Hermes: Federating Fog and Cloud Domains to Support Query Evaluations in Continuous Sensing Environments

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

Abstract—As networked sensing devices continue to proliferate, the storage and processing capabilities of *fog* or *edge* devices have also increased substantially while offering low energy consumption and hardware costs. These complimentary technologies enable unique opportunities for performing decentralized analysis on the edges of the network while also leveraging the capabilities of public and private clouds for coordination and long-term storage. Our framework, HERMES, enables federated query evaluations that transition between cloud and fog nodes seamlessly. The system selectively samples from observational streams to reduce communication and memory consumption on fog nodes; our evaluation over a real-world observational dataset demonstrates an 89% reduction in dataset size while maintaining a mean absolute error of less than 0.25%.

Index Terms—Fog storage, Raspberry Pi, federated query evaluations, sensor networks

1 INTRODUCTION

ADVANCEMENTS in miniaturization, improved network connectivity, and falling costs have led to a proliferation of sensing devices that underpin the Internet of Things (IoT). Gartner and ABI project upwards of 20-30 billion devices on the Internet of Things by 2020. These IoT systems are often backed by *fog* architectures, where fog nodes at the edges of the network are responsible for preliminary storage and processing of observations. Fog nodes then coordinate with cloud-based services to synchronize state or publish updates. The measurements reported by fog nodes in this study are spatiotemporal and encompass multiple *features* (dimensions) including temperature, humidity, visibility, etc. Given the recording frequencies of the sensors we consider, the resulting datasets are voluminous and cannot be managed solely by fog nodes.

Herein we present HERMES, a framework that bridges the fog and cloud support analysis through federated query evaluations. Queries may target arbitrary spatiotemporal scopes, statistical properties of the feature space, and temporal events. To facilitate these queries, our novel *spillway* data structure extends reservoir sampling [1] to provide variable-precision samples of incoming data streams in resource-constrained environments. The fog nodes we consider in this paper are Raspberry Pi single-board computers that provide cost savings and energy efficiency, but have lower-end processing capabilities. Though our fog nodes are implemented on the Raspberry Pi, HERMES and the spillway data structure can be used with other devices such as desktops, Banana Pi, ODroid, etc.

• *M. Malensek, S. Pallickara, and S. Pallickara are with the Computer Science Department, Colorado State University, Fort Collins, CO 80523.*

1.1 Scenario

Consider an electric utility company that specializes in wind power generation. Before turbines can be constructed, a geographical survey must be performed to locate regions with consistent, non-turbulent wind. While these attributes are largely a function of the surrounding geography and weather patterns, national meteorological data is too coarse-grained to guide turbine placement. To acquire accurate, fine-grained climate data, the organization deploys wireless sensing devices on and around the premise to monitor weather patterns, including wind speed, temperature, humidity, pressure, etc. for a year or more at the rate of multiple readings per second [2]. These observations are collected by fog nodes on-site and managed by HERMES, which provides query capabilities over high-resolution samples collected recently, as well as historical data stored in the cloud. After successful planning and construction of the wind farm, the organization continues to use these devices in conjunction with HERMES to optimize their power generation strategy by analyzing weather patterns, long-term trends, and sensing data from the turbines themselves.

1.2 Research Questions and Approach

To guide the development of HERMES, we explored the following research questions:

RQ1 How can we harness resource-constrained fog nodes to support query evaluations in a federated environment?

RQ2 Given the data volumes at hand, how can we minimize information exchanged between fog nodes at the edges of the network and the cloud?

RQ3 Considering the distributed and decentralized nature of the system, how can we ensure that query evaluations are timely and accurate?

Our methodology for achieving these goals is divided between the HERMES system architecture (§2) and our spillway data structure (§3). At the system level, cloud and fog nodes must coordinate during storage and retrieval operations, with fog nodes handling recently-collected data and cloud nodes responsible for historical storage. This involves controlling client access to fog nodes, restricting communication within the system, and caching frequently-accessed data in the cloud for analytics (§2.1). We also employ a geospatially-aware data partitioning scheme to divide storage and processing workloads (§2.2). Each fog node maintains a spillway data structure to ensure efficient memory management for incoming sensor readings. This is achieved through a novel, hierarchical arrangement of reservoir samples (§3.2) and a temporal aging strategy that allows for a variable accuracy gradient. Each reservoir also maintains online summary statistics to provide information about the full-resolution data distributions (§3.3), and observations are placed in a red-black tree to facilitate time series queries (§3.4).

