A Performance Study of Geo-Distributed IoT Data Aggregation for Fog Computing

Shigeru Imai, Carlos A. Varela, and Stacy Patterson
Department of Computer Science, *Rensselaer Polytechnic Institute*
imais2@rpi.edu, {cvarela, sep}@cs.rpi.edu

***Abstract*—We investigate MapReduce-based data aggregation for Internet-of-Things data in a multi-tier, geo-distributed datacenter architecture. Specifically, we consider 1) end-to-end hierarchical data aggregation and 2) query response for aggregated data requests made by geo-distributed clients. We first develop a realistic performance model based on previous empirical studies. We then study application performance for various deployment architectures, ranging from a purely cloud-based approach to a geo-distributed architecture that combines cloud, fog, and edge resources. From simulations created based on U.S. Census data, we characterize the trade-off between end-to-end data aggregation time and query response time. Our experiments show that for data aggregation, a purely-cloud based deployment is 53% faster than a deployment with edge resources; however, for query response, the edge approach is 46% faster due to the edge resource proximity to query clients.**

***Index Terms*—geo-distributed, map-reduce, edge, fog**

I. Introduction

In 2017, Gartner forecasted that there will be 20.4 billion Internet-of-Things (IoT) devices by 2020 [1]. IoT devices, such as smartphones or power plant sensors, are distributed across the world and will generate an enormous amount of *geo-distributed data*. The scalability of cloud computing has enabled centralized processing of big data; however, considering the unprecedented scale of growing IoT adoption, transferring data directly from IoT devices to cloud datacenters is not always feasible due to higher latencies, large data volume, and bandwidth consumption [2], [3].

To fill the gap between cloud and IoT devices, a new computing paradigm called fog computing [2] (or edge computing [3]), has been proposed. Fog computing envisions to bring computing power closer to network edges so that the data produced by IoT devices can be processed with lower latency. These edge servers can then transmit aggregated data to the cloud for further processing, reducing bandwidth utilization in the core network. This approach can naturally be extended to a hierarchical architecture, where computing resources of varying power are available for different computing tasks. For example, local micro data centers can process IoT data within a single municipality. Aggregated data can then be transmitted to county and/or state data centers to generate statistics for multiple municipalities, and these results can be passed to the cloud to perform analysis on regional data.

This research is partially supported by the DDDAS program of the Air Force Office of Scientific Research, Grant No. FA9550-15-1-0214 and NSF Awards, Grant No. 1462342, 1553340, and 1527287. The authors would like to thank an AWS Cloud Credits for Research award.

Motivating applications are location-based information services, such as Dark Sky (weather information) [4] and Waze (traffic information) [5], which provide useful local information to users. These services have two aspects that make them good candidates for hierarchical processing: (1) aggregation of a massive amount of geo-distributed IoT sensor and/or crowdsourced data (*e.g.*, barometer data collected from smartphones [4], [6]), and (2) processing queries for the aggregated results requested from geo-distributed users.

We consider a hierarchical data-aggregation topology based on multiple map and reduce operations, as shown in Fig. 1 (described in detail in Sec. II). The hierarchical data aggregation

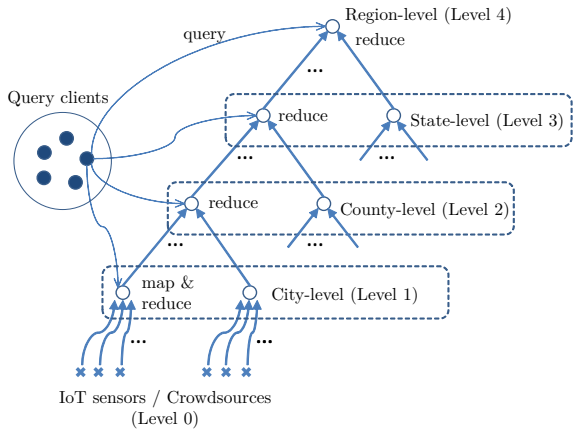


Fig. 1. Logical topology of hierarchical data aggregation with *map* and *reduce* operations.

provides users with useful location-dependent information for multiple geographical levels, such as cities, counties, states, and regions (*e.g.*, northeast & midwest U.S.). Users request this information through queries, for example, "What is the average temperature of city X?" and "What are the current traffic conditions in county Y?". Information obtained from IoT sensors and crowdsources are gradually aggregated towards the root of the aggregation topology. Each node corresponds to a geographical area and offers aggregated data about its own area. We deploy this topology over multi-tier geo-distributed datacenters. Such hierarchical data processing has been studied for two-tier MapReduce [7], [8] and recently for a multi-tier serverless computing framework [9].

