Uploading and Replicating Internet of Things (IoT) Data on Distributed Cloud Storage

Akshay Kumar
Centre for Development of Advanced Computing
Mumbai, India
akshay@cdac.in

Nanjangud C. Narendra
Ericsson Research
Bangalore, India
nanjangud.narendra@ericsson.com

Umesh Bellur
Indian Institute of Technology Bombay
Mumbai, India
umesh@cse.iitb.ac.in

***Abstract*—The Internet of Things (IoT) phenomenon is creating a world of billions of connected devices generating enormous amounts of data items. In particular, for purposes of high availability and disaster recovery, replication of this data on cloud storage needs to be implemented efficiently. To that end, in this paper, we investigate the combined problem of uploading IoT data from a set of sensor gateways and efficient replication of this data on distributed cloud storage. While other efforts do exist in this space, either they do not consider replication in context of small size and high number of data items, which is inherent nature of IoT data or they concentrate on the access time after replication. Our solution assumes the existence of multiple distributed cloud data centers, called *mini-Clouds*, among which data can be replicated. We model our problem comprehensively based on various parameters such as effective bandwidth of the IoT network, available number and size of data items at each mini-Cloud; and we present our problem as a collection of various sub-problems based on subsets of these parameters. We prove that the exact solution to the problem is intractable, and we present a number of heuristic strategies to solve it. Our results show that the performance of any heuristic is bounded by the read and write latency of mini-Clouds and the best we can do is often 12 times the worst we can do for a given number of data items to be uploaded and replicated from the gateways to the mini-Clouds in test setup.**

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 data explosion is expected to overwhelm the capacity of even these data centers.

Additionally, transferring all this data to a centralized location would render data analysis difficult. Thus distributed data storage, among multiple geographically distributed mini-data centers (which we call *mini-Clouds*), is called for, as depicted in Figure 1. Note that, in this system setup, a group of sensors upload their data to a single sensor gateway which is then responsible for pushing the data collected over a period of time to the mini-Clouds.

One of the key research problems in such a scenario is how to upload data from the sensor gateways and replicate the data in multiple mini-Clouds to cater to high availability and disaster recovery requirements. This calls for the data to be optimally replicated among the mini-Clouds in a minimal amount of time. The time taken is a function of many parameters such as network bandwidth, available data storage on each mini-Cloud, number of data items at each mini-Cloud and amount of data to be replicated. To that end, in this paper, we propose an approach to address this problem. As an optimal solution of this problem is intractable we propose a number of heuristics for solving this problem.

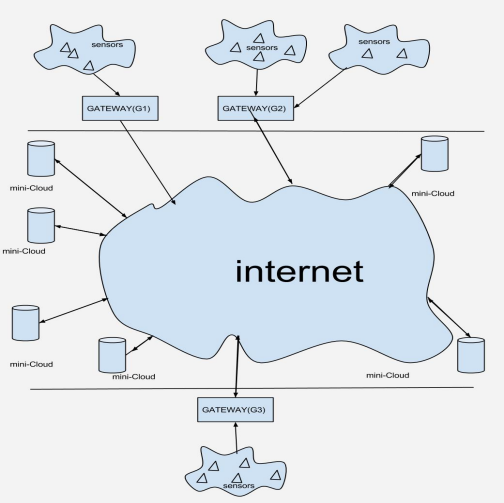


Fig. 1. Cloud-based IoT Network

We use simulation for evaluating different data configurations and the heuristics we propose. Our results show that scheduling the data items for transfer from the gateways to the mini-Clouds based on the highest bandwidth requirement first is optimal in most cases and can be up to 12 times as effective as other schedules based on different ordering heuristics.

The key contributions of this paper are:

- Formulating the problem of uploading data from a set of sensor gateways to multiple distributed mini-Clouds taking replication requirements for each data item into account. The formal problem is presented with the parameters of interest.
- Investigation of the performance of eleven possible or-

2159-6190/16 $31.00 © 2016 IEEE
DOI 10.1109/CLOUD.2016.92

derings of the data items used in the uploading and replication. These orderings are based on the parameters considered in the problem.
- An analysis on simulation based on the parameter space with results.

The rest of this paper is organized as follows. Section II analyzes existing work in this space. We then present a formal description of the problem in Section III. Our solution approach follows this in Section IV. Finally the paper concludes in Section V with suggestions for future work.

