithm to determine the node with the highest number of records in a sample taken from the Split's data, called the **DataHandler** node and forwards it the Split wrapped in a **StorageRequestMessage**. Thus the process of properly georeferencing gets distributed across the cluster nodes.

6.2 Distributed georeferencing, storage, and metadata extraction

The process of georeferencing in RADIX+ is extended from the relaxed georeferencing scheme in Reference 10 which is distributed and asynchronous. The data points are evaluated data-parallel over the cluster and merged through one final data-interchange among the cluster nodes at the end of the data ingestion process, thus leveraging concurrency and the DHT's data locality and also reducing the amount of data exchange between the nodes.

The incoming StorageRequestMessages on each DataHandler node are loaded in an **UnprocessedMessageQueue** from which they get taken and processed by a thread-pool of **MessageHandlers**. Each MessageHandler performs geo-referencing on the records and maps them to the node to which they belong. From here, the data storage and metadata extraction follow two possible paths. First, in case for records that get mapped to the local node by the partitioning algorithm, the MessageHandler simply saves/appends the records to existing data blocks and updates or adds the metadata information of the new data elements on the **MetadataGraph**. Second, records belonging to a foreign node are maintained in an in-memory **ForeignPlotDataMap**. The ForeignPlotDataMap gets cleared at a regular interval when new entries have not been observed for a plot for more than a time threshold or when the record count crosses 2000—the data for this plot is compressed (with Snappy[44]) and wrapped into a StorageRequestMessage and forwarded to their respective nodes, which would process them from their UnprocessedMessageQueue in the above manner, thus completing the process of geo-referencing. The aggregation of foreign plot records in-memory before transferring them to their respective foreign nodes prevents excessive disk I/O during this phase of ingestion and limits the amount of message exchanged between the nodes.

6.3 Long-term data storage

Access to historical data is critical to analyze phenotyping data. RADIX+ stores data in CyVerse's IRODS to achieve long-term storage and easy-to-use integration with the cloud-based computing environments and provides plot-wise data management scheme in IRODS. The final phase of ingestion process requires interaction between nodes in the cluster to combine any geo-referenced data for Shared Plots that might be held on multiple nodes. Thus, DataHandlers, on completion of their local ingestion process, wait for a threshold amount of time to declare themselves as idle in order to initiate the data backup process. This wait is to allow for the inter-exchange of properly geo-referenced messages of StorageRequestMessage among nodes to transpire. Each idle node notifies a TransferCoordinator node of its idle status along with a list of plotID-s whose data it has for a lock on those plots.

The TransferCoordinator consults the **SharedPlotMap** (Section 5.2) to identify plots(if any) whose spatial bounds might span multiple DHT nodes. If so, the TransferCoordinator allows the first node in the map corresponding to the plotID to claim this lock and rejects subsequent lock requests for this plot from other nodes. It then sends back a list of plots whose locks are granted, along with names of nodes that might have data for the overlapping plots. The DataHandler can then request the data for those overlapping plots from those nodes or in case of no overlap, simply bundle the data paths for offload and initiate data backup through the IRODS API into paths in IRODS that have similar relative paths as that in RADIX+. Along with offloading the data paths, each DataHandler also appends the paths and the CRC checksum of each data block along with their timestamp to a remote file in IRODS called the **RHIGDump**, which is the main component for our Data Discovery Framework and helps create and maintain the indexing and lookup data structure for the data blocks on IRODS.

6.4 Data retrieval with trackable integrity

In **Radix+**, data blocks may reside in one or multiple components of the framework such as a short-term storage system, archival service, and in the hard-disk drive of a computing device. To provide assurance of data consistency over the distributed setup, we apply an integrity tracking scheme (see Section 5.3) for data retrieval and archival managed by the **Data Discovery Framework (DDF)** which provides API for scientists/users to efficiently identify and fetch portions of the geo-referenced data-blocks from the IRODS storage using spatiotemporal and/or feature queries or update their existing data repository by identifying only the stale/missing data-elements. A DDF can be loaded on any external user node, which we call **DDF Coordinator** and the query structure supported by the DDF API is that in Listing 3.

