Towards Cloud-based Decentralized Storage for Internet of Things Data

Nanjangud C. Narendra, Koundinya Koorapati, Vijayalaxmi Ujja

***Abstract*—The Internet of Things (IoT) phenomenon is creating a world of billions of connected devices generating enormous amounts of data. That data needs to be stored efficiently so that it can be retrieved easily on demand and acted upon. Current cloud-based solutions, focusing on centralized data collection and storage, would not be adequate for this task, given the sheer volume of data expected to be generated by IoT devices. Despite this, however, *very little* work has been reported on cloud-based storage tailored for IoT data. To that end, in this paper, we present a decentralized cloud-based storage solution specifically tailored for IoT data. The salient features of our solution are: usage of object storage (such as Ceph) for software-defined storage; and optimal distribution of data among distributed *mini-Clouds*, which are mini-data centers. We present approaches for the following: optimal mini-Cloud placement to minimize latency of data collection from IoT devices; and data migration among mini-Clouds with a view towards addressing storage capacity issues while minimizing access latency. Throughout our paper, we illustrate our ideas via a realistic running example in the Smart Cities domain, and present experimental results via a proof of concept prototype.**

I. Introduction

The Internet of Things (IoT) phenomenon is expected to usher in as many as 25 billion devices by the year 2020[1]. As these devices (sensors, actuators, and gateways, routers & switches to manage the same) are not equipped with extensive storage, their data has to be stored outside them. Typical solutions so far have included storing all this data in centralized cloud data centers [1]. However, the expected data explosion is expected to overwhelm the capacity of even these data centers[2]. Additionally, transferring all this data to a centralized location would render data analysis difficult. Thus the *research problem* to be solved, is to determine a way to store IoT-generated data in geographically distributed mini-data centers for greater speed of access, closer to the point where the data was generated. A corollary to this problem, would be to optimally migrate data among the mini-data centers in case any of them runs out of storage space.

To address the aforementioned research problem, we propose a scalable software-defined object storage based decentralized storage solution for IoT data using mini-Clouds and Cloud. Mini-Clouds are decentralized and widely dispersed Internet infrastructure components whose compute cycles and storage resources can be leveraged by nearby connected devices and computers. Our solution will be using Ceph [2], an open source storage platform whose distributed object store is being exploited to store IoT data as objects which in itself is an innovative idea of using Ceph's object storage capabilities optimized for IoT. Additionally, storing IoT data as objects allows us to store structured as well as unstructured data, which is essential given the heterogeneity of IoT data.

Therefore, these are our *key contributions*:

- Decentralized software defined storage solution for IoT data incorporating the idea of mini-Clouds
- Innovative use of an object-based software-defined storage system for storing IoT data
- Approach for optimal placement of mini-Clouds in an IoT-based network for minimizing overall latency of data collection from IoT devices
- Once the mini-Clouds have been optimally placed, algorithms to migrate data among mini-Clouds to address storage capacity issues among mini-Clouds, with emphasis on minimizing access latency by users requesting access to the data

This paper is organized as follows. The following section presents our running example, which we will be using throughout this paper to illustrate our ideas. In Section III, we present background material used in the rest of our paper. Section IV presents our system architecture. Our mini-Cloud placement and optimal data storage algorithms are presented and illustrated in Section V. Experimental results via our proof of concept prototype are then presented and discussed in Section VI. Comparison of our approach to related work is presented in Section VII. Our paper concludes in Section VIII with suggestions for future work.

II. Running Example

Our running example is from the Smart Cities domain [3] (e.g., SmartSantander[3]) and it relates to smart traffic management on highways and surface roads in a city. We assume the city comprises a collection of suburbs with their own surface road network, connected together with one or more highways that act as "ring roads" encircling the suburbs. In such a system, smart traffic management would involve several scenarios: ensuring smooth traffic flow with minimal disruptions, especially at rush hour or during special events such

Nanjangud C. Narendra, Ericsson Research, Bangalore, India, ncnaren@gmail.com; Koundinya Koorapati & Vijayalaxmi Ujja, EMC Software and Services India Private Limited, Bangalore, India, {koundinya.x,vijayaujja}@gmail.com;

[1] http://www.gartner.com/newsroom/id/2905717

[2] From [1]: "The recent trend to centralize applications to reduce costs and increase security is incompatible with the IoT. Organizations will be forced to aggregate data in multiple distributed mini data centers where initial processing can occur. Relevant data will then be forwarded to a central site for additional processing."

[3] www.smartsantander.eu

978-1-4673-8566-4/16 $31.00 © 2016 IEEE
DOI 10.1109/CCEM.2015.9

as music concerts at particular venues; diverting/regulating traffic during emergency situations such as accidents, natural disasters, etc; and cutting off sections of roads/highways for repairs/maintenance, and regulating traffic in other parts of the Smart City. Monitoring and managing such a system requires collecting large amounts of data from several sensor networks: traffic sensors, road sensors and weather sensors. Depending on the frequency at which this data is generated, decision makers could get inundated with huge amounts of data, most of which would not be relevant to meet their particular requirements at a particular point of time. For example, the traffic commissioner of police for the city might be interested in the overall traffic data for the entire city or even historical data. Likewise a traffic policeman for a particular area might be interested in the real time data on the traffic for his area only. These access patterns of a particular user requirement would dictate the way the storage is accessed which in our case is the Cloud or the mini-Cloud. This example, therefore, raises the following crucial questions: given the huge amount of data emitted by traffic related sensors, how can the data be stored in an efficient manner so as to not only use the storage on the mini-Clouds and Cloud efficiently, but also enable faster analytics on the stored data. One possibility is store the traffic data related to the city on the Cloud and the traffic data meant for the suburbs on the mini-Clouds.

