A Measurable Framework for Run-time Data Sampling in Large-scale Datacenter

Hedong Yan, Shilin Wen, Rui Han*
Department of Computer Science, Beijing institute of technology, Beijing, China
*Email address: hanrui@bit.edu.cn

***Abstract*—In large-scale data center, collecting run-time data is a very effective method which can be used to analyze and monitor the performance of data centers. But due to the huge size of data centers, limited computing resources and the requirement of low delay, it is very difficult and unrealistic to collect all the data in large-scale data centers. Therefore, to solve the serious problem, sampling partial data from all data is a common method at present. However, existing researches only focus on designing some efficient data sampling methods to reduce resource and time overhead in datacenters, but these works do not provide a unified and measurable framework to quantity the quality and practicability of other sampling methods. In this paper, we propose a measurable framework for general run-time data sampling in large-scale data center by modeling underlying recovering hypothesis explicitly. The proposed framework is mainly composed of four processes: sampling, collecting, recovering, and comparing. It could measure sampling bias degree accurately. And we design and implement three sampling methods with different recovering hypothesis. The experimental results demonstrate that the proposed framework can help us find a better run-time data sampling method effectively which has a lower sampling bias degree with same sampling rate.**

Keywords— large-scale datacenter, run-time data collecting, measurable framework, sampling bias degree, recovering hypothesis.

I. INTRODUCTION

Nowadays, monitoring the performance of large-scale datacenters in real time is vital to ensure more efficient resource scheduling, workflow scheduling, and PUE (Power Usage Effectiveness) monitoring [12]. There have been several studies for monitoring datacenters' performance based on run-time data (such as CPU utilization, Memory utilization and Disk usage, etc.) which can be used to analyze the state of every mechanical equipment in datacenters [14] [15] [16]. However, due to huge datacenter scale, limited computing resources, and the requirement of low delay, collecting run-time data of all machines or Linux Containers (Docker [13], etc.) in datacenters is pretty difficult and even impossible. That is to say, under such strict constraints of the actual big data scenario [19], how to achieve accurate and efficient run-time data collection faces with great challenges in large-scale datacenters.

To address these challenges, sampling partial devices from all of them becomes an effective approach at present. The recent work about monitoring the data center [8] used the compressed sampling that developed basis selection algorithm and adaptive-rate sampling to detect various system-level faults. Their previous work [9] is to evaluate the feasibility of compressed sampling as a monitoring approach for cloud datacenters. However, their researches only focus on evaluate compressed sampling methods, not including other sampling methods and models. Thus, the bottleneck of human' fitting ability couldn't be broken through without measurements and efficient sampling will hardly be reliable and practical when datacenter characteristics can't observed directly.

It's necessary to have a good and general measurement if we want to acquire a better data-driven sampling model which could learn the characteristics of runtime data to reduce sampling cost and sampling bias dramatically using some machine learning or deep learning methods. Therefore, in this paper, we focus on designing a measurable framework for general run-time data sampling in large-scale data center. Our contributions are mainly as follows:

(1) We propose a measurable framework for general run-time data sampling methods in large-scale data center by modeling underlying recovery hypothesis explicitly. The proposed framework is mainly composed of four processes: sampling, collecting, recovering, and comparing. In the framework, we can effectively evaluate the biasness of different run-time data sampling methods, which can help us find more accurate and efficient sampling methods to collect data in large-scale data center.

(2) We design and implement three sampling methods with Neighbor Uniform assumption or Markov assumption based on Logical clocks. Then we conduct some verification experiments on Alibaba trace [5]. The experimental results demonstrate that for different sampling methods and recovering assumption, we can always obtain sampling bias degree through our proposed framework. Therefore, which sampling method is more accurate is clear. Also, it provides possibility to optimize sampling and recovering model end to end.

This paper is organized as follows. Section II introduces some related work about run-time data collecting and existing data sampling techniques in large-scale data center. Section III details our proposed framework. In Section IV, we design and implement three different data sampling methods. We also show our verification experiments and the experimental results in section V. Finally, we conclude this paper and introduce future work in Section VI.

II. BACKGOUND AND RELATED WORK

In this section, we mainly describe the overview of our related work which consists of the following two parts: run-time data collecting and some existing data sampling techniques. They are introduced in details afterwards.