II. Related Work

Most replica placement problems target to minimize access time. [2] presents a replication strategy for reducing access latency and bandwidth utilization in data grid environments. The strategy used here is to applying k-means and p-center model based on the weighted average of response time. In [3], the authors present a replication strategy for optimally replicating objects in CDN (Content Distribution Network) servers using heuristics based on object popularity and storage capacity of the nodes using different topologies. In [4], the proposed replication system tries to reduce delay, energy consumption, and cost for cloud uploading of IoT applications given the massive number of devices with tiny memory sizes. In [4], the author proposes deployment of local cloud computing resources to address the LTE architectural bottlenecks within the radio access network and proposes a memory replication protocol. In [5], the authors propose an object replication and placement scheme for wireless mesh networks (WMNs) through a divide and conquer strategy. In [6], the authors exploit graph partitioning to obtain a hierarchical replica placement scheme in WMNs.

[7] presents a replication strategy which balances the load among the servers in storage of large volume of data in Cloud data centers. In [8], the authors propose a method to evenly distribute data among the available storage by using pseudo random data distribution function. In [9] the authors present a replication placement scheme for cloud storages for an IoT Environment in the health care domain. It uses a MOX (Mosquitoes Oviposition Mating) algorithm [10] to find a data replica placement solution. The implementation and evaluation of the proposed scheme is done in CloudSim [11].

In [12], a distributed algorithm for the replication placement problem has been discussed. This algorithm tries to improve the efficiency of object replication in a distributed group of storages. In [13], the reliability of clustered and declustered replica placements in data storage systems is discussed. In [14], a self managed key-value store is proposed which dynamically allocates the resources of a data cloud to several applications and thus maintains differentiated availability guarantee for different application requirements.

All of the efforts presented here attempt to store data in a way so as to minimize cost of accessing it later and none of them consider user specified policy specifications associated with data items. Although, the scenario discussed in the paper [4] is almost similar to ours, our focus in this paper is to minimize the time taken to upload the data from the gateway and replicate the data amongst the different mini-Clouds.

Our earlier work [15] presents an approach for optimal distribution of data among various mini-Clouds in a cloud-based IoT network, and also presents optimal data migration algorithms for migrating excess data between mini-Cloud storages to mitigate storage capacity issues. However, this work did not consider data replication as a requirement.

In comparison to the above, our approach has two primary advantages. First, it is based on the infrastructure and inherent property of IoT data. It considers data replication in IoT from multiple parameters, and proposes a framework by which the replication problem can be solved using some way of ordering by using different inherent properties in this scenario. Second, our approach is specific to IoT, since it takes care of small size data items which are abundant in number and tries to determine the nature of time spent for data transfer as well as amount of data transferred with each transfer.

III. Problem Formulation

Our system environment is modeled as $E =< G, C, P >$, where G is a set of gateways, C is a set of mini-Cloud storages and P is a set of policies. Gateways are responsible for receiving sensor data and forwarding it to a mini-Cloud. A gateway is modeled as $G =< I, D >$ where I is the gateway identifier and D is the set of data items in the Gateway. A data item is modeled as $D =< S', L >$ where S' is the data identifier and L is size of the data in bytes. A mini-Cloud is modeled as $C =< T, A, R, W >$, where T is total capacity in bytes, A is available capacity in bytes, R is read access latency and W is write latency (bytes/sec) per unit data size. A policy P is modeled as $P =< S', N >$ where S is a data item, and N is the required number of replicas.

Our research problem therefore is: given a network topology connecting the gateways and the mini-Clouds and available bandwidths of the links in this topology, minimize the total time required in transferring the data from all of the gateways to the mini-Clouds and replicating each data item according to the defined policy. It can be formally defined as follows:

$$\text{minimize} \max_{d=1,\ldots,m}\left[\min_{c=1,\ldots,n}\left\{T_{dj}+\left(\left(\frac{1}{B_{jc}}+R_j+W_c\right)*L_d\right)+\max_{C'\subset C}\left(T_{dc}+\left(\frac{1}{B'_{cc'}}+R_c+W_{c'}\right)*L_d\right)\right\}\right]$$

