Analytic Queries over Geospatial Time-Series Data using Distributed Hash Tables

Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara, *Members, IEEE*

Abstract—As remote sensing equipment and networked observational devices continue to proliferate, their corresponding data volumes have surpassed the storage and processing capabilities of commodity computing hardware. This trend has led to the development of distributed storage frameworks that incrementally scale out by assimilating resources as necessary. While challenging in its own right, storing and managing voluminous datasets is only the precursor to a broader field of research: extracting insights, relationships, and models from the underlying datasets.
The focus of this study is twofold: *exploratory* and *predictive* analytics over voluminous, multidimensional datasets in a distributed environment. Both of these types of analysis represent a higher-level abstraction over standard query semantics; rather than indexing every discrete value for subsequent retrieval, our framework autonomously learns the relationships and interactions between dimensions in the dataset and makes the information readily available to users. This functionality includes statistical synopses, correlation analysis, hypothesis testing, probabilistic structures, and predictive models that not only enable the discovery of nuanced relationships between dimensions, but also allow future events and trends to be predicted. The algorithms presented in this work were evaluated empirically on a real-world geospatial time-series dataset in a production environment, and are broadly applicable across other storage frameworks.

Keywords—Exploratory analytics, predictive analytics, multidimensional data, distributed hash tables

1 Introduction

Data volumes are growing rapidly in several domains. Many factors have contributed to this growth, including *inter alia* proliferation of observational devices, miniaturization of sensors, improved logging and tracking systems, and improvements in the quality and capacity of both networks and disk storage. Analyzing such data provides insights that can be used to guide decision making. To be effective, analysis must be timely and cope with data scales. The scale of the data and the rates at which they arrive make manual inspection infeasible. The primary objective of this paper is to simplify the process of gleaning insights and allow analysts to arrive at decisions faster through continuous, interactive exploration.

In this study, we consider geospatial, time-series datasets. Data with spatio-temporal properties are commonly found in epidemiology, atmospheric and climate modeling, environmental and ecological systems, traffic and congestion analysis, and commercial sales tracking systems. Observations or *features* of interest are measured across geographical locations over long periods. Data points include both coordinates (expressed as latitude-longitude tuples) and also the time at which the observation was recorded. Each data point (or observation) is multidimensional, encompassing several features of interest such as humidity, wind speed, temperature, pressure, etc.

Gleaning insights involves exploring characteristics of the data. Besides identifying properties associated with individual features, this also includes identifying relationships: how features correlate with each other and also how their values change with respect to each other. Also of interest are the probabilities associated with these features, for example pairwise conditional probabilities between features that allow use of Bayesian statistics to predict one feature when measurements are unavailable for the other feature. A broader insight users might be interested in is how the feature space is impacted by time and geography. Our objective is to provide a set of queries that allow:

- Speeding up the discovery process
- Fast, near real-time exploration of the feature space
- A base to facilitate more complicated processing (using MapReduce [1])

We are interested in supporting two types of analytic queries: *exploratory* and *predictive*. Exploratory queries help with understanding the feature space: how features evolve, and how they relate to each other. The objective in exploratory analytic queries is to get a deeper understanding of the data. An example of an exploratory query is: "Retrieve locations where temperature is negatively correlated with humidity". In some cases, an exploratory query might return a probability density function (PDF) instead of a concrete answer. These queries may also involve summary statistics associated with features at both global scales and for a particular geographical scope.

Predictive analytic queries assist in forecasting what is likely to happen in the future. An example of a predictive analytics query is: "What will the temperature be in Seattle at 4:00 pm tomorrow?" Another example of predictive analytics involves confirmation or rejection of the statistical *null hypothesis*. Consider the case where a store has run an advertising campaign and would like to determine whether or not it resulted in more sales. We can contrast the mean and standard deviation of sales after the point where the campaign started, and use specified *p-value* thresholds to either reject or validate the null hypothesis that the campaign resulted in no change to sales. Another example would entail confirming the failure rate of new instrumentation; unlike the previous example, the manufacturers of the instrument wish to validate the null hypothesis that the instruments are reliable and the failure rate has not been impacted by the changes that were made.

- *M. Malensek, S. Pallickara, and S. Pallickara are with the Department of Computer Science, Colorado State University, Fort Collins, CO 80523. E-mail: {malensek,sangmi,shrideep}@cs.colostate.edu*

1.1 Challenges

Supporting exploratory and predictive analytics over voluminous, time-series, and geospatial datasets involves several challenges, including:

1) *Data Volumes*: The quantity and overall size of the data is high, precluding visualization of every point or sifting through the data sequentially.
2) *Continuous Arrivals*: Data is streamed into the system on a regular basis, requiring the knowledge base to be constantly updated.
3) *Data Dimensionality*: Each observation has several features (dimensions) of interest. Furthermore, evolution of the feature space has spatio-temporal characteristics that must be accounted for.
4) *Managing the Memory Hierarchy*: While main memory accesses complete in nanoseconds, disk access times are much slower. Due to dataset sizes, queries often target large quantities of information, which makes careful management of in-memory data structures critical.
5) *Circumvention of I/O Hotspots*: When some of the distributed nodes take a disproportionate amount of the storage load, hotspots will occur. This means that data partitioning must be done in a balanced manner.

Given the data volumes being dealt with, it is possible that some queries will be compute-intensive and hence have longer turnaround times. We must balance these processing overheads to ensure high throughput is maintained.

1.2 Research Questions

The goals and challenges associated with this study led us to formulate the following research questions:

1) How can we discover and account for relationships between features at a particular geographic scope? How can this be used to inform predictions and explorations?
2) Can we account for the spatio-temporal characteristics of different features at various geographical extents?
3) How can we ensure near real-time query evaluation given the data volumes and the speed differential between the memory and disk access times?
4) How can we facilitate exploration at scale while maintaining fast response times and high overall query evaluation throughput?
5) Given the continuously arriving data streams, how can we maintain an up-to-date knowledge base?

1.3 Summary of Approach

Our approach to supporting analytic queries over voluminous, time-series, geospatial datasets takes a holistic view that targets: (1) alleviation of I/O hotspots, (2) coping with the differential between memory and disk speed, (3) construction of multiple models to ensure specificity, and (4) online updates to summary statistics that capture feature space evolution.

Information is managed by our distributed storage framework, Galileo [2], [3], [4]. Galileo is based on Distributed Hash Tables (DHTs), which have well-known scaling properties [5], [6], [7]. In the system, datasets comprise a set of observations with spatio-temporal attributes. The observations may be measurements reported by instruments, annotations from subject matter experts on observed phenomena, or the result of analytics operations. Data is dispersed based on Geohashes [8] that are computed from the latitude-longitude tuples associated with each data point. A subset of nodes is responsible for each particular Geohash, with dispersion over the subsets based on data density and chronological ordering.

To cope with I/O latencies, ***DiscoveryGraphs*** maintain feature synopses in memory. These include statistics at each vertex, with leaf nodes pointing to physical data blocks on disk. DiscoveryGraphs improve upon traditional indexing techniques by providing direct access to higher-level information about the underlying data points, including their distributions, trends, and cross-feature correlations. This enables real-time knowledge extraction to derive both coarse- and fine-grained insights across voluminous datasets. The graphs can also be reoriented to reduce their memory footprints (via reduction of vertices) or query evaluation times (via reduction of edges that must be traversed). The breadth and depth of information represented by the DiscoveryGraph directly impacts our ability to avoid costly disk accesses.

DiscoveryGraph queries are evaluated as MapReduce computations. Galileo storage nodes that do not contain relevant data points are eliminated from the search space during preprocessing, and then the queries are resolved in parallel across the remaining nodes. During the reduce phase, DiscoveryGraph instances are merged to produce a final result set that can be traversed and used directly for analysis. Galileo supports both singular and continuous queries [9].

Both exploratory and predictive queries depend on the construction of data models that capture feature relationships. We rely on a multiplicity of distributed models rather than a single all-encompassing model. This approach allows for specificity, where each model is responsible for a particular geographical scope. Since the models are dispersed over multiple machines and localized, this approach scales with increases in data volumes as well as reductions to finer-grained geographical scopes. Models include multiple linear regression and artificial neural networks to capture both linear and non-linear relationships, respectively. To avoid the effects of collinearity, we detect and account for features that are highly correlated.

1.4 Contributions

This paper describes a distributed framework for evaluation of analytic queries over multidimensional time-series datasets. The work described makes the following contributions:

1) We model feature characteristics at various scales, including how the features evolve spatio-temporally and with respect to each other.
2) Support for validation or rejection of null hypotheses based on specified p-value thresholds.
3) Dynamic construction of linear and nonlinear models, including ARIMA for time-series projection.
4) Probabilistic estimates for features, as well as queries that return results in the form of PDFs or joint PDFs.
5) The algorithms described are broadly applicable to other storage systems, such as Cassandra [5], Dynamo [6], and Chord [7], among others.

A key innovation in this work is achieving these contributions at scale and in a timely fashion. Our empirical evaluations are performed over a real-world ***1 Petabyte*** dataset encompassing over ***20 billion*** files.

1.5 Experimental Setup

The dataset used in this study was sourced from the National Oceanic and Atmospheric Administration (NOAA) North American Mesoscale Forecast System (NAM) [10]. The NAM conducts frequent collection of atmospheric data in GRIB format, and contains several features across a timespan of over ten years. Some of the features highlighted in this study were the surface temperature, visibility, cloud cover, precipitation, humidity, snow fall, vegetation, and albedo.