Our aim is to characterize the performance trade-offs of such hierarchical MapReduce jobs in different fog-based ar-

chitectures vs. a purely cloud-based approach. We develop performance models for hierarchical data aggregation with MapReduce and for query response for aggregated data. Using these models, we systematically investigate how different mappings between the logical hierarchical aggregation topology and the physical datacenter resources affect the performance of geo-distributed data aggregation and query response. To perform realistic performance simulations, we use real-world geography and population data obtained from the U.S. Census [10], [11]. The simulation results confirm that there is a trade-off relationship between end-to-end data aggregation time and query response time. A purely-cloud based deployment executes the MapReduce tasks 53% faster than a deployment with edge resources; however, for query response, the approach with edge resources 46% faster due to client proximity.

The rest of this paper is organized as follows. In Sec. II, we define our logical and physical architectures. In Sec. III, we present a performance model for end-to-end hierarchical data aggregation and an average query response time model to retrieve aggregated data. Sec. IV gives performance results comparing several mappings of logical data aggregation topologies to multi-tier datacenters. We conclude in Sec. V.

II. Hierarchical Data Aggregation

In this section, we describe how to aggregate geographically distributed data through hierarchical map and reduce operations. The hierarchical data aggregation provides users with statistical information for multiple geographical levels.

From the MapReduce application developers' perspective, they only need to provide a pair of *map* and *reduce* functions; however, to enable applying reduce operations repetitively, the reduce function must be associative. Our model assume that data aggregation is performed periodically with a fixed time interval, *i.e.*, micro-batch processing.

A. Logical Topology

We model the logical topology of a hierarchical MapReduce job as a tree of nodes, as shown in Fig. 1. The tree consists of L levels, where level L corresponds to the root node and level 1 corresponds to leaf nodes that receive raw input data from level 0 sensors. We consider four levels of nodes in this work: city-level (level 1), county-level (level 2), state-level (level 3), and region-level (level 4). Each node represents a cluster of machines, in a to-be-defined physical location, that processes data from lower level nodes and transmits results to a parent node, which corresponds to a wider geographical area. Level 1 nodes apply a map operation to the input data, followed by a reduce operation. All the upper level nodes only use reduce operations to aggregate data. We assume the logical topology is a perfect tree in this work for simplicity; however, the models we present in Sec. III can be applied to arbitrary trees (*e.g.*, level 0 sensors can be connected to level 3 nodes directly). Since each node only sends data to its parent, there is no need to shuffle data across multiple nodes. Instead, shuffle operations are performed within each node.

There are query clients that send queries to different levels of nodes to request aggregated data. We assume aggregated data for the node's level are stored in a storage system at that logical node and are available for requests from clients.

B. Mapping to Virtual Machine Clusters

We denote the i-th logical node at level ℓ by x_i^ℓ and the set of nodes at level ℓ by $X^\ell = \{x_i^k |\ k = \ell\}$ for $\ell = 1, ..., L$. There is a set of N datacenters $DC = \{dc_1, dc_2, ..., dc_N\}$, and a cluster of virtual machines (VMs) may be allocated in each datacenter. We denote a cluster in dc_j by y_j and a number of allocated VMs for the cluster y_j by $m(y_j)$. We define a mapping from a logical node x_i^ℓ to a cluster y_j with $y_j = f(x_i^\ell)$. A cluster is managed by a resource management system such as YARN [16] and the VMs associated with the cluster can be shared by multiple logical nodes. Note that we assume a cluster only processes one level of logical node data at a time, and thus it cannot start the next micro-batch until it has finished with all of the data from the current micro-batch.

The mappings we study shown in Fig. 2. Mapping A uses datacenters at all levels and assigns each logical node to its corresponding area's datacenter. Thus, f is a one-to-one mapping. Since this mapping places city-level logical nodes at edge city datacenters, we can interpret it as an *edge computing* deployment. Mapping B does not use city and county datacenters, but uses state and region datacenters only. Logical city and county nodes are mapped to their state's datacenters. Mapping C only uses the root region datacenter, which means f is an all-to-one mapping. We can interpret this mapping as a *cloud computing* deployment.

