Alleviating I/O Inefficiencies to Enable Effective Model Training Over Voluminous, High-Dimensional Datasets

Daniel Rammer, Walid Budgaga, Thilina Buddhika, Shrideep Pallickara, and Sangmi Lee Pallickara
Department of Computer Science
Colorado State University, Fort Collins, USA
{rammerd, wbudgaga, thilinab, shrideep, sangmi}@cs.colostate.edu

***Abstract*—There has been an exponential growth in data volumes in several domains. Often these voluminous datasets encompass a large number of features. Fitting models to such high-dimensional, voluminous data allows us to understand phenomena and inform decision-making. The analytics process is naturally iterative as scientists explore the set of features, data fitting algorithms, portions of the dataspace, and the particular algorithm's hyperparameters to guide their model-building process. It often takes several model-fitting attempts before one arrives at a satisfactory solution that may then be subjected to further refinements. Each of these model-building attempts is itself time-consuming and dominated by I/O and data movement costs. In this study, we present our methodology for significantly alleviating I/O-induced inefficiencies during model training. Rather than work with the raw data, we generate and work with sketches of the data. Our framework, Fennel, is independent of the libraries or analytical engines preferred by users. Our empirical benchmarks have been performed with datasets from diverse domains (weather, epidemiology, and music) and we profile several aspects of our methodology.**

***Index Terms*—model training; data sketches; multidimensional datasets, in-memory analytics;**

I. Introduction

Data generation and storage in several domains have grown exponentially. Contributing factors include the proliferation of sensing environments, falling costs for networked sensors, monitoring of diverse natural and man-made phenomena, simulations, and social media.

These data offer unprecedented opportunities for innovation and discovery. The data can be used to construct models that inform understanding of phenomena, planning, and decision making. In commercial settings, such models have economic implications and are known to drive sales, promotional campaigns, make inventory decisions, and guide investment strategy. The models we consider in this study are the product of analytic processes that leverage statistical and machine learning algorithms to fit models to the data. These are distinct from physical and mathematical models derived by subject-matter-experts in a particular domain.

This growth in data volumes has coincide with the proliferation of new data-fitting algorithms, open-source libraries for statistical and machine learning, and analytical engines that facilitate distributed orchestration of analytic tasks at scale. These software provide turnkey solutions for model construction; all that is required is provisioning of computing infrastructure and data to train the models on.

The starting point for analytics libraries is data. Building models is predicated on access to on-disk data. However, the I/O subsystem is 5-6 orders of magnitude slower than the CPU the analytics process is I/O bound. Furthermore, this data is distributed and discovery of the precise data to build the models on can itself be I/O intensive.

While data staging plays an important role in the efficiency of data accesses, it is impossible for any staging algorithm to (divine and) optimize for all possible "future" analytic operations. Users may focus their model-fitting operations to specific combinations of features and/or particular portions of the feature space. For example, a user may seek to understand the impact of disease spread using a mix of different control strategies such as movement controls and vaccination. Some features may be common in both analyses such as those that describe disease biology. The range of possibilities to construct models using different sets of features becomes combinatorially explosive as the features increase; with N features there are 2^N possibilities.

Misalignments in how the data is stored and how they are accessed are inevitable. The crux of this paper is to alleviate I/O inefficiencies during model construction in settings where the data is voluminous, dispersed over multiple machines, and encompasses a large number of features.

A. Challenges

There are several challenges to constructing analytical models over voluminous datasets. These include:

- *I/O costs:* On-disk data must be accessed and be memory-resident before model building operations can commence. The speed-differential across the memory hierarchy compounds these challenges. The disk I/O subsystem is sluggish with access times and transfer rates that are orders of magnitude slower than memory.
- *Data movement costs:* Completion times for analytics processes are prolonged when there is no data locality. In such cases, data needs to be *pulled* to the machine where the analytics process is executing. Data accesses in such cases incur network I/O costs in addition to disk I/O.

- *Shared clusters:* Analytic processes often execute in clusters that are shared across users and their applications. Processes on a single machine are subject to interference when they access the disk simultaneously; this interference results in plummeting throughputs during transfers from disk to memory. Network transfers are subject to these interference inefficiencies as well.
- *Iterative nature of the analytics process:* The analytics process is naturally iterative involving multiple, repeated sweeps over the data as the coefficients associated with the features are continually adjusted to improve model performance. A data scientist may also choose to explore multiple model fitting algorithms each of which may also involve hyper-parameter tuning.
- *Data volumes:* Data volumes exacerbate the aforementioned challenges relating to I/O, data movements, interference and plummeting throughputs in shared clusters. These inefficiencies result in prolonged execution times for analytic operations that inhibit explorations.

B. Research Questions

Since data volumes worsen inefficiencies, can we reduce data volumes without adversely impacting the accuracy of model construction? In particular, we propose to ***sketch*** the data and use this as the basis for explorations. This study is guided by the following research questions.

RQ-1: *How can we sketch the data to significantly reduce data volumes while preserving characteristics of the feature space?* In particular, compaction must also include preservation of the distribution characteristics of individual features and cross-feature covariance.

RQ-2: *How can we leverage the sketch to facilitate training of models?* The sketch must be amenable to querying, selection and retrievals, and effective generation of training data. Training data generated from the sketch must be statistically representative of the original full-resolution data.

RQ-3: *How can we facilitate fast, effective explorations model building using these sketches?* In particular, this involves preserving timeliness without compromising on the accuracy of the generated models.

C. Approach Summary

Our methodology targets significant reduction in I/O requirements during model fitting operations. In particular, our methodology relies on sketching the data, and then using these sketches to identify relevant portions of the feature space. Finally, we also leverage the sketches to construct models. Our framework, encompassing the sketching algorithm and systems optimizations, is codenamed ***Fennel***.

There are two key characteristics in our methodology. First, on-disk data is accessed during sketching operations, but is never modified (or deleted). Our sketch facilitates fast exploratory analytics and the full-resolution, on-disk data can continue to be used for deeper explorations once some of the exploratory analyses show promise. Second, the Fennel sketch is significantly more compact than the on-disk data and thus amenable to memory-residency, fast query evaluations, and subsequent retrievals.

