iGen: A Realistic Request Generator for Cloud File Systems Benchmarking

Zujie Ren[1], Biao Xu[1], Weisong Shi[2], Yongjian Ren[1], Feng Cao[3], Jiangbin Lin[3] and Zheng Ye[1]

[1]*School of Computer Science*
Hangzhou Dianzi University
Hangzhou, China
{renzju,xubiao.codeyz}@gmail.com

[2]*Department of Computer Science*
Wayne State University
Detroit, USA
weisong@wayne.edu

[3]*Pangu Team*
Alibaba Cloud Computing Ltd.
Beijing, China
{feng.cao,jiangbin.lin}@alibaba-inc.com

***Abstract*—Benchmarking is a traditional approach for system performance evaluation and optimization. Over the past decades, a variety of file systems, *e.g.*, GFS, HDFS and Ceph, have been designed and implemented, serving as the key components in cloud infrastructures. With the mature of those cloud file systems, the demands for performance evaluation and comparison are also rising. However, due to the complexity and heterogeneity of I/O workloads in cloud infrastructures, it is still challenging to generate realistic I/O workloads. System developers often use traditional file system benchmarks and make inaccurate assumptions on workload generation, yielding to misleading results.**

To address this problem, we investigate the characteristics of I/O requests in a production cloud infrastructure at Alibaba Cloud Computing, which is one of the biggest cloud providers in Asia. We proposed a flexible framework *iGen* to mimic I/O request arrivals. One of the salient features of the iGen is that the request arrival process is modeled by three statistics properties, *request arrival rate*, *inter-arrival time distribution*, and *request periodicity*. According to these properties, the iGen can determine the sequence of requests and the inter-arrival time between two subsequent requests. We use the iGen to emulate a real workload that collected from Alibaba cloud platform. Experimental results show that high accuracy and flexibility of the iGen.

***Keywords*-cloud file systems; benchmarking; workload generator;**

I. Introduction

Last decade has witnessed that cloud infrastructures gain a rapid increase of popularity. Some well-known cloud infrastructures, *e.g.*, Amazon EC2 and Microsoft Azure attract millions of users. As the storage-layer components of cloud infrastructures, various cloud file systems have been designed and implemented. Typical examples include GFS (Google File System), HDFS (Hadoop Distributed File System), Ceph [1], GlusterFS [2], and so on. Notable research efforts have been devoted to improve their scalability, reliability[3], fault-tolerance[4], *etc*.

As the cloud file systems mature, the demand to evaluate the performance of these file systems rises. Although hundreds of file system benchmarks have been proposed [5] in the past decades, most of them are unsuitable for cloud file systems because their workload characteristics differ significantly. Previous studies consistently reported that the I/O workloads in cloud file systems are much more heterogeneous and dynamic than traditional file workloads [6][7]. Though existing file system benchmarks, *e.g.*, Postmark [8] and SPC[9] , can provide a variety of options to generate diverse workloads, they have trouble generating realistic I/O workloads on cloud infrastructures, making the benchmark results inaccurate.

To address this problem, we collected a two-week I/O workload trace from Alibaba cloud file system, called Pangu, which supports object storage, block storage and file storage for a variety of Alibaba cloud services. The workload trace is characterized in terms of multiple properties, including the request arrival pattern, session behaviors, periodicity pattern, request size, and so on[6]. One of key observations in [6] is that the request arrivals do not follow a Poisson process. Another is that the request arrival process presents multiple periodicities. The pattern of request arrival process is potentially significant but has been overlooked unfortunately in the past. Most of workload generators simply assume the requests arrive in a constant rate, or simply regard the request arrivals as a Poisson process, failing to capture the realistic arrival pattern.

Motivated by the previous observations, we perform a more in-depth investigation on the model of request arrival, and propose a novel framework *iGen* to generate realistic request arrivals. In the iGen, the arrival process is modeled by three statistics properties.

1) *Request arrival rate*. Request arrival rate reflects the time-varying intensity of workload. It is usually defined as the number of arrived requests per second. In the iGen, the arrival rate is time-varying, rather than constant.
2) *Inter-arrival time distribution*. The inter-arrival time distribution regulates the intervals of two subsequent requests arrival time.
3) *Request periodicity*. At a long-time scale, I/O requests exhibit a clear periodicity, such as daily or weekly. I/O workloads may contain other periods, which may be identified using the workload's auto-correlation [10]. Different cloud services may generate cyclic request streams with different periods, so the periodicity should be miscellaneous.