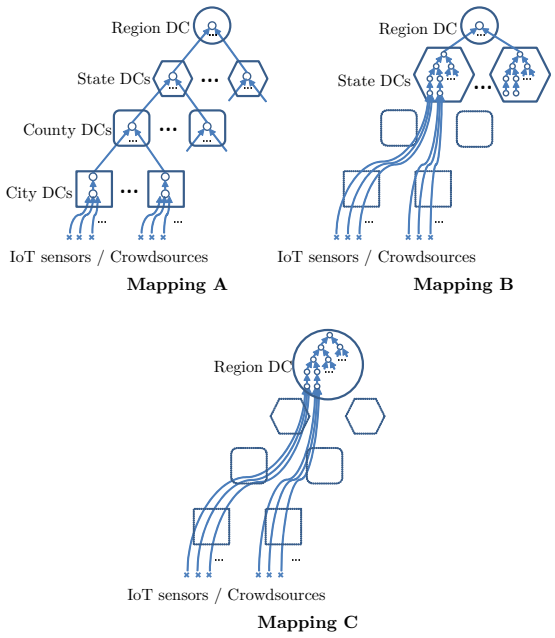


Fig. 2. Mappings between logical topologies and datacenters. Mapping A: all level DCs are used. Mapping B: city and county DCs are not used. Mapping C: only the region DC is used.

III. Performance Models

In this section, we present our performance models for end-to-end data aggregation and query response.

A. Data Aggregation Time Model

Our processing time models for map and reduce operations are defined as follows, respectively:

$$t_{\mathrm{m}}(m, d; \beta) = \frac{d}{\beta \cdot m}, \tag{1}$$

$$t_{\mathrm{r}}(m, d; \gamma) = \frac{d}{\gamma \cdot m}, \tag{2}$$

where m is the number of VMs. For both models, d is the amount of data to process in Mbytes. This linear scaling, in the number of VMs, of both map and reduce operations matches the observations reported in [12]. In [12], the throughput for a massively parallel, compute-intensive Hadoop application ranges from 0.1 to 1.2 Mbytes/sec/VM. As a middle ground between these throughput values, we use $\beta = 0.5$ Mbytes/sec/VM for map. Assuming the required computational resources for reduce is much less than that of map, as observed in [12], we use a higher throughput value of $\gamma = 1.0$ Mbytes/sec/VM for reduce. In Sec. IV-B, we use these models with up to $m = 112$ VMs, which is comparable to the number of VMs studied in [12]. We note these models may not accurately capture performance with significantly larger numbers of VMs because the throughput of map and reduce operations cannot scale up infinitely.

As a communication time model, we use the following:

$$t_{\mathrm{comm}}(y_1, y_2, d; \sigma_0, \sigma_1) = \sigma_0 + \sigma_1 \cdot \delta(y_1, y_2) + \frac{d}{\omega(y_1, y_2)}, \tag{3}$$

where d is the amount of data to transfer in Mbytes, $\delta(y_1, y_2)$ is the distance between two clusters y_1 and y_2 in miles, and $\omega(y_1, y_2)$ is the network bandwidth between the two clusters in Mbytes/sec. In this model, we extend the Hockney and Berry's communication time model [13] to include a linearly increase latency term (*i.e.*, $\sigma_1 \cdot \delta(y_1, y_2)$) as observed in [14]. We use $\sigma_0 = 4.852 \cdot 10^{-3}$ sec and $\sigma_1 = 0.022 \cdot 10^{-3}$ sec/mile as reported in [14]: Depending on the type of communication between source and destination datacenters, we use three different bandwidths: ω_{mobile} for mobile, ω_{LAN} for LAN, and ω_{WAN} for WAN communications. For mobile and WAN communications, we use the bandwidth values reported from Speedtest for Q1-Q2 2017 [15]: $\omega_{\mathrm{mobile}} = 1.064$ Mbytes/sec and $\omega_{\mathrm{WAN}} = 2.849$ Mbytes/sec. For LAN communication, we measured actual bandwidth using iperf on Amazon EC2 with two m5.large instances and obtained $\omega_{\mathrm{LAN}} = 1288.8$ Mbytes/sec.

Next, we describe how we estimate the time to aggregate and transfer data for each logical node as shown in Fig. 3. A logical node x_i^ℓ at level ℓ receives some amount of data $d(x_i^\ell)$ from lower level nodes or sensors. When $\ell = 1$, it receives input data from a set of IoT sensors at level 0. When $\ell > 1$, it receives input data from its children nodes in level $\ell - 1$, $\{x_k^{\ell-1} \mid k \in C(x_i^\ell)\}$, where $C(x_i^\ell)$ is the set of children of x_i^ℓ.