The rest of this paper will therefore present our approach in coming up with a decentralized software-defined storage solution for IoT data using Cloud and mini-Clouds which addresses these crucial questions.

III. Background

In this section we present some background material that we will be using throughout the rest of this paper.

A. Software Defined Storage

Software Defined Storage (SDS) refers to storage infrastructure that is managed and automated by intelligent software as opposed to by the storage hardware itself. In this way, the pooled storage infrastructure resources in a software-defined storage (SDS) environment can be automatically and efficiently allocated to match the application needs of an enterprise. By separating the storage hardware from the software that manages the storage infrastructure, software-defined storage enables enterprises to purchase heterogeneous storage hardware without having to worry as much about issues such as interoperability, under or over utilization of specific storage resources, and manual oversight of storage resources. The software that enables a software-defined storage environment can provide functionality such as deduplication, replication, thin provisioning, snapshots and other backup and restore capabilities across a wide range of server hardware components. The key benefits of software-defined storage over traditional storage are increased flexibility, scalability, automated management and cost efficiency.

B. Object Storage

Object storage is a highly scalable, simple, cost effective, distributed storage for the cloud that manages data as objects. Object storage is vastly more scalable than traditional file system storage because it's vastly simpler. Instead of organizing files in a directory hierarchy, object storage systems store files in a flat organization of containers (called **buckets** in Amazon S3) and use unique IDs (called **keys** in S3) to retrieve them. The upshot is that object storage systems require less metadata than file systems to store and access files, and they reduce the overhead of managing file metadata by storing the metadata with the object. This means object storage can be scaled out almost endlessly by adding nodes. Reliability is achieved on ordinary hardware and disk drives by replicating objects across multiple servers and locations.The HTTP interface to object storage systems allows for fast, easy access to files for users from anywhere in the world.

C. Ceph

Ceph [2] is a free software-defined storage (SDS) solution which is completely distributed with an unified approach of providing all data access methods (file, object, block). In the true spirit of SDS solutions, Ceph can work with commodity hardware and is not dependent on any vendor-specific hardware. A Ceph storage cluster is intelligent enough to utilize storage and compute the powers of any given hardware, and provides access to virtualized storage resources through the use of ceph-clients or other standard protocol and interfaces. Ceph storage clusters are based on Reliable Automatic Distributed Object Store (RADOS), which uses the CRUSH algorithm (Controlled Replication Under Scalable Hashing) [4] to stripe, distribute and replicate data. Ceph's main goals are to be completely distributed without a single point of failure, scalable to the exabyte level, and freely-available. We have chosen Ceph since it provides sophisticated functionality for storage placement & redistribution which we will leverage in our data migration algorithms. Such features are typically not native to generic key-value based storage solutions. Key-Value Store storage systems are very simple systems, similar to Hash Tables, that store any type of data indexed by a key. Due to its low complexity they are generally very highly performant, but not very flexible. In such system's all data accesses must be made using the given key and there is no way to query data items using any of the contents of the data (unless they are the key). Object Storage from Ceph offers the additional advantage of a simpler storage model with flat organization that enables scale out, a highly essential feature in cloud environments.

IV. System Architecture

Our proposed system architecture is shown in Fig. 1 (the networking layer shown in Fig. 1 is not part of our architecture, but is shown for the sake of completeness). In our mini-Cloud based architecture, the data from the IoT devices/sensors is stored on the mini-Clouds which form a first level tier with the data being distributed among the mini-Clouds. The