A. Run-time data collecting

In large-scale datacenter, run-time data collecting is not only a necessary process but also a challenge due to limited computing resources and the requirement of low delay. [3] proposes an open-source program which was built by UC Berkeley to monitor thousands of nodes in data centers or clusters. Xia, etc. [1] presents μDC^2 (unified data collection for data centers) which includes data sourcing step, data transfer step and data storage step and devotes to address the heterogeneous collecting challenge of datacenter and

978-1-7281-2345-5/19/$31.00 ©2019 IEEE

develop a real-time, extensible and scalable data collection framework. A perspective of Google's network traffic and load monitoring system was given in [2]. It avoids sampling the network traffic by using request/response pairs per type, which enables intra-application monitoring. And adaptive sampling was used at full URL and host information to get a high throughput [2]. However, when they were applied to large-scale datacenter, it was not enough to reach a real-time performance of monitoring if sampling techniques were not used.

B. *Data sampling techniques*

Sampling techniques often used in many fields, such as [6] [7] [10]. In large-scale data center, several related researches have developed different sampling approaches to overcome the challenge that making a tradeoff between collecting cost (including gathering cost, transferring cost and aggregation cost) and collecting bias. Among them, the most related-works are [8] [9]. [8] evaluates a compressed sampling method for runtime data monitoring and [9] develops the idea to adaptive compressed sampling that used a strategy based on adaptive-rate. The difference between our work and [8] [9] is that [8] [9] tried to implement adaptive compressed sampling in datacenter. However, our framework is not only made it measurable for compressed sampling, but also for other sampling methods.

III. PROPOSED FRAMEWORK

A. *Overview*

Run-time data sampling is to choose partial run-time data samples from population, and those samples should be representative and unbiased [4]. In this way, we could monitor the whole datacenter by collecting only a small part of run-time data. Our goal is to set a flexible framework so that we could measure general sampling methods' biasness accurately. Table I is notations we will used in this paper.

TABLE I. NOTATIONS

Symbols	Meaning
C	Clocks set in increasing order
P_i	Real runtime data population at clock i
S_i	Collected runtime data sample at clock i
A_i	Sample IDs at clock i
B_i	All IDs at clocks i
H	History table, which records latest runtime data of every device
R_c	Recovered population at clock c
f_0	Sampling function without the prior
h	Collecting function
g	Excellent reconstruct function
r	Sampling rate
d_c	Discrepancy between real data and recovered data at clock c
q	parameter of R^3S^2 sampling method

The proposed framework could be separated into four parts as Figure 1: sampling, collecting, recovering, and comparing. Sampling will select IDs of devices we will collect in the following procedure. Collecting is a set of procedures including sending instructions, data gathering, transferring, and aggregation. Performance of devices will be collected by IDs in this step. Recovering will try to reconstruct population by uncompleted data we collected under certain hypotheses. And comparing is to compare the discrepancy between recovered population and real population. The measurement of difference (such as MAD, RMSE, KL divergence etc.) is often various when we focus on different things due to all kinds of reason. The final loss value indicate the sampling bias degree of the sampling method under certain recovering hypothesis. A definition of sampling bias degree was given as follow for reference.

Definition: At a certain time, each individual in population was allocated to an estimator to predict this individual's data by a sampling model and an underlying recovering model, sampling bias degree of the sampling model with the underlying recovering model is defined as discrepancy between those estimators' expectation and real data.

Actually, sampling bias degree refers to selection bias only in this paper, measurement bias which is regarded as zero by authors will not be discussed here. The sampling bias degree we defined above indicates sampling method's ability to select representative samples to reconstruct population.

In the framework, we can measure the biasness of sampling methods in a more accurate, intuitive and clarified way. So we could optimize run-time data sampling model and explicit hypothesis model in a data-driven approach. Also we could compare the performance of different run-time sampling methods for customized datacenters without direct observation of target datacenters' characteristics (such as workflow, network topology [11] [18], and scheduling mechanism) as Figure 1.