We now describe the end-to-end phases that comprise Fennel; some of these phases are one-time operations and once the data is sketched and the metadata organized, the discovery and analytic operations can be completed in a timely fashion with substantially reduced I/O.

The groundwork phase lays the foundation for sketching. We use the Welford's method to construct statistical properties of the feature space. This include summary statistics associated with individual features and cross-feature covariances. Next we explore binning (or discretization) of individual features. For each feature, the bins are generated using an Online Kernel Density Estimation (oKDE) [1], [2] function such that the probability of placing an observation within each bin is equal. Bins are generated offline with respect to a sample and a configurable error threshold defined based on the Normalized Root Mean Square Error (NRMSE). The number of bins are incremented until the discretization error is below the preconfigured NRMSE threshold. The costs associated with the groundwork phase are amortized over a large number of subsequent analytic tasks.

During the sketching phase on-disk data is sketched, and elements of the sketch are shuffled within the cluster to facilitate fast querying, retrievals, and memory-consumption reduction. In particular, we normalize the data based on the summary statistics for each feature, and then discretize the normalized feature values. The binned feature(s) are then used to compute *discretized feature vectors* (DFVs), which are an ordered concatenation of binned feature values.

As the data is sketched on individual disks, the DFVs and their assorted frequencies are shuffled to get an accurate global count of the frequencies associated with a particular DFV for the entire dataset. Fennel uses DHTs (distributed hash tables) as the logical overlay to organize the DFVs. The DFVs and their concomitant frequencies maintained within the DHT comprises the distributed Fennel sketch. The Fennel DHT is organized as a one-hop DHT with consistent hashing. The hash code associated with the DFVs is generated by using a configurable number of feature bins within the DFV as the input to a CRC-32 function. This allows us to ensure that DFVs representing proximate portions of the feature space are collated. DFVs at each Fennel-DHT node are organized using an in-memory data structure with support for aging contents partially to disk in situations where memory contention is high. The organization of DFVs does not preclude searching or retrieving portions of the feature space for an arbitrary set of features.

The discovery phase supports search and retrievals of relevant portions of the feature space. Queries that we support include: range queries, SQL, and statistical queries. We are also support queries based on top-K or bottom-K DFVs; this allows users to retrieve either the k-most (or rarely) occurring DFVs. During query evaluations DFVs that satisfy query constraints are retrieved from the sketch and streamed back

to the client. These DFVs are organized within our GIST data structure that is the entry point for several analytic operations.

The pre-processing phase supports incremental refinements of the GIST data structure. The GIST data structure is used for generating synthetic data. In particular, DFVs and their occurrence frequencies are used to generate datasets that are normalized, proportional, and statistically representative of the observed feature space. GIST is also used to inform partitioning synthetic datasets as training, test, and validation datasets. A refinement supports generation of folds as needed in k-fold cross-validation.

The model fitting and assessment phase includes generation of synthetic datasets from GIST, support for just-in-time generation of data and using the data to complete model training. Several customizations such as controlling the number of observations to be generated, the number of machines involved during the analytics phase, and partitioning the synthetic dataspace into decorrelated datasets as needed in ensemble training are also supported. Fennel supports automated assessment of model performance.

D. Paper Contributions

Our methodology facilitates rapid explorations of the feature space. Fennel does not preclude building models from on-disk data. Our contributions include:

1) Our sketch achieves significant compaction rates while preserving representativeness of the feature space.
2) Our sketch is amenable to memory-residency, querying, and targeted retrievals.
3) GIST, which in essence is a sketchlet, serves as the basis for fast model training and assessments. GIST supports just-in-time generation of synthetic data to substantially alleviate memory footprints during training.
4) Model fitting operations using the sketch can be performed using diverse libraries and analytical engines.

Since we target efficiencies at the data and I/O level, our methodology is suitable for a broad class of model fitting algorithms be they regression, classification, or clustering.

E. Paper Organization

The remainder of the paper is organized as follows. Section II provides an overview of Fennel's architecture and its building blocks followed by a detailed discussion on our approach in Section III. System benchmarks are presented in Section IV and related work is discussed in Section V. Section VI outlines conclusions and future work.

II. SYSTEMS OVERVIEW

The Fennel Architecture can be partitioned into three components, namely the Storage Cluster, Analytics Cluster, and Clients. This architecture, along with the sketched data distribution communication, is presented in Figure 1a.

The Fennel Storage Cluster contains a number of machines responsible for storing sketched data. The feature space is partitioned among nodes using a single hop DHT, which uses consistent hashing for deterministic DFV placement. Figure 1b outlines important internals of each storage node. Node functionality is divided between the control and data planes. The control plane handles the DHT and gossip between nodes. Alternatively, the data plane is responsible for sketch initialization, querying, and data pipelines which are used for efficient data insertion.

The Analytics Cluster comprises machines we use to generate synthetic datasets and train models. During analytics, each machine receives a desirable portion of sketched data from the storage cluster. We train a variety of models by generating a statistically representative synthetic dataset using sketched data. Observations are generated and fed into model training entirely in-memory.

Fennel Clients are responsible for submitting requests for sketch initialization, data insertion, and data queries. Users are able to initialize sketch definitions containing any variety of dataset features and feature discretization boundaries. Clients may also initialize Data Pipelines at multiple Fennel Storage Nodes and use them to insert data. Finally, users can submit queries over the sketched data; instructing Fennel Storage Nodes to distribute data to the Analytics Cluster and kickstart model training operations.

The core abstraction underpinning data insertions is data pipelines. Pipelines contain a chain of stages (source, transform, and shuffle) connected with blocking queues. Each stage operates with multiple threads to efficiently leverage modern multi-core CPU architectures. Using these structures we effectively distribute computationally expensive operations (ie. feature discretization, DHT lookups, DFV shuffling, etc) over the Fennel Storage Nodes.