Our data ingest process involves reading data from the NAM and streaming it into the system as samples from discrete spatial locations. Each sample consisted of about 10 KB of raw feature data, along with another 40 KB of graphical tiles designed to be overlaid on a map for visualization. The total dataset encompassed 20 billion files at approximately 50 KB each for a total dataset size of 1 Petabyte.

The benchmarks included in this paper were carried out on a heterogeneous 78-node cluster consisting of 48 HP DL160 servers (Xeon E5620, 12 GB RAM) and 30 HP DL320e servers (Xeon E3-1220 V2, 8 GB RAM). Each machine was equipped with four hard disk drives. The hardware was divided into 13 groups of 6 nodes each. Galileo was run under the OpenJDK Java runtime, version 1.7.0_65.

1.6 Paper Organization

Section 2 outlines the system architecture used in the paper, while Section 3 describes the DiscoveryGraph indexes and their query resolution process. Sections 4 and 5 describe our exploratory and predictive analytics functionality, respectively, followed by a performance evaluation in Section 6. Section 7 provides a survey of related work, and Section 8 outlines our conclusions and future research direction.

2 System Overview

Galileo is a high-throughput distributed storage framework that supports range-based, exact-match, geospatial, and approximate queries over multidimensional time series data. The system is implemented as a Distributed Hash Table (DHT), allowing incremental scalability and flexible data partitioning. Contrasting with the traditional DHT model used in Chord [7] or Pastry [11], Galileo is a *zero hop* (or *one-hop*) DHT. This means that enough state information is maintained at each node in the DHT to route requests directly to their destination without intermediate hops through the network. Examples of other zero-hop DHTs include Amazon's Dynamo [6] and Apache Cassandra [5].

To support the management of voluminous scientific datasets, Galileo places *storage nodes* in several groups (or subgroups), creating a hierarchical network layout. Combined with a multi-tiered hashing scheme, this functionality enables a balance to be struck between uniform load balancing and ordered data placement. Galileo includes several domain-specific features to ease working with scientific datasets such as built-in spatial and temporal data structures, along with support for scientific formats such as NetCDF [12], BUFR, or HDF5 [13].

While most DHTs support only key-based retrieval semantics, Galileo provides expressive query support through a multi-tier indexing scheme. Each storage node in the system maintains a unique *metadata graph* that indexes local files, while the globally-distributed *feature graph* [3] contains a coarse-grained representation of all the information in the system. Both graphs rely on an eventually-consistent gossip scheme to disseminate state information. The metadata graph is primarily used for locating and retrieving records, whereas the feature graph helps reduce the search space of distributed queries by eliminating storage nodes that do not contain relevant data.

Users do not need to be aware of the structure or contents of the graphs, and queries are formulated in an intuitive, declarative syntax. Any of the nodes comprising the DHT can accept and evaluate queries. The receiving node computes the *subset* of nodes likely to hold data of interest; we use the feature graph and Bloom Filters [14] to ensure that there are no false negatives generated by this process. The receiving node then forwards the query on to applicable nodes for evaluation.

2.1 Data Partitioning

We use the Geohash [8] algorithm to partition data across the storage groups in Galileo. Geohash divides the Earth into a hierarchy of bounding boxes that are referenced by Base 32 strings. The longer the string, the smaller (and therefore more precise) the spatial bounding box. For example, the coordinates of N 30.3321 W 81.6556 would translate to the Geohash string *DJMUTCU1Q*. Figure 1 demonstrates two successive iterations of the algorithm, where our example location would fall within the *DJ* and *DJM* bounding boxes. Each subsequent iteration of the algorithm adds another character to the string and introduces 32 new spatial subdivisions.

Geohash strings are used by Galileo to succinctly represent spatial points or regions, as well as assign data to groups in a flexible manner that enables both scaling up or down as necessary. In this study, the first two characters of the Geohash for a data point are used to determine group membership. This results in groups being responsible for regions of approximately 1030×620 km, which also helps reduce the search space of distributed spatial queries.

2.2 Indexing and Retrieval Operations

Galileo extracts metadata from incoming records and organizes it in hierarchical, graph-based indexes to enable fast query resolution and retrievals. Each level in the hierarchy manages a unique feature type, with vertices representing configurable ranges of values known as *tick marks*. Tick marks allow varying degrees of index granularity to be achieved; given the data volumes Galileo is designed for, simply storing every discrete value in memory is not feasible. This results in similar values being condensed to a single index entry — values of 0.091 and 0.092 might be stored on a vertex managing values from 0.0 to 0.1, for instance. These tick marks can be chosen explicitly by the user or generated autonomously by the system [15]. One key contribution proposed in this work involves maintaining synopses to improve the expressiveness of a vertex without storing individual values in memory.

As data is streamed into the system, its metadata is extracted and indexed before being stored to disk. The indexing process begins by transforming values into graph *paths* that represent an ordered traversal through the index hierarchy. As the graph is traversed, neighboring vertices are created or updated as necessary to reflect the data points in the new path. The final step in an insertion depends on the type of graph being updated: in the case of a metadata graph, the last vertex in the path contains a list of relevant file locations on disk. On the other hand, final vertices in the global feature graph contain a list of storage nodes with matching data points to aid in search space reduction. Figure 2 provides an illustrative example of a sample graph layout and its feature hierarchy. The graphs can also be *reoriented* at run time to improve query response times or conserve memory by modifying feature ranks, which impacts the positioning and quantity of vertices as well as the number of edges in the graph.

Retrieving information from the graphs proceeds in a manner similar to data insertion, with each clause specified in the query used to incrementally narrow down the search to a subset

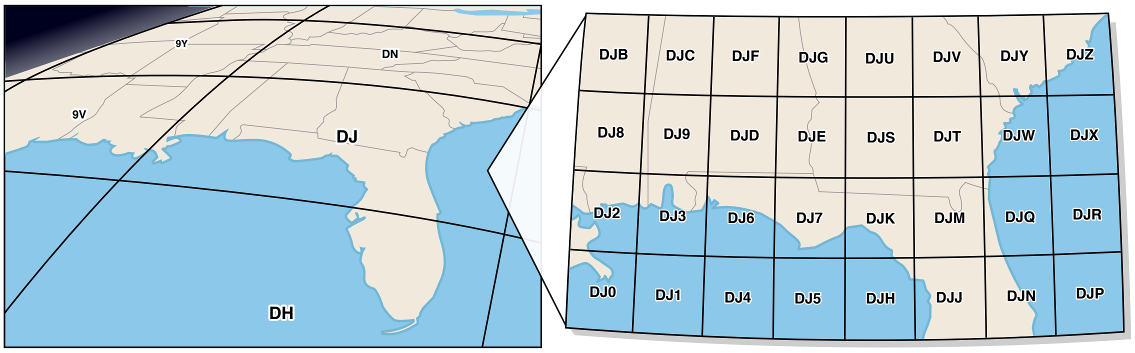


Fig. 1. A visual overview of successive spatial subdivisions generated by the Geohash algorithm. In this example, the area represented by Geohash *DJ* is divided into 32 smaller subregions, each about 133×155 km in size.

of relevant vertices. These vertices are streamed back to clients as orientation-neutral paths that can be organized as a tabular display or used to create traversable *result graphs*. If further analysis is warranted, the graphs allow the system to guide the execution of MapReduce [1] computations on the cluster that exploit data locality. This abstraction also helps reduce the amount of data that client hardware has to handle, allowing files on physical media to be accessed selectively.

2.3 Spatiotemporal Queries

Geohashes are augmented by bitmap-based *Geoavailability Grids* [4], [16] to provide higher-precision spatial lookup functionality. This includes queries that are constrained by arbitrary polygons, and also enables proximity queries such as, "retrieve readings closest to Toronto, Ontario with a temperature lower than 12° C". Similar to the feature graph, Geoavailability Grids also support search space reduction as a preprocessing step. Queries that include both spatial and feature constraints can be combined, essentially resolving all data points that intersect both parameters.

Temporal information can be stored directly in the feature and metadata graphs as either *timestamps* or *intervals*. Besides the usual query operators, interval-based records can be selected using Allen's Interval Algebra [17], [18], an expressive calculus for temporal reasoning. This enables queries that make inexact comparisons across temporal events; for instance, when finding points that overlap, lie outside a specific range, or happen during a common time period.

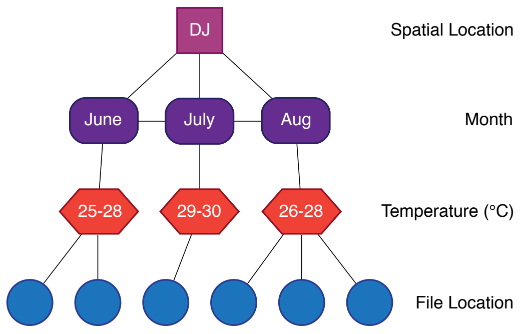


Fig. 2. An example graph index; traversals lead to the location(s) of files in the network (in the feature graph) or on disk (in the metadata graph).

3 The DiscoveryGraph: Methodology

By quantizing indexed values and reducing the overall search space, the feature graph enables low-latency distributed queries across voluminous datasets. However, quantization with adaptive tick mark boundaries still results in a small reduction of index fidelity. Furthermore, simply locating relevant records is only the first step in most analytics activities; given a subset of the data, distributed computations or batch processing must occur to glean insights from the dataset. To address these use cases, we developed the ***DiscoveryGraph***, an index that enables efficient resolution of both *exploratory* and *predictive* analytic queries.