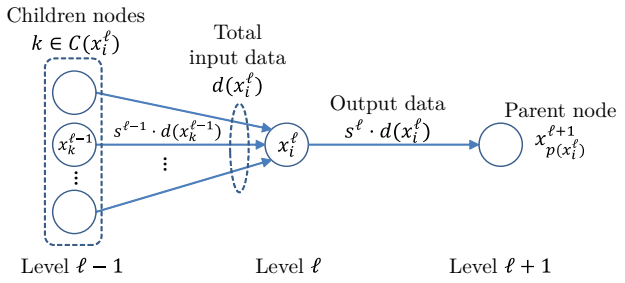


Fig. 3. Data processing at logical node x_i^ℓ and data transfer to its parent node for $\ell > 1$.

Once the logical node x_i^ℓ receives the input data, it aggregates the data and transfers the aggregated data to its logical parent node. The amount of aggregated output data is $s^\ell \cdot d(x_i^\ell)$, where $s^\ell > 0$ is the data selectivity as the result of data aggregation. The amount of total data, in Mbytes, received by x_i^ℓ is defined as follows:

$$d(x_i^\ell) = \begin{cases} \mathcal{I}_i & \ell = 1, \\ \sum_{k \in C(x_i^\ell)} s^{\ell-1} \cdot d(x_k^{\ell-1}) & \text{otherwise}, \end{cases} \tag{4}$$

where $\mathcal{I}_i$ is the amount of input data received from level 0 sensors, in Mbytes. As shown in Fig. 1, data aggregation is a combination of map and reduce operations for level 1 nodes and a reduce operation for upper-level nodes. Let $s_{\mathrm{m}} > 0$ and $s_{\mathrm{r}} > 0$ represent the data selectivities for map and reduce operations, respectively; then, $s^\ell = s_{\mathrm{m}} \cdot s_{\mathrm{r}}$ for $\ell = 1$ and $s^\ell = s_{\mathrm{r}}$ otherwise.

We consider a logical node x_i^ℓ as a single map or a reduce job (a map & reduce job for level 1) deployed on a mapped cluster $y_j = f(x_i^\ell)$. When multiple map/reduce jobs run on the same cluster y_j, we assume a scheduler (*e.g.*, YARN [16]) assigns a fraction of the computing power of the cluster's VMs to each logical node; that fraction is proportional to the logical node's input data size. Let $Z(y, \ell)$ represent the set of level-ℓ logical nodes mapped to cluster y (*i.e.*, $Z(y, \ell) = \{x_i^\ell \mid y = f(x_i^\ell)\}$). We define the total amount of data processed by the map function, denoted, $D_{\mathrm{m}}(y)$ and the total amount of data processed by the reduce function, denoted $D_{\mathrm{r}}(y, \ell)$, as follows:

$$D_{\mathrm{m}}(y) = \sum_{x_i^1 \in Z(y,1)} d(x_i^1), \tag{5}$$

$$D_{\mathrm{r}}(y, \ell) = \begin{cases} s_{\mathrm{m}} \cdot D_{\mathrm{m}}(y) & \ell = 1, \\ \sum_{x_i^\ell \in Z(y,\ell)} d(x_i^\ell) & \text{otherwise}. \end{cases} \tag{6}$$

For $x_i^\ell \in X^\ell, \ell = 1, ..., L$, we define the time to aggregate data from its children nodes or sensors and to transfer the aggregated data to its parent node, denoted $x_{p(x_i^\ell)}^{\ell+1}$, as:

$$\begin{aligned} t_{\mathrm{aggr}}(x_i^1) = {} & t_{\mathrm{m}}\left(m(f(x_i^1)) \cdot \frac{d(x_i^1)}{D_{\mathrm{m}}(f(x_i^1))},\ d(x_i^1)\right) \\ & + t_{\mathrm{r}}\left(m(f(x_i^1)) \cdot \frac{s_{\mathrm{m}} \cdot d(x_i^1)}{D_{\mathrm{r}}(f(x_i^1), 1)},\ s_{\mathrm{m}} \cdot d(x_i^1)\right) \\ & + t_{\mathrm{comm}}\left(f(x_i^1),\ f\left(x_{p(x_i^1)}^2\right),\ s^1 \cdot d(x_i^1)\right) \end{aligned} \tag{7}$$