subject to

$$(i)\ c \neq c^{'}\ i.e.\ source\ cannot\ be\ destination\ for\ data\ d$$

$$(ii)\ \sum_{d=1,\ldots,m} L_d < \sum_{c=1,\ldots,n} A_c$$

$$(iii)\ \sum_{d'\subset D} L_{d'} < A_c$$

where

$n\ is\ the\ number\ of\ mini-Clouds$

$m\ is\ the\ number\ of\ data\ items$

$L_d\ is\ size\ of\ D_d\ placed\ on\ Gateway\ G_j$

$\left|C^{'}\right| = R_d - 1$
R_d *is number of replications required for* D_d
T_{dj} *is waiting time of data* D_d *at Gateway* G_j
T_{dc} *is waiting time of data* D_d *at mini* $-$ *Cloud* C_c
B_{jc} *is the rate of data transfer*
from Gateway G_j *to mini* $-$ *Cloud* C_c
$B^{'}_{cc'}$ *is the rate of data transfer*
from mini $-$ *Cloud* C_c *to mini* $-$ *Cloud* $C_{c'}$
R_j *is read latency at Gateway* Gj
R_c *is read latency at mini* $-$ *Cloud* C_c
$W_c(resp.\ W_{c'})$ *are write latencies at any*
$(resp.\ destination)$ *mini* $-$ *Cloud*

The time complexity of the exact solution to the above problem can be reasoned as follows:

1) Let there be K data items. Each data item would be uploaded to the mini-Cloud. Hence if there are R number of replications needed, each data item would be placed at R mini-Clouds.
2) Hence there must be at least one appropriate assignment of R mini-Clouds to each data item that would minimize replication time.
3) The gateway to mini-Cloud storage link would be used at most once for each data item.
4) The process of uploading can therefore be viewed as choosing first mini-Cloud for each data item out of the selected R mini-clouds for that data item; the other data transfers are to the remaining (R-1) mini-Clouds.
5) Hence the total time complexity is of two parts: choose R from C for each of the K data items, and choose one from the selected R mini-Clouds for each of the data items.
6) So, total complexity of this problem is $[K\binom{C}{R}] * R * K$.

Since this time complexity is obviously intractable, we present a heuristic solution in the next Section.

IV. Our Solution

We now present our replication strategy which is based on ordering of data items to assign a sequence to all the data items for uploading from gateways to mini-Cloud. We must assign the order of transfer of these data items at each iteration till all required replicas are stored on the mini-Clouds. We are assuming here a store and forward type of transfer, hence for any transfer to be started, a completely copied data item should be present at source.

For implementation of our heuristic based on ordering, we need an abstraction for simulating the transfer or copying among mini-Clouds from the gateways. We call this abstraction a JOB. We model each JOB as a 4-tuple comprising the following: source identifier, destination identifier, value of bandwidth of the link between source and destination identifier, and data identifier. A JOB represents the transfer of a data item from either the gateway to a mini-Cloud or from one mini-Cloud to another (the latter for replication purposes). The transfer occurs when the following conditions are satisfied: (1) the data identifier of a JOB should be available at the source identified by source identifier in the JOB, (2) the data item should *not* be present at the destination identified by destination identifier of the JOB and (3) no transfer is taking place between the source and destination at the moment. In other words, each JOB simulates a transfer process in the program for particular data items for a pair of source and destination for a required span of time. Source of the transfer may be gateways or mini-Clouds and destination of transfer would be always mini-Cloud.

Given the set of all possible JOBS at any point in time, ours is a greedy heuristic which makes the following assumption: any data item will be sent only once from the gateway to a mini-Cloud. Since the up links from the gateways are usually not as reliable and robust as the links between the mini-Clouds, replication of data items is done from the initial destination mini-Cloud to other mini-Clouds opportunistically and greedily. However, the sequence in which the data items will be sent to the mini-Cloud as well as be replicated amongst different mini-Clouds will be one of many possible sequences depending on how we order these transfers. There are eleven possible such sequences as we shall show below. Our algorithm is formally described in Algorithm 1 and encodes a greedy strategy in that we keep all network links busy as long as there is a data item at the source end of the network link that has not had its replication factor satisfied.

As stated earlier, we customize our greedy algorithm using one of the following orderings in which the JOBs will be scheduled.