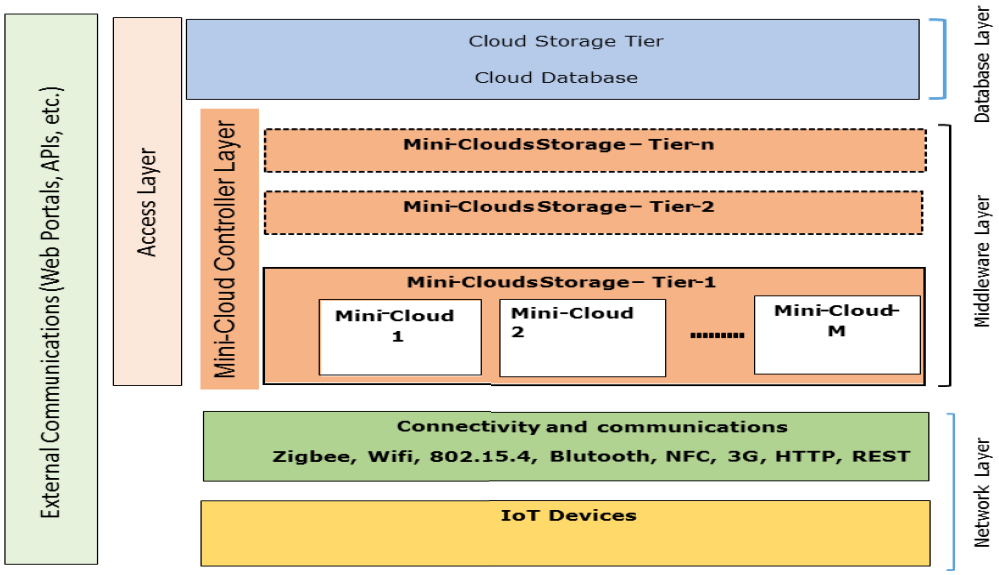

Fig. 1. System Architecture

computers/nodes that form the cluster of mini-Clouds, each consist of internal storage such as disk drives, flash drives, or PCIe flash cards. Although this internal storage in each node is not large, the storage on each node can be aggregated using a software layer abstraction to produce object storage. Thus a software-defined distributed object storage with a common name space can be realized where the storage is shared among all the nodes of the mini-Cloud. Each of the mini-Cloud node serves like an application server to the IoT devices/sensors. Each of the mini-Cloud nodes share access to all the data.

As shown in Fig. 1, each mini-Cloud node would correspond to a ***Ceph Node*** and the collection of these Ceph Nodes, i.e mini-Clouds, forms the Ceph Storage Cluster. On each mini-Cloud node, there would be a listener process (in the mini-Cloud Controller Layer) which would receive the data from one or more sensors or queries originating from users. The listener process would then hand off the data or the query to the process which interfaces with the distributed object store. Ceph provides S3-compatible API and Swift-compatible APIs to convert the data from the sensors into object form. Likewise, responses to the queries are also sent to the application through the listener process.

V. Data Storage Optimization

Given our system architecture as in Fig. 1, two key research issues need to be investigated before such an architecture becomes operational. First, the mini-Clouds need to be optimally placed throughout the IoT sensor network for the purpose of minimizing overall latency and cost,subject to a limitation on the number of mini-Clouds due to cost considerations. Second, since the amount of sensor data transmitted has the potential to overwhelm the (limited) storage capacity of a mini-Cloud, data would sometimes have to be migrated to mini-Clouds that possess the spare capacity, or to the Cloud itself. This migration will also have to be implemented with a view towards minimizing overall latency of access by users; this latency is measured in terms of the number of mini-Clouds over which the data requested by the user is distributed, with a lower number denoting lower latency and thereby faster access.

A. Optimal mini-Cloud Placement

We model the sensor network as a graph $G = (V, E)$ [5]. We say that any two nodes v_i and v_j are *directly connected* to each other (i.e., via a single hop [6]) if there is a direct data transfer between them without any intermediate node. Each such node in the sensor network can be a sensor, actuator, gateway, router, etc. The direct connections between any pair of nodes is modeled via an edge $e_i \in E$. The edge weight w_{ij} for each edge would model the latency of data transfer between the two nodes that make up the edge. (Latency in sensor networks can be measured via techniques such as those described in [7]; the exact method of latency measurement is beyond the scope of this paper.)

Mini-Clouds are placed at specific points in the sensor network. Each mini-Cloud would be configured to interact with a specified set of nodes, resulting in a partition of the sensor network such that each node would transmit data to only one mini-Cloud. We assume that the total number of mini-Clouds that can be installed in the network is limited by a number N due to cost considerations. Also let l_i be the latency of data transmission from a node v_i to the mini-Cloud

to which it is connected. Hence our optimization problem is to minimize the maximum value of this latency, and is expressed as follows:

$$\begin{aligned} \min \max \quad & l_i \\ \text{s.t.} \quad & x_i \in \{0,1\} && \forall i \in V \\ & \sum_{i \in V} x_i \leq \mathcal{N} \\ & \sum_{i \in V} y_{ik} * x_i = 1 && \forall i \in V \end{aligned}$$

Briefly, there are N mini-Clouds and V nodes, on which the N mini-Clouds have to be placed so that the maximum latency from a node to a mini-Cloud is minimized. (Please note that in this formulation we are not considering the available bandwidths in the sensor network and the amount of data to be transferred; rather, in a manner similar to that described in [6], we are considering a more abstract formulation of latency, leaving more detailed treatment of latency vis-a-vis bandwidth to future work.)

The x_i refers to the placement of the mini-Clouds on selected nodes of the network. The inequality constraint depicts the limit on the number of mini-Clouds. The y_{ik} in the equality constraint shows that node v_i sends data to the mini-Cloud positioned at node v_k, hence y_{ik} is 1, otherwise it is zero, $y_{ik} \in E$. It is to be noted that our problem formulation does *not* make any assumptions about the storage capacity of the mini-Cloud.

It is clear that this optimization problem, derivable from the well-known 0-1 integer programming problem [8], is NP-Complete. Hence we provide an approximation algorithm.