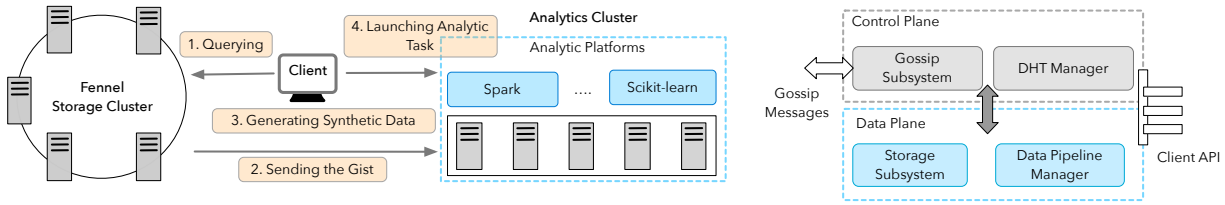
III. METHODOLOGY

Our methodology for effective model training encompasses the following distinct phases.

1) The groundwork phase focuses on distributed calculation, and subsequent exchange, of global statistical properties of the feature space. [**RQ-1**]
2) We then sketch on-disk data to ensure effective compactions while preserving representativeness of the feature space. Portions of the sketch are shuffled to conserve memory footprints and fast query evaluations. [**RQ-1**]
3) The distributed Fennel sketch facilitates interactive explorations and retrievals of the feature space. This is supported by enabling fast query evaluations and retrieval of results. [**RQ-1, RQ-2**]
4) The pre-processing phase leverages our GIST data structure, which organizes portions of the sketch that satisfy specified query constraints. GIST supports generation of representative synthetic datasets. [**RQ-2, RQ-3**]
5) The model fitting and assessments phase facilitates training of models while performing I/O frugally to facilitate fast completion times. [**RQ-3**]

A. Groundwork Phase

The groundwork phase constructs statistical properties of the feature space and uses online kernel density estimation



(a) Fennel's three architectural components and interactions

(b) Fennel storage node internals.

Fig. 1: Fennel Architectural Overview.

functions to compute feature discretization boundaries. We designed a MapReduce job that does a single pass over the distributed dataset in the Map phase, and a computationally inexpensive Reduce phase.

We compute statistical properties of the feature space using Welford's online method. During the Map phase we iteratively update the data structure with each observation. The Reduce phase leverages the well-defined algorithm to merge separate instances of Welford's from each node, resulting in accurate evaluation over the entire feature space.

We compute feature discretization boundaries based on a data sample of around 50,000 observations. During the Map phase, we create a reservoir sample at each host with a proportional number of observations relative to the entire dataset. Reservoir sampling is an online sampling technique, which ensures that each observation has an equal probability of being present in the final sample. In the Reduce phase, we combine the samples from each host and compute an online kernel density estimation (oKDE) function for each feature. We iteratively increase the number of bins until the normalized root means squared error (NRMSE) between the discretized values and the sample is below a configurable, predefined threshold (0.025 in our experiments). During each iteration, discretized boundary values are computed to split the oKDE into bins containing equal observation frequencies. Tuning the NRMSE threshold is tradeoff between dataset compaction rates and the statistical representativeness. The result of this operation is a list of bounds that split the feature space into bins with equal observation frequencies.

B. Sketching On-Disk Data

We initialize sketches in Fennel by defining a set of active features and their corresponding discretization boundary values. This enables defining multiple sketches over the same dataset encompassing any subset of features and producing myriad compaction rates and levels of synthetic data accuracy.

Discretized Feature Vectors (DFVs): We handle generation of DFVs in the transform stage of the Data Pipeline. Each observation is presented as a vector of numeric values, one for each feature. For each value we perform a binary search over the defined discretized boundaries for that feature to retrieve a boundary index. Fennel tracks frequencies of unique DFVs.

Storing DFVs: We shuffle DFVs among Fennel nodes during the shuffle stage, which serves two purposes:

1) It enables equal distribution of data to each Fennel node. This balance results in more effective distribution of computations over the cluster.
2) We are able to optimize some queries based on known data placement. Queries that constrain specific features may only need to contact a subset of Fennel Storage Nodes. This results in more efficient data queries.

Fennel uses a Distributed Hash Table (DHT) to effectively distribute DFVs over the cluster. We allow each physical node to partition itself as multiple virtual nodes to allow for responsibility of different proportions of the hash space. This is advantageous when deploying in a cluster where machines have varying hardware capabilities or when the DHT experiences data "hot spots".

We determine the node responsible for a DFV by hashing a subset of the DFV features using the CRC32 algorithm. Consistent hashing among Fennel nodes ensures nearly uniform DFV placement. The hashed features are defined during sketch initialization. Domain specific knowledge aids in choosing the features as Fennel queries may leverage the DHT to improve query performance.

At each node, we store unique DFVs and their accompanying frequencies. Employing a hash map, as opposed to a tree structure, reduces overheads. During query evaluations, a tree-based organization requires traversal of many levels; data with high dimensionality presents a combinatorially explosive tree (e.g.; 2000 features, with 30 discretized values each, results in 30^{2000} possible leaf nodes). Query wildcards where some features are unconstrained, results in an exponential increase in the number of tree path traversals.

To provide efficient queries we construct indices based on a subset of the feature space. The data structure is a *tree* where each level pertains to a single feature, internal nodes are discretized feature values, and leaf nodes contain a list of DFVs and frequencies. The result are shallow indices that reduce the search space when querying highly dimensional data where many features are left unconstrained.

C. Exploring and Retrieving Portions of the Feature Space

Fennel supports lightweight SQL queries over data sketches. Queries may specify a subset of features to retrieve. As datasets are intrinsically numeric we support a number of numeric feature constraints ($<$, $<=$, $=$, $>$, $>=$) as well

as boolean operations (AND, NAND, OR, XOR). Query execution proceeds in steps.

- Discretization of query parameters: Since all feature values defined in sketches are discretized we need to convert the query parameters. Obviously not all queries will fall directly on discretization boundaries, therefore we dynamically relax query constraints towards outlying boundaries.
- Determine responsible nodes: Depending on sketch definitions, Fennel redirects queries to a subset of Storage Nodes. DFVs are distributed in the DHT based on a subset of features, if the query contains those features we can guarantee the location of requested data and reduce the number of Nodes involved in evaluation.
- Execute query at each node: Each responsible Fennel Storage Node executes the query (in parallel), leveraging suitable indices to improve performance.

DFVs that satisfy specified query constraints are organized into our GIST data structure. Fennel clients submit queries to each node in parallel and are returned a stream of DFVs and their corresponding frequencies. The GIST structure is an agglomerated list of ⟨DFV, frequency⟩ combinations.