6.4.1 | RHIG population

DDF generates and maintains the Radix+ Hierarchical Integrity Graph(RHIG) to track data integrity across the Radix+ framework against an external short-term storage. The individual offloaded block-paths along with their checksum from the RHIGDump are used to construct the Global RHIG on an external node using the DDF API. The Local RHIG is constructed as the node starts fetching data and gets constructed off the relative paths of the downloaded blocks in the local storage and their checksums that get computed during download. RHIGs can be persisted locally by the DDF Coordinator in a serialized json file from which they can be reloaded during startup.

A DDF Coordinator can be initialized using a **rhig_init** command to our DDF API that fetches the IRODS RHIGDump file for the requested filesystem and populates them into the in-memory Global RHIG - this represents the master index to the persistent data-backup in IRODS. Existing Global RHIGs can also be updated through a **rhig_update** command that fetches the RHIGDump from a specified offset to fetch the newly appended paths to the RHIGDump and updating the Master with new paths (if any). It is to be noted that in order to update an existing RHIG(local or global) only the nodes that lie in the newly added paths need to have their checksums updated once at the end of their addition into the RHIG.

6.4.2 | Path evaluation

In a cold-start scenario, the Local RHIG is empty and the Master is evaluated against the user query. Unlike the Metadata Query evaluation, the paths are not evaluated all the way to the leaf, but the topmost nodes in the RHIG that satisfy a query (refer Algorithm 4). A bundled fetch request is made for all the directories and blocks that result from the RHIG evaluation—this helps reduce the number of individual requests we would have to make for individual blocks otherwise. The fetched entries and checksum of the requested nodes are checked against the matching nodes of the Global RHIG to evaluate the integrity of the fetched data from IRODS. The Local RHIG (which is currently empty) is then appended with the new paths and its nodes' checksums/Merkle Trees are updated.

In an update scenario, we run Algorithm 4 against the Global Local RHIGs to evaluate the new paths (if any) that need fetching. The rest of the process follows the cold-start scenario. As we can see, this comparison between the Master and the Local RHIG contents helps greatly reduce the amount of data-transfer needed in a partially incomplete/corrupted local storage.

6.5 | Metadata-driven visual analytics

End users can interact with the Radix+ DFS through a light-weight front-end web-application that fetches data to be loaded on a Google Map-based[45] terminal using MetadataQueries as shown in Listing 2. The MetadataGraph is the backbone that supports these query analytics. A metadata query gets forwarded randomly to an available cluster node, that uses the partitioner to multicast it to those cluster nodes, whose spatiotemporal scope match the scope of the query in each of the node, the MetadataGraph is evaluated to list of paths to all the leaf nodes in the tree that satisfied all the attributes of the query. The summaries from each of these leaf nodes are combined and returned as a key-value JSON response (which is used to generate the visual representations), to be visualized over the user interface.

6.6 | Containerized front-end UI

The front-end visualization UI, along with the RESTful web service that helps redirect front-end user analytics queries into the Radix+ distributed system is currently on our own Apache server.[46] We also provide it in Docker[47]-based container, ready to be deployed on any external node, without further configuration.

7 | SYSTEM EVALUATION

7.1 | Datasets

We have used two sets of data for the evaluation of our system—(1) sensor data collected over ∼ 1800 plots using heterogeneous autonomous sensors(explained in Section 3.2) from Subterra corn-fields in Maricopa, AZ from 4 days in 2019, each amounting to a processed data size of around 2 GB, totaling to 8 GB (D_R) and (2) a ∼1 TB synthetic dataset, modeled on the real Maricopa data. This consists of a collection of source files ∼5GB each representing data collected for 200 unique dates for the year 2019 (D_S). The main purpose of using D_S is to be able to evaluate the scalability of our system in resource-intensive scenarios.