1) Arrival of data - earliest data first.
2) Smallest remaining bandwidth link first.
3) Largest remaining bandwidth link first.
4) Smallest available space at destination first - tightest destination first.
5) Largest available space at destination first - loosest destination first.
6) Data item with smallest data size first.
7) Data item with largest data size first.
8) Smallest ratio of (data size/bandwidth available) first
9) Largest ratio (data size/bandwidth available) first
10) Smallest value of (bandwidth available*data size) first
11) Largest value of (bandwidth available*data size) first

We have evaluated all these orderings in our simulation. Note that our heuristic is greedy in the following ways:

- JOBs are scheduled as soon as links over which the transfer needs to happen become free. Thus if a gateway or mini-Cloud has more than one up link to the network, more than one JOB can be in progress at the same time.
- As soon as a JOB completes, a new set of JOBs are immediately spawned for the new set of available data items and free links until all requirements of the specification replication factor of the data item has been satisfied.
- The ordering of the set of all JOBs is done on the basis of one of the selected orderings in any single simulation.

The outermost $while$ loop executes till such time that no

ALGORITHM 1: Greedy Heuristic

Data: Given a set of gateways with data items, a set of mini-Clouds with a known capacity and a set of communication links among them

Result: Time taken to transfer all data items from the gateways and to replicate all data items

Initialization;

▷ let job J be composed of data item D_j, source $Source_j$, destination $Dest_j$ and link L_j reflecting a transfer event;

▷ Let S be the set of all possible JOBs at any point in time. ▷ For D data items, M mini-Clouds and L links/gateways, S will have $D \times M \times L$ JOBs initially.

repeat

  Order S based on ordering heuristic; **foreach** $JOB_j \in S$ **do**

    if $Dest_j$ *has no space for* D_j **then**

      Delete JOB_j;

    end

    if L_j *is not busy* **then**

      Schedule JOB_j;

      Delete all JOB_k from S such that $D_k = D_j$ and $Source_k = Source_j$;

      Mark L_j busy;

    end

  end

  In parallel, when any JOB_j completes Mark L_j free;

  delete JOB_j;

  If (D_j needs to be further replicated) Add new JOB_k to S, such that $Source_k = Dest_j$, $Dest_k \neq Source_k$ and $Dest_k \neq gateway$;

until $S \neq \varnothing$;

more JOBs exist. JOBs are added to S both initially when the algorithm starts as well as when a JOB completes and the data item has not yet been replicated sufficient number of times. Upon completion of one JOB all other JOBs that are for the same data item and from the same source are removed. Thus S grows and shrinks as the algorithm proceeds but will always become empty when all data items has been replicated sufficient number of times. Hence the algorithm will always terminate.

A. Experimentation Setup

We evaluate and analyze our approach using a model that captures different possible system configurations. The configurations are derived from the following parameters: (1) mini-Cloud capabilities (read and write latency, whether we can write multiple data items in parallel or not) (2) network characteristics (variable vs constant bandwidth links) (3) the limitations of storage capacity at the mini-Clouds (unlimited vs constrained) and (4) the variability in the size of the data items to be stored (variable vs constant). The system configuration is represented as a tuple of a collection of these parameters. Figure 2 illustrates the possible system configurations presented as a tree. A leaf node of this tree is a combination of these 4 parameters with specific values. For example: ACFH represents a configuration where mini-Clouds have non-zero read and write latencies for their storage, network links have variable bandwidth, the mini-Clouds have unlimited storage each and all the data items are of constant size. Other configurations such as BCFI and ADFH can be similarly interpreted.

We evaluated the different suitable configurations corresponding to every one of the 16 leaf nodes of the tree in Figure 2, for our greedy approach (Algorithm 1) with its eleven types of considered ordering presented above. We also varied the number of data items and present experimental results for the same.

Our simulation assumes the following fixed parameter values for all cases:

- Number of gateways: 50
- Number of mini-Cloud storages: 30
- Number of uplinks or downlinks: 2 for each gateway or mini-Cloud storage
- GC link bandwidth: 1 Mbps to 5 Mbps
- CC link bandwidth: 25 Mbps to 100 Mbps
- Read Latency at gateway: 25MB/sec - 50MB/sec
- Read Latency at cloud : 50MB/sec - 125 MB/sec
- Write latency at cloud storage: 10MB/sec -25 MB/sec.
- Size of data: 100 bits to 2000 bits
- Number of data items: 500 to 2000

B. Experimental Results

The graph shown in Figure 3 is for the system configuration BCEI. On the graph are shown four different sets of histograms each of which correspond to the number of data items transferred. In each set are 11 histograms - one for each of the 11 ways in which JOBs can be ordered as enumerated earlier. The Y-axis represents time in micro seconds for data transfer from gateway to mini-Cloud as well as the replication amongst mini-Clouds of all data items. We have implemented our simulation for different input data sets with number of data items being 500, 1000, 1500 and 2000.