D. Preprocessing Phase

Refinements of the GIST are necessary for the incremental nature of analytics. A GIST may be further filtered given additional feature constraints using the SQL-like query language. Operationally, refinement requires iteration over each ⟨DFV, frequency⟩ combination in the structure to identify all desired observations. Since a GIST is compact enough to fit in memory, filtering is computationally inexpensive.

We produce synthetic datasets by iterating over the GIST; operating on a specific DFV and frequency combination during each iteration. Synthetic observations are created by generating a random value between the lower and upper boundaries of each discretized feature following a uniform distribution. For example, say a feature is discretized with values 0, 5, 10, and 15. DFVs with a discretized value 0 would then produce a randomly generated value between 0 and 5.

We provide the ability to specify the number of synthetic observations that should be generated. This functionality is implemented as a multiplier of the original observation count. For example, if a GIST contains 100,000 observations a multiplier of 2.0 will generate 200,000 observations, whereas a 0.5 multiplier will only generate 50,000 observations. This functionality is especially useful for data sampling operations. When using the multiplier synthetic datasets remain representative of the original feature space by applying the multiplier to each DFVs frequency.

GIST is amenable to partitioning synthetic data into separate training, testing, and validation datasets. Each partition is guaranteed to be statistically representative of the original feature space. Once a synthetic dataset is available it is then partitioned into non-overlapping training, testing, and validation datasets (by default 80%, 10%, 10%).

TABLE I: Benchmark dataset attributes including observation count, feature count, and original dataset size.

Dataset	Observations	Features	Size
NOAA	4.2 billion	56	2.94 TB
Texas Epidemiology	899,952	2595	26 GB
Million Songs	463,715	91	425 MB

E. Model Fitting and Assessments

The learning process improves the efficiency of the models gradually in an iterative manner by optimizing an objective function (loss function) that measures the degree to which the ideal case is satisfied. To build a model that *generalizes* well (i.e., it performs well on unseen data), this process needs to be repeated several times for fine tuning hyper-parameters in the learning process and finding the most influential features. Model fitting is time-consuming and becomes prohibitive for voluminous datasets with high dimensionality. For example, the process for finding the most influential features (called feature selection) needs multiple accesses to a voluminous dataset to extract a different subset of features and evaluate their importance during the training process. Fast access to data speeds up model fitting.

1. Support for just-in-time Generation of Synthetic Data: Fennel operations work entirely with sketched data and generate synthetic data only when it is necessary, or "just-in-time" for processing. In Fennel we are able to query and distribute sketched data, encapsulated within the GIST, over the Analytics Cluster and generate synthetic data at each machine. The advantage of this approach is (1) data is small enough to be memory resident and therefore bypasses (magnitudes slower) disk accesses, and (2) network I/O incurred during data transfer is vastly reduced.

2. Completion of Model Training: The time to train a model depends on multiple factors including the size of the training data, how the data is hosted, and the data fitting algorithm. For example, performing batch learning to fit one complex model on voluminous data will be infeasible and leads to prolonged training time because an optimizing step will be done after making a full pass over the entire training data. Reducing the training time could be achieved by relying on different strategies such as (1) employing mini-batch learning in parallel to train one complex model, (2) constructing a bagging ensemble comprising decorrelated models that can be trained in parallel, (3) building specialized models, each of which has been trained to be expert for different portion of the feature space, or (4) relying on sampling to create a small version of the voluminous data. Each of these different strategies requires a different aspect of the data. Fennel retains sketches in memory and relies on different strategies to enable fast handling of the queries.

3. Assessment of Model Performance: The synthetic data generated by Fennel can be partitioned into training (80%), test (10%), and validation datasets (10%) to evaluate models performance with a different set of the hyper-parameters. Cross-

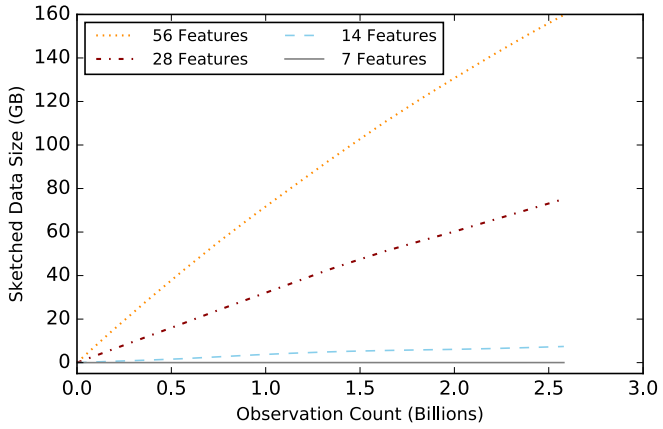


Fig. 2: Sketch sizes using the NOAA dataset for varying features. Note the logarithmic trend where increasing amounts of data improves compaction rates.

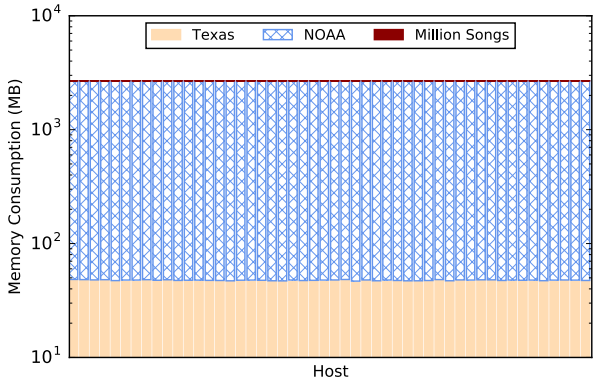


Fig. 3: Memory consumed at each Fennel storage node for different datasets. This highlights Fennel's load balancing.

validation can also be used for the same purpose. The validation data will be used to assess the model's performance using different hyper-parameter values. In our systems benchmarks, we assess models using external test data that is a subset of the original data and have not been staged to the system. For regression problems, we use the NRMSE to measure the models' performance on unseen data. Furthermore, we perform bias-variance decomposition to divide the prediction error into two components (Bias and Variance). With this analysis, we can improve the models' performance further by targeting either the bias or variance components of model error.