7.2 | Distributed cluster configuration

We have used two different cluster configurations for D_R and D_S scenarios. For D_R, we have used a cluster of 20 nodes. Each node in our distributed cluster is an HP Z420 with the configuration: 8-core Xeon E5-2560V2, 32 GB RAM and 1 TB disk. The data is partitioned uniformly over the cluster based on the first 8 characters of their Geohash. In case of D_S we have used a similar setting but increased the number of cluster nodes to 200 to allow for high-volume distributed ingestion.

7.3 | Data ingestion benchmarks

7.3.1 | Voluminous data ingestion

We have profiled the ingestion rate of our system over D_S by reproducing scenarios where the RADIX+ Ingestion Module is started on multiple cluster nodes simultaneously for ingestion. Figure 6A compares four scenarios where 2,10,20 and 40 nodes in the cluster are simultaneously tasked with ingesting ∼5GB of data each accounting for ∼10,50,100, and 500 GB of total simultaneous data ingestion respectively. The boxplots show the range of time taken by various cluster nodes to complete the process of data interchange, geo-referencing, metadata update and data partitioning. Expectedly, the total ingestion time increases as increasingly greater amount of data needs to be ingested and also the percentage of cluster nodes are involved in both initial partial-georeferencing of incoming data, as well as the actual geo-referencing and storage for incoming data messages from other nodes.

The throughput in this scenario is calculated by taking in consideration the slowest node in the cluster. Figure 8 compares the ingestion throughput of the RADIX+ cluster for these four scenarios and we can see that the overall throughput increases with more nodes involved in ingesting greater amount of data. However, the rate of improvement in throughput drops as more cluster nodes have to perform the dual responsibility of running an Ingestion Module and perform distributed geo-referencing and storage.

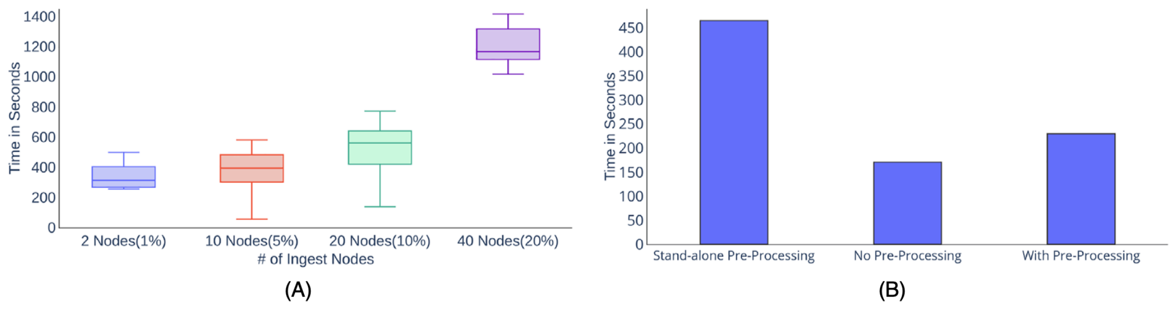

FIGURE 6 Heterogeneous sensor data ingestion time: (A) with 2,10,20, and 40 ingestion nodes, resp., each absorbing ∼5GB synthetic data (B) for 8 GB real sensor data with, without data-preprocessing and with distributed preprocessing, resp.

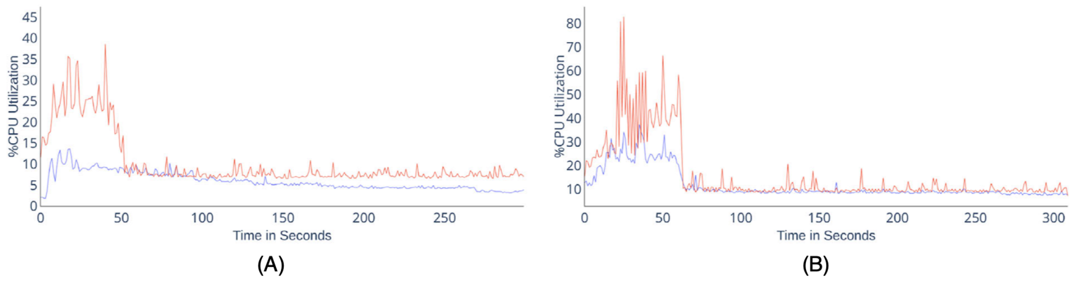

FIGURE 7 Comparison of avg. CPU utilization between ingest and non-ingest nodes during parallel ingestion through (A) 2 ingest nodes and (B) 20 ingest nodes.