Figure 3 suggests that for all numbers of data items, the ordering 8 (Smallest ratio of $datasize/bandwidthavailable$ first) results in the best performance, followed by ordering 11 (Largest value of $bandwidthavailable \times datasize$ first). Another point worth noting is that the ratio of performance of the best ordering to the worst ordering is almost 7x suggesting that it is extremely important to pick the right ordering for each configuration.

The configurations represented by BCEH, BCFI and BCFH result in behavior very similar to that of the BCEI configuration, i.e., the orderings 8 (Smallest ratio of $datasize/bandwidthavailable$ first) results in the best performance, followed by ordering 11 (Largest value of $bandwidthavailable \times datasize$ first). Given the similarity of results, we have omitted those graphs from the paper.

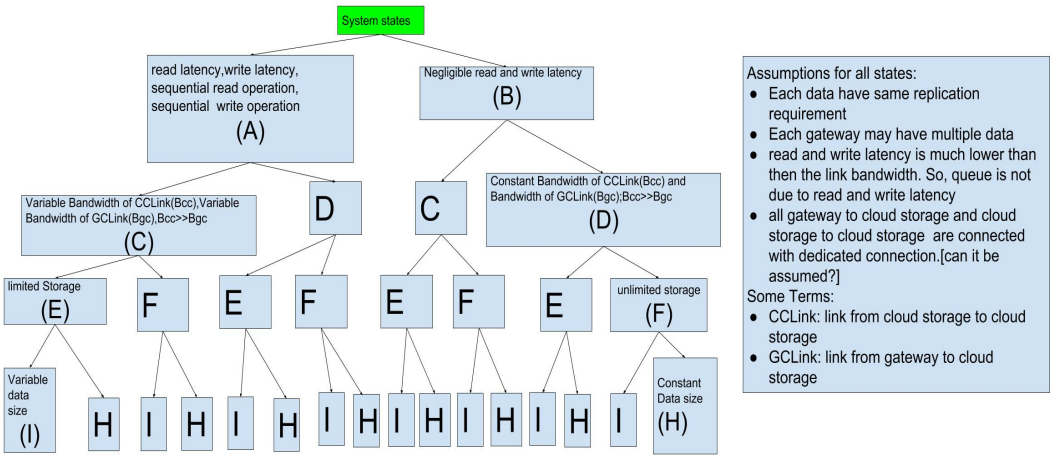


Fig. 2. The System Configuration Space as a Tree

Fig. 3. BCEI Configuration

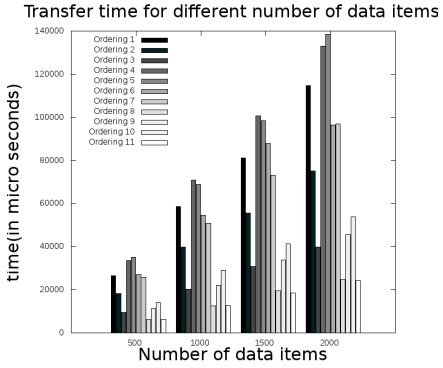
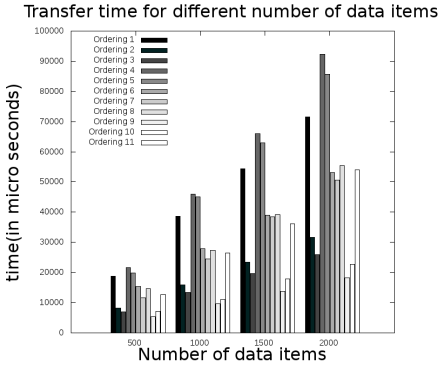


Fig. 4. BDEI Configuration



The graph with system configuration BDEI is shown in Figure 4. Here, we observe that orderings 9 and 10 result in better performance, which suggests that larger data size/bandwidth ratio is a better criterion for ordering in this scenario. The bandwidth of the links between mini-Clouds is constant in this configuration. It is also worth noting that the total time taken to transfer and replicate the data items is much lower for this configuration compared to the BCEI configuration for both the best and worst cases.