The DiscoveryGraph improves upon (and supersedes) the feature and metadata graphs by making knowledge extraction part of the indexing process. As records are streamed into the system, the DiscoveryGraph maintains a variety of statistics at each vertex that describe the underlying data distributions and their interactions. Maintaining this information boosts index fidelity, greatly improves the speed of queries meant to generate synopses, and serves as a platform for development of additional functionality. The DiscoveryGraph operates in two modes: a coarse-grained, globally distributed, and *eventually consistent* instance that replaces the feature graph, along with finer-grained local instances maintained on each storage node to replace the metadata graphs.

3.1 Vertex Statistics

By placing incoming data points into discrete bins or "tick marks," Galileo can index voluminous datasets quickly and evaluate queries without excessive memory consumption. While our autonomous tick mark reconfiguration functionality does provide some degree of insight into the data distributions, the DiscoveryGraph further improves expressiveness by calculating statistical synopses of the information that was placed under each vertex. The synopses include:

- Number of data points placed at the vertex
- Smallest and largest values observed
- Mean, variance, and standard deviations

Given that the range of values placed at a vertex is generally small, these statistics provide finer-grained insight into the behavior of the underlying data. Since new information is continually streaming into the system, we use the online method proposed by Welford [19] to avoid re-calculating these values each time a new data point is added. This involves maintaining the count of samples observed thus far, n, along with the current mean of the values, $\bar{x}$, and the sum of squares of differences from the current mean, S_n. When a new data point x_n is added at a particular vertex, the components are updated as follows:

$$\bar{x}_0 = 0, S_0 = 0$$
$$\bar{x}_n = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}$$
$$S_n = S_{n-1} + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n)$$

With this information, the sample variance can be calculated as $\sigma^2 = S_n/(n-1)$. These calculations are lightweight, enabling us to update statistics for each vertex affected by the insertion of a new multidimensional data point efficiently. In situations that require updates or removals, the running statistics can be adjusted by effectively reversing the procedure outlined above. When a query or system operation involves multiple vertices, the statistics from each vertex can be merged to provide an aggregate summary. A sample merge operation of vertices A and B follows:

$$n' = n_A + n_B$$
$$\bar{x}' = \frac{n_A\bar{x}_A + n_B\bar{x}_B}{n'}$$
$$S' = \frac{S_A + S_B + n_A n_B(\bar{x}_A - \bar{x}_B)^2}{n'}$$

Table 1 evaluates the performance of this algorithm by testing each operation over 1000 iterations and averaging their computational overhead in microseconds. Updating the statistics consumes the most CPU time because it triggers several calculations, but all five operations (update statistics, remove data point, calculate mean and standard deviation, merge two instances) complete in less than a microsecond on average; even while accounting for additional traversal overhead, updating a DiscoveryGraph with 100 unique features takes less than 1 ms.

TABLE 1
A performance evaluation of vertex statistics maintenance operations, averaged over 1000 iterations.

Operation	Mean Time (μs)	σ (μs)
Add Data Point	0.942	0.137
Remove Data Point	0.913	0.126
Calculate Standard Deviation	0.416	0.023
Calculate Mean	0.339	0.025
Merge Instances	0.279	0.018

3.2 Data Insertion

Data is added to the DiscoveryGraph in a fashion similar to the feature and metadata graphs; incoming multidimensional data points are first converted to *paths* that represent a hierarchical graph traversal. Once a path is constructed, summary statistics are updated (or created, if necessary) as the vertices are placed in their appropriate tick mark ranges. The final vertex in the path includes a list of pointers to files on physical storage media that match the traversal hierarchy. For globally-distributed DiscoveryGraphs that augment the functionality provided by the feature graph, the final vertices contain a list of Galileo storage nodes that hold relevant data.

Figure 3 contains a demonstration DiscoveryGraph populated with five records from our dataset that were extracted from Florida, USA, in July of 2013. Each record contains three dimensions: temperature, cloud cover, and humidity. The feature values of the paths are shown in Table 2, identified by the letters A through E. Note that each level in the graph hierarchy contains a particular feature type, and that the summary *count* at each level sums to the total number of data points seen at the parent vertex. Vertices are split or merged based on their *coefficient of variation*, which is the relationship between the standard deviation σ and the mean μ of the values: $C = \sigma/\mu$. This allows vertices with high feature dispersion values (such as the cloud cover vertex handling values from 50-100%, marked Δ) to be broken up into smaller ranges that exhibit less dispersion, thus improving the specificity of the summary statistics [15].

Table 3 provides an intuition for how summaries in the DiscoveryGraph change over time by updating each statistic at the 50-100% cloud cover vertex (marked Δ in Figure 3) as the paths in Table 2 were inserted. Column **A** contains the vertex state with only one data point (66%). Note that the *count* increases up until column **D**, where the path insertion did not involve the particular vertex being used for demonstration. Once path **E** has been inserted, the summary statistics match those shown in Figure 3.

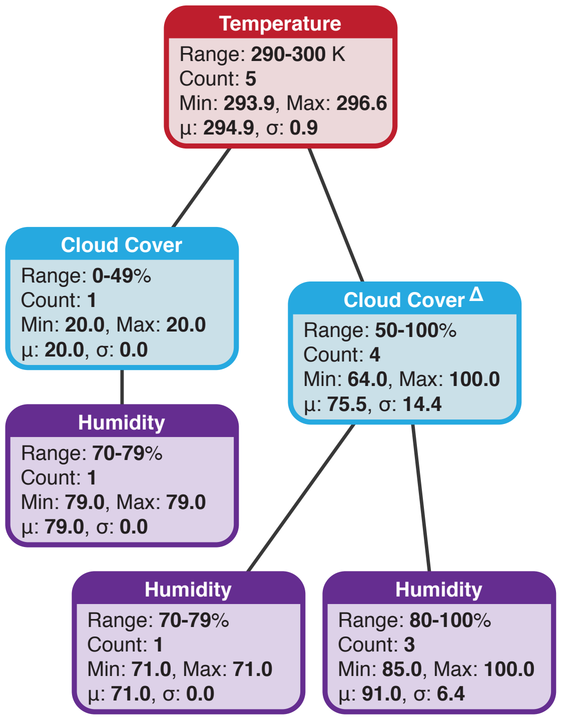


Fig. 3. A DiscoveryGraph populated with multidimensional data points from Table 2. File/node pointers are omitted for brevity.

TABLE 2
Sample multidimensional data points used to create the DiscoveryGraph shown in Figure 3.

Path	Temperature	Cloud Cover	Humidity
A	295.4 K	66.0%	88.0%
B	296.6 K	64.0%	100.0%
C	294.5 K	72.0%	71.0%
D	293.9 K	20.0%	79.0%
E	294.1 K	100.0%	85.0%

TABLE 3
A step-by-step view of DiscoveryGraph statistics at the cloud cover node handling values from 50-100% (Δ in Figure 3) as each successive path from Table 2 is inserted into the graph.

Statistic	A	B	C	D	E
Count	1	2	3	3	4
Min	66.0	64.0	64.0	64.0	64.0
Max	66.0	66.0	72.0	72.0	100.0
Mean (μ)	66.0	65.0	67.3	67.3	75.5
St. Dev. (σ)	0.0	1.0	3.3	3.3	14.4

3.3 Managing Vertex Specificity

Merging summary statistics across vertices enables Galileo to provide both a coarse- and fine-grained vantage point over the entire dataset: when a linear correlation or event of interest only occurs under specific conditions, statistics at the appropriate leaf vertices capture relationships without being influenced by the rest of the data. On the other hand, vertices near the root of the graph tend to capture highly general information about the subgraphs beneath them. By managing the inherent trade-offs of this *specificity gradient*, we can ensure accuracy is maintained while improving query response times and overall system performance. Figure 4 illustrates this concept, where two sample graphs have been constructed using the same data but different orientations. Vertices in the figure are highlighted with four different colors to represent four unique feature types; as vertex positions change their respective counts change accordingly.

DiscoveryGraph orientations are configured manually by the user or dynamically based on usage trends observed by the system. This requires reorienting the graph to place the desired fine-grained synopses at the leaf nodes, while the coarse-grained synopses are placed near the top of the hierarchy. For example, if a large quantity of queries involve the average temperature of a region, placing temperature vertices at the top of the hierarchy would avoid traversing further through the graph. On the other hand, if a feature is always accessed along with many other features (such as a request for subgraphs containing a snow depth greater than 1 cm, temperatures less than 0° C, and a time span of November through February) then it should be placed towards the bottom. Galileo maintains a table of query *feature sets* to monitor these trends. Additionally, ensuring the graph can be reoriented requires each leaf node to maintain *path synopses* that track each incoming path so vertex statistics can be redistributed during a reorientation.

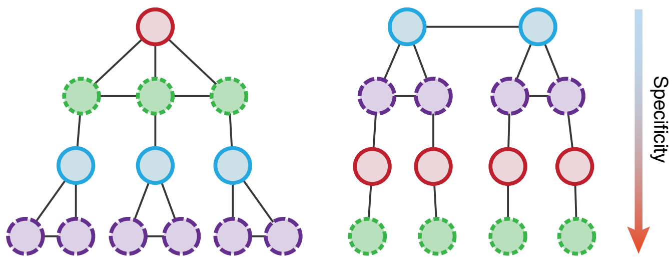


Fig. 4. Two graphs modeling the same data with different orientations. Vertices near the top of the hierarchy contain generalized information, whereas vertices closer to the bottom are highly specific.