$$t_{\text{aggr}}(x_i^\ell) = t_{\text{r}}\left(m(f(x_i^\ell)) \cdot \frac{d(x_i^\ell)}{D_{\text{r}}(f(x_i^\ell),\ell)},\ d(x_i^\ell)\right) + t_{\text{comm}}\left(f(x_i^\ell),\ f\left(x_{p(x_i^\ell)}^{\ell+1}\right),\ s^\ell \cdot d(x_i^\ell)\right), 1 < \ell < L, \quad (8)$$

$$t_{\text{aggr}}(x_i^L) = t_{\text{r}}\left(m(f(x_i^L)) \cdot \frac{d(x_i^L)}{D_{\text{r}}(f(x_i^L),\ell)},\ d(x_i^L)\right). \quad (9)$$

Finally, assuming the data aggregating operations are synchronized (*i.e.*, processing at logical level $\ell + 1$ starts when aggregated data from all of level ℓ nodes are transferred), we define the end-to-end data aggregation time as follows:

$$T_{\text{A}} = \sum_{\ell=1}^{L} \max_{x_i^\ell \in X^\ell} t_{\text{aggr}}(x_i^\ell). \quad (10)$$

We take a synchronized approach in (10), and thus the slowest node at each level dictates the final data aggregation time. Asynchronous processing may lead to faster data aggregation, in which case our model provides an upper bound on aggregation time.

B. Query Response Time Model

For modern data stores, it is reported that the maximum read throughput linearly scales up as the number of machines increases [17]. It is also observed when input read workload is less than the maximum read throughput, the read latency decreases linearly as the workload decreases [17]. We capture these characteristics in the following query time model:

$$t_{\text{query}}(m, q; \theta, \lambda) = \frac{q}{\theta \cdot m} \cdot \lambda, \quad (11)$$

where m is the number of VMs, q is the read requests per second, θ is the maximum read throughput per VM in requests/sec/VM, and λ is the read latency at maximum read workload in sec. The observed value of θ ranges from 1,750 to 14,000 requests/sec/VM, depending on the type of the storage system [17]; we use $\theta = 5,000$ requests/sec/VM.

Next, we model the average query response time for K query requests. Each query $k = 1, ..., K$ is associated with a source client c_k and a logical destination node x_k. For a cluster $y = f(x_k)$, we denote the query requests per second received by the cluster y by $q(y)$. Let the data sizes for a query and its response be d_{query} (= 128 bytes) and d_{resp} (= 2 Kbytes), respectively, the response time for the k-th query is:

$$\begin{aligned} t_{\text{resp}}(c_k, x_k) = {} & t_{\text{comm}}\left(c_k, f(x_k), d_{\text{query}}\right) \\ & + t_{\text{query}}\left(m(f(x_k)), q(f(x_k))\right) \\ & + t_{\text{comm}}\left(f(x_k), c_k, d_{\text{resp}}\right), \end{aligned} \quad (12)$$

where t_{comm} is as defined in (3). The average query response time over all K queries is:

$$T_{\text{Q}} = \frac{1}{K} \sum_{k=1}^{K} t_{\text{resp}}(c_k, x_k). \quad (13)$$

IV. Performance Study

We now use our models to study the application performance for the three mappings shown in Fig. 2. Our study uses real-world geographical and population data obtained from the U.S. Census [10], [11] to determine the IoT data generation rate and client and sensor locations.

A. Evaluation Setup

We consider data generated from IoT sensors or crowdsources spread across the nine Northeastern U.S. states: Connecticut, Maine, Massachusetts, New Hampshire, Rhode Island, Vermont, New Jersey, New York, and Pennsylvania. To process the logical hierarchical topology in Fig. 1, we assume there is a datacenter in every city (including towns and villages), county, and state in the Northeastern U.S., based on the locations defined in the U.S. Census [11]. We also have the regional datacenter located in Northern Virginia, as AWS hosts its U.S. East region datacenter there. The mobile network is used for level 0 sensors to transfer the input data to level 1 nodes. LAN is used for communications between logical nodes within the same datacenter while WAN is used for communications between logical nodes across different datacenters. Since the value of λ differs by orders of magnitude for different storage systems [17], we test $\lambda = \{0.1, 1, 10, 100\}$ ms. in Sec. IV-C.