B. Approximation Algorithm

For our approximation algorithm, we make a key assumption central to the way sensor networks are usually configured. Typically data from sensor nodes is transmitted through the network and is finally sent to a gateway via routers, with the gateways being directly to cloud data centers. Hence, regardless of the sensor network topology (modifying our algorithm to take into account sensor network topologies will be investigated as part of future work), we can assume the existence of a collection of gateways that need to be connected to our set of N mini-Clouds.

For every gateway G_i connected mini-Cloud C_j, we represent the latency as $l(G_i, C_j) = max(L_{G_i}) + l'_{ij}$, where L_{G_i} refers to the maximum latency of any sensor node connected to the gateway G_i, and l'_{ij} refers to the latency of directly connecting G_i to C_j. We can therefore represent these latency values via a matrix $M(G, C)$. Our problem then reduces to finding the appropriate mapping between gateways and mini-Clouds so that the maximum value of $l(G_i, C_j)$ is minimized.

To that end, we present our approximation algorithm as depicted in Algorithm 1. The algorithmic complexity is $O(nlogn)$ due to the initial sorting of the matrix $M(G, C)$; otherwise it is a linear approximation algorithm for our problem formulated in Section V-A.

TABLE I
LATENCY MATRIX (LATENCIES IN MILLISECONDS)

	C_1	C_2	C_3
G_1	20	200	∞
G_2	28	40	204.8
G_3	32.4	16.4	∞
G_4	16.04	24.4	∞
G_5	32.4	78	122
G_6	166.4	∞	20.4
G_7	220.8	140.4	17.6
G_8	120.8	92	69.2

Algorithm 1 Approximation Algorithm for Optimal mini-Cloud Placement

1: **Description:** Optimally connect mini-Clouds with Gateways based on Minimizing Max Latency
2: **Input:** The matrix $M(G, C)$
3: **Output:** Sorted list of (G_i, C_j) pairs
4:
5: Sort the entries in $M(G, C)$ in increasing order to create a sorted list M_s
6: List of Gateways = G
7: AllottedGateways G' = ϕ // Empty set
8: **for** For each entry G_i in the sorted list **do**
9: **if** $G_i \notin G'$ **then**
10: G' = $G' \cup G_i$
11: G = G - $\{G'\}$
12: **end if**
13: **if** G = ϕ **then**
14: $exit()$;
15: **end if**
16: **end for**

Referring to our Smart Cities running example, assume eight gateways $G1$ through $G8$, and assume that the user has the budget for up to 3 mini-Clouds. Each gateway is assumed to service a suburb of a city, and represent the interface to the sensor network installed in that suburb. Table I lists the latency matrix with maximum latencies listed in milliseconds. After running Algorithm 1, the mini-Cloud-gateway mapping is as follows: $(C_1 : G_1, G_5, G_4; C_2 : G_2, G_3; C_3 : G_6, G_7, G_8)$.

A short explanation of Table I is in order. In some entries the latency is listed as ∞, which signifies that certain gateways are not able to connect to a particular mini-Cloud; such a situation would arise in an urban scenario where such a connection would be difficult due to specific structural issues (in the case of a wired connection, or if the signal from the gateway (in case of wireless connection) is considered too feeble to connect to the particular gateway in question. Indeed, such a situation is to be expected in cities which could conceivably contain a heterogeneous mix of sensor networks of varying capacity and quality, installed in stages over a period of time.

One extension to the above approach would be to extend our (currently) single tier mini-Cloud deployment to multiple

tiers, as depicted in Fig. 1. This would require us to implement our approximation algorithm at each tier. The key difference, however, is that the latency matrix would now need to model latencies between mini-Clouds at tier i and those at tier $i+1$.

C. Optimal Data Migration among Mini-Clouds

Once the mini-Clouds have been optimally placed as described above, situations may arise when storage capacity of certain mini-Clouds may fill up. Since all mini-Clouds are not expected to be of the same storage capacity, some other mini-Clouds may contain spare capacity. Hence this would necessitate data migration among the mini-Clouds, while ensuring that this migration is still able to meet the overall demand from users of the data.

Prior work on optimal data migration was shown to be NP-Hard [9], and this necessitated the investigation of approximation algorithms, in particular, the algorithm described in [10]. However, the algorithm in [10] was primarily about moving data around purely in response to changing demand; we need to tweak this algorithm to ensure that data is migrated only from those mini-Clouds whose capacity is reaching its limit.

For any mini-Cloud d, let $L(d)$ denote the objects on that mini-Cloud. Corresponding to any placement $\{p_i\}$ (which denotes the set of mini-Clouds that each object is stored on), we define the corresponding flow graph $G_p(V, E)$, as follows. We add one node a_i to the graph for each item $i \in 1...m$. We add one node d_j for each mini-Cloud $j \in 1...N$. We add one source vertex s and one sink vertex t. We add edges (s, a_i) for each object i. Each of these edges has user demand $\texttt{demand}(i)$, hence we denote $\texttt{demand}(i)$ as the weight of the edge (s, a_i). We also add edges (d_j, t) for each mini-Cloud j. These edges have capacity L (where L is the load capacity of mini-Cloud j). For every mini-Cloud j and for every object i in mini-Cloud j, we add an edge (a_i, d_j) with capacity L, such that the size of the object on mini-Cloud d_i does *not* exceed L.