3.4 Integrated Analytic Models

While the DiscoveryGraph provides a wealth of information about the underlying dataset, its interfaces and functionality also form the basis for more advanced models that facilitate analytic queries. Both exploratory and predictive queries (discussed extensively in the following sections) build on this technology, with online models ranging from correlation analysis and two-dimensional linear regression to neural networks and ARIMA forecasting. Exploratory queries generally involve the discovery of unknown patterns or attributes in the dataset, so we ensure that their models are lightweight (about 8 bytes per feature) and available at every vertex. On the other hand, predictive queries are intended for situations that require guidance from the user to define the problem domain; while a multiple linear regression or ARIMA model could incorporate every reading received at a storage node, targeted insights are usually much more valuable. Predictive models involve two major phases: *creation* and *maintenance*.

During the creation of predictive models, *ModelGraphs* are used to constrain the scope of model inputs to particular subsets of the DiscoveryGraph. User-defined queries filter and select the vertices that compose a ModelGraph, resulting in a bounded "view" of the overall dataset. In cases where historical data is required for training, creation of a ModelGraph may involve disk accesses. Once the ModelGraph is created, nodes under its purview will trigger the maintenance process as they receive new observations. Maintenance intervals determine how often the models are updated, which can be continuous, periodical, or performed based on the availability of a specific number of observations. By default, models are constrained to the same size geographical regions as storage node groups (1030 × 620 km in this study) and 12-month time spans, but any geographical, chronological, or feature scope can be incorporated.

Limits on both the scope of model inputs and their lifetimes can be specified by the system administrator to manage resource usage. By default, predictive models that have been unused for more than 24 hours are automatically garbage collected and removed from main memory. However, state variables and model calculations generally require consistent and predictable amounts of memory and CPU time, so resource limits can be configured to allow certain models to be maintained for longer time spans (or even indefinitely, with removal performed manually). If storage space is available, these models can also be serialized to disk for future use. In situations where models incorporate future observations, their lifetimes are specified upfront during creation.

To distribute workloads and exploit data parallelism, models with inputs that span multiple storage nodes are maintained separately and can also be queried independently to analyze how different geographic regions or distributions of the data impact the models. However, the most common query pattern involves merging model instances prior to their use to generate an aggregate model. This process is carried out in parallel by our MapReduce framework.

3.5 Query Evaluation: MapReduce Framework

While the summary statistics and models in the DiscoveryGraph are lightweight and incur minimal processing costs, they provide nuanced insights about the underlying dataset. Galileo includes rich retrieval functionality to allow this information to be queried and used individually by end users, aggregated across dimensions, or used to locate phenomena and features of interest autonomously. DiscoveryGraph queries are implemented as dynamic MapReduce [1] computations that are *pushed* out to relevant storage nodes for parallel evaluation. A query may involve subsets of graph nodes or outputs from analytic models; Figure 5 provides an example of both.

As noted previously, the first step of a distributed query in Galileo is to eliminate nodes that do not contain relevant data from the search. This involves performing a lookup on the global DiscoveryGraph, which receives state updates that are gossiped through the network in an *eventually consistent* manner. While the search space reduction process does not guarantee that the remaining nodes will have relevant information, it does not produce false negatives. An additional benefit afforded by the summary statistics maintained at each vertex is that irrelevant storage nodes can be detected with greater accuracy. For example, consider a query requesting records with humidity values greater than 32%. In this case, a vertex that is responsible for data points ranging from 25-35% produces a positive match. However, if the vertex in question has a minimum value of 30%, the false positive can be avoided.

After the initial pruning process, queries are pushed to candidate storage nodes for evaluation in the *Map* phase. Query results are represented by traversable *subgraphs* or serialized model state information, which is then merged by a subset of the nodes in the *Reduce* phase before being streamed back to the client. DiscoveryGraph synopses facilitate the reduction phase by providing additional information about the data distributions of the query; if a single storage node contains a large percentage of the records being queried, it is selected as a reducer to decrease serialization overhead and network I/O.

The reduce phase operates in one of three modes depending on the end user's goals, requirements, or hardware constraints. These modes include the *tabular layout*, *traversable graph*, or *summary graph*. Modes are selected during query submission, and each has its own set of trade-offs and use cases:

- A ***Tabular Layout***, which provides raw outputs with minimal post-processing occurring in the reduce phase. Results are streamed as table rows directly to clients.
- ***Traversable Graphs*** that retain relationships between records. Graph paths that appear more than once are represented as a single path, reducing output sizes.
- ***Summary Graphs*** that merge DiscoveryGraph vertices at the same level in the feature hierarchy to summarize the data with fewer records.

The differences between each of these output types are shown in Figure 6. Tabular records are best suited for model outputs or streaming applications that prioritize fast responses, while the traversable graphs offer additional data inspection and exploration features at smaller output sizes. Summary graphs are ideal for fast, ad-hoc exploration and analysis of the feature space, allowing large quantities of data to be sifted and represented in a compact fashion.

```
QUERY Humidity, Temperature WHERE
Date IN [Feb 2014 TO Jun 2014]
AS SUMMARY_GRAPH
```

```
PREDICT USING 'Temp-DJ_ARIMA'
WHERE Date =
    [Jun 15 2013 TO Jun 30 2013]
AS TABULAR
```

Fig. 5. Example queries and requested output types. The first query retrieves temperature and humidity statistics, whereas the second issues a model query to predict temperature values at Geohash *DJ*.

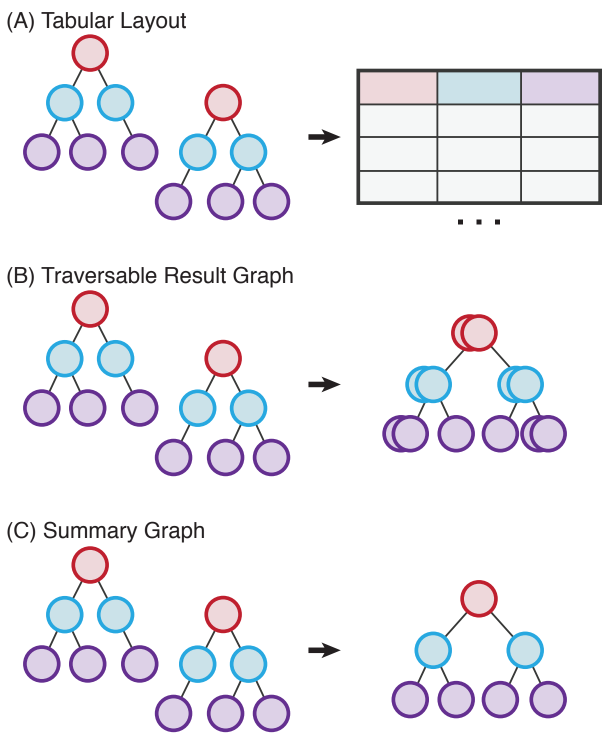


Fig. 6. Output dataset types supported by Galileo. Intermediate results (left) are transformed by the multi-stage MapReduce process into the final output data structures (right).

4 Exploratory Analytics: Methodology

Often, voluminous datasets contain a multitude of relationships and insights that may not be obvious or intuitive. For this reason, we provide support for *exploratory analytics* functionality to pinpoint information, trends, and properties of interest in the underlying data. These components include online algorithms for generating both broad and specific statistical synopses, discovering correlations between dimensions, evaluating the significance of feature variations, and analyzing the probabilities associated with events.

4.1 Detecting and Quantifying Feature Relationships

Most real-world systems involve several interrelated features. These relationships may represent dependencies or correlations between events or variables; for instance, absolute humidity is impacted by temperature and pressure, and precipitation may be classified as rain, hail, or snow depending on the current temperature (along with other atmospheric conditions). While it is important to note that correlation does not imply causation, the relationships between variables may prove to be valuable from a research perspective and warrant further study.

To autonomously track correlations between features, we augmented the DiscoveryGraph by adding the capability to calculate the Pearson product-moment correlation coefficient (PCC) across two-dimensional feature combinations. PCC measures the degree of a linear relationship between two variables, ranging from $[+1, -1]$. A correlation coefficient of $+1$ or -1 between variables represents a perfect positive or negative linear relationship, respectively, whereas a value of 0 would imply the absence of a linear correlation. Both sides of this spectrum can provide useful insights into how features interact and influence one another.

A correlation query is issued by specifying feature tuples of interest along with any additional constraints (such as limiting results to a specific region or time), and will return a set of correlation coefficients and their two-tailed p-values. p-values represent the probability of producing the same test results if there was no relationship between the features. In other words, a very low p-value (often less than 0.01 or 0.05, depending on user requirements) represents a significant relationship that is unlikely to occur by chance. Client applications have the option to request that these statistics be merged into a single set of correlation coefficients or returned as subgraphs for further exploration.

Table 4 contains correlations found between several different features during the month of July in Wyoming, USA, along with their corresponding p-values to test the significance of the relationships. These results indicate a negative correlation between sky visibility and total precipitation, a positive correlation between humidity and precipitation, and a negligible relationship between temperature and ground vegetation. As one might expect, snow depth did not exhibit correlations with any of the other features during this summer month, highlighting the fact that both temporal and spatial aspects must be considered when drawing conclusions from the data.