1.3 Related Work

Bonomi et al. [3], [4] surveys several use cases associated with fog computing, including smart traffic lights and wind farms, and proposes architectural designs for dealing with unique challenges such as geo-distribution and coordination between the fog and cloud. In the *fog abstraction layer*, policies define how requests are routed between participating components. However, the proposed fog analytics use case does not consider the memory efficiency of the data structures employed. *Context awareness* is also a crucial aspect of the IoT, characterizing entities and related events [5]. HERMES employs query-based context distribution and performs filtering and aggregation during context acquisition.

Tang et al. [6] outlines the design of a multi-tier fog architecture for processing and detecting events in data streams. The system includes intermediate nodes between the fog and cloud nodes, and demonstrates both the latency benefits and reduced communication of a federated approach. Jayaraman et al. [7] proposes cost models for data transmission and power consumption in fog-based analytics contexts. Performing analytics on fog nodes and applying data reduction techniques results in substantial energy savings over simply transmitting all observations to the cloud.

Alternative approaches for sampling time series data include applying bias functions to a reservoir to boost the representativeness of recent samples [8] as well as timestamp-based windows that update sample instances based on priorities [9]. Weighted reservoir sampling may also boost the accuracy of recently-observed data points by gradually increasing insertion probabilities over time.

2 SYSTEM ARCHITECTURE

HERMES is a two-tiered storage and analysis framework consisting of *cloud nodes* and *fog nodes* (see Figure 1-a for an architectural overview). While cloud nodes are responsible for heavyweight analytics and long-term storage, fog nodes are tasked with processing incoming sensor data. As a result, fog nodes provide low-latency access to recent observations, with cloud nodes managing historical data and coordinating activities across the system. This separation of concerns reduces network traffic because sensing devices are able to communicate directly with fog nodes in close geographical proximity rather than transmitting data to a central location. Additionally, certain types of near-term analysis can be conducted entirely on the fog nodes, which reduces the amount of cloud resources necessary for operation.

Both the cloud and fog components support the same set of query operations but diverge in how they handle storage. Fog nodes sample from the incoming data streams and hold high-resolution data for shorter periods of time, whereas cloud nodes coordinate long-term storage by migrating observations to a distributed storage platform such as Galileo [10], [11], HDFS, or Apache Cassandra.

2.1 Query Coordination

A key aspect of fog computing is edge devices are generally low-powered from both computational and energy efficiency perspectives. Consequently, fog nodes become overwhelmed by requests faster than traditional server hardware. To provide control over request rates and limit query activities that may impact the management of incoming observations, we require client applications to submit query **requests** to cloud nodes rather than directly to fog nodes. During query resolution, the coordinating cloud node determines the locations of relevant data points based on the spatial scope(s) requested (see §2.2), and grants authorization for communication with fog nodes in the form of non-reusable *access tokens*. In situations where historical data in the cloud satisfies a query, the client will be directed to relevant cloud nodes instead.

Requiring clients to coordinate with cloud nodes as a precursor to query resolution has the added benefit of providing the system with usage metrics. If a particular query is executed frequently by clients, cloud nodes will retrieve and cache the specific observations for future use. This approach does not result in duplicate communications because all records will eventually be migrated to the cloud. For types of analysis that require frequent access to recently-recorded data, the system grants *long-term* access tokens for fixed queries. By default, long-term access tokens expire after one hour and are revoked in situations where load on the fog node(s) is too high to maintain average response latencies.

2.2 Spatial Partitioning

HERMES accounts for the geospatial dispersion of sensing equipment and fog nodes by partitioning data with the *Geohash* algorithm [12]. Geohash divides the Earth into a hierarchy of spatial bounding boxes that are identified by Base-32 strings. Longer strings describe finer-grained spatial regions in the hierarchy, while shorter strings represent coarser-grained areas. For instance, the Geohash *9Q* spans a region including the majority of California and Nevada in the United States, while *9QQJ* refers to a subregion surrounding the city of Las Vegas, Nevada.

Given a region under study, HERMES assigns responsibility for sensor readings to fog nodes based on Geohash proximity. Each fog node represents one or more *stations* that process incoming data for a particular geospatial area; for example, all sensors within the *9Q* Geohash could be configured to report to a single node. To cope with diversity in the underlying data streams and ensure load balancing, the Geohash precision used for assigning stations is configurable. During query operations, target spatial locations are specified by the user as Geohash strings or spatial bounding boxes that are subdivided into Geohashes before being distributed to relevant fog nodes. Evaluating such queries involves computing a string prefix match against the list of available sensor recording stations.