The graph with system configuration BDEH (negligible latency, constant bandwidth, limited storage and constant data size) is shown in Figure 5. This also shows results similar to that of BCEH, BCEI, BCFH and BCFI. However, a key result to be noted from BDEH, is that, with data size and bandwidth being constant, varying JOB ordering on the basis of remaining storage capacity of mini-Cloud storage does not yield significantly better results.

The next set of configurations considered are BDFI (neg-

Fig. 5. BDEH Configuration

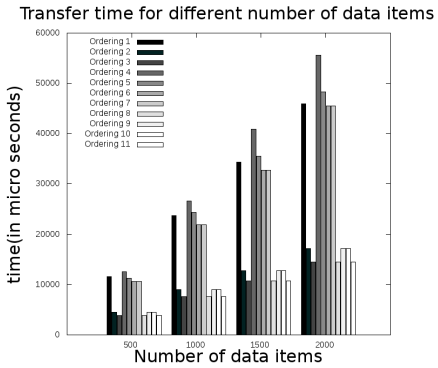


Fig. 6. BDFI Configuration

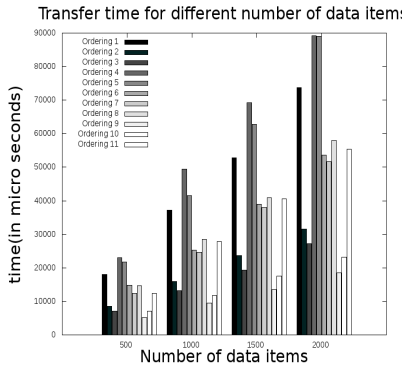
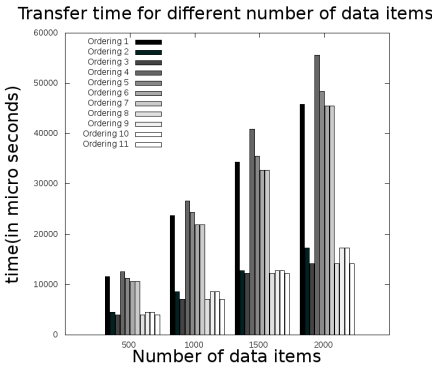


Fig. 7. BDFH Configuration



ligible latency, constant bandwidth, limited storage and variable data size; see Figure 6) and BDFH (negligible latency, constant bandwidth, unlimited storage and constant data size; see Figure 7). For both these configurations, we observe that orderings 2 and 3 show the better results, along with orderings 9 and 11. One would conclude from this that data size plays a vital role with the bandwidth in replication time in these instances.

The graph with system configuration ACEI (latency, variable bandwidth, limited storage and variable data size) is shown in Figure 9. This graph suggests that orderings 9 (larger data size/bandwidth first) and 11 (larger bandwidth*data size first) yields the best results. Hence this leads us to conclude that data size and bandwidth are key criteria for optimal performance in this case and in general, bandwidth and the data size plays an important role in our problem scenario. An important observation is that the performance for this configuration even in the best case is several times worse than the performance for the B* configurations. For 2000 data items the best performance is almost 7 to 8 times worse than the best performance for BDFI.

The graph with system configuration ACFI (latency, variable bandwidth, unlimited storage and variable data size) is shown in Figure 10. This also shows that orderings 7 and 9 provide better performance. We see similar results from configuration ADEI (latency, constant bandwidth, limited storage and variable data size; see Figure 11.

We see similar results from configurations ACFH (latency, variable bandwidth, unlimited storage and constant data size), ACEH (latency, variable bandwidth, limited storage and constant data size), ADEH (latency, variable bandwidth, limited storage and constant data size) and ADFH (latency, variable bandwidth, unlimited storage and constant data size). But this does not show any appreciable difference among the orderings, since constant data size appears to invalidate the other ordering criteria. But as each gateway has a large number of data items, sequencing does not perform well in this case.

The final configuration is ADFI (latency, constant bandwidth, unlimited storage and variable data size; see Figure 12). Here, orderings 2 and 3, and ordering 10, show better results, leading us to conclude that smaller bandwidth jobs need to be scheduled first in such a configuration.