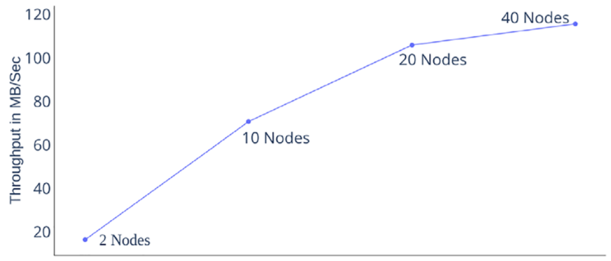

FIGURE 8 Data ingestion throughput versus number of ingestion nodes.

7.3.2 | Preprocessing overhead

We have evaluated the effect of preprocessing of incoming data (e.g., sensor-based geodesic calculations) on the overall ingestion latency on D_R. Figure 6B demonstrates the following three scenarios in order: (1) standalone pre-processing of raw data followed by single node ingestion, (2) absorption of preprocessed data split over four ingestion nodes, and (3) scenario 2, but with unprocessed data, thus, making use of the inbuilt pre-processing module on each ingestion node before actual ingestion. Expectedly, latency in scenario 2, for ingesting already processed data (~170.8 s) is the lowest (~25% lesser than scenario 3, which performs the extra data preprocessing in each Ingestion Manager). However, this overhead is low considering scenario 3's latency for complete distributed preprocessing and ingestion of the data is ~2.05x lower than scenario 1.

7.3.3 | Resource utilization

We have compared the CPU utilization of the RADIX+ system between scenarios with 1% and 10% of the total cluster nodes involved in ingestion. Figure 7A,B shows the CPU utilization pattern of these two scenarios, respectively, for the first 5 min of the ingestion process—the red curve shows the average utilization of a random sample of ingestion nodes and the blue demonstrates the average CPU utilization of a subset of the noningestion nodes. In both scenarios, the nodes running Ingestion modules show high initial utilization due to the execution of the dual task, but the utilization goes down to a more stable pattern with time once the VIM is done with its processing.

7.4 | Distributed query evaluation

7.4.1 | RADIX+ metadata query

We have evaluated the performance of the metadata-driven query over the RADIX+ framework. As explained before, both the visualization-related metadata queries and block-related queries have similar formats and refer the distributed metadata graph for results—they only differ in which of the leaf elements get returned from the graph. Figure 9 compares the effect of query size over its latency. We compare four sets of spatiotemporal requests, where, keeping the temporal component constant, we expand the spatial polygon incrementally to include 10%, 33%, 50%, and 100% of the total plots. We can see that the average query latency grows with query size but stays relatively low, mainly due to the fact that the in-memory metadata graph ensures that no disk I/O is necessary for visualization.

7.4.2 | Block query

Table 1 compares the evaluation latency for block queries, which are supported both by the RADIX+ distributed system, as well as the RHIG standalone module, due to the structural similarity between the Metadata Graph and the RHIG. The query involved here (Q_{25}) is for all blocks for a single day (25% of data coverage) totaling to ~1700 data blocks and Table 2 compares the path evaluation time for the same query over a locally loaded RHIG (170 ms) and the RADIX+ metadata graph (560 ms). It is evident that evaluating paths through RADIX+ requires contacting a cluster node and distributed evaluation, thus leading to a slightly longer evaluation time than a local RHIG.