3 Federated Storage and Retrieval

To enable federated storage and retrieval functionality, HERMES employs: (1) statistically robust sampling methods to manage the space-accuracy trade-off at fog nodes, (2) full-resolution monitoring of data distributions, and (3) expressive query support.

3.1 Reservoir Sampling

Reservoir sampling is a probabilistic, random sampling algorithm proposed by J. S. Vitter that facilitates sample generation when the source dataset is of an unknown size or too large to fit in main memory [1]. Reservoir instances are maintained as fixed-size arrays, resulting in predictable memory consumption. These characteristics make reservoir sampling particularly useful in resource-constrained environments such as embedded systems, fog nodes, or IoT devices.

While there are several variations of the reservoir sampling algorithm, the particular implementation we employ in this work was designed to be suitable for distributed applications. During initialization, reservoirs are created with a backing array of size n. As data points stream into the system, the first n entries are placed into the reservoir immediately and the observation counter, C, is incremented. Entries may be objects of any type, including numeric values or multidimensional records. Once the initial array is full, subsequent entries are assigned a randomized key k in the range [0,1] that determines whether the entry will be inserted into the reservoir. Entries where $k < n/C$ are

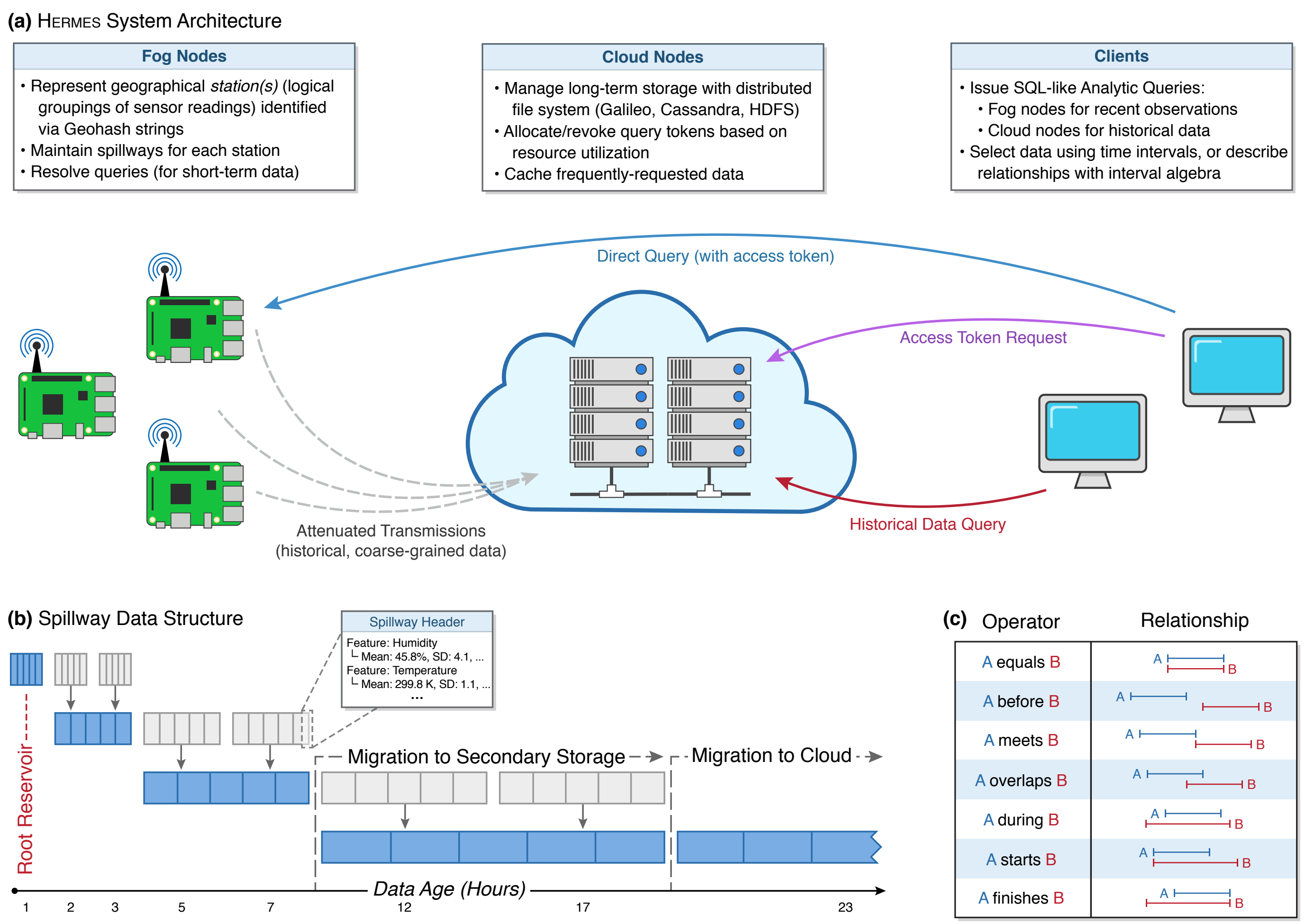
(a) HERMES System Architecture
Fog Nodes
• Represent geographical station(s) (logical groupings of sensor readings) identified via Geohash strings
• Maintain spillways for each station
• Resolve queries (for short-term data)
Cloud Nodes
• Manage long-term storage with distributed file system (Galileo, Cassandra, HDFS)
• Allocate/revoke query tokens based on resource utilization
• Cache frequently-requested data
Clients
• Issue SQL-like Analytic Queries:
• Fog nodes for recent observations
• Cloud nodes for historical data
• Select data using time intervals, or describe relationships with interval algebra
Direct Query (with access token)
Access Token Request
Historical Data Query
Attenuated Transmissions (historical, coarse-grained data)
(b) Spillway Data Structure
Spillway Header
Feature: Humidity
└ Mean: 45.8%, SD: 4.1, ...
Feature: Temperature
└ Mean: 299.8 K, SD: 1.1, ...
...
Root Reservoir
Migration to Secondary Storage
Migration to Cloud
Data Age (Hours)
1
2
3
5
7
12
17
23
(c) Operator
Relationship
A equals B
A before B
A meets B
A overlaps B
A during B
A starts B
A finishes B