For each cluster y_j, we use the following resource allocation policies. These policies apply only to the clusters with associated logical nodes (*e.g.*, no VMs are allocated for city datacenters in mapping C).

a) Population: We determine the number of VMs proportional to the number of people:

$$m(y_j) = \left\lceil \frac{population(y_j)}{5 \times 10^5} \right\rceil, \quad (14)$$

where $population(y_j)$ is the population of the local government to which the cluster y_j belongs. Table I shows the number of datacenters and the calculated number of VMs.

TABLE I
DATACENTERS AND VMS ALLOCATED BY THE POPULATION POLICY.

	City	County	State	Region
Number of DCs	8384	217	9	1
Number of VMs				
Average	1.01	1.07	12.11	
Minimum	1	1	1	
Maximum	5	5	39	112
Total	8396	234	109	

b) Fixed: We assign the fixed number of VMs: 1, 4, 16, 64 VMs for city, county, state, and region clusters, respectively.

B. Data Aggregation Performance

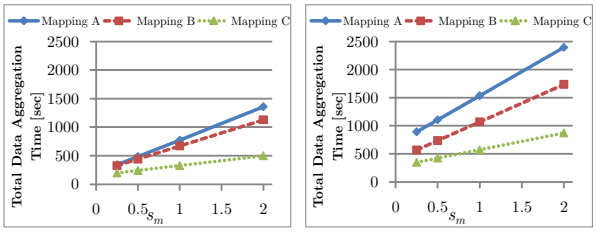
1) Parameters: For the map selectivity, we use $s_{\text{m}} = \{0.25, 0.5, 1.0, 2.0\}$ to test different data amounts while the reduce selectivity is fixed as $s_{\text{r}} = 0.6$. We assume a fraction of IoT sensors or users in each city produces some amount of

data over a certain time window. For each level 1 city node $x_i^1 \in X^1$, we specify the amount of input data size as follows:

$$d(x_i^1) = population(x_i^1) \times sensors\_per\_person \times bytes\_per\_sensor\_per\_time\_window, \quad (15)$$

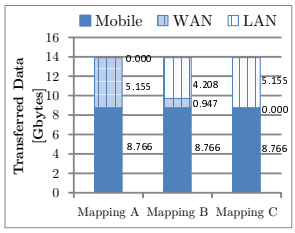
where $population(x_i^1)$ represents the number of people in the city for x_i^1, $sensors\_per\_person = 1$, and $bytes\_per\_sensor\_per\_time\_window = 128$ bytes.

2) Results: The data aggregation results for mappings A, B, and C, for $s_{\mathrm{m}} = \{0.25, 0.5, 1.0, 2.0\}$ are shown in Fig. 4. Figs. 4(a) and 4(b) give the total data aggregation time for the



(a) Total data aggregation time for *population* policy.

(b) Total data aggregation time for *fixed* policy.



(c) Transferred data between clusters.

Fig. 4. A comparison of data aggregation results between three mapping strategies for $s_m = \{0.25, 0.5, 1.0, 2.0\}$.

population and fixed policies, respectively. Since the amount of transferred data between clusters is not affected by the resource allocation policies, it is the same for both policies as shown in Fig. 4(c).

On average, the population policy is 84% faster than the fixed policy. This is due to the lower number of VMs allocated for the region datacenter by the fixed policy (fixed: 64 VMs vs. population: 112 VMs). Another reason is load imbalance between logical nodes caused by the fixed VM allocation policy. The population policy achieves more balanced load since it allocates VMs in proportion to population, and thus in proportion to the amount of data that is processed.

For both policies, the total data aggregation time increases linearly as the value of s_{m} increases. The reason is that $s^1 = s_{\mathrm{m}} \cdot s_{\mathrm{r}}$ dictates the amount of data coming out of level 1 nodes, and it linearly affects the aggregation and communication time, as we can see in (1), (2), and (3).

The relationship between the total data aggregation time for the different mappings does not change, as observed in Figs. 4(a) and 4(b). Mapping C is always the fastest, followed by mappings B and A. On average, mapping C is 53% and 62% faster than mapping A for the population and fixed policies, respectively. This is caused by asymmetric communication performances between the three network types and the number of VMs available for datacenters. In Fig. 4(c), all the mappings have the same amount of mobile communication; however mapping A has no LAN, but has WAN communication, whereas mapping C has no WAN, but has LAN communication. Mapping B has about 1 Gbytes of LAN and 4.2 Gbytes of WAN communication. Since LAN is about 450 times faster than WAN ($\omega_{\mathrm{LAN}} = 1288.8$ Mbytes/sec vs. $\omega_{\mathrm{WAN}} = 2.849$ Mbytes/sec), the ratio of different types of communication affects the data transfer time significantly. Typically, mobile and WAN communications are charged while LAN communication is free of charge. In terms of monetary cost for data transfer, mapping A is the most expensive since it only contains mobile and WAN communications while other mappings contain LAN communication.