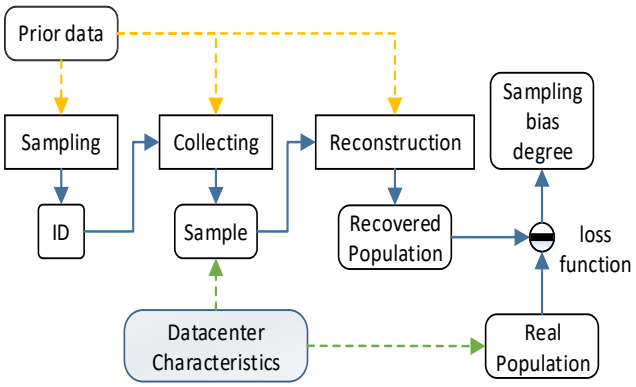

Fig. 1. Proposed framework

B. *Prior knowledge and prior data*

1) *Run-time data sampling and recovering without prior knowledge and data*

Assuming that it is clock c now, f_0 is a sampling function without the prior, mapping from clock c to sample IDs A_i. And h is a physical and practical function from A_i to S_i. It may include sending introduction to hosts which ID is in A_i, gathering their data, transferring data to central node and aggregating the collected data. And g is an excellent reconstruct function, which means that it can recover run-time data of all objects completely from partial collected run-time data. Of course, it seems that an excellent reconstruct function g is what we want, but we couldn't attain both the form and parameters of the function g in general cases without a prior knowledge and data.

2) *Run-time data sampling and recovering with prior knowledge*

It is almost impossible to learn the distribution characteristics of run-time data of the datacenter without any prior data. So that it is very hard to obtain the most appropriate run-time data sampling, collecting and recovering model according to its characteristics. Therefore, prior data for sampling is necessary to learn the distribution characteristics of run-time data quickly and easily. However, how much prior data should be introduced and how to use it is the key to the problem. The following are some prior knowledge.

a) Neighbor Uniform assumption

In traditional definition of unbiased sampling, expectation is often used to judge the unbiasedness of sampling method, which couldn't tell the concrete details between each individual of sample and population accurately. Although the sampling method is unbiased, it couldn't guarantee every sample is unbiased. When difference of expectation of whole sample and whole population is chosen as the measurement of how biased the sampling method is, hidden recovering hypothesis is that sampling between sample and population is neighbor uniform. It means that we could recover runtime data by scaling up sample to the size of population uniformly, which seems like expansion of balloon. The hypothesis usually failed in most cases because relation between ID and its runtime data is complicated. So that sampling bias can't represent the true biasness of sampling methods microscopically. Therefore, sampling methods can't achieve representative sample if each sample plays a different role in population.

b) Markov assumption based on Logical clocks

Considering we have to collect runtime data many times, each individual (host, VM, container, etc.) in datacenter is allocated a logical clock. When this individual is being sampled by central node, its logical clock will move on. And we assume that each individual is subjected to Markov assumption in its own logical clocks. It means that we could predict one device's current state completely only through its last state when its logistic clock moves on if we know the underlying state-change rule. The idea of Markov assumption based on Logical clocks is that there may not exist an accurate and synchronized clock for each individual. Only when these individuals are collected by central node in datacenter, these individuals should be considered in order to reduce computing during the interval.

IV. IMPLEMENTATION AND EVALUATION

A. Dataset and preprocess

Cluster-trace-v2017 in Alibaba trace [5] includes run-time data of Alibaba datacenter about 1300 machines in a period of 12 hours. The container data we used was collected by 144 times distributed uniformly in 12 hours. The timestamps are from 39600 to 82500 and the interval of neighbor timestamp is 300, which means 5 minutes. The number of container varies from 10260 to 10358 mostly, and the number of container was 5563 at two specific timestamp. In a certain timestamp, IDs of containers is continuous from 1 to more than 10000 mostly, expect for some discontinuities occasionally. We recommend reader to [17] for more details about Cluster-trace-v2017.

In order to simplify the collecting problem, we only chose CPU, memory and disk utilization from all 10 indicators as the performance of containers. And we would drop containers whose id was greater than 10000, so that number of containers was no more than 10000 in our experiments. In fact, number of container is usually smaller than 10000 and larger than 9000. In the preprocess, we removed five incomplete container data, whose at least one indicator of the three was NULL. Especially, if we sampled an id between 1 to 10000 but there is no runtime data with regard to the id at this clock, it would be regarded that runtime data of this container was collected although the result is NULL. The futile collecting mentioned would influenced comparison of sampling result merely because this operation would be applied to all sampling methods.