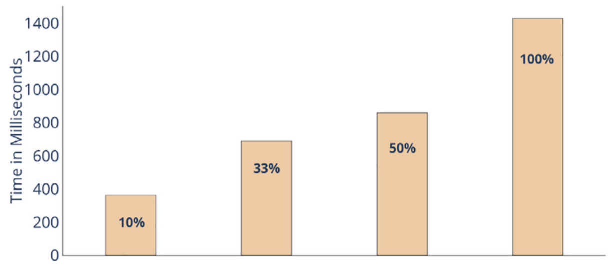

FIGURE 9 Metadata query latency versus size.

TABLE 1 Data fetch latency.

	Path evaluation	Data fetch	Integrity check	Overall
Using RHIG (S1)	0.17(0.004%)	3842.76(99.9%)	1.389(0.036%)	3844.18
Using metadata graph (S2)	0.56(0.006%)	8462.57(99.8%)	12.880(0.16%)	8476.01

TABLE 2 RHIG creation and regeneration.

Cold-start RHIG construction (S1)	**26.03 s**
RHIG regeneration (S2)	0.123 s

7.5 | Data discovery framework evaluation

We now explain our evaluations over the identifying and fetching of archived historical data from the CyVerse-IRODS server using the data discovery framework for the D_R dataset.

7.5.1 | RHIG generation

As shown in Table 2, in a cold-start scenario, RHIG generation on any node by fetching and loading the RHIGDump from IRODS takes ∼26.03 s on average. However, restarting a previously loaded RHIG from a locally persistent serialized RHIG copy takes a substantially lower time(S2), primarily due to the communication overhead and the delay in response from the IRODS server. Hence, we can conclude that a key to reducing download time for raw data-blocks would be to accurately identify the relevant blocks against a query and reduce unnecessary communication.

7.5.2 | Data retrieval with RHIG

Here, we evaluate the performance of the RHIG framework while downloading data against Q_{25} from scratch on a single machine – the estimated size of the data to be downloaded is ∼2 GB. Table 1 also compares the download and integrity evaluation time using paths assessed using both RHIG(S1) and the metadata graph(S2). It is evident that both retrieval and integrity check time for S2 is substantially higher since unlike a RHIG, a metadata graph contains block paths at the leaf nodes always evaluates to the leaf nodes against a query- this leads to a greater number of fetch requests over IRODS and separately evaluating their integrity. The RHIG on the other hand, evaluates to the directory represented by the topmost node in its hierarchical structure that matches the query and thus, requires lesser communications with the IRODS server and lesser successive authentication.

7.5.3 | Local data update with RHIG

It is to be noted that the download of data takes a lion's share of the entire data fetch operation, as shown in Table 1. In Figure 10 we outline scenarios, where a previously downloaded local data dump, that may have become stale is updated using the RHIG. Identification of the correct stale

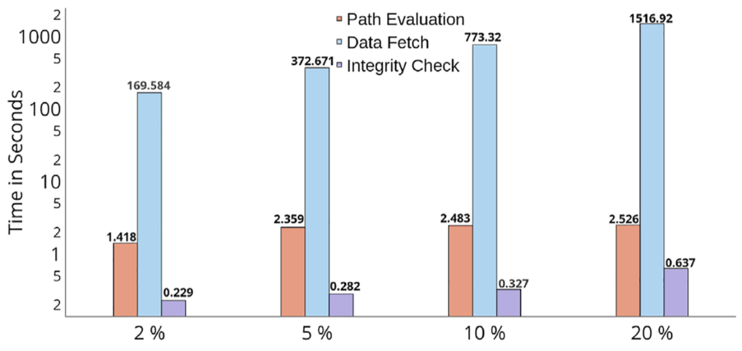

FIGURE 10 RHIG data discovery: latency for identification of varying fractions of stale blocks.