In this work, we have implemented iGen[1] as the workload generator module of a file system benchmark, which allows users to specify multiple parameters of workload properties. We address the issue of generating realistic request arrivals commonly observed in cloud infrastructures. The main contributions of this work can be summarized as follows.

1) We showed that request arrival process is a complex stochastic process. The request arrival rates follow a lognormal-like distribution and inter-arrival times follow a Pareto-like distribution.
2) We proposed that request arrival process can be modeled by three configurable properties, *request arrival rate*, *inter-arrival time distribution*, *request periodicity*.
3) We implemented a flexible framework to mimic realistic requests, called iGen. The iGen is designed with high flexibility to generate a customized workload based on a group of configurable parameters. As a case study, we employed the iGen to generate request arrivals observed in the Pangu, showing the high accuracy of the iGen.

The rest of this paper is organized as follows. In Section II, we briefly review some related work on big data system benchmarks and illustrate the motivations of our work. We describe the arrival generation model in Section III. Section IV presents the design and implementation of the iGen. The evaluation and validation of the iGen are shown in Section V. We conclude this paper in Section VI.

II. Related Work

The topic of workload modeling and generation has been discussed in many previous literatures. A myriad of related literatures have shed light on this work. In this section, we review some related work, and illustrate the motivations of our work.

With the advent of big data techniques, there have been plenty of benchmarks developed to benchmark the performance of various big data systems. Table I presents an overview of the state-of-the-art big data benchmarking efforts. These benchmarks are widely used for evaluating different target systems, *e.g.* database systems and virtualized platforms. As the target systems of those benchmarks are different, the workload types and arrival patterns differ with each other.

For example, CloudBench[11] is an IaaS benchmark tool for deploying complex applications and running applications inside acquired VMs. It is quite useful for customers to choose appropriate cloud providers. YCSB (Yahoo! Cloud Serving Benchmark) [31], was proposed to compare the performance of transactional processing systems including Cassandra, HBase, PNUTS, and a simple sharded MySQL implementation. YCSB supplies several workloads with different combinations of `insert`, `read`, `update` and `scan` on database tables. Pitchumani *et al.* [25] proposed a YCSB-based key-value storage benchmark. In the work, the authors modified YCSB to generate workloads based on three three categories, including Poisson process, self-similar process and envelope-guided process.

Shi *et al.* [32] developed two benchmarks with a collection of structured queries, to compare the performance of Cassandra, HBase, Hive and HadoopDB. While in the field of MapReduce-style computing, GridMix2[13], HiBench[33] and SWIM[14] contain some representative offline analytical jobs and target on benchmarking MapReduce-style data analytical systems. Our work is orthogonal to these benchmarks because we focus on evaluating the performance of file systems in cloud infrastructures. The types of workloads and their characteristics are totally different.

Traeger *et al.* [5] examined 415 file system benchmarks from over 100 papers, such as PostMark [8] and SPC[9], the authors found that in many cases benchmarks do not provide adequate evaluation of file system performance. In contrast to traditional cluster computing systems, the workload in a cloud platform is much more heterogeneous, complex and dynamic [7][34]. The characteristics of I/O requests in clouds are quite different and complex. Due to the lack of a publicly available trace, file system workload analysis in cloud platforms has not been well studied yet.

In contrast to traditional file systems, workload characterization on cloud file systems poses special challenges for benchmarking efforts. These challenges arise from the characteristics of cloud file systems, including (1) system complexity, which makes it difficult to develop request processing models; (2) workload heterogeneity and dynamicity, which hamper efforts to identify representative workload behavior; (3) high data volume and large cluster scale, which make it challenging to replay the workload and reproduce system behavior; and (4) rapid system evolution, which requires benchmarks to accommodate changes in the underlying systems. Without real-life empirical knowledge, system researchers and engineers could easily make incorrect assumptions about their own workloads, thus yielding inaccurate performance evaluation results.

Traditional models for generating requests include statistics-based models and queue theory based models[6][17]. In these models, data characteristics like probability distribution and stochastic process are analyzed firstly. For example, Meisner *et al.* [35] describe BigHouse a simulation infrastructure for data