IV. System Benchmarks

We profiled several aspects of our methodology to assess:

1) Effectiveness of the Fennel sketch in terms of it compaction rates (Section IV-B, **RQ-1**) and representativeness of the feature space (Section IV-D, **RQ-1**).
2) Systems characteristics of the distributed Fennel sketch. This includes its memory footprints, how amenable it is to querying and retrievals, and how it load balances memory utilization. (Section IV-C, **RQ-2**)
3) Support for effective model fitting in terms of training times, and the accuracy of the models constructed using the sketch. (Section IV-E, **RQ-3**)

A. Experimental Setup

1) Datasets: We have benchmarked Fennel using three different datasets (presented in Table I) to highlight functionality under a variety of dataset dimensionality and sizes. NOAA is a popular weather dataset. We have chosen to analyze 2 years (2013 and 2014) and the 56 features that are uniformly reported by all vantages during that duration. The Texas Epidemiology dataset is a high dimensional dataset tracing livestock epidemics. Million Songs outlines a variety of song characteristics and the year that song was recorded.

2) Hardware: We deployed Fennel on a cluster of HP-DL60-G9-E5-2620v4 machines running Fedora 27. These are outfitted with 16GB RAM and an Intel Xeon E5-2620 which has 8 cores (16 hyper-threads) @ 2.10GHz. The storage and analytics clusters are deployed on 50 and 47 unique machines respectively.

B. Sketching Effectiveness

Figure 2 depicts the compaction effectiveness of Fennel over the NOAA dataset. We iteratively loaded each month and noted sketch memory consumption.

The cumulative sketches compaction rates improve from the first month to the last increasing from 10x—13x, 25x—27x, 250x—280x, and 15,000x—87,000x for 56, 28, 14, and 7 features respectively. We see that (1) reducing the number of features has a significant impact on sketched data size and (2) sketched size follows a logarithmic trend where compaction rates improve as the data volumes increase.

C. Profiling the Data Structure for Organizing DFVs

Figure 3 depicts the memory consumption at each individual node comprising the distributed Fennel sketches. Each of the 50 bars reflects a singe node and for each node we depict memory consumed by each of the three datasets. The standard deviations of memory usage for each dataset at each node are 0.0234MB, 0.3844MB, and 0.4053MB for Million Songs, Texas, and NOAA respectively. This benchmark highlights the Fennel DHT's ability to balance DFV placements within the cluster.

Next, we profile individual query evaluations. This experiment was performed by submitting queries from a single client to a single storage node. We have chosen to evaluate under these constraints because the Fennel DHT may optimize queries by restricting the number of storage nodes a query contacts, meaning queries returning the same number of observations may execute with highly variable performance. To more accurately catalogue query evaluations we have isolated a single node.

In Figure 4, the selectivity percentage represents the portion of records stored at the node that was returned by the query. The profiled node contained 40 million total observations. The y-axis reports elapsed time (in seconds) for each operation. For each query we report two separate values. (1) the

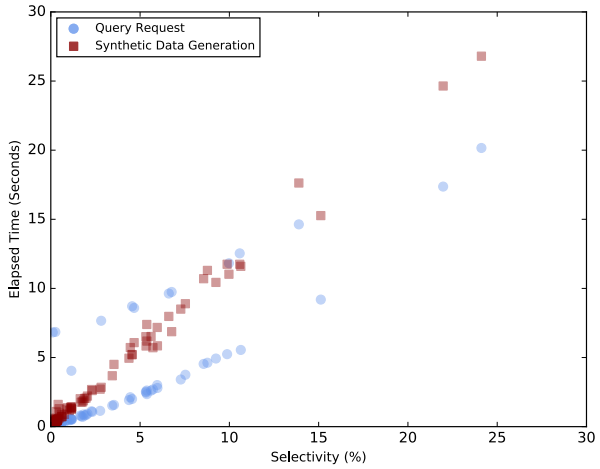


Fig. 4: Query latency and synthetic generation time for random queries on a single Fennel Storage Node.

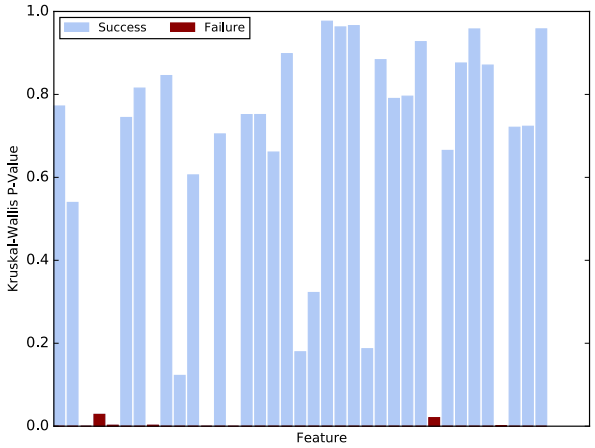


Fig. 5: P-values of the Kruskal-Wallis test for each feature in the NOAA dataset (Original and Synthetic).

latency incurred by sending a query to the node and receiving portions of the sketch (2) the duration to generate synthetic observations based on the GIST.

We see that latency for servicing a query increases linearly with selectivity. For example, 5% selectivity (∼2 million observations) takes roughly 2.5 seconds and 10% selectivity (∼4 million observations) takes about 5 seconds. Query requests are not throttled by network I/O and therefore may be executed in parallel among Fennel storage nodes.

D. Generation of Synthetic Datasets

Figure 4 profiles synthetic dataset generation. Generation of observations scales linearly with the query selectivity.

To evaluate statistical representativeness of the synthetic data we have employed the Kruskal-Wallis test to compare if the original dataset and synthetic dataset are drawn from the same population. In this test, the standard p-value less than 0.05 is used to reject the null hypothesis that the two samples are drawn from separate populations. We have provided a plot of p-values for each feature in the NOAA dataset in Figure 5. We found that only 7 of 56 features failed the Kruskal-Wallis test. Four of the failures we caused by features with already discretized values. In this case, the latitude and longitude of each collection point and a yes (1) / no (0) flag for measurable snow and rainfall. The other three were a construct of heavily skewed feature distributions. We generate synthetic values based on a uniform distribution within each bin, decreasing the size of bins over dense portions of the KDE would improve synthetic data accuracies.