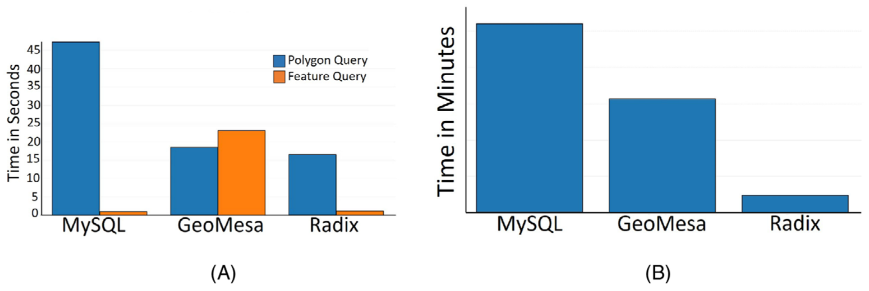

FIGURE 11 RADIX Performance comparison. (A) Query evaluation time comparison. (B) Time to fully index and store a 10 GB file.

directory/blocks, if any, is paramount reducing the overall fetch time. For our experiments, we have randomly corrupted 2, 5, 10, and 20% of the local dump for Q_{25} and executed an update over the same query (Q_{25}). We can see that the download time is significantly low that a fresh download for all of these scenarios, primarily because of the reduced amount of data that is required to be fetched from the server. Hence, in a real-world scenario, a slightly out-of-date local copy can be quickly updated using server data through the RHIG framework.

7.6 | Performance comparison with other frameworks

We demonstrate the utility of the nested hash grid and metadata graph by evaluating the decrease in the metadata query latency and the ingestion time of our initial RADIX[10] framework against other well-known frameworks such as MySQL,[22] GeoMesa.[27]

We evaluated polygon queries, where each system must identify all points intersecting a user-defined polygon, as well as feature queries, where their features must match the user's query. For MySQL queries, an index was created for the features queried, and a spatial index created on the spatial column. Our GeoMesa configuration consisted of a spatiotemporal index, with feature index. For query evaluation experiments, the RADIX cluster and GeoMesa Hadoop[28] cluster each consisted of 25 nodes from the base cluster, with 10 ingest nodes for RADIX and 10 threads for GeoMesa. MySQL was run on a single machine with 8 threads to insert data to maximize insertion speed. The additional time in MySQL required to create indices was not accounted for. Figure 11A shows query evaluation time for each system for both polygon queries (blue) and feature queries (orange). We see that even with a spatial index, polygon queries are slow for MySQL. Additionally, feature queries are 20 times slower in GeoMesa than for MySQL or RADIX, with the latter maintaining a good balance between latencies for both feature and polygon queries.

Figure 11B compares the ingestion time for a 10 GB file into RADIX against the others. RADIX outperforms both systems by ~8x. GeoMesa suffers a penalty during ingestion due to the fact that indexing is not pipelined.

8 | CONCLUSION

We have presented RADIX+, a distributed framework enabling efficient curation of high-velocity spatiotemporal sensor data that supports query-based spatiotemporal analytics with a trackable data-retrieval and updating framework.

RQ1: Our hierarchical nested hash grid scheme allows for both coarse and fine-grained spatial indexing, with the latter being used only in case of a collision of spatial boundaries, thus reducing excessive evaluation time or memory usage.

RQ2: The spatiotemporal analytics queries are evaluated against the in-memory MetadataGraph or the hash grid that helps avoiding disk I/O and providing quick aggregated responses to aid interactive visualizations. Additionally, metadata extraction is pipelined in the ingestion process, thus ensuring eventual consistency between the metadata and raw data.

RQ3: The simultaneous backup of the RHIGDump, containing checksums, with the plot data into IRODS enables our Data Discovery Framework, coupled with the RHIG, to verify the integrity of sub-repositories being maintained in temporary storages by end-users to their counterpart in the historical data-storage on IRODS. RHIG, being an in-memory data structure, also ensures quick and compact evaluation of stale/missing contents of a local repository against a given spatiotemporal query.

ACKNOWLEDGMENTS

This research was supported by the Department of Energy's ARPA-E program, the National Science Foundation [OAC-1931363, ACI-1553685], the National Institute of Food & Agriculture [COL0-FACT-2019], and a Cochran Family Professorship.

CONFLICT OF INTEREST STATEMENT

We are not aware of any potential conflict of interests regarding this submission.