TABLE 4
Correlations and p-values between several features recorded during July in Wyoming, USA.

Feature A	**Feature B**	**Correlation**	**p-value**
Precipitation	Visibility	-0.49	3.39E-39
Humidity	Precipitation	0.37	3.08E-22
Pressure	Visibility	0.36	1.61E-20
Vegetation	Temperature	-0.06	0.12
Temperature	Snow Depth	0.0	1.0

In addition to the correlation coefficient, we also maintain enough information to generate linear models between feature types. These models are provided by two-dimensional linear regression, and include the coefficient of determination, r^2, which measures the predictive quality of the linear models. Features that exhibit strong linear correlations can often be used to make accurate predictions. Our implementation follows the least-squares approach, wherein a straight line is fit to the data such that the sum of squared residuals is minimized. The slope and intercept of the regression line can be retrieved by clients, or the model can be used directly to extrapolate the value of a particular feature.

While updating vertex statistics is computationally lightweight, several more operations are necessary to provide the correlation coefficient, linear regression functionality, and r^2 values. Table 5 describes the performance characteristics of the two-dimensional vertex statistics management capabilities added to the DiscoveryGraph. Once again, adding new data points incurs an upfront cost, but subsequent calculations are fast: all operations were completed in less than 2 μs.

4.2 Significance Evaluation

Regions in close spatial proximity often exhibit similar climactic trends, but variations in the surrounding geography may result in significantly different weather patterns. Similarly, the average temperature for a particular region will vary from year to year, but a researcher would be most interested in situations where the difference was *statistically significant*. While Galileo supports calculating the significance of the variations between two samples using p-values, we have also added functionality to enable the evaluation of queries that invert this problem: given a feature or set of features, a *significance query* locates relevant locations, time spans, or data points across the overall dataset where variations are statistically significant. The features in question are often chosen based information gained from previous exploratory analysis.

TABLE 5
Performance evaluation of the augmented two-dimensional vertex statistics mechanism, averaged over 1000 iterations.

Operation	**Time (μs)**	**σ (μs)**
Add Data Point	1.489	0.044
Calculate Correlation	0.723	0.023
Calculate r^2	0.101	0.003
Predict y	0.381	0.016
Merge 2D Instances	0.919	0.103

We implement significance queries as multi-stage MapReduce computations; each relevant storage node begins by evaluating a standard query with the given parameters, and then retrieves the statistical synopses of the feature in question from the resulting subgraph. To reduce network I/O costs, the synopses are combined and transmitted to the node that was projected to contain the largest number of matching records by the initial DiscoveryGraph pruning step, which then performs *Welch's t-test* between synopses. A *t-test* uses the *t-distribution* to determine whether there is a relationship between observations, and Welch's version of this test is used in situations where the variances between two samples are possibly unequal. This produces a set of p-values that measure inter-group variance. In the final step, both the individual and group results are streamed back to the client based on a *p-value threshold* that is specified at query time based on the client's desired significance level. This reduces the amount of information that must be processed on the client side and enables users to quickly evaluate perceived relationships across the dataset. To test this functionality, we submitted queries involving randomized feature sets with significance levels of either 5% or 1% to our cluster. The results were produced in 41.0 ms, on average (over 1000 iterations) with a standard deviation of 0.13 ms.

4.3 Probability Density Queries

Feature analysis not only involves the values a particular feature has taken, but also the probabilities associated with them. For instance, high temperatures are more likely to occur during summer months and less likely during winter months. The probability densities associated with these events provide insight as to how features behave under specific conditions, as well as how they evolve over time. Figure 7 contains the results of a *probability density query*, illustrating the likelihood of measurable precipitation occurring as the amount of atmospheric cloud cover changed in Wyoming, USA during the month of July in 2013. While the presence of clouds being associated with a heightened probability of rain is a fairly intuitive relationship, this query yields the probability of precipitation as a function of cloud cover; with this information, we know that rain is highly unlikely when when cloud cover is under 50% in this region. The next step in our analysis might involve inspecting the relationship between cloud cover and precipitation across a variety of spatial locations.

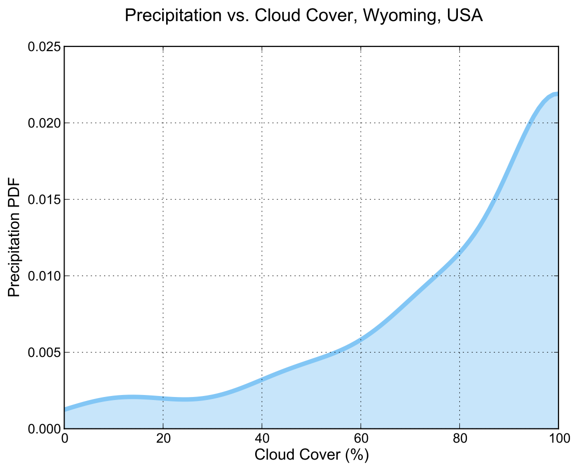


Fig. 7. Results of a probability density query showing the probability of precipitation occurring as the amount of atmospheric cloud cover varies.

Client applications can issue probability density queries across ranges of feature values, geospatial regions, or time spans. For query shown in Figure 7, Galileo uses feature synopses in the DiscoveryGraph to determine the number of times precipitation was greater than zero under the specified constraints and produces a collection of vertices to be passed on to the reduce phase. During the reduce phase, vertices are merged to produce a final histogram that contains the counts for each tick mark range. The resulting set of vertices can either be returned directly to the client to produce a histogram, or passed through a *kernel density estimation* to compute the approximate probability density function. On our test dataset, this query involved a total of six storage nodes and completed in 18.26 ms (averaged over 1000 iterations, with a standard deviation of 0.11 ms).

4.4 Joint Probability Queries

To model cross-feature influences, *joint probability queries* provide the probability densities of multiple events or feature values occurring at the same time. For example, consider the joint influence of temperature and humidity on the human body's perception of heat: days with high temperatures coupled with high humidity *feel* hotter than those with the same temperature and a low relative humidity (often colloquially referred to as "dry heat"). Figure 8 contains the probability density of each temperature-humidity tuple in Florida, USA during the month of July in 2013. Note the prominent peak signaling a high probability of both high humidity and high temperatures for the region. The query involved six storage nodes and was evaluated in 240.10 ms (averaged over 1000 iterations, with a standard deviation of 30.0 ms).

Similar to a probability density query, evaluating joint probability requires creating a subgraph of intersecting feature values and then performing a kernel density estimation during the reduce phase. As always, the client-side request for a joint probability query can be combined with our existing query constructs to manage specificity. Figure 9 provides a contrasting view of the same query evaluated across the entire continental United States, which contacted all 78 nodes in parallel and completed in 424.19 ms (averaged over 1000 iterations, with a standard deviation of 45.16 ms). Note that in this scenario a peak similar to that in Figure 8 exists due to the overall hot and humid summer months, but the probabilities of other temperature-humidity tuple combinations have also risen to account for other regions; particularly, the frequency of high heat and low humidity is much more prominent in this example. In both cases, additional information can be retrieved from the subgraphs generated by the query; for instance, a range query could be evaluated against the aggregate subgraphs to determine the locations in the continental USA with high temperatures and low humidity.

5 Predictive Analytics: Methodology

After exploring the interactions between features in the dataset, *predictive analytics* enables us to capitalize on these relationships through modeling and dynamic forecasting to discover and exploit trends, project future events, and reason about how the dataset is evolving over time. These types of analysis involve making assertions about the strength of relationships to validate statistical hypotheses, projecting future events and conditions with linear and nonlinear models, and making probabilistic estimations. Using this functionality, the wealth of knowledge already stored in the system is leveraged to provide accurate and timely insights into the evolution of the feature space and its corresponding events.

5.1 Hypothesis Testing

The exploratory analysis process leads to the development of hypotheses about the dataset and its interactions; an increase in sales might be the result of a successful advertising campaign, or simply due to fluctuations in shopping trends. To evaluate these hypotheses in a statistically sound manner, Galileo provides built-in *hypothesis testing* functionality. Hypothesis testing begins with formulating the *null* and *alternative* hypotheses. The null hypothesis is often considered the "default position" for statistical tests, which states that there is no relationship between particular phenomena. On the other hand, the alternative hypothesis represents the opposite case where outcomes indicate a relationship. When a relationship is present in the data, we *reject* the null hypothesis — depending on the test being carried out, this may or may not be desirable. For instance, it may be advantageous to replace a hardware component with a cheaper version, but only if the null hypothesis holds for failure rates before and after the change.

Hypothesis testing is performed by comparing aggregate summary graphs produced by two queries. Storage nodes that are projected to contain a majority of the relevant data for a query orchestrate the reduction process, merging vertices as they are received to produce the final summary graphs. A t-test is then carried out on the graphs, enabling calculation of the p-value as described in the previous section. We support both one- and two-tailed tests: a one-tailed test is used for relationships that change in only one direction, whereas a two-tailed test is applied when the relationship could be both negative or positive. For example, an advertising campaign could both increase or decrease sales; if the advertisements are negatively received or alienate previous customers, sales may decline. The p-value is then used to decide whether the null hypothesis should be rejected or not, with significance levels of 5% or 1% commonly used to denote a strong presumption against the null hypothesis. To instrument a hypothesis test, clients provide two queries that represent the states being compared, along with a desired significance level. Backed by our vertex synopses, hypothesis testing is an efficient operation: a randomized test involving 250,000 vertices and over 2 million records executed across 24 nodes completed in 26.66 ms on average over 1000 iterations, with a standard deviation of 1.64 ms.