Fig. 1. **(a)** An overview of the components and system architecture of HERMES. **(b)** A demonstration of our *spillway* data structure, which manages a set of hierarchical reservoirs to preserve a variable accuracy gradient across observational data. Insertions are performed on the *root reservoir*, and reservoirs are merged as they age to manage memory consumption. In this example, 18 hours worth of observations are shown. **(c)** The set of interval algebra operators supported by HERMES for analysis.

selected for insertion, randomly replacing an existing entry; as a result, the probability that a given sample will be assimilated into the reservoir decreases over time as the observation count C increases.

In addition to raw data points, each reservoir entry tracks its randomly-generated insertion key. This enables reservoirs from disparate nodes or time frames to be merged, forming an aggregate reservoir. During a merge, entries are sorted by their insertion keys, with the smallest n keys retained in the aggregate reservoir.

3.2 Spillways

Reservoir sampling is an effective means for producing random samples over unbounded streams. However, this *unbiased* approach may not be desirable in all scenarios as it tends to mask recent evolution in the stream. Consider a reservoir sample of temperature data over a month-long timespan; if an abnormal weather pattern causes large fluctuations in temperature, the probability that the observations will be included in the sample is lower due to the age of the reservoir. While this behavior may be desirable for long-term climate monitoring, it limits short-term analysis of weather phenomena. To facilitate these use cases, our *spillway* data structure is designed to allow short-term analysis while preserving long-term data samples.

Spillways are implemented as hierarchical collections of reservoirs with bounded lifetimes. Each reservoir within a spillway contains the same number of samples but may span varying temporal scopes; for example, a spillway could be configured with an initial timespan of 12 hours, followed by 24-hour reservoirs, and so on. Each successive level in the spillway hierarchy describes a broader duration of time and therefore represents a coarser-grained sample. Insertions are performed on the *root* reservoir, which manages the current time slice. When the root *matures*, a new root is created to accommodate incoming observations for the next time slice. This process continues until the spillway size limit is reached, at which point two reservoirs are selected to be merged. Figure 1-b provides an overview of a simplified spillway instance and its components.