DATA AVAILABILITY STATEMENT

The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

ORCID

Saptashwa Mitra https://orcid.org/0000-0003-2459-8417

REFERENCES

1. Mayer A. Making agriculture part of the climate change solution: researchers seek new ways to sustainably increase food production. *BioScience*. 2019;09;69(10):771-777. doi:10.1093/biosci/biz097
2. Arend D, Lange M, Pape JM, et al. Quantitative monitoring of Arabidopsis thaliana growth and development using high-throughput plant phenotyping. *Sci Data*. 2016;3(1):1-13.
3. Lobet G, Draye X, Périlleux C. An online database for plant image analysis software tools. *Plant Methods*. 2013;9(1):38.
4. Cruz JA, Yin X, Liu X, et al. Multi-modality imagery database for plant phenotyping. *Mach Vision Appl*. 2016;27(5):735-749.
5. Global Phenotype Database for Plant Scientists. 2020, https://cordis.europa.eu/article/id/150886-global-phenotype-database-for-plant-scientists.
6. Pound MP, Atkinson JA, Townsend AJ, et al. Deep machine learning provides state-of-the-art performance in image-based plant phenotyping. *Gigascience*. 2017;6(10):gix083.
7. Tsaftaris SA, Minervini M, Scharr H. Machine learning for plant phenotyping needs image processing. *Trends Plant Sci*. 2016;21(12):989-991.
8. Deutsch P, Gailly J. Zlib compressed data format specification version 3.3. *RFC*. 1950.
9. Merkle RC. Protocols for public key cryptosystems. *1980 IEEE Symposium on Security and Privacy IEEE*. IEEE Computer Society; 1980:122.
10. Pallickara S, Roselius M. Radix: Enabling high-throughput georeferencing for phenotype monitoring over voluminous observational data. *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*. IEEE; 2018:1112-1119.
11. Yang W, Feng H, Zhang X, et al. Crop phenomics and high-throughput phenotyping: past decades, current challenges, and future perspectives. *Molecular Plant*. 2020;13(2):187-214.
12. Heidsieck G. *Distributed management of scientific workflows for high-throughput plant phenotyping*. PhD thesis, Université Montpellier; 2020.
13. Zhao C, Zhang Y, Du J, et al. Crop phenomics: Current status and perspectives. *Front Plant Sci*. 2019;10:714.
14. Chopin J, Laga H, Huang CY, Heuer S, Miklavcic SJ. RootAnalyzer: a cross-section image analysis tool for automated characterization of root cells and tissues. *PloS One*. 2015;10(9):e0137655.
15. Pratapa A, Doron M, Caicedo JC. Image-based cell phenotyping with deep learning. *Current Opinion Chem Biology*. 2021;65:9-17.
16. Paszke A, Gross S, Massa F, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances Neural Informat Process Syst*. 2019;32:1-12.
17. Pang B, Nijkamp E, Wu YN. Deep learning with tensorflow: A review. *J Educat Behavioral Stat*. 2020;45(2):227-248.
18. Chen KH, Su HP, Chuang WC, et al. Apache submarine: a unified machine learning platform made simple. *Proceedings of the 2nd European Workshop on Machine Learning and Systems*. Association for Computing Machinery; 2022:101-108.
19. Alam MM, Torgo L, Bifet A. A survey on spatio-temporal data analytics systems. *ACM Comput Surv*. 2021;54:1-38.
20. Regin R, Rajest SS, Singh B. Spatial data mining methods databases and statistics point of views. *Innovat Informat Commun Technol Series*. 2021;103-109.
21. Dombrovskaya H, Novikov B, Bailliekova A. *PostgreSQL Query Optimization*. Springer; 2021.
22. Widenius M, Axmark D, Arno K. *MySQL reference manual: documentation from the source*. O'Reilly Media, Inc.; 2002.
23. Bradshaw S, Brazil E, Chodorow K. *MongoDB: the definitive guide: powerful and scalable data storage*. O'Reilly Media; 2019.
24. Barsai G. Getting to know ArcGIS PRO. *Photogramme Eng Remote Sens*. 2018;84(4):181-182.
25. Pandey V, van Renen A, Kipf A, Kemper A. How Good Are Modern Spatial Libraries? *Data Sci Eng*. 2021;6(2):192-208.
26. Giachetta R. A framework for processing large scale geospatial and remote sensing data in MapReduce environment. *Comput Graphics*. 2015;49:37-46.