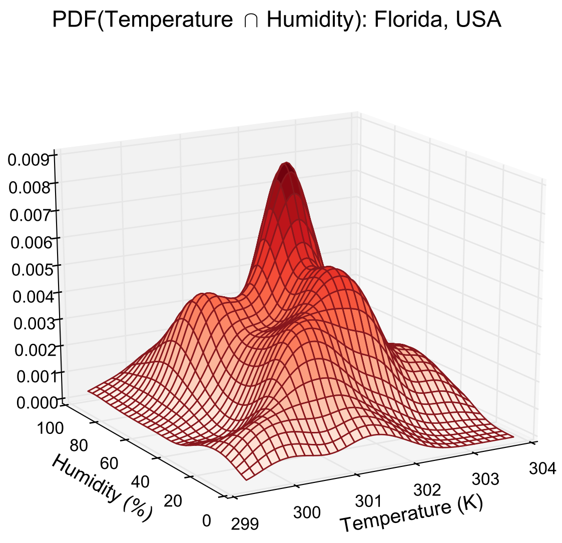


Fig. 8. Joint probability density of temperature and humidity values in Florida, USA, during July of 2013.

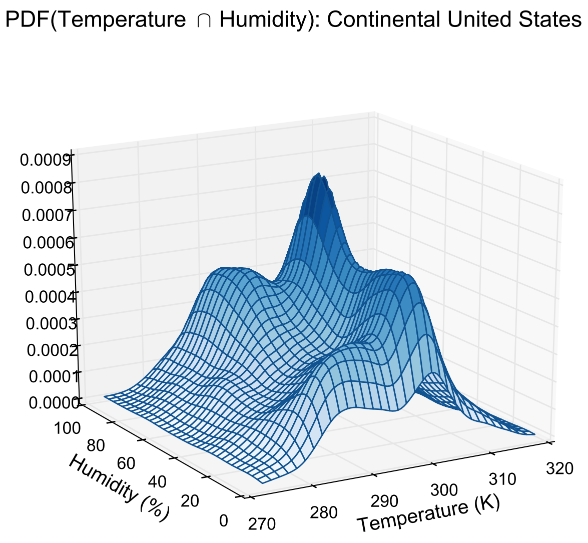


Fig. 9. Joint probability density of temperature and humidity values across the continental USA during July of 2013.

5.2 Multiple Linear Regression

While our vertex synopses support two-dimensional regression functionality, there are cases where several explanatory variables can predict a single outcome. To handle these situations, we also support *multiple linear regression* across features to build predictive models that are updated continuously as data is streamed into the system. Since multiple linear regression spans several features, instances are not required at each vertex in the graph. Instead, we support selective placement of regression instances; the default placement is not linked to any particular location in the graph, assimilating all new readings as they arrive. However, we have also incorporated support for user-defined limitations on the scope of the regression models to particular subsets of the data to ensure expressivity. For example, each vertex representing a month of the year might maintain its own multiple linear regression instance. Users can place or remove multiple linear regression models during index creation or dynamically at runtime, and their meager memory requirements enable a substantial number of instances to be maintained in the index.

In situations where variables in the regression model are highly correlated and begin to exhibit *multicollinearity*, coefficient estimates for individual predictors may become inaccurate. For this reason, we autonomously detect multicollinearity and warn users of its presence using the *variance inflation factor* (VIF). The VIF provides a measure of multicollinearity that helps judge whether certain variables should be excluded, which is essential when dealing with the large number of models our framework maintains. Client applications can also specify custom VIF thresholds based on their particular use cases.

Like our two-dimensional vertex synopses, multiple linear regression instances are lightweight. Table 6 contains performance statistics for various operations applied to the regression instances, which include adding data, calculating r^2, and making predictions. We also include the time taken to compute the root-mean-square error (RMSE) of a regression operation, which measures the accuracy of the predictions using the same units as the dependent variable. Vertices that maintain a multiple linear regression instance create a predictive model for each feature type, meaning fast and efficient updates are critical to ensure overall system performance.

TABLE 6
Dynamic multiple linear regression performance evaluation, averaged over 1000 iterations.

Operation	**Time (μs)**	σ **(μs)**
Add Data Point	1.51	0.28
Calculate r^2	0.78	0.02
Calculate RMSE	0.79	0.13
Make Prediction	2.54	0.51

To benchmark the effectiveness of our regression framework, we used models built with data collected in Wyoming, USA during July over a three-year period (2011 through 2013) to predict rainfall. The models included each of the feature types indexed in this study, and were tested with new feature readings from 2014. Figure 10 contains a scatter plot of the residuals (difference between predicted and actual precipitation). The RMSE of this test was 0.31 kg$/m^2$ of rainfall. While an exact measure of rainfall is a useful metric, we can also answer the common question “do I need my umbrella today?” with a binary classifier. To create the classifier, we considered any prediction over 0.31 kg$/m^2$ to imply that it would indeed rain, whereas a value lower than the threshold would indicate little to no rainfall. In this case, we predicted rain correctly 92% of the time when compared to the actual rainfall data from 2014. While client applications can request multiple linear regression model parameters directly for a particular vertex or feature set and have them streamed back for analysis, they are also given the option to create customized binary classifiers similar to the example described. A classification query specifies features of interest, spatial and temporal ranges, and classification thresholds, and returns a computational model that can be used to predict and classify future events.

5.3 Artificial Neural Networks

For some datasets or feature types, linear methods may restrict model fidelity or reduce prediction accuracy; for instance, consider the constant fluctuations present in foreign exchange rates and stock prices or the sinusoidal variations in temperatures that occur over the course of a day. In these cases, nonlinear methods provide an alternative that can produce models that are a better fit for the underlying data. Our framework includes an interface that enables arbitrary models and prediction methods to be placed at graph vertices in a similar fashion to the multiple linear regression instances, and can produce output datasets in the form of classifications, function approximations, or forecasts. We have incorporated support for online artificial neural networks (ANNs) provided by the Encog [20] machine learning library to accommodate nonlinear predictive models.

Compared to the linear methods discussed in previous sections, ANNs generally involve more complex computations for training. Furthermore, they often do not completely converge on one final set of model parameters, so training is an inexact and iterative process. A neural network created with the Wyoming dataset from the previous section took 682.08 ms on average to train with a single new record (1000 iterations, standard deviation of 128.13 ms). For these reasons, our framework supports a *batch ingest* mode to amortize training costs by collecting a set number of observations before training occurs. Training is also executed as a lower-priority background thread to avoid impacting query throughput, maintaining performance while still keeping predictions updated. This allows us to ensure consistent overall system performance while still providing the more compute-intensive functionality afforded by ANNs.

5.4 Time-series Forecasting: ARIMA

Autoregressive integrated moving average (ARIMA) models are specifically designed for time-series data and allow predictions to be made on non-stationary data types. ARIMA models are parameterized by three parameters, p, d, and q, which correspond to the autoregressive, integrated, and moving average components of the model, respectively. Our implementation allows these parameters to be chosen autonomously by the system or specified at query time by client applications. Query parameters also control which features should be considered by the model and the time bounds of interest (which may include historical data).

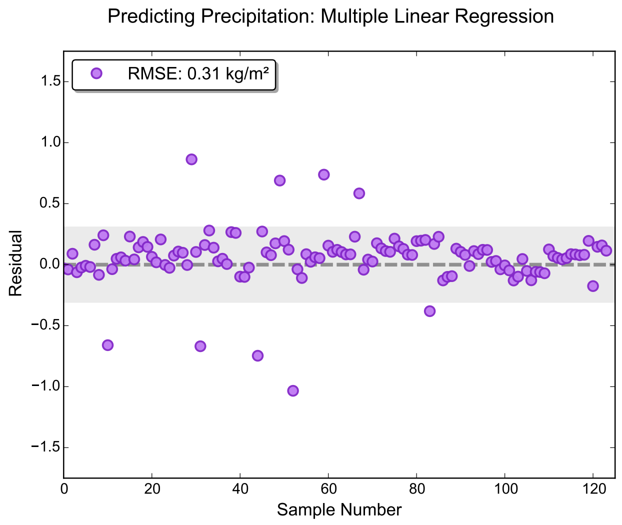


Fig. 10. Scatter plot of the residuals between predicted and actual precipitation in Wyoming, USA, in July of 2014. The shaded region above and below the reference line represents the RMSE of the predictions.

Figure 11 illustrates the effectiveness of ARIMA for predicting temperatures in Florida and Wyoming, USA, respectively during July of 2013. These models were populated with data from the first 23.5 days of the month, and then were used to forecast subsequent temperatures for the remainder of the month. Predictions for Florida resulted in a root-mean-square error (RMSE) of 0.077 K, while predictions from the model responsible for Wyoming produced an RMSE of 0.818 K; in both cases, the forecasts were highly accurate, and responded to the cyclical temperature shifts that occur during a typical day in both locations. Note that each day generally exhibits the highest temperatures during midday and lowest temperatures during night hours. Compared to Wyoming, temperature fluctuations are less severe in Florida, which likely accounts for some of the difference in prediction accuracy.