E. Model Construction

1) Training Times: In Figure 6 we explore resource utilization and duration of Fennel data sampling. We compare these results to TODA (traditional on-disk analytics). Model training times will be identical between the systems for each selectivity interval so we have focused on reporting the variances in data sampling. For these experiments we monitored disk I/O, network I/O, and duration based on various sample sizes of the NOAA dataset.

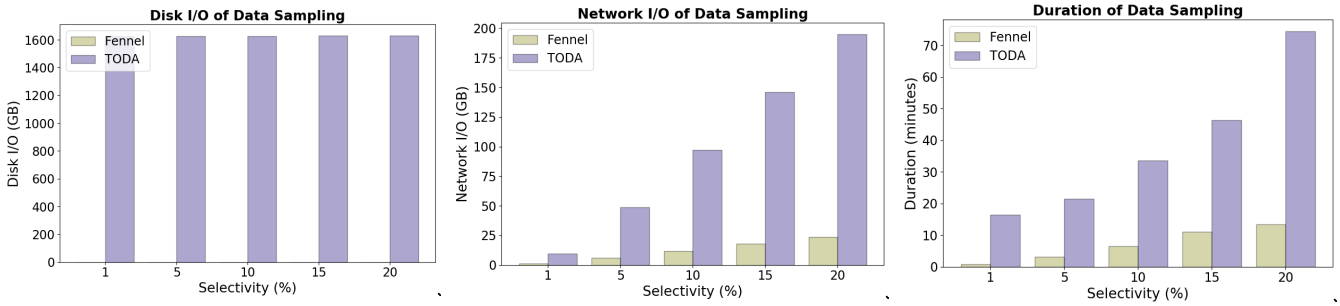
Figure 6a reports disk I/O during data sampling; as can be seen Fennel requires no disk I/O. Figure 6b reports on network I/O between the Storage and Analytics Clusters. We see that Fennel significantly reduces network I/O. **For each selectivity test Fennel shows an 88% reduction in network I/O.**

Figure 6c displays the overall duration for data sampling. **We observe a 76% - 92% reduction in data sampling times.** However, the tradeoff between savings of disk/network I/O and time required to generate accurate synthetic datasets is tunable for higher performance.

Figures 7 and 8 present sampling times for varying selectivity over the Million Songs and Texas datasets respectively. We see similar improvements as in Figure 6c showing that Fennel's advantage is not unique to the NOAA dataset.

2) Model Performance - Accuracy: We conducted experiments to asses how much accuracy is sacrificed by using the Fennel sketch that significantly reduces training times. We trained models on data sampled from the actual and the synthetic data generated from sketches and evaluated their accuracy on test data sampled from the actual full-resolution data. Figures 9a, 9b and 9c contrast the accuracy of models built on actual data with the ones trained on synthetic data for the three datasets. In Figure 9a we can see that the models trained on actual and synthetic data of Million Song dataset have similar accuracy for all sampling sizes. In this case, we significantly reduced the training time without accuracy reduction. For other datasets, models built on synthetic data are less accurate, but as can be seen the differences are not significant. Furthermore, the accuracy improves as the training data sizes become larger.

Also, we created ensembles comprising instances of the models that have already trained on the Million Song and Texas datasets. Each ensemble makes a prediction by averaging the predictions of the constituent models for a given observation. The ensembles could improve the accuracy as shown in Figures 9a and 9b and their creation does not increase



(a) Disk I/O incurred during data sampling. Fennel requires no disk I/O whereas TODA is required to read the entire dataset.

(b) Network I/O incurred during data sampling. Fennel shows a consistent 88% reduction in network I/O.

(c) Duration of data sampling. As selectivity of data increases, model training using Fennel improves from 76% - 92%.

Fig. 6: Characteristics of data sampling based on selectivity of NOAA dataset.

the training time. We can also use Fennel query to train less correlated models and construct the ensembles that will improve the accuracy further.

We used a Fennel query to partition the multidimensional feature space of the NOAA dataset into 64 sub-regions; the target that we considered for model building was *surface temperature* measured in *Kelvin*. We sampled each sub-region to create a specialized model to make predictions only for observational feature vectors belonging to the particular sub-region. We then assessed the accuracy of these specialized models by contrasting their accuracy with the best accuracy obtained from global models built on data sampled from the *entire* data. The accuracy (RMSE) achieved by leveraging specialized models was 1.04, while that using the global model was 1.39. This benchmark demonstrates the suitability of using Fennel to construct specialized models targeted for particular portions of the feature space.

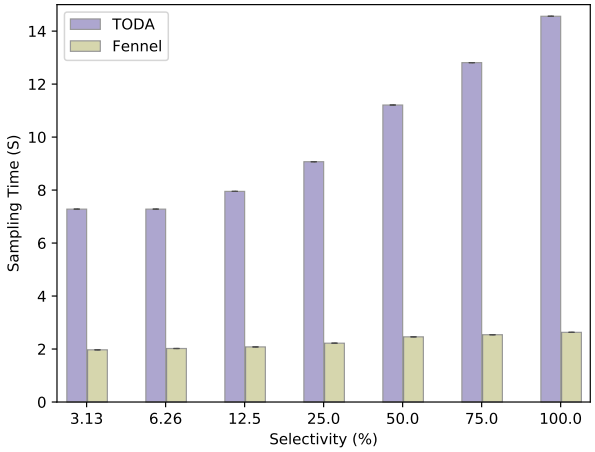


Fig. 7: Sampling times for the Million Song dataset.