27. Hughes JN, Annex A, Eichelberger CN, Fox A, Hulbert A, Ronquest M. Geomesa: a distributed architecture for spatio-temporal fusion. *Geospatial Informatics, Fusion, and Motion Video Analytics V*. Vol 9473. International Society for Optics and Photonics; 2015:94730F.
28. White T. *Hadoop: The definitive guide*. O'Reilly Media, Inc.; 2012.
29. Zaharia M, Xin RS, Wendell P, et al. Apache spark: a unified engine for big data processing. *Commun ACM*. 2016;59(11):56-65.
30. Perrin JG. *Spark in Action: Covers Apache Spark 3 with Examples in Java, Python, and Scala*. Manning Publications; 2020.
31. Buddhika T, Malensek M, Pallickara SL, Pallickara S. Synopsis: A distributed sketch over voluminous spatiotemporal observational streams. *IEEE Trans Knowledge Data Eng*. 2017;29(11):2552-2566.
32. Mitra S, Khandelwal P, Pallickara S, Pallickara SL. Stash: Fast hierarchical aggregation queries for effective visual spatiotemporal explorations. *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE; 2019:1-11.
33. Mitra S, Rammer D, Pallickara S, Pallickara SL. Glance: A Generative Approach to Interactive Visualization of Voluminous Satellite Imagery. *2021 IEEE International Conference on Big Data (Big Data)*. IEEE; 2021:359-367.
34. Pallickara SL, Pallickara S, Zupanski M, Sullivan S. Efficient metadata generation to enable interactive data discovery over large-scale scientific data collections. *2010 IEEE Second International Conference on Cloud Computing Technology and Science IEEE*. IEEE; 2010:573-580.
35. Subterra. 2022, https://www.subterra.org/.
36. Lal R. Carbon sequestration. *Philosophical Trans Royal Soc B: Biolog Sci*. 2008;363(1492):815-830.
37. Heun JT, Attalah S, French AN, et al. Deployment of lidar from a ground platform: customizing a low-cost, information-rich and user-friendly application for field phenomics research. *Sensors*. 2019;19(24):5358.
38. Eling C, Klingbeil L, et al. UAV real-time. Data use in a lightweight direct georeferencing system. *GPS World*. 2015;44:55.
39. Nørremark M, Griepentrog HW, Nielsen H, Blackmore S. A method for high accuracy geo-referencing of data from field operations. In: Stafford JV, Werner A, eds. *Proceeding of the 4th European Conference on Precision Agriculture*. Wageningen Academic Publishers; 2003:463-467.
40. Malensek M, Pallickara SL, Pallickara S. Galileo: A framework for distributed storage of high-throughput data streams. *2011 Fourth IEEE International Conference on Utility and Cloud Computing IEEE*. IEEE; 2011:17-24.
41. Hill LL. *Georeferencing: The geographic associations of information*. Mit Press; 2009.
42. Niemeyer G. *Geohash*; 2022, http://www.geohash.org/.
43. CyVerse. 2022, https://cyverse.org/.
44. Snappy. 2022, http://google.github.io/snappy/.
45. Google Maps. 2022, https://developers.google.com/maps/documentation/javascript/reference/.
46. Radix. 2022, http://radix.cs.colostate.edu/.
47. Docker. 2022, https://www.docker.com/.

How to cite this article: Mitra S, Roselius M, Andrade-Sanchez P, McKay JK, Pallickara SL. RADIX+: High-throughput georeferencing and data ingestion over voluminous and fast-evolving phenotyping sensor data. *Concurrency Computat Pract Exper*. 2023;35(8):e7484. doi: 10.1002/cpe.7484