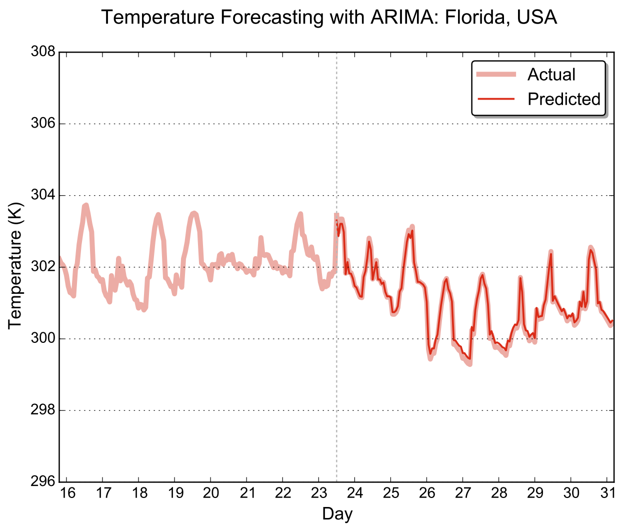
5.5 Conditional Probability and Naive Bayes Classification

Conditional Probability queries answer questions such as, "what is the probability of rain given cloud cover is greater than 80%?" Computing the answer to this type of query involves determining the intersection of a particular set of events, which can be performed efficiently with the DiscoveryGraph; client-side turnaround times averaged 38.66 ms for queries over Florida, USA (across 1000 iterations with a standard deviation of 2.89 ms). Based on the frequencies stored in vertex synopses, individual probabilities associated with each event can be retrieved through synopsis combinations. This type of probabilistic analysis can also uncover relationships with no direct interaction; two features may seem unrelated until an additional dimension is considered. Determining whether such a relationship exists requires traversing backwards through the graph hierarchy until a common link is found.

A ***Naive Bayes classifier*** uses the probabilities associated with events or features in the dataset to make predictions. The key assumption of this type of model is that all features are *independent*: completely unrelated to any of the other features. Despite the fact that this assumption may not always hold, naive Bayes has proven to be effective in practice for a variety of classification tasks, including text categorization and medical diagnosis. To classify a given set of samples, naive Bayes uses the combined probabilities of the events to choose the most probable outcome. Given variables such as the current cloud cover, humidity, and temperature, naive Bayes can determine the probability of rain or other events in the dataset. Using our test data from Wyoming in July of 2013 we were able to achieve 85% accuracy for rainfall predictions in 2014, a query that took 17.68 ms on average (across 1000 iterations with a standard deviation of 1.66 ms, after contacting 6 storage nodes). While the predictions were less accurate than our multiple linear regression models, naive Bayes can also be used in situations where multiple linear regression does not apply, such as text classification.

6 Sever-Side Query Performance: Throughput

Our framework introduces exploratory and analytic queries that require an involved evaluation process. To ensure scalability and support near real-time analysis, our algorithms are designed to be lightweight and efficient. The computational profiles of these algorithms fall into three retrieval categories: (1) precomputed synopses (summary statistics, correlation analysis, conditional probability, naive Bayes classifiers), (2) multi-stage processing and analysis (PDFs, joint PDFs, significance queries, hypothesis testing), and (3) model calculations (multiple linear regression, artificial neural networks). To test the



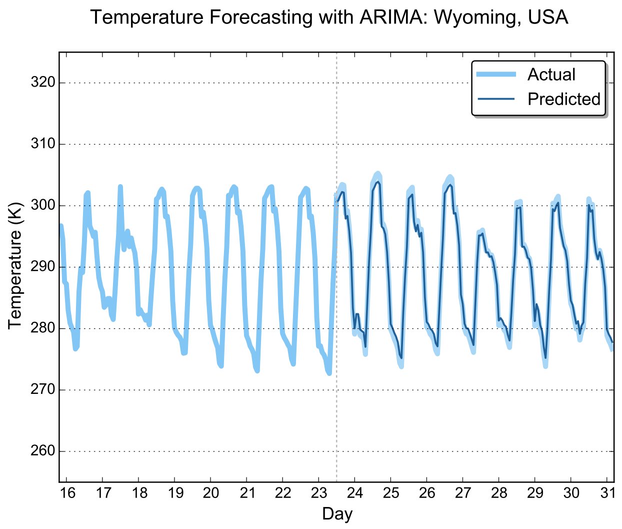


Fig. 11. ARIMA forecast of temperature values in Florida (left) and Wyoming (right), USA during the month of July in 2013.

performance of these algorithms we conducted a throughput evaluation; Table 7 provides the number of queries processed per second on our 78-node cluster. During the tests, the major page fault rate across all the nodes never went above 0.01/s. Query types were represented uniformly in this test. Precomputed synopses and multi-stage analytic queries both involve graph traversals and the construction of subgraphs, whereas model calculations generally require very few traversal steps.

TABLE 7
Cumulative throughput across query categories for our 78-node cluster (with four disks per node). Results are averaged over 1000 iterations.

Category	Cumulative Queries/s	St. Dev.
Precomputed Synopses	131400	191
Multi-Stage Analysis	90690	244
Model Calculations	592200	810

6.1 Concurrent Workload Evaluation

To illustrate the impact of concurrent workloads on throughput, we executed our tests again with varying levels of storage requests being performed in parallel. Galileo uses non-blocking I/O, enabling processing and storage activities to be interleaved. Table 8 compares the performance impact of concurrent storage operations expressed as a percentage of query throughput (e.g., 10000 queries per second with a 50% workload mix would result in 15000 total operations being processed per second).

TABLE 8
Impact of concurrent workloads on query throughput, averaged over 1000 iterations.

Workload Mix	Δ Queries/s (%)	St. Dev. (%)
25% Storage	-0.82%	0.21
50% Storage	-8.65%	1.32
75% Storage	-16.25%	2.85

6.2 Model Creation with Historical Data

Models that incorporate future observations or information that is already available in the DiscoveryGraph can largely avoid disk I/O. On the other hand, models that require high-fidelity historical data will have to retrieve on-disk observations during initialization. Table 9 provides insight into model creation times when disk accesses are necessary. In this example, data from a 1030×620 km region is retrieved from disk to initialize models that incorporate all available features across successively larger time spans. Ultimately, model creation times are dependent on disk speeds and how much of the underlying dataset must be retrieved. Once model initialization is complete, new observations will be assimilated as they become available.

TABLE 9
Model creation times for a 1030×620 km region when disk accesses are required, averaged over 100 iterations.

Time Span	Size (GB)	Creation (s)	St. Dev. (s)
1 Month	36.16	9.81	1.81
6 Months	227.36	56.30	3.19
12 Months	453.36	102.37	8.27

7 Related Work

The graphs used in Galileo share some common features with k-d trees [21], but do not employ binary splitting and allow much greater fan-out as a result. Similar to Tries [22], identical attributes in a record can be expressed as single vertices, which simplifies traversals and can reduce memory consumption. However, Galileo graphs support multiple concurrent data types, maintain an explicit feature hierarchy (that can also be reoriented at runtime), and employ dynamic quantization through configurable tick marks.

MongoDB [23] shares several design goals with Galileo, but is a document-centric storage platform that does not support analytics directly. However, MongoDB has rich geospatial indexing capabilities and supports dynamic schemas through its JSON-inspired binary storage format, BSON. MongoDB can use the Geohash algorithm for its spatial indexing functionality, and is backed by a B-tree data structure for fast lookup operations. For load balancing and scalability, the system supports sharding ranges of data across available computing and storage resources, but imposes some limitations on the breadth of analysis that can be performed on extremely large datasets in a clustered setting.

Facebook's Cassandra project [5] is a distributed hash table that supports column-based, multidimensional storage in a tabular format. Like Galileo, Cassandra allows user-defined partitioning schemes, but they directly affect lookup operations as well; for instance, using the random data partitioner backed by a simple hash algorithm does not allow for range queries or adaptive changes to the partitioning algorithm at runtime. This ensures that retrieval operations are efficient, but also limits the flexibility of partitioning schemes. Cassandra scales out linearly as more hardware is added, and supports distributed computation through the Hadoop runtime [24]. Predictive and approximate data structures are not maintained by the system itself, but could be provided through additional preprocessing as new data points are added to the system.

Considerable research has been conducted on supporting query types beyond the standard *get* and *put* operations of DHTs. For instance, Gao and Steenkiste [25] maps a logical, sorted tree containing data points to physical nodes, enabling range queries. Chawathe et. al [26] outlines a layered architecture for DHTs wherein advanced query support is provided by a separate layer that ultimately decomposes the queries into *get* and *put* operations, decoupling the query processing engine from the underlying storage framework.

Apache Hive [27] is a data warehousing system that runs on Hadoop [28] and HDFS [29]. As an analysis platform, it is capable of a wide range of functionality, including summarizing datasets and performing queries. Unlike Galileo and a number of other storage frameworks, the system is intended for batch use rather than online transaction processing (OLTP). In Hive, users can perform analysis using the HiveQL query language, which transforms SQL-like statements into MapReduce jobs that are executed across a number of machines in a Hadoop cluster. The *Metastore*, a system catalog, provides an avenue for storing pre-computed information about the data stored in the system. Hive emphasizes scalability and flexibility in its processing rather than focusing on low latency.

BlinkDB [30] provides approximate query processing (AQP) functionality by augmenting the Hive [27] query engine. Two methods of sampling are supported: a broad dataset-wide sample, and a focused sample that includes frequently-accessed items. The sampling framework in BlinkDB also supports stratification to better represent outliers or underrepresented data points. By generating indexes based on samples, queries can be resolved quickly and avoid reading information from disk. However, using the system in a predictive capacity is generally limited to the insights that can be derived directly from the available data.