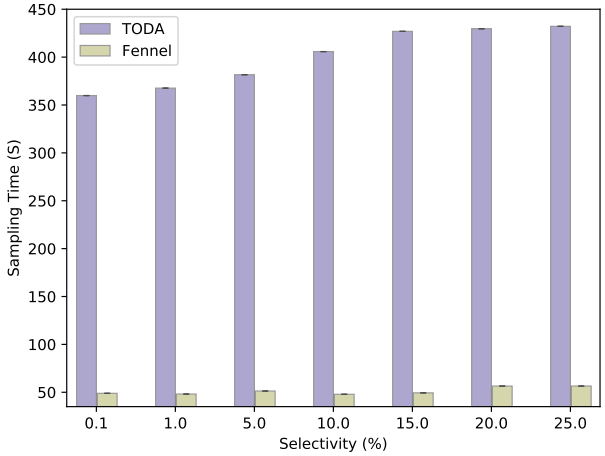
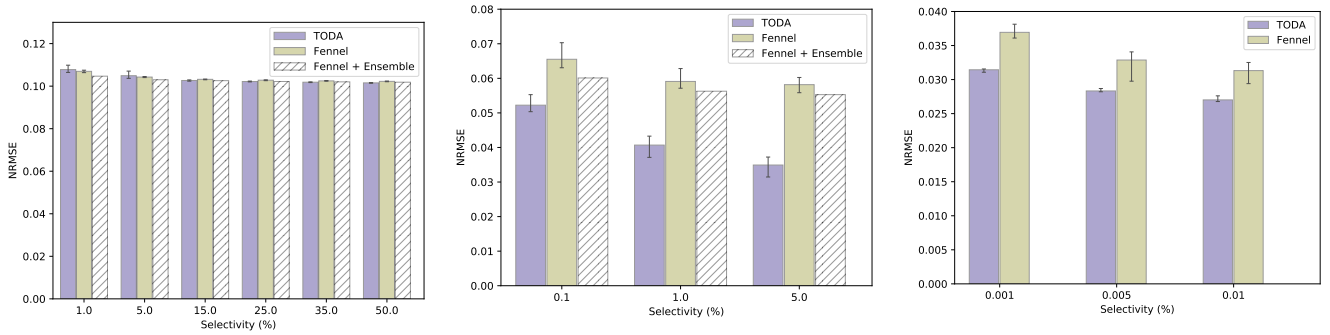


Fig. 8: Sampling times for the Texas dataset.

V. Related Work

A broad spectrum of solutions have been proposed to enable learning from large datasets with the focus on reducing training times without sacrificing accuracy. Sampling is a commonly used strategy for dealing with voluminous datasets. Different strategies [3], [4], [5] have been proposed to create a smaller representative subset of the original dataset. Although sampling enables analysis of voluminous data, it becomes infeasible for optimizing problems with a large number of parameters that involve voluminous data to achieve an acceptable solution. Full use of data that is both large in volume and variability promises opportunities to explore more hidden structures in such data [6]. Agarwal et al. [7] have shown that increasing the sampling size of such data usually leads to better accuracy.

Other solutions rely on distributed systems to build a complex model using voluminous data distributed across many networked machines. Some approaches [8], [7], [9], [10] have relied on the optimization of the MapReduce framework, and others [11], [12] employed a graph abstraction to express computations. Also, some frameworks [13], [14], [15], [16] have been especially proposed to support a broader class of machine learning algorithms. The cost of moving voluminous data among distributed machines is prohibitive, and cost concerns have forced the proposed systems to support data-centric computation. Collocating the computations with distributed



(a) Million Song Data (*Target: Release year*)

(b) Texas Data (*Target: Disease duration measured in days*)

(c) NOAA Data (*Target: Precipitable water measured in millimetres*).

Fig. 9: The accuracy (NRMSE) of models trained on different datasets sampled from the actual and synthetic data

data portions involves exploiting the natural decomposability of the objective function over the training examples or, if possible, modifying the learning algorithm to become system friendly.

Divide and conquer is another technique for enabling learning from voluminous data. This approach relies on partitioning the original dataset into subsets, fitting a predictive model over each subset independently, and combining the trained models. Several approaches [17], [18], [19] rely on this technique to allow diverse solutions for different domain problems. Some approaches employ ensemble techniques to learn from voluminous data.

Data preparation is the key to achieving efficient large-scale data analytics. Usually, each of these solutions are customized for the particular domain. Our solution stages and organizes the data such that it facilitates a diverse set of data fitting techniques efficiently.

Distributed data management and analytics are trending towards memory-based systems to alleviate magnitudes slower disk I/O. There are efforts such as Redis [20], which provide distributed in-memory key-value stores, but the lack of data compactions precludes their use. Spark SQL [21], using it's Dataframe API, provides a coupling of relational and procedural frameworks over Apache Spark [22]. Alternatively, Tachyon [23] targets the storage layer for in-memory optimizations using a hierarchical data lineage subsystem built over traditional HDFS to make *popular* data memory resident. These approaches aim to provide memory resident analytics, but may still require disk I/0 during extensive analytics. Fennel compacts datasets using data sketches allowing the system to store the sketches in-memory providing two advantages. (1) substantial alleviation of disk I/O and (2) significant reductions in network I/O during data transfers.

Frequency based sketches are designed to represent the observed frequency distribution of a dataset which is a useful construct in designing approximate query processing systems. Count Sketch [24], Count-Min [25], Misra-Gries algorithm [26], and Counting-Quotient filters [27] are examples of frequency based sketches. The sketches sizes are sub-linear to the dataset size and can be controlled through parameters associated with the accuracy and probability of exceeding accuracy bounds. These sketches though designed primarily for streaming systems, can be used as the sketching algorithm within Fennel to represent the frequency distributions of DFVs with controlled accuracy. Generating sketches over spatio-temporal observational streams is studied in [28], [29] where spatial and/or temporal attributes of data inform(s) the organization of the sketch. Synopsis comprises a set of micro-sketches arranged as a prefix tree based on the spatial attributes of the data.

There have been several efforts to perform analytics over scientific data collections [30]. Often these efforts have targeted efficiencies at the storage layer [31], [32], [33], scheduling [34], [35], and queries [36], [37]. Our methodology differs from these aforementioned approaches in our support for sketches and generation of synthetic datasets to facilitate fast analytics.

VI. Conclusions and Future Work

This study describes our methodology and reference implementation to sketch on-disk datasets, facilitate distributed organization of the sketch, and to use the sketch as the basis for feature space explorations and model training.

RQ-1: The size of the individual bins and total number of bins should be informed by the distributions associated with individual features and the NRMSE of the particular binning strategy. We track the observed feature combinations using DFVs. Compromising the resolution of the features using discretization and tracking occurrence frequencies allows us to preserve representativeness of the feature space. Multidimensional DFVs are generated based on distributed calculation of the statistical properties of the feature space.