For every round of data migration, we pick every pair of mini-Clouds d_i and d_j, where d_i belongs to the mini-Cloud which needs to migrate data (sender mini-Cloud), and d_j belongs to the mini-Cloud that can receive the data (recipient mini-Cloud). We modify the placement $\{p_i\}$ to $\{p'_i\}$ by moving objects from d_i to d_j, and we calculate the max-flow in the flow graph for the placement $\{p'_i\}$. After doing this for all pairs, we pick the one with the greatest max-flow value. For the selected mini-Cloud d_j with capacity L, we decrement L by the size of the object that is migrated.

In the next round, we pick another pair of mini-Clouds such that the earlier mini-Cloud d_i is *not* picked, and repeat the above procedure. The algorithm stops once all objects in the mini-Clouds d_i are migrated. We call this algorithm the Basic algorithm.

To cater to latency-sensitive IoT environments, we propose the following improvements to the Basic algorithm:

A1 We can sort the set of sender mini-Clouds $\{d_i\}$ by decreasing order of demand, and select the mini-Clouds d_j for the d_i in the sorted order.

A2 In order to minimize overall latency of access, we can modify the above algorithm to ensure that only the minimal set of mini-Clouds d_j are selected. This would ensure that the migrated data would be "spread" across the minimal set of mini-Clouds. Such a minimal distribution would be crucial for storing unstructured data as objects; typically, such objects may need to be integrated from multiple mini-Clouds and integrated before presentation to the user.

To that end, our improvement here would be to sort the mini-Clouds d_j by decreasing order of available storage capacity before running the above algorithm.

A3 Improvements **A1** and **A2** can be implemented together first sorting mini-Clouds d_i by decreasing order of demand, along with sorting of mini-Clouds d_j by decreasing order of available storage capacity. (Due to space constraints, we will be reporting implementation results from this in a future paper.)

VI. Implementation and Experimentation

The proof of concept prototype is implemented as a simulation program in Python, which also simulates our Ceph cluster (due to lack of space, we will be reporting the results of implementation on the actual Ceph cluster in a separate paper). For our implementation, we have chosen to highlight our data migration approach, since we believe it is the more critical problem, given the need to optimally store increasing amounts of IoT data. For our simulation, we have made the following assumptions:

- Our example would be based on N=100 mini-Clouds and we will not be considering replicas for the object, although in Ceph at any given point in time there are three copies of an object stored in the cluster.
- The trigger for the migration is always when an object to be written on a mini-Cloud does not meet the space requirements. In Ceph, this happens when the ceph-osd daemon fails the write from the client when the Cluster is 85% full.
- When the migration algorithm is invoked to make space for a new object on a mini-Cloud, it will migrate all the existing objects on the mini-Cloud to a new mini-Cloud.
- For the sake of simplicity, we assume that all objects are of the same size.
- For our example, we assume the bandwidth or load handling capacity (L) per mini-Cloud is 500 units, each object is of 100 units in size making the mini-Cloud capable of holding not more than five objects which is the Storage Capacity (K). However, this would not in any way affect our algorithm.

The framework for our experiments is as follows.

1) For the N=100 mini-Cloud nodes considered, create an initial Layout wherein we have one of the five nodes having reached full capacity of storing five objects.
2) Create a Target Layout wherein we adopt one of the following approaches at a time.

a) This approach implements the basic one round migration algorithm. On the node that has reached full capacity, let us assume that we have a write request to store a new object. Now we need to migrate all the already stored five objects to other four nodes. This becomes the source node for migration. So, for each object on the source node to be migrated, we select a target node such that the target node has free capacity to accommodate the object. Each object on the source node that gets migrated to a new target node, has its object placement map updated as well. Likewise, the table tracking the used and free space available for each node is also updated accordingly.
b) This is a variant of the first approach wherein the mini-Clouds that require migration are sorted based on the decreasing order of demand. This would ensure that those mini-Clouds whose migration is most crucial from user demand perspective would be considered first for migration.
c) This approach would select only a minimal set of mini-Clouds as the target for migration aiming for a minimal distribution on a minimal set of nodes.
d) This method combines the two approaches presented above wherein the source mini-Clouds are sorted based on the decreasing order of demand and the target mini-Clouds are sorted based on the decreasing order of available storage capacity.

3) After a certain number of rounds required to migrate the initial layout to the target layout, we would be interested in knowing the number of mini-Clouds to which the extra storage items (objects) have been distributed.
4) Going further our experiments would vary K and L and capture data for the algorithms proposed and determines the number of mini-Clouds used to accommodate the extra storage items as a consequence of data migration.