Similar to the summary statistics maintained in our approach, incoming data streams can be represented as wavelets to avoid indexing every data point while still maintaining an approximate model. Cormode et al. [31] employs several different types of wavelets for creating synopses or approximations of incoming data and reviews their efficacy. This approach enables very large datasets to be accurately summarized. Yousefi et al. [32] also shows the feasibility of using wavelets for prediction. However, methods that rely on wavelets are generally very problem- or dataset-specific and can limit the feasibility or efficiency of resolving arbitrary queries over the underlying data.

The TPR*-tree [33] provides predictive spatio-temporal query functionality that can retrieve the set of moving objects likely to intersect a particular spatial window at some future point in time. Intended use cases include meteorology, mobile computing, and traffic control. TPR*-Tree improves upon the Time Parameterized R-tree (TPR-tree) [34] by adding a probabilistic model that accurately predicts disk accesses involved with resolving a query, along with new insertion and deletion algorithms to enhance performance. While TPR*-tree and its related data structures focus on object movements, Galileo also considers predictions across a wide array of dimensions.

FastRAQ [35] considers both the storage and retrieval aspects associated with range-aggregate queries. To manage the error bounds of these approximate queries, partitioning is based on stratified sampling: a threshold is used to control the maximum relative error for each segment of the dataset. Like Galileo, data is assigned hierarchically to groups and then physical nodes. Queries are resolved using adaptive summary statistics that are built dynamically based on the distributions of the data.

SAUNA [36] proposes a technique for automatically relaxing the constraints user-defined queries. SAUNA operates on a standard relational database management system, and uses histograms to estimate the cardinality (number of results) of incoming queries. In situations where a query is estimated to return a small number of results, the input ranges of the query are relaxed to retrieve a broader range of items that are close to the desired parameters. This optimization helps reduce the overall number of queries that will be submitted by users of the system, increasing throughput.

8 Conclusions and Future Work

Support for analytic queries over voluminous datasets entails accounting for: (1) the speed differential between memory accesses and disk I/O, (2) how metadata is organized and managed, (3) the performance impact of the data structures, (4) dispersion of query loads, and (5) the avoidance of I/O hotspots. These factors enable us to provide a rich set of exploratory analysis functionality as well as predictive models that produce insights beyond just the trends present in the dataset.

One key aspect of our approach is minimizing disk accesses. This is achieved by carefully maintaining metadata graphs that retain expressiveness for query evaluations but preserve compactness to ensure memory residency while avoiding page faults and thrashing. The graphs remain compact even in situations where individual nodes store hundreds of millions of files. Further, statistical synopses ensure the knowledge base is continually updated as live streams occur. We achieve this via the use and adaptation of online algorithms, compact data structures, and lightweight models. This also allows us to perform query evaluations at multiple geographic scales.

We avoid query hotspots by propagating the queries to nodes likely to satisfy them, performing in-memory evaluations and avoiding disk accesses. This reduces the likelihood of queries building up and overflowing request queues at individual nodes. By targeting only a specific subset of the nodes, we minimize cases where queries are evaluated that produce no results. Our use of Geohashes also allows us to localize queries efficiently. Hotspot avoidance ensures faster overall turnaround times for individual queries. Combined with efficient pipelining, this allows multiple queries to be evaluated concurrently at a high rate, which is validated by our empirical results.

Our future work will target support for Bayesian networks and causality analysis. This work will also target pruning of the feature space and ordering to ensure tractability, which includes detection of v-structures, use of heuristics to support local causality detection, and Hidden Markov Models.

Acknowledgments

This research has been supported by funding (HSHQDC-13-C-B0018 and D15PC00279) from the US Department of Homeland Security, the US National Science Foundation's Computer Systems Research Program (CNS-1253908), and the Environmental Defense Fund (0164-000000-10410-100).

References

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, 2008.

[2] M. Malensek, S. Pallickara, and S. Pallickara, "Exploiting geospatial and chronological characteristics in data streams to enable efficient storage and retrievals," *Future Generation Computer Systems*, 2012.

[3] ——, "Expressive query support for multidimensional data in distributed hash tables," in *Utility and Cloud Computing (UCC), 2012 Fifth IEEE International Conference on*, nov. 2012.

[4] ——, "Polygon-based query evaluation over geospatial data using distributed hash tables," in *Utility and Cloud Computing (UCC), 2013 Sixth IEEE International Conference on*, dec. 2013.

[5] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[6] D. Hastorun, M. Jampani, G. Kakulapati, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *SOSP*. Citeseer, 2007.

[7] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[8] G. Niemeyer. (2008) Geohash. [Online]. Available: http://en.wikipedia.org/wiki/Geohash

[9] C. Tolooee, M. Malensek, and S. L. Pallickara, "A framework for managing continuous query evaluations over voluminous, multidimensional datasets," in *Proceedings of the 2014 ACM Cloud and Autonomic Computing Conference*, ser. CAC '14. ACM, 2014.

[10] National Oceanic and Atmospheric Administration. (2015) The north american mesoscale forecast system. [Online]. Available: http://www.emc.ncep.noaa.gov/index.php?branch=NAM

[11] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware 2001*. Springer, 2001, pp. 329–350.

[12] R. Rew and G. Davis, "Netcdf: an interface for scientific data access," *Computer Graphics and Applications, IEEE*, 1990.

[13] Q. Koziol and R. Matzke, "HDF5—a new generation of hdf," *National Center for Supercomputing Applications, Champaign, Illinois, USA, http://hdf.ncsa.uiuc.edu/nra/HDF5*, 1998.

[14] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[15] M. Malensek, S. Pallickara, and S. Pallickara, "Autonomously improving query evaluations over multidimensional data in distributed hash tables," in *Proceedings of the 2013 ACM International Conference on Cloud and Autonomic Computing*, aug. 2013.

[16] ——, "Evaluating geospatial geometry and proximity queries using distributed hash tables," *Computing in Science Engineering*, vol. 16, no. 4, pp. 53–61, July 2014.

[17] J. F. Allen, "An interval-based representation of temporal knowledge." in *IJCAI*, vol. 81, 1981, pp. 221–226.

[18] ——, "Maintaining knowledge about temporal intervals," *Communications of the ACM*, vol. 26, no. 11, pp. 832–843, 1983.

[19] B. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[20] Heaton Research, Inc. Encog machine learning framework. [Online]. Available: http://www.heatonresearch.com/encog

[21] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.

[22] E. Fredkin, "Trie memory," *Commun. ACM*, vol. 3, no. 9, pp. 490–499, Sep. 1960.

[23] mongoDB Developers, "Mongodb," *http://www.mongodb.org/*, 2013.

[24] The Apache Software Foundation. Apache Hadoop. [Online]. Available: http://hadoop.apache.org

[25] J. Gao and P. Steenkiste, "An adaptive protocol for efficient support of range queries in dht-based systems," in *Network Protocols, 2004. ICNP 2004. Proceedings of the 12th IEEE International Conference on*, Oct 2004, pp. 239–250.

[26] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, and J. Hellerstein, "A case study in building layered DHT applications," in *Proceedings of the 2005 SIGCOMM*, 2005.

[27] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.

[28] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–10.

[29] A. Bialecki *et al.*, "Hadoop: A framework for running applications on large clusters built of commodity hardware," *Wiki at http://lucene. apache. org/hadoop*, 2005.

[30] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica, "Blinkdb: queries with bounded errors and bounded response times on very large data," in *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 2013, pp. 29–42.

[31] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine, "Synopses for massive data: Samples, histograms, wavelets, sketches," *Foundations and Trends in Databases*, vol. 4, no. 1–3, Jan. 2012.

[32] S. Yousefi *et al.*, "Wavelet-based prediction of oil prices," *Chaos, Solitons & Fractals*, vol. 25, no. 2, pp. 265–275, 2005.

[33] Y. Tao, D. Papadias, and J. Sun, "The TPR*-tree: An optimized spatio-temporal access method for predictive queries," in *Proc of 29th Internatl Conf on Very Large Data Bases*, 2003.

[34] S. Šaltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez, "Indexing the positions of continuously moving objects," *SIGMOD Rec.*, vol. 29, no. 2, pp. 331–342, May 2000.

[35] X. Yun, G. Wu, G. Zhang, K. Li, and S. Wang, "Fastraq: A fast approach to range-aggregate queries in big data environments," *Cloud Computing, IEEE Transactions on*, 2014.

[36] A. Kadlag, A. V. Wanjari, J. Freire, and J. R. Haritsa, "Supporting exploratory queries in databases," in *Database Systems for Advanced Applications*. Springer, 2004, pp. 594–605.

Matthew Malensek is a Ph.D. student in the Department of Computer Science at Colorado State University. His research involves the design and implementation of large-scale distributed systems, data-intensive computing, and cloud computing.

Sangmi Pallickara is an Assistant Professor in the Department of Computer Science at Colorado State University. She received her Masters and Ph.D. degrees in Computer Science from Syracuse University and Florida State University, respectively. Her research interests are in the area of large-scale scientific data management, data mining, scientific metadata, and data-intensive computing.

Shrideep Pallickara is an Associate Professor in the Department of Computer Science at Colorado State University. He received his Masters and Ph.D. degrees from Syracuse University. His research interests are in the area of large-scale distributed systems, specifically cloud computing and streaming. He is a recipient of an NSF CAREER award.