RQ-2: Shuffling the DFVs comprising the sketch allows us to collocate DFVs, achieve compaction, and ensure accurate occurrence counts. To conserve memory footprints the in-memory data structure organized as a hash map. The memory-residency and reduction in traversals facilitates fast query evaluations that allow data scientists to interactively explore the feature space.

RQ-3: The Fennel sketch can be queried to retrieve precise portions of the feature space that are of interest. The compactness of the Gist data structure and the accompanying metadata facilitates creation of synthetic datasets that are highly representative of the on-disk data as evidenced by our statistical tests reported in Section IV-D. Model training times are significantly reduced because no disk I/O is performed during model training.

As part of future work, we will be investigating deeper integration with analytical engines. In particular, this includes seamlessly generating synthetic datasets using core engine APIs for example, DataFrames and TFRecords.

ACKNOWLEDGMENT

Research supported by funding from the Department of Homeland Security [D15PC00279]; the National Science Foundation [ACI-1553685, CNS-1253908].

REFERENCES

[1] M. Rosenblatt *et al.*, "Remarks on some nonparametric estimates of a density function," *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 832–837, 1956.

[2] E. Parzen, "On estimation of a probability density function and mode," *The annals of mathematical statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.

[3] H. Wang, R. Zhu, and P. Ma, "Optimal subsampling for large sample logistic regression," *Journal of the American Statistical Association*, no. just-accepted, 2017.

[4] R. Zhu, P. Ma, M. W. Mahoney, and B. Yu, "Optimal subsampling approaches for large sample linear regression," *arXiv preprint arXiv:1509.05111*, 2015.

[5] R. Zhu, "Poisson subsampling algorithms for large sample linear regression in massive data," *arXiv preprint arXiv:1509.02116*, 2015.

[6] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, 2014.

[7] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford, "A reliable effective terascale linear learning system," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1111–1133, 2014.

[8] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "Haloop: efficient iterative data processing on large clusters," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 285–296, 2010.

[9] Y. Zhang, Q. Gao, L. Gao, and C. Wang, "imapreduce: A distributed computing framework for iterative computation," *Journal of Grid Computing*, vol. 10, no. 1, pp. 47–68, 2012.

[10] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of USENIX NSDI*. USENIX Association, 2012, pp. 2–2.

[11] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *ACM SIGMOD*. ACM, 2010, pp. 135–146.

[12] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein, "Graphlab: A new framework for parallel machine learning," *arXiv preprint arXiv:1408.2041*, 2014.

[13] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[14] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi, "Naiad: a timely dataflow system," in *SOSP*. ACM, 2013, pp. 439–455.

[15] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server." in *OSDI*, vol. 1, no. 10.4, 2014, p. 3.

[16] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu, "Petuum: a new platform for distributed machine learning on big data," *Big Data, IEEE Transactions on*, vol. 1, no. 2, pp. 49–67, 2015.

[17] C.-J. Hsieh, S. Si, and I. Dhillon, "A divide-and-conquer solver for kernel support vector machines," in *International Conference on Machine Learning*, 2014, pp. 566–574.

[18] Q. Guo, B.-W. Chen, F. Jiang, X. Ji, and S.-Y. Kung, "Efficient divide-and-conquer classification based on feature-space decomposition," *arXiv preprint arXiv:1501.07584*, 2015.

[19] Y. Tian, X. Ju, and Y. Shi, "A divide-and-combine method for large scale nonparallel support vector machines," *Neural Networks*, vol. 75, pp. 12–21, 2016.

[20] J. L. Carlson, *Redis in action*. Manning Pubs., 2013.

[21] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi *et al.*, "Spark sql: Relational data processing in spark," in *ACM SIGMOD*. ACM, 2015, pp. 1383–1394.

[22] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[23] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Tachyon: Reliable, memory speed storage for cluster computing frameworks," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2014, pp. 1–15.

[24] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2002, pp. 693–703.

[25] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[26] J. Misra and D. Gries, "Finding repeated elements," Cornell University, Tech. Rep., 1982.

[27] P. Pandey, M. A. Bender, R. Johnson, and R. Patro, "A general-purpose counting filter: Making every bit count," in *ACM SIGMOD*. ACM, 2017, pp. 775–787.

[28] T. Buddhika, M. Malensek, S. L. Pallickara, and S. Pallickara, "Synopsis: A distributed sketch over voluminous spatiotemporal observational streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 11, pp. 2552–2566, 2017.

[29] Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias, "Spatio-temporal aggregation using sketches," in *Proceedings-International Conference on Data Engineering*, vol. 20, 2004, p. 214.

[30] S. L. Pallickara, S. Pallickara, M. Zupanski, and S. Sullivan, "Efficient metadata generation to enable interactive data discovery over large-scale scientific data collections," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE, 2010, pp. 573–580.

[31] M. Malensek, S. L. Pallickara, and S. Pallickara, "Galileo: A framework for distributed storage of high-throughput data streams," in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*. IEEE, 2011, pp. 17–24.

[32] C. Tolooee, S. L. Pallickara, and A. Ben-Hur, "Mendel: A distributed storage framework for similarity searching over sequencing data," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2016, pp. 790–799.

[33] M. Malensek, W. Budgaga, R. Stern, S. Pallickara, and S. Pallickara, "Trident: Distributed storage, analysis, and exploration of multidimensional phenomena," *IEEE Transactions on Big Data*, 2018.

[34] T. Buddhika, R. Stern, K. Lindburg, K. Ericson, and S. Pallickara, "Online scheduling and interference alleviation for low-latency, high-throughput processing of data streams," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3553–3569, 2017.

[35] S. L. Pallickara and M. Pierce, "Swarm: Scheduling large-scale jobs over the loosely-coupled hpc clusters," in *SWARM: Scheduling Large-Scale Jobs over the Loosely-Coupled HPC Clusters*. IEEE, 2008, pp. 285–292.

[36] M. Malensek, S. Pallickara, and S. Pallickara, "Analytic queries over geospatial time-series data using distributed hash tables," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 6, pp. 1408–1422, 2016.

[37] M. Malensek, S. L. Pallickara, and S. Pallickara, "Fast, ad hoc query evaluations over multidimensional geospatial datasets," *IEEE Transactions on Cloud Computing*, vol. 5, no. 1, pp. 28–42, 2017.