A. Example

We shall first illustrate the basic algorithm for migration using an example with the parameters N, L and K as mentioned in the previous section; for ease of exposition, however, we will set N=5. Let us assume an initial layout as shown here for the mini-Clouds C1 through C5:

TABLE II
INITIAL LAYOUT

H	B	A	C	
	D		G	
	E		I	
	F		J	
			K	

The used and free space on each mini-Cloud node would be as listed below:

mini-Cloud	Used	Free
C1	100	400
C2	400	100
C3	100	400
C4	500	0
C5	0	500

Now let us assume that we need to write another new object **L** to the mini-Cloud C4, which has reached full capacity. This is when the migration algorithm is invoked to migrate all the objects on the mini-Cloud C4 to other mini-Clouds which have space. In the basic algorithm we do not have any particular order in which the mini-Clouds are chosen; hence let us assume that C1 is chosen first which has space for four objects. So the objects C, G, I, J are migrated to C1. Likewise if C2 is chosen as the next target for migration, the object K is migrated to C2. Thus after migrating all the objects of C4, the object L finds its place on C4, as listed below.

H	B	A	L	
C	D			
G	E			
I	F			
J	K			

The used and free space on each mini-Cloud node would be as listed below:

mini-Cloud	Used	Free
C1	500	0
C2	500	0
C3	100	400
C4	100	400
C5	0	500

What is interesting to observe in this example is the absence of smart and intelligent algorithms to minimize the number of mini-Clouds chosen to distribute the data. For instance, although mini-Cloud C5 was free and could accommodate all the migration data, it was not chosen and instead the extra storage required was provided by two mini-Clouds C1 and C2. This is our Basic algorithm.

We will first see another improvement in our basic algorithm which is A1. In this improved algorithm we sort the sender mini-Clouds by decreasing order of demand. In other words, the mini-Clouds are sorted based on their used capacity which makes them candidates for migration in the right priority. Considering the same initial layout as show in II, but with the mini-Clouds sorted based on their usage capacity, the storage usage of mini-Clouds is as listed below:

mini-Cloud	Used	Free
C4	500	0
C2	400	100
C1	100	400
C3	100	400
C5	0	500

Evidently, C4 has completely exhausted all the capacity and needs migration. This becomes the case of a proactive

migration wherein we could imagine some kind of a process which monitors the mini-Clouds periodically and initiates the migration for mini-Clouds whose migration is most crucial from user demand perspective. The destination mini-Cloud for C4 can be selected in any random order as in our Basic algorithm. Once C4 has been fully migrated, the used and free space of all the mini-Clouds are recomputed and the mini-Clouds are again sorted in the decreasing order of their usage capacity. Another mini-Cloud whose demand is crucial becomes the candidate for migration.

We will next see how the improvement A2 to the basic algorithm works. In this algorithm we can choose mini-Clouds in such a way that the capacity demand for the extra storage required for migration can be met more efficiently. In this case, the mini-Clouds will be sorted based on the decreasing order of available storage capacity. Consider the same initial layout as used in the previous example, as shown in Table II, but with the mini-Clouds sorted based on their free capacity.

mini-Cloud	Used	Free
C5	0	500
C1	100	400
C3	100	400
C2	400	100
C4	500	0

Now let us assume that we need to write our new object **L** to the mini-Cloud C4, which has reached full capacity, the extra storage can be provided by just C5 which has enough capacity to store the objects migrated from C4. The target layout for the mini-Clouds C1 through C5 would be as listed below:

H	B	A	L	C
	D			G
	E			I
	F			J
				K

It is interesting to note that there is minimal disruption to the initial layout and only a minimum number of mini-Clouds have been selected for migration.

Using this example as the basis, we next run the migration algorithms for larger data sets, in particular for a data set of N=100 mini-Clouds. Keeping the capacity K as constant, we vary the bandwidth L, starting from 500 to 5000 in steps of 500 and record the different values corresponding to the number of mini-Clouds required to meet the extra storage requirements for the migration demand. For a given input, the recorded value of number of mini-Clouds will be the average of all the mini-Clouds that have provided the capacity for the corresponding number of migrations performed in that run of the algorithm. One set of experiments are repeated with K=5 and varying L and the other set are repeated with K=10 and varying L. The corresponding graphs for these experiments are shown in Figures 2 and 3.

As can be seen from the experiments, the improvement A2 performs consistently better than the other approaches; it is able to distribute storage to the least number of mini-Cloud nodes, followed by A1 and the Basic algorithm.

It is to be noted, however, that the Basic algorithm is able to perform better than improvement A1. This is probably because the criteria for selecting the destination mini-Cloud is the same in both cases; main difference between Basic and A1 being that the trigger for data migration for the former is more reactive than proactive. We will investigate this further in future work.

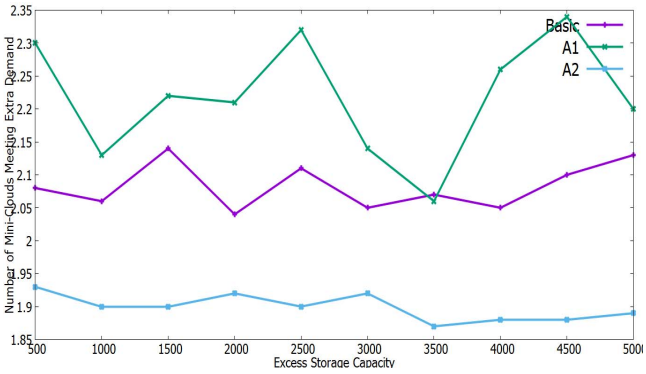

Fig. 2. Basic, A1 and A2 for K=5 and N=100. L is varied from 500 to 5000 in steps for 500.