During a merge operation, the **oldest** reservoirs from the **finest** temporal granularity are targeted first. Consequently, merges tend to occur on short-term reservoirs, with long-term data merged less frequently. This helps maintain a gradual accuracy curve over the entire spillway rather than producing a steep drop in granularity after a particular amount of time. Merge functionality is parameterized in two ways: first, the *temporal curve* describes the increase in timespan size at each level in the hierarchy. The example in Figure 1-b demonstrates a temporal curve of $f(x) = 2^x$. Second, the *merge threshold*, T, specifies how many additional reservoirs of the same timespan must exist before a merge can take place on a per-level basis; HERMES defaults to a minimum merge threshold of $T = 2$, but the threshold can be increased to improve accuracy or elongate particular levels of the hierarchy. For example, a study configured with hour-long root reservoirs may require fine-grained samples for the first 24 hours, in which case setting $T = 26$ for the first level would ensure merges only occur when more than 24 hours of observations have been collected.

Over time, merges occur in a cascading fashion with samples becoming coarser-grained. For any given set of spillway parameters, there is a point where no sufficient reservoirs exist to merge. When this occurs, the oldest reservoir in memory is selected for migration to secondary storage (generally flash memory). During migration the reservoir is compressed and written to the storage pool sequentially, allowing future retrieval based on temporal ranges. After migration is complete, the reservoir entry is removed from memory and a new root reservoir is allocated. To ensure incoming observations are not lost during this procedure, the spillway is allowed to temporarily surpass its size limit. Once complete, the compressed and migrated reservoirs will be pulled to the cloud for historical storage on a set interval during idle periods. We call this delayed migration to the cloud *attenuated transmissions*.

Spillways support configuration of both the temporal curve and merge threshold, which in turn determines the size of the data structure. A base unit size must also be provided, such as ‘one hour,’ ‘seven days,’ etc. For many problem types, simply configuring the unit size and using the default settings will provide reasonable performance that can be further fine-tuned at run time.

3.3 Spillway Headers

Reservoirs sample from the underlying data stream, but tend to overlook extreme or anomalous events. Furthermore, data sources that exhibit large variations in values will be described by their average case, which may mask true behavior patterns. To provide information about extreme events and measure how well a reservoir captures its source data stream, we include *spillway headers* that contain running summary statistics for each reservoir instance. As multidimensional observations arrive, we use Welford’s method [13] to update online statistics for the current root reservoir. This approach does not require access to previous data points, which may be absent from the current root. Summary statistics include the running mean, variance, standard deviation, minimum/maximum values, and cross-feature relationships such as correlations and regression coefficients. The total number of full-resolution observations that were received during the active timespan of the reservoir is also available, allowing the system to report exact sample sizes for each reservoir instance — regardless of data arrival rates.

A key feature of these reservoir summaries is that they can be merged to form an aggregate summary without inspecting the underlying data points. We exploit this property during reservoir merge operations to ensure each reservoir includes up-to-date summary information. This is also the rationale behind our minimum merge threshold of 2; if partial reservoirs were merged, spillways could no longer maintain accurate summary information about the full-resolution streams. An example header in the 3rd level of the spillway hierarchy is shown in Figure 1-b.

3.4 Time Series Query Support

While the data from entire reservoirs within a spillway can be requested by client applications, time series query

TABLE 1
Fog storage performance evaluation using the F2FS file system. Each disk was tested for read, write, and random seek performance, with the storage type listed in the first column; *S* indicates an SD card, *U* indicates a USB-based flash drive, and *H* represents a traditional 2.5" hard disk drive at 5400 RPM over USB. We also include results from benchmarking spillway migration operations, listed in the final column.

Disk	**Manufacturer**	**Model**	**Capacity**	**Read (MB/s)**	**Write (MB/s)**	**Rand. Seek/s**	**Migrate (MB/s)**
$S1$	PNY	P-SDU128U185EL-GE	128 GB	34.0	9.2	805	10.9
$S2$	Samsung	MB-ME128DA/AM	128 GB	24.6	14.5	1254	15.4
$S3$	Samsung	MB-ME32DA/AM	32 GB	33.3	12.9	1690	14.4
$S4$	Samsung	MB-MP32DA/AM	32 GB	45.0	15.4	2371	18.3
$S5$	SanDisk	SDSQUNC-032G-GN6MA	32 GB	45.1	14.8	2665	16.8
$U1$	Lexar	LJDS45-128ABNL	128 GB	42.3	21.8	1746	22.1
$U2$	Samsung	MUF-128BB/AM	128 GB	43.4	22.1	1299	21.3
$U3$	SanDisk	SDDD2-128G-G46	128 GB	41.4	8.9	569	7.1
$H1$	Western Digital	WD6400BPVT-75HXZT1	640 GB	40.0	31.8	172	33.5

support enables increased precision and control over retrieval operations. As observations are inserted into the root reservoir, they are also placed in a red-black tree based on timestamps. Red-black trees are self-balancing binary search trees that enable range queries and exact-match lookups over the dataset. One advantage of red-black trees is they tend to have high insertion and removal performance for moderately-sized datasets. Once indexed, observations can be retrieved by client applications in fine-grained temporal intervals without requiring knowledge of the underlying reservoirs. To ensure the red-black tree stays consistent, observations that are removed during merge operations from the reservoirs are also deleted from the tree.