Finally, we present a comparison of all ordering strategies for all possible system configurations on one graph in Figure 8. This clearly shows that the best performance we can hope to achieve with any ordering with the A* configurations is significantly worse than the performance we can achieve with the B* configurations leading us to conclude that performance is highly sensitive to read and write latency of storage at the mini-Clouds. In fact, the ratio of the best to worst performance for 2000 data items is more than 12 times which is significant.

C. Observations

Based on the experimental results above, we can draw the following conclusions:

1) The time for replication increases roughly linearly with the number of data items, suggesting that our heuristic is scalable. This is shown by Figure 8.
2) Selecting jobs based on higher bandwidth requirement generally provides appreciably better performance.
3) Selecting jobs based on larger data size also provides better performance, but the improvement seems to be rather marginal.
4) Storage capacity based replication should not be employed in general. However, if at all this ordering needs to be employed, it would be better to replicate data to mini-Clouds possessing larger storage capacity.
5) Whenever bandwidth is constant, data elements with variable data size modify the trends of applying orderings. In case of configurations ADFI (Figure-12), ADEI (Figure-11), BDEI(Figure-4) and BDFI(Figure-6), orderings 9 and 10 are optimal but otherwise orderings 8 and 11 are optimal. This leads us to conclude that jobs with larger data size usually provide better performance as already stated above.