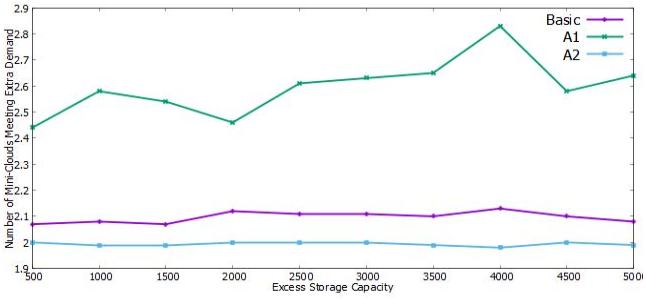

Fig. 3. Basic, A1 and A2 for K=10 and N=100. L is varied from 500 to 5000 in steps for 500.

VII. Related Work

Industrial Perspectives: The fact that IoT will cause a rethink in the way storage solutions are designed has recently been pointed out by several industry experts; in particular, the benefits of object storage for IoT have already been emphasized[4]. Similar sentiments have been expressed by the Storage Network Industry Association (SNIA), the premier standards body for storage [5].

[4]http://www.whatiswith.com/2015/05/guest-post-iot-object-storage-redefining-storage-s

html

[5]http://www.snia.org/sites/default/files2/DSI2014/presentations/StorTech/TomLeyden_The_Internet_of_Things.pdf

Cloud Data Storage: The need for object-based storage for IoT data has been introduced and explained in [11]. Also, in [12] the authors present a system that enables the specification of multi-tiered cloud storage instances. However, [12] mainly discusses cloud storage in a single cloud, and does not discuss distributed cloud storage. The citation [13] presents a survey of data placement and migration algorithms in disk-based storage systems. We will be investigating some of the ideas presented in [13] for our future work. The authors of [14] discuss efficient data migration in self-managing storage systems. Their method automatically detects hotspots and uses a bandwidth-to-space ratio metric to greedily reconfigure the system while minimizing the resulting data copying overhead. While our approach is agnostic of any data copying or replication, we consider the approach in [14] to be complementary to our data migration approach. The citation [15] presents a dynamic algorithm that dynamically allocates data objects to nodes, redistributing minimum amount of objects when new nodes are added or existing nodes are removed to maintain the balanced distribution.

In [16], on the other hand, the authors present distribution of storage among clouds from the security perspective. They present a method to divide data into pieces and distributing them among the available cloud providers so that no less than a threshold number of cloud providers can take part in successful retrieval of the whole data. We view this approach as being complementary to ours, and we will investigate it for future work.

More recent work has focused on data migration from desktops to remote sites[6]. That work has shown that it is feasible to use information about the tree structure of a file system to minimize the amount of data copied, thereby providing a more efficient data migration mechanism.

IoT Data Storage: Some recent work on storage has focused on storage for IoT, e.g., [17]. That approach proposes the use of a NoSQL database, along with an ontology-based data sharing mechanism that implements sharding for optimal data storage. We view this approach as being complementary to our work, and we will investigate integrating it in our future work. Gonzalez [18] proposed a model called RFID-Cuboids to store massive RFID data, based on data warehousing. Ding, et al. [19] proposed a database clustering framework for managing and querying sensor data. However, [19] proposes the use of standard relational database management systems, whereas we have argued in our paper for the usage of object storage for (multi-modal) IoT data.

VIII. Conclusions

The emergence of Internet of Things (IoT), with the expected huge increases in storage needed to store IoT data and retrieve it quickly on demand, has raised the issue of optimal storage of this data. To address this issue, our paper has presented a decentralized cloud-based solution for storage of IoT data, using Cloud and mini-Clouds. The key features of our paper are the proposal to use an object storage product such as Ceph; algorithms for optimal mini-Cloud placement in an IoT environment; as well as algorithms for optimal migration of data among the mini-Clouds to address storage capacity issues while minimizing access latency. We have also demonstrated the operation of our algorithms via a simulation program.

Future work would involve the following: (1) implement our simulation program on a real-life Ceph cluster (this is already underway); (2) test our approach on larger case studies; and (3) incorporate reconfiguration/re-placement of mini-Clouds along with data migration.

[6]http://drona.csa.iisc.ernet.in/~gopi/docs/kamala-MSc.pdf

Acknowledgments

The authors would like to thank Prof. K. Gopinath for his assistance. The authors would also like to thank the following for their comments: Prof. Umesh Bellur, Prof. D. Janaki Ram, Prof. N.D. Gangadhar, Prof. P.V.R. Murthy, V.S Yerragudi and Karthikeyan Ponnalagu. The authors also acknowledge the support of MRSUAS.

References

[1] G. Inc. (2014; accessed on 6th August, 2015) Gartner press release. [Online]. Available: http://www.gartner.com/newsroom/id/2684616

[2] I. S. Inc. (2015) Ceph storage. [Online]. Available: http://ceph.com/ceph-storage

[3] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *Internet of Things Journal, IEEE*, vol. 1, no. 1, pp. 22–32, 2014.