C. Query Response Performance

1) Parameters: We determine the number of adult smartphone owners based on the U.S. Census, which reports that 77.4% of the population are adults, and of those, 70% own smartphones [11]. We assume these people are the query issuers for our location-based information service. For each city in the Northeastern U.S, n_{query} queries are sent per second, where

$$n_{\mathrm{query}}(city) = 0.774 \times 0.7 \cdot population(city) \times queries\_per\_smartphone\_per\_hour/3600.$$

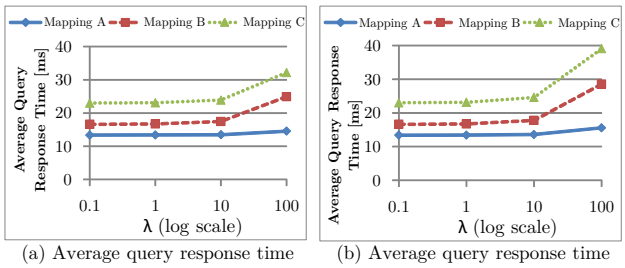
We use 1.0 for $queries\_per\_smartphone\_per\_hour$.

We model each city as a perfect circle; we estimate the radius of the city from its land area and randomly select the source location for each query within the circle. To determine the logical destination nodes for these queries, we randomly choose them from the following *local-heavy* probability distribution:

- *City-level*: Own city=0.56, a city in own county=0.14
- *County-level*: Own county=0.12, a county in own state=0.03
- *State-level*: Own state=0.08, a state in own region=0.02
- *Region-level*: Own region=0.05

2) Results: The average query response time for mappings A, B, and C with $\lambda = \{0.1, 1, 10, 100\}$ ms are shown in Fig. 5. Figs. 5(a) and 5(b) are the results for the population and the fixed policies, respectively. For both policies, the relative performance of the mappings is the reverse of the data aggregation time results. Mapping A is consistently the fastest among the three mappings, regardless of the value of λ. On average, mapping A is 46% and 47% faster than mapping C for the population and fixed policies, respectively. This is due to the proximity between edge data centers and client locations, as well as the local-heavy distribution of the query destinations. The value of λ does not affect the average query response time until $\lambda = 10$, and the response time starts to diverge when $\lambda = 100$. The reason is that the query time is so small compared to the communication time until $\lambda = 10$;

however, when $\lambda = 100$, the query time becomes significant and starts affecting the total response time. For the same reason, the effect of the difference in the numbers of VMs is small between the population and fixed policies: on average, the population policy is only 4% better than the fixed policy.

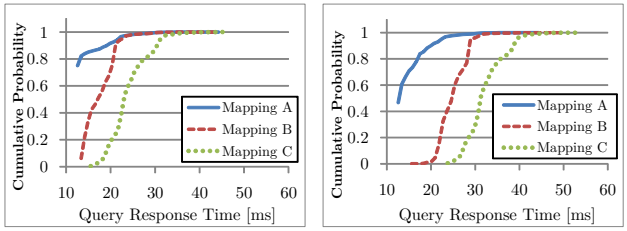


(a) Average query response time for *population* policy.

(b) Average query response time for *fixed* policy.

Fig. 5. A comparison of average query response time results for mappings A, B, and C with $\lambda = \{0.1, 1, 10, 100\}$ ms.

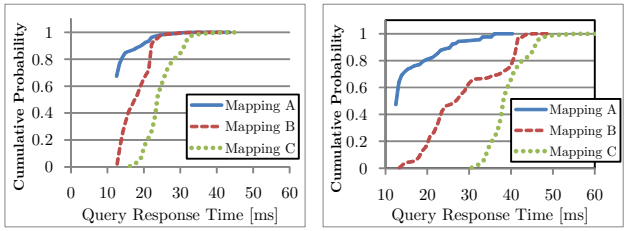
Fig. 6 shows the cumulative probabilities of query response times for mappings A, B, and C with $\lambda = 10$ ms and $\lambda = 100$ ms. Figs. 6(a) and 6(b) compare the cumulative probabilities between the population and fixed policies. For both policies, the effect of large λ can be confirmed: the curves in Fig. 6(a-1) are shifted to the right by about 10 ms. in Fig. 6(a-2).