Fig. 8. Ordering trends vs Configuration for 2000 data items

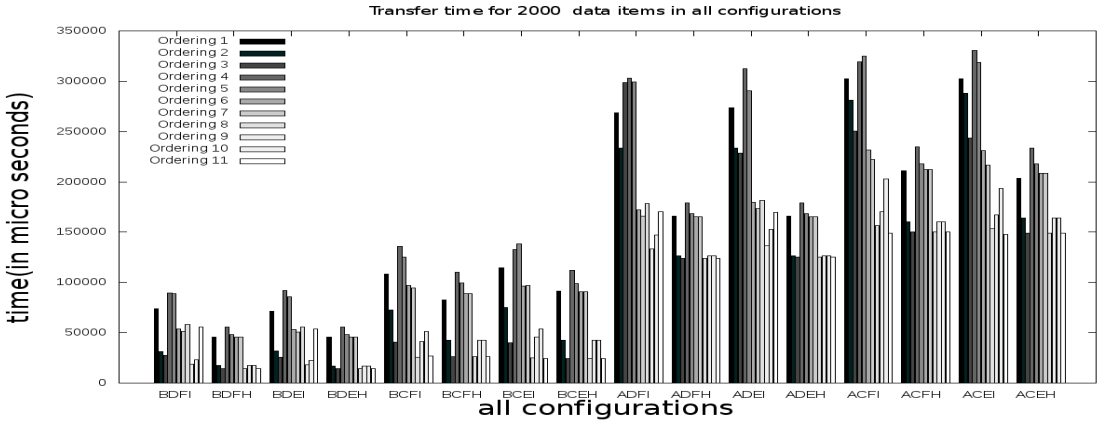


Fig. 9. ACEI Configuration

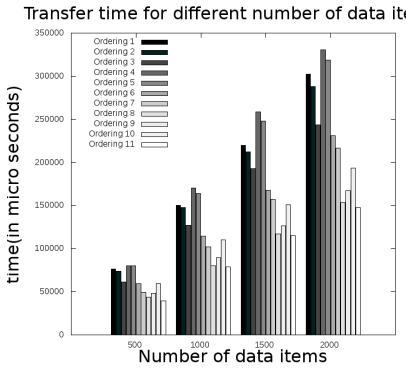


Fig. 10. ACFI Configuration

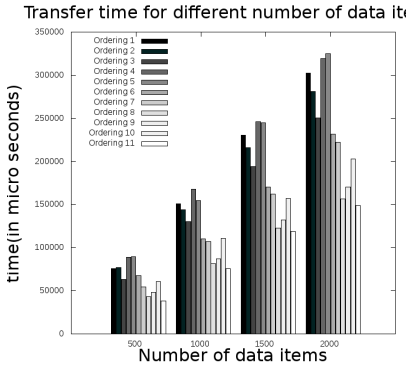


Fig. 11. ADEI Configuration

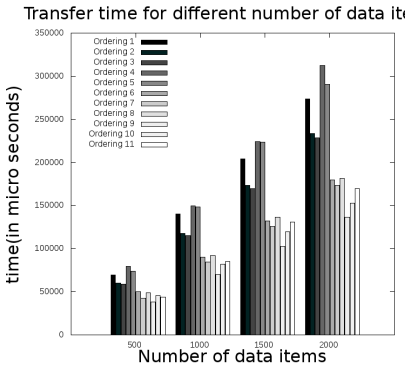
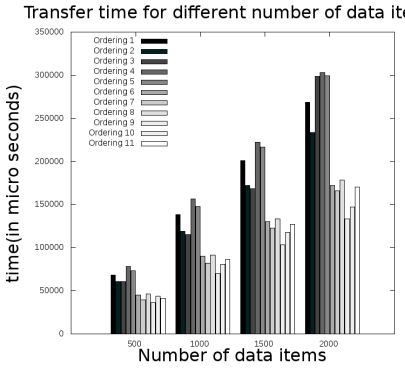


Fig. 12. ADFI Configuration



V. Conclusions

In this paper, we have investigated a crucial problem in integrating cloud computing and Internet of Things (IoT): given a distributed IoT network comprising many mini-Clouds, how to implement optimal data replication in the network. By optimal, we mean the time taken for data replication which could be quite considerable given the size of data generated in IoT networks. We have characterized the problem from parameters such as latency, link bandwidth, data size, and storage capacity of the mini-Clouds. Since the general problem is intractable, we have presented a greedy heuristic, comprising several orderings derived from the parameters. We have also presented detailed experimental results, and derived useful conclusions useful to further research, as well as to any IoT network operator that need to implement data replication in an IoT network.

For future work, we will be investigating the impact of specific IoT network topologies on data replication, and also how our various ordering strategies would perform in such topologies. We will also be conducting further experiments on larger data sets.

References

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

[2] R. Rahman, K. Barker, and R. Alhajj, "Replica placement strategies in data grid," *Journal of Grid Computing*, vol. 6, no. 1, pp. 103–123, 2008.

[3] J. Kangasharju, J. Roberts, and K. W. Ross, "Object replication strategies in content distribution networks," *Comput. Commun.*, vol. 25, no. 4, pp. 376–383, Mar. 2002.

[4] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Replisom : Disciplined tiny memory replication for massive iot devices in lte edge cloud," *Internet of Things Journal, IEEE*, vol. PP, no. 99, pp. 1–1, 2015.

[5] Z. Al-Arnaout, Q. Fu, and M. Frean, "A divide-and-conquer approach for content replication in wmns," *Comput. Netw.*, vol. 57, no. 18, pp. 3914–3928, Dec. 2013.

[6] Z. Al Arnaout, Q. Fu, and M. Frean, "Exploiting graph partitioning for hierarchical replica placement in wmns," in *Proceedings of the 16th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWiM '13. New York, NY, USA: ACM, 2013, pp. 5–14.

[7] Q. Zhang, S. Q. Zhang, A. Leon-Garcia, and R. Boutaba, "Aurora: Adaptive block replication in distributed file systems," in *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on*, June 2015, pp. 442–451.

[8] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "Crush: Controlled, scalable, decentralized placement of replicated data," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006.

[9] B. Zhang, X. Wang, and M. Huang, "A data replica placement scheme for cloud storage under healthcare iot environment," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2014 11th International Conference on*, Aug 2014, pp. 542–547.

[10] F. u. A. A. Minhas and M. Arif, "Mox: A novel global optimization algorithm inspired from oviposition site selection and egg hatching inhibition in mosquitoes," *Appl. Soft Comput.*, vol. 11, no. 8, pp. 4614–4625, Dec. 2011.

[11] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[12] S. Zaman and D. Grosu, "A distributed algorithm for the replica placement problem," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 9, pp. 1455–1468, Sept 2011.

[13] V. Venkatesan, I. Iliadis, C. Fragouli, and R. Urbanke, "Reliability of clustered vs. declustered replica placement in data storage systems," in *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, July 2011, pp. 307–317.

[14] N. Bonvin, T. G. Papaioannou, and K. Aberer, "A self-organized, fault-tolerant and scalable replication scheme for cloud storage," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 205–216.

[15] N. Narendra, K. Koorapati, and V. Ujja, "Towards cloud-based decentralized storage for internet of things data," in *Cloud Computing for Emerging Markets (CCEM), 2015 IEEE Conference on*. IEEE, 2015.