To provide more expressive temporal queries, we support a variety of operators from Allen's interval algebra [14], shown in Figure 1-c. These operators allow the comparison of *events* by selecting data points that match particular criteria. For instance, HERMES can evaluate queries such as "retrieve observations where precipitation and cloud cover were correlated **during** June 5th through the 12th." Note that these relationships operate specifically on time bounds; for example, a study of extreme climate conditions may request: "retrieve timespans when the temperature was above 100° F and **overlap** with spans that saw humidity levels of over 90%." HERMES provides an SQL-like query interface that allows range, exact match, statistical, and temporal operators to be composed and chained.

4 EXPERIMENTAL EVALUATION

Our fog testbed was composed of 48 Raspberry Pi 3 Model B nodes (1.2 GHz, 1 GB RAM, 160 GB flash storage) running Arch Linux. Connectivity to the cluster, located in Colorado, was provided by 100 Mbps Ethernet. Cloud nodes were sourced from the Amazon Elastic Compute Cloud (EC2), with 16 `m4.large` instances split evenly between the US West (Oregon) and US East (N. Virginia) regions. Finally, test clients consisted of `t2.small` EC2 instances located in the US West (N. California) region. All EC2-based instances were configured with HVM virtualization, CentOS 7.2, and HERMES was executed on Oracle JDK version 1.8.0.

We used two datasets to evaluate the performance of HERMES. The first dataset was collected from the NOAA NAM Forecast System from 2013–2015. The NAM contains atmospheric observations with a variety of features such as the date, time, location, surface temperature, humidity, wind speed, and precipitation. With the subset of features used for this study, the total dataset size was 15 TB. The second dataset was recorded directly by our fog nodes and included several ambient conditions from a busy computer lab with over 50 workstations at Colorado State University, including the temperature/humidity (DHT-11 sensor), sound level (CM108 audio controller), number of logged in users, system load averages, and overall memory usage; given its relatively small set of features, the total dataset size was 1.4 GB. However, while the NAM dataset reports several times per day, the lab dataset measurements were taken approximately every two seconds.

4.1 Fog Storage Media

While cheap, reliable storage is broadly available in commodity workstations and servers, fog nodes tend to be constrained by their I/O interfaces and available storage technologies. Our Raspberry Pi nodes have two viable storage interfaces, MicroSD and USB 2.0. We surveyed several options in this space to determine which were the most cost-effective and performant for our usage scenario. Table 1 provides the results of our performance evaluation for a variety of SD and USB storage technologies on our fog hardware. We used the Flash-Friendly File System (F2FS) [15] for these benchmarks with the exception of disk *H1* (traditional HDD), as performance was 5-20% better across all devices when compared to the commonly-used ext4 or XFS file systems. This performance boost is likely because of F2FS's flash-specific physical storage layout. Ultimately, we used a combination of disks *S5* and *U1* in each Raspberry Pi unit to achieve a total storage pool of 160 GB per machine, based on cost, availability, and performance.

4.2 Spillway Accuracy

To evaluate the accuracy of our spillway data structure, we issued a series of queries on both of our test datasets and compared the results with full-resolution source data. The NAM configuration divided North America among our