B. Algorithm design and implementation

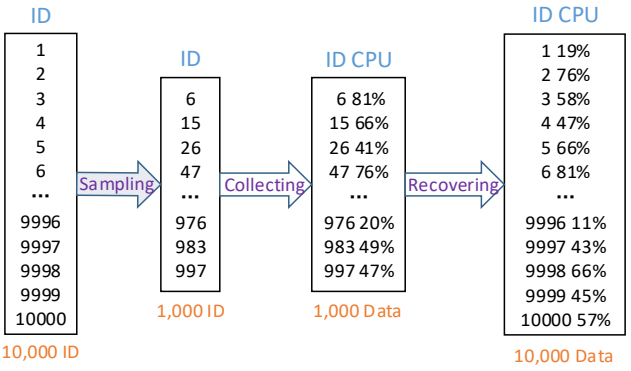

Fig. 2. An implemented instance of proposed framework without comparing

An naïve instance of the framework is as Figure 2. Frist we sample 1000 ID randomly from 10000 containers' ID, and then we collect the runtime data with regard to these 1000ID. Then we try to recover runtime data of all 10000 containers under a certain hypothesis. At last, we compare difference between recovered runtime data of 10000 containers and real runtime data. In this paper, we will choose MAD (Mean Absolute Derivation), rather than RMSE (Root Mean Squared error), as measurement of difference between recovered run-time data and real run-time data due to computational convenience. Actually, MAD is often smaller than RMSE.

There are three algorithms we designed to validate the feasibility of our framework. Algorithm 1 uses simple random sampling and recovers following Neighbor Uniform hypothesis. As the simplest case, the uncollected containers' data will be set to its nearest neighbor by ID. Algorithm 2 applies simple random sampling under Markov hypothesis based on Logical clocks. As a specific case of this, the run-time data between adjacent clocks is regarded as unchanged. Under the same hypothesis with algorithm 2, Algorithm 3 uses repeat-reducing simple random sampling to collect all containers' runtime data at once in shorter time. These three algorithms' abbreviations are called SRS+NU, SRS+ML and R^3S^2+ML in this paper respectively.

Algorithm 1 is an implementation of our framework, which could measure sampling bias degree of simple random sampling *under neighbor uniform hypothesis*. In this case, the discrepancy computed will tell us how accurate SRS is, if real run-time data are also collected. First, m ids were randomly selected from n ids as container ids that we needed to collect its data. After we collected partial runtime data, those uncollected containers' runtime data was

regarded as same as its nearest neighbor whose runtime data had been collected simply. Then we could estimate the discrepancy between recovered runtime data and real runtime data of all containers.

The main pseudo code of algorithm 1 at clock c is as follow. And we will not give specific implementation of some subfunctions. The readers who are interested in implementation details of these three algorithms could go to *https://github.com/herdonyan/MeasureSamplingBias* for source codes.

Algorithm 1: Evaluation of simple random sampling under uniform neighbor hypothesis (SRS+NU)

Input: containers' ID set B_c, real runtime data P_c, sampling rate r
Output: discrepancy d_c
$A_c \leftarrow \emptyset$, $S_c \leftarrow \emptyset$, $R_c \leftarrow \emptyset$, $d_c \leftarrow$ -1
$A_c \leftarrow$ SRS (B_c, r)
$S_c \leftarrow h$ (A_c)
for i **in** B_c **do**
  $j \leftarrow i$
  if S_c [i] *is null* **then**
    j $\leftarrow$ FINDNEARSTNEIGHBOR (A_c, i)
  end if
  R_c [i] $\leftarrow S_c$ [j]
end for
$d_c \leftarrow$ COMPARE (R_c, P_c)

Different from algorithm 1, the recovering hypothesis of algorithm 2 is *Markov assumption based on Logical clock*, although same sampling methods are used. Algorithm 2 tries to compute the loss of method SRS+ML. A history table H, which records and updates the newest run-time data of each individual, is the key we take advantage of this assumption. The length of this table is the same as number of containers and n units of the table are allocated to those n containers respectively. And when a newer runtime data of the same container was collected, the older would be replaced by the newer. The similar mechanism to history table is widely adopted, such as Cache table in computer operating system, although for different purposes and with different update algorithms. Then the recover procedure depends on a history table H instead of the table of collected partial runtime data S. The brief pseudo code of SRS+ML is as following.