[4] S. Weil. (2006) Crush algorithm. [Online]. Available: http://ceph.com/papers/weil-crush-sc06.pdf

[5] C. Inzinger, B. Satzger, W. Hummer, and S. Dustdar, "Specification and deployment of distributed monitoring and adaptation infrastructures," in *2nd International Workshop on Performance Assessment and Auditing in Service Computing*, ser. PAASC'12, 2012.

[6] M. Alicherry and T. V. Lakshman, "Network aware resource allocation in distributed clouds," in *INFOCOM*, 2012, pp. 963–971.

[7] A. Ageev, D. Macii, and D. Petri, "Experimental characterization of communication latencies in wireless sensor networks," in *Proc. 16th Intl. Symp. Exploring New Frontiers of Instrumentation and Methods for Electrical and Electronic Measurements (IMEKO TC4)*, 2008, pp. 258–263.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.

[9] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu, "Approximation algorithms for data placement on parallel disks," in *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA.*, 2000, pp. 223–232. [Online]. Available: http://dl.acm.org/citation.cfm?id=338219.338255

[10] S. R. Kashyap, S. Khuller, Y. J. Wan, and L. Golubchik, "Fast reconfiguration of data placement in parallel disks," in *Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments, ALENEX 2006, Miami, Florida, USA, January 21, 2006*, 2006, pp. 95–107. [Online]. Available: http://dx.doi.org/10.1137/1.9781611972863.10

[11] Y. Peng, D. Xie, and A. Shemshadi, "A network storage framework for internet of things," *Procedia Computer Science*, vol. 19, pp. 1136–1141, 2013.

[12] A. Raghavan, A. Chandra, and J. B. Weissman, "Tiera: towards flexible multi-tiered cloud storage instances," in *Proceedings of the 15th International Middleware Conference*. ACM, 2014, pp. 1–12.

[13] B. Seo, "Survey on data placement and migration algorithms in distributed disk systems," in *Proceedings of 2004 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'04), Las Vegas, Nevada, USA*. Citeseer, 2004.

[14] V. Sundaram, T. Wood, and P. Shenoy, "Efficient data migration in self-managing storage systems," in *Autonomic Computing, 2006. ICAC'06. IEEE International Conference on*. IEEE, 2006, pp. 297–300.

[15] G. Zhang, L. Chiu, and L. Liu, "Adaptive data migration in multi-tiered storage based cloud environment," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE, 2010, pp. 148–155.

[16] Y. Singh, F. Kandah, and W. Zhang, "A secured cost-effective multi-cloud storage in cloud computing," in *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*. IEEE, 2011, pp. 619–624.

[17] T. LI, Y. LIU, Y. TIAN, S. SHEN, J. Boleng, and W. MAO., "A storage solution for massive iot data based on nosql," *GREENCOM '12 Proceedings of the 2012 IEEE International Conference on Green Computing and Communications*, 2012.

[18] H. Gonzalez, J. W. Han, L. X. Lei, and et al, "Warehousing andanalyzing massive rfid data sets," *Proceedings of the 22nd International Conference on Data Engineering*, 2006.

[19] Z. Ding, J. Xu, and Q. Yang, "Seaclouddm: a database cluster framework for managing and querying massive heterogeneous sensor sampling data," *The Journal of Supercomputing*, vol. 66, no. 3, pp. 1260–1284, 2013.

[20] N. Tran, M. K. Aguilera, and M. Balakrishnan, "Online migration for geo-distributed storage systems." in *USENIX Annual Technical Conference*, 2011.

[21] J. MacCormick, N. Murphy, V. Ramasubramanian, U. Wieder, J. Yang, and L. Zhou, "Kinesis: A new approach to replica placement in distributed storage systems," *ACM Transactions On Storage (TOS)*, vol. 4, no. 4, p. 11, 2009.

[22] L. Zhong, "Efficient, balanced data placement algorithm in scalable storage clusters," *J Commun Comput USA*, vol. 4, no. 7, 2007.

[23] M. Rouse. (2013) Software defined storage. [Online]. Available: http://searchsdn.techtarget.com/definition/software-defined-storage

[24] M. Satyanarayanan, P. Bahl, C. Ramón, and N. Davies., "The case for vm-based cloudlets in mobile computing," *IEEE*, vol. 8, pp. 14–23, 2009.

[25] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn., "Crush: Controlled, scalable, decentralized placement of replicated data," *IEEE*, 2006.

[26] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta, J. Boleng, and K. Ha., "The role of cloudlets in hostile environments," *IEEE*, 2013.

[27] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami., "Internet of things (iot): A vision, architectural elements, and future directions," *IEEE*, 2010.

[28] H. Wang, "Accelerating mobile-cloud computing using a cloudlet," Master's thesis, University of Rochester, May 2013.

[29] Z. M. Ding and X. Gao, "A database cluster system framework for managing massive sensor sampling data in the internet of things," *Chinese Journal of Computers*, vol. 35, pp. 1175–1191, 2012.