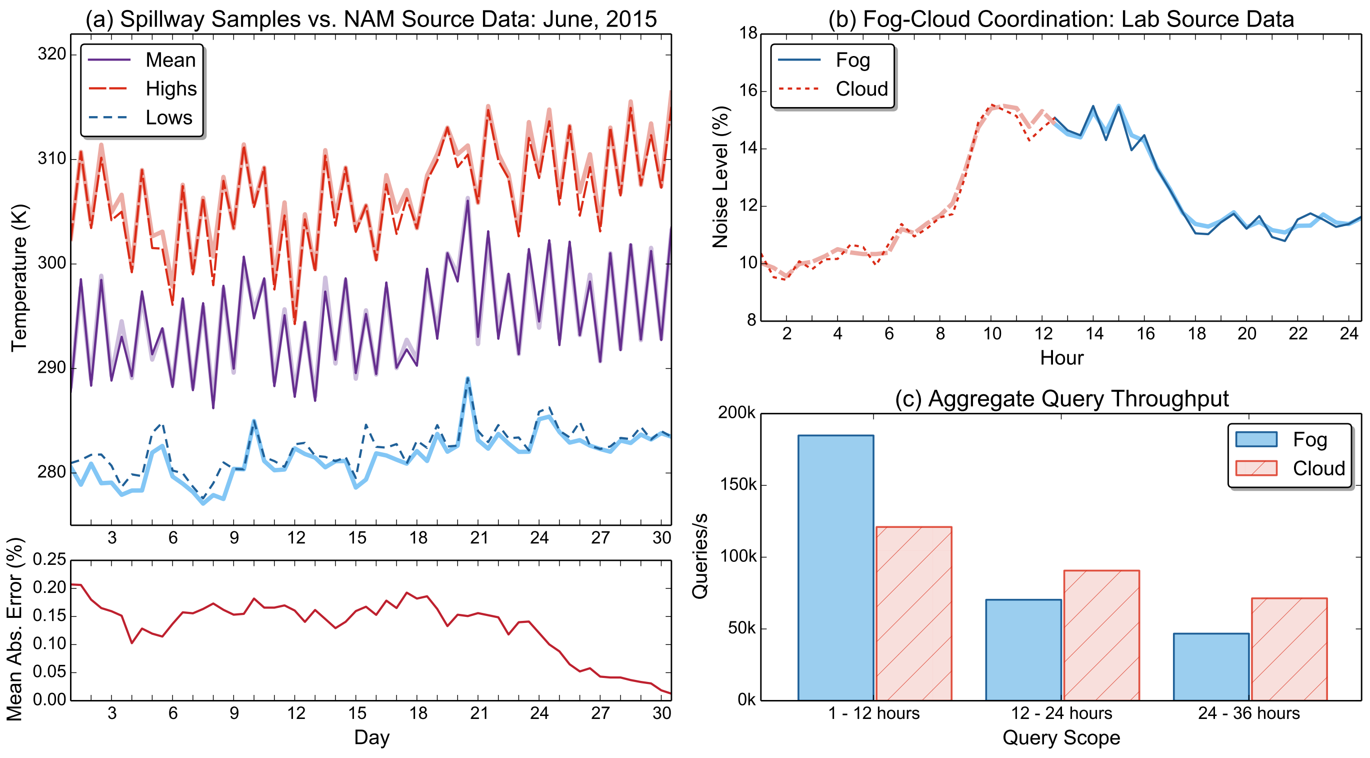

Fig. 2. HERMES performance evaluation: **(a)** A comparison of spillway values and source data measured by the aggregate mean absolute percentage error. Using fine-grained reservoirs, the most recently-recorded data exhibits the lowest error, with a maximum error of less than **0.25%**. **(b)** Noise level query against our lab dataset, which was handled by both fog and cloud nodes. A clear pattern of heightened noise during working hours is apparent. **(c)** Comparison of aggregate queries (across 48 and 16 fog and cloud nodes, respectively) resolved per second.

48 fog nodes with two-character Geohash strings. Each fog station was also configured to maintain 32 stations (three Geohash characters), for a total of 1536 spillways in the system. Each spillway consumed 25-28 MB of main memory on average, and were parameterized with a unit size of 24 hours, a temporal curve of $f(x) = 2^x$, and merge thresholds of $T = [1, 2, 4, 5]$ for each of the four levels in the hierarchy (thresholds listed starting with root reservoirs). Figure 2-a contains the high, low, and mean temperatures for station *9XJ* over 30 days in June of 2015. We retrieved these values by issuing queries across 12-hour intervals. The figure also contains the mean absolute percentage error of the results compared to the source data, which demonstrates how the spillway accuracy curve ensures the most recent data (near the end of the month) is maintained with the highest accuracy. At this particular accuracy level, the configuration resulted in an **89%** reduction in dataset size, allowing an entire month of NAM data to reside in main memory across the fog cluster.

In our second error evaluation, we artificially constrained the system to demonstrate coordination between fog and cloud nodes. A single fog node was tasked with managing our lab sensing dataset, with a spillway configuration of 30-minute units, a temporal curve of $f(x) = x + 1$, and merge thresholds for three levels of reservoirs: $T = [3, 2, 2]$. Figure 2-b reports the lab noise level measured by our CM108 sensor; audible activity increases during working hours, although it is evident that several students continued using the lab into the night. The figure highlights the components responsible for resolving the query, with the fog and cloud each returning 12 hours of data.

4.3 Query Throughput Microbenchmark

To determine the query processing throughput of our cluster configuration, we launched randomized queries from our test clients across three different timespans on the NAM dataset: 1–12 hours, 12–24 hours, and 24–36 hours. Longer ranges of time are more computationally expensive and tend to incur higher message serialization costs, whereas retrieving observations from a small number of time units is generally faster. Figure 2-c contains the query throughput for each timespan, which includes query resolution and serialization. Notably, the 48 fog nodes were able to achieve higher throughput than the 16 cloud nodes for small ranges of time, likely due to the first benchmark being I/O-bound rather than CPU-bound.

5 CONCLUSIONS

Our methodology makes innovative use of reservoir sampling to support accurate, real-time, federated query evaluations with our *spillway* data structure (**RQ1**). To minimize communication, fog nodes buffer incoming observations in memory and secondary storage before migrating data to the cloud periodically, while cloud nodes cache frequently-requested data points (**RQ2**). Preserving variable-granularity samples at the fog nodes allows queries to achieve high accuracy while minimizing latency and I/O (**RQ3**). Finally, our empirical evaluations demonstrate the suitability of our approach in federated fog and cloud sensing environments.

Acknowledgments

This research was supported by the US Department of Homeland Security [HSHQDC-13-C-B0018, D15PC00279], the US National Science Foundation [ACI-1553685, CNS-1253908], the Environmental Defense Fund [0164-000000-10410-100], and a Monfort Professorship.

References

[1] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, vol. 11, no. 1, pp. 37–57, Mar. 1985.

[2] EWEA, *Wind energy - The Facts: A Guide to the Technology, Economics and Future of Wind Power*. Routledge, 2012.

[3] F. Bonomi *et al.*, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, 2012, pp. 13–16.

[4] ——, *Fog Computing: A Platform for Internet of Things and Analytics*. Springer International Publishing, 2014, pp. 169–186.

[5] C. Perera *et al.*, "Context aware computing for the internet of things: A survey," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[6] B. Tang *et al.*, "A hierarchical distributed fog computing architecture for big data analysis in smart cities," in *Proceedings of the ASE BigData & SocialInformatics 2015*, 2015, pp. 28:1–28:6.

[7] P. P. Jayaraman *et al.*, "CARDAP: A scalable energy-efficient context aware distributed mobile data analytics platform for the fog," in *Advances in Databases and Information Systems*. Springer, 2014, pp. 192–206.

[8] C. C. Aggarwal, "On biased reservoir sampling in the presence of stream evolution," in *Proceedings of the 32nd International Conference on Very Large Data Bases*, 2006, pp. 607–618.

[9] B. Babcock, M. Datar, and R. Motwani, "Sampling from a moving window over streaming data," in *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002, pp. 633–634.

[10] M. Malensek, S. L. Pallickara, and S. Pallickara, "Fast, ad hoc query evaluations over multidimensional geospatial datasets," *IEEE Transactions on Cloud Computing*, p. (To Appear).

[11] ——, "Analytic queries over geospatial time-series data using distributed hash tables," *IEEE TKDE*, vol. 28, no. 6, pp. 1408–1422, 2016.

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

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

[14] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, Nov. 1983.

[15] C. Lee, D. Sim, J.-Y. Hwang, and S. Cho, "F2FS: A new file system for flash storage," in *Proceedings of the 13th USENIX FAST*, 2015, pp. 273–286.

Matthew Malensek is a PhD student in the Department of Computer Science at Colorado State University. His research interests include distributed systems, data-intensive computing, and cloud computing. Malensek received his MS in Computer Science from Colorado State University and can be reached at malensek@cs.colostate.edu.

Sangmi Lee Pallickara is an Assistant Professor in the Department of Computer Science at Colorado State University. Her research interests are in the area of large-scale scientific data management and data mining. She received her Masters and Ph.D. degrees in Computer Science from Syracuse University and Florida State University, respectively. She is a recipient of the NSF CAREER award. Contact her at sangmi@cs.colostate.edu.

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