Algorithm 2: Evaluation of simple random sampling and Markov assumption based on Logical clocks (SRS+ML)

Input: containers' ID sets B_c, real runtime data P_c, sampling rate r
Output: discrepancies d_c
Constraint: c belongs to C
$A_c \leftarrow \emptyset$, $S_c \leftarrow \emptyset$, $R_c \leftarrow \emptyset$, $H \leftarrow \emptyset$, $d_c \leftarrow$ -1
for c **in** C **do**
  $A_c \leftarrow$ SRS (B_c, r)
  $S_c \leftarrow h$ (A_c)
  for i **in** A_c **do**
    H [i] $\leftarrow S_c$ [i]
  end for
  for i **in** B_c **do**
    $j \leftarrow i$
    if H [i] *is null* **then**
      j $\leftarrow$ FINDNEARSTNEIGHBOR (A_c, i)
    end if
    R_c [i] $\leftarrow H$ [j]
  end for
  $d_c \leftarrow$ COMPARE (R_c, P_c)
end for

We noticed that in algorithm 2 the loss at early timestamp is too large. In order to lower relative error at start, we tried not to sample same containers many times at start to collect all containers once at least faster. Here, if runtime data of an individual was collected this time, it would not be collected next serval times probably until almost all containers had been sampled once at least. This will reduce relative error from 8.9% to 7.1% if sampling rate is 0.1. And we also believed that if we sampled runtime data of containers in turn to balance containers' waiting time to be collected, the performance of sampling model will improve. In experiments, it helped us reducing average *sampling bias* to 6.4% from 7.1%. Also, we found an "average trick" that if we use average utilization of the same container' runtime data over all past clocks, rather than its last state only, the relative error computed will reduce from 6.4% to 6.0% when sampling rate is 0.1. Pseudo code of algorithm 3 is evaluation of R^3S^2+ML without this trick.

Algorithm 3: Evaluation of Repeat-reducing simple random sampling and Markov assumption based on Logical clocks (R^3S^2+ML)

Input: containers' ID sets B_c, real runtime data P_c, sampling rate r
Output: discrepancies d_c
Constraint: c belongs to C
$A_c \leftarrow \emptyset$, $S_c \leftarrow \emptyset$, $R_c \leftarrow \emptyset$, $H \leftarrow \emptyset$, $d_c \leftarrow$ -1, $q \leftarrow 0.99$
for c **in** C **do**
  if $H.size < B_c.size * q$ **then**
    $L_c \leftarrow B_c$ - $H.ids$, $r_l \leftarrow r * (B_c.size/L_c.size)$
    $A_{c1} \leftarrow \emptyset$
    if $r_l > 1$ **then**
      $r_h \leftarrow (r * B_c.size - L_c.size)/H.size$
      $A_{c1} \leftarrow$ SRS (H, r_h)
    end if
    $A_{c2} \leftarrow$ SRS (L_c, r_l)
    $A_c \leftarrow$ Merge (A_{c1}, A_{c2})
  else
    $H \leftarrow \emptyset$
    $A_c \leftarrow$ SRS (B_c, r)
  end if
  $S_c \leftarrow h$ (A_c)
  for i **in** A_c **do**
    H [i] $\leftarrow S_c$ [i]
  end for
  for i **in** B_c **do**
    $j \leftarrow i$
    if H [i] *is null* **then**
      j $\leftarrow$ FINDNEARSTNEIGHBOR (A_c, i)
    end if
    R_c [i] $\leftarrow H$ [j]
  end for
  $d_c \leftarrow$ COMPARE (R_c, P_c)
end for

C. Experimental evaluation

We regard the runtime data of Alibaba Trace[5] as real population, and we simulate collecting procedure by selecting some containers' runtime data as collected sample.

Which container should be selected is determined by sampling methods.

Our machine for experiment is Intel(R) Core(TM) i7-4790 CPU @3.60GHz, the memory size is 32.0GB, and the system is Windows 10.

Table II is three different configurations of the framework and we choose different sampling rate to see the sampling methods' performance changing over sample rate. The number of containers is about 10000, and average CPU, memory and disk utilization over the 144 timestamps are 9.52%, 44.30% and 14.33% respectively.

In order to show that recovering hypothesis could not be neglected and it plays a vital role if we want to get lower sampling bias degree, the comparison between algorithm SRS+NU and SRS+ML were presented as Figure 3 when sampling rate is 0.1. When the Neighbor Uniform hypothesis is extended to Markov assumption with Logical clock, the relative error is 1/5~1/3 of the former. It shows the importance of underlying hypothesis of recovering procedure. So, the sampling bias will be influenced by underlying recovering hypothesis.

Comparing algorithm 2 and 3, we know that we could improve the performance of sampling methods by changing part configuration of this framework. The accuracy of algorithm 3 improved about 1/3 comparing to algorithm 2. Finally, relative error of algorithm 3 could reach 6% when sampling rate is 0.1 using container data of Alibaba Trace.