(a-1) $\lambda = 10$.

(a-2) $\lambda = 100$.

(a) Cumulative probabilities for *population* policy.



(b-1) $\lambda = 10$.

(b-2) $\lambda = 100$.

(b) Cumulative probabilities for *fixed* policy.

Fig. 6. A comparison of query response time results between three mappings A, B, and C for the fixed policy.

The results from Sections IV-B2 and IV-C2 show that there is a performance trade-off between data aggregation and query response. A cloud-based approach achieves faster data aggregation times, while query response time is minimized with an edge-based deployment.

V. Conclusion

We have presented a performance model for end-to-end hierarchical MapReduce-based data aggregation and query response for aggregated data, executed in highly geo-distributed multi-tier datacenters. We studied application performance for three different mappings created from real-world geography and population data from the U.S. Census. These evaluations demonstrated that there is a trade-off between end-to-end data aggregation time and query response time. For data aggregation, a mapping with cloud resources only is on average 53% faster than a mapping with edge resources due to the cloud's fast network and plentiful compute resources; however, for query response, the edge hierarchy is 46% faster due to resource proximity to query clients. This work is a first step towards optimizing the deployment of hierarchical MapReduce jobs over fog computing environments.

References

[1] Gartner, Inc., "Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016," 2017. [Online]. Available: https://www.gartner.com/newsroom/id/3598917

[2] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for Internet of Things and analytics," in *Big data and Internet of Things: A roadmap for smart environments*. Springer, 2014, pp. 169–186.

[3] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[4] The Dark Sky Company, "Dark Sky," 2018. [Online]. Available: https://darksky.net/

[5] Waze Mobile, "Waze: Free Community-based GPS, Maps and Traffic Navigation App," 2018. [Online]. Available: https://www.waze.com

[6] K. Waddell, "How Phones Can Help Predict Thunderstorms," 2018. [Online]. Available: https://www.theatlantic.com/technology/archive/2016/08/how-phones-can-help-predict-thunderstorms/495389/

[7] Y. Luo and B. Plale, "Hierarchical MapReduce programming model and scheduling algorithms," *Proc. IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (CCGrid 2012)*, pp. 769–774, 2012.

[8] S. Dolev, P. Florissi, E. Gudes, S. Sharma, and I. Singer, "A survey on geographically distributed big-data processing using mapreduce," *arXiv preprint arXiv:1707.01869*, 2017.

[9] S. H. Mortazavi, M. Salehe, C. S. Gomes, C. Phillips, and E. de Lara, "Cloudpath: a multi-tier cloud computing framework," in *Proc. ACM/IEEE Symp. on Edge Computing*, 2017, p. 20.

[10] Brandbert Ltd., "US cities list," 2018. [Online]. Available: http://www.uscitieslist.org/

[11] U.S. Department of Commerce, "United States Census," 2018. [Online]. Available: https://www.census.gov/

[12] A profile of Apache Hadoop MapReduce computing efficiency, "Cloudera," 2010. [Online]. Available: http://blog.cloudera.com/blog/2010/12/a-profile-of-hadoop-mapreduce-computing-efficiency-continued/

[13] R. Hockney and M. Berry, "Public international benchmarks for parallel computers: Parkbench committee: Report-1," *Scientific Computing*, vol. 3, no. 2, pp. 100–146, 1994.

[14] R. Goonatilake and R. A. Bachnak, "Modeling latency in a network distribution," *Netw. and Commun. Technologies*, vol. 1, no. 2, p. 1, 2012.

[15] Speedtest, "Speedtest market reports Q1-Q2 2017 - United States," 2017. [Online]. Available: http://www.speedtest.net/reports/united-states/2017/

[16] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache Hadoop Yarn: Yet another resource negotiator," in *Proc. ACM Symp. Cloud Computing*, 2013, pp. 5:1–5:16.

[17] T. Rabl, S. Gómez-Villamor, M. Sadoghi, V. Muntés-Mulero, H.-A. Jacobsen, and S. Mankovskii, "Solving big data challenges for enterprise application performance management," *Proc. VLDB Endowment*, vol. 5, no. 12, pp. 1724–1735, 2012.