TABLE II. RELATIVE ERROR IN DATACENTER WITH 10,000 CONTANIERS

Sampling rate	Sampling methods under some hypotheses	Relative error of CPU utilization (%)	Relative error of Memory utilization (%)	Relative error of Disk utilization (%)	Average relative error (%)
0.1	SRS+NU	39.7	25.8	28.9	31.5
	SRS+ML	17.4	4.2	5.2	8.9
	R^3S^2+ML	12.9	2.4	2.7	6.0
0.2	SRS+NU	29.0	13.3	20.0	20.8
	SRS+ML	11.2	2.2	2.6	5.3
	R^3S^2+ML	8.8	1.3	1.3	3.8
0.4	SRS+NU	17.1	6.7	8.4	10.7
	SRS+ML	6.4	0.9	1.0	2.8
	R^3S^2+ML	5.4	0.6	0.6	2.2

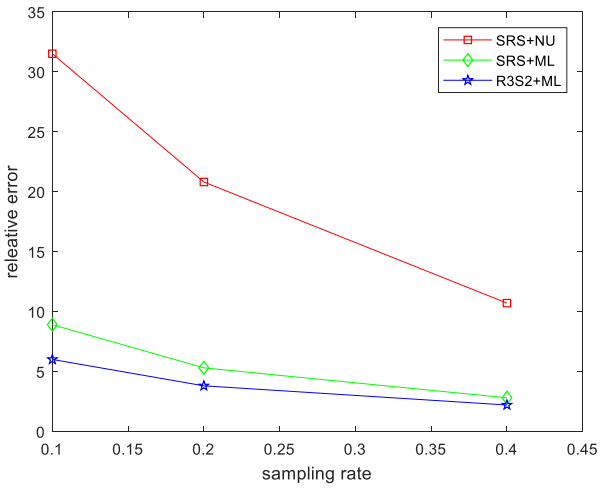

Fig. 3. Relative error of different sampling methods

And we also tested R^3S^2+ML in different scale datacenters by using different number of containers of Alibaba Trace [5]. For example, 1000 containers means that only containers whose ids are 1~1000 will be considered. When sampling rate equals 0.1, the result is as Table III and Figure 4. The scale of datacenter affects the relative error of this sampling method little. And the biasness of sampling on CPU utilization is much larger than memory utilization dramatically. Despite of properties of CPU and memory, CPU utilization is usually below 10%, which is much smaller. So, the relative error of CPU utilization may seems higher than memory utilization if their fluctuations are not very different.

TABLE III. RELATIVE ERROR OF R^3S^2+ML IN DIFFERENT SCALE DATACENTERS

Number of containers	Relative error of CPU utilization (%)	Relative error of Memory utilization (%)	Relative error of Disk utilization (%)
1000	13.1	2.5	2.8
2000	13.0	2.4	2.7
3000	13.2	2.4	2.8
4000	13.0	2.4	2.8
5000	12.7	2.4	2.6
6000	12.7	2.4	2.7
7000	12.9	2.4	2.7
8000	13.0	2.4	2.7
9000	12.9	2.4	2.8
10000	13.0	2.4	2.7

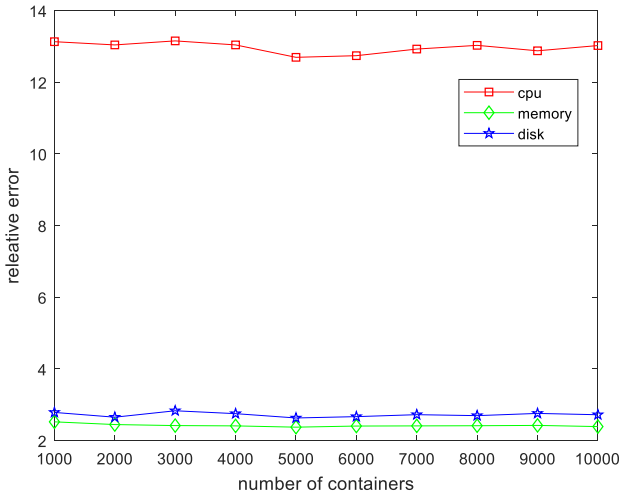

Fig. 4. Relative error of R^3S^2+ML in different scale datacenters

V. CONCLUSION

In this paper, we propose a measurable framework for general run-time data sampling in large-scale data center. By using our proposed framework, we can effectively evaluate the quality and practicability of different run-time data sampling methods in datacenters. And in order to evaluate the validity of our framework, we design and implement three sampling methods based on Neighbor Uniform distribution and Markov assumption based on Logical clocks. Then we conduct some verification experiments on Alibaba Trace and the experimental results demonstrate that for different sampling methods and data population recovering assumption, we can obtain sampling bias degree of different sampling methods through our proposed framework. So, we can obtain a better sampling method which has a lower sampling bias degree with same sampling rate.

In the future, in order to obtain more accurate and efficient sampling methods for monitoring the performance of datacenters, we will focus on the representation and learning of run-time data sampling and reconstruction model in large-scale datacenters.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (Grant No. 61872337). And, thanks for Zifeng Liu's useful advices about dataset.

REFERENCES

[1]. Xia W, Wen Y, Xie H, et al. μDC^2: unified data collection for data centers[J]. The Journal of Supercomputing, 2014, 70(3): 1383-1404.

[2]. Ryckbosch F, Polfliet S. Efficient Data Center Monitoring: U.S. Patent Application 14/492,176[P]. 2015-3-26.

[3]. Massie, M.L., B.N. Chun and D.E. Culler, The ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing, 2004. 30(7): p. 817-840.

[4]. Lohr S L. Sampling: design and analysis[M]. Nelson Education, 2009.

[5]. "Alibaba trace," https://github.com/alibaba/clusterdata, 2017.

[6]. Xu Z, Zhang L, Shen J, et al. MRCS: matrix recovery-based communication-efficient compressive sampling on temporal-spatial data of dynamic-scale sparsity in large-scale environmental IoT networks[J]. EURASIP Journal on Wireless Communications and Networking, 2019, 2019(1): 18.

[7]. Ramakrishnan S K, Jayaraman D, Grauman K. Emergence of exploratory look-around behaviors through active observation completion[J]. Science Robotics, 2019, 4(30): eaaw6326.

[8]. Huang T, Kandasamy N, Sethu H, et al. An efficient strategy for online performance monitoring of datacenters via adaptive sampling[J]. IEEE Transactions on Cloud Computing, 2016.

[9]. Huang T, Kandasamy N, Sethu H. Evaluating compressive sampling strategies for performance monitoring of data centers[C]//Proceedings of the 9th international conference on Autonomic computing. ACM, 2012: 201-210.

[10]. Li Y, Dai W, Zou J, et al. Structured sparse representation with union of data-driven linear and multilinear subspaces model for compressive video sampling[J]. IEEE Transactions on Signal Processing, 2017, 65(19): 5062-5077.

[11]. Singh A, Ong J, Agarwal A, et al. Jupiter rising: A decade of clos topologies and centralized control in Google's datacenter network[J]. ACM SIGCOMM computer communication review, 2015, 45(4): 183-197.

[12]. Barroso L A, Hölzle U. The datacenter as a computer: An introduction to the design of warehouse-scale machines[J]. Synthesis lectures on computer architecture, 2009, 4(1): 1-108.

[13]. Anderson C. Docker [software engineering][J]. IEEE Software, 2015, 32(3): 102-c3.

[14]. Mazak A, Lüder A, Wolny S, et al. Model-based generation of run-time data collection systems exploiting AutomationML[J]. at-Automatisierungstechnik, 2018, 66(10): 819-833.

[15]. Spinner S, Filieri A, Kounev S, et al. Run-Time Models for Online Performance and Resource Management in Data Centers[M]. Self-Aware Computing Systems. Springer, Cham, 2017: 485-505.

[16]. Walter J, Di Marco A, Spinner S, et al. Online learning of run-time models for performance and resource management in data centers[M]. Self-Aware Computing Systems. Springer, Cham, 2017: 507-528.

[17]. Cheng Y, Anwar A, Duan X. Analyzing Alibaba's co-located datacenter workloads[C]//2018 IEEE International Conference on Big Data (Big Data). IEEE, 2018: 292-297.

[18]. Yantao Sun,Jing Cheng,Konggui Shi, et al. Data Center Network Architecture[J]. ZTE Communications, 2013, 11(1):54-61. DOI:10.3969/j.issn.1673-5188.2013.01.010.

[19]. Han R, John L K, Zhan J. Benchmarking big data systems: A review[J]. IEEE Transactions on Services Computing, 2017, 11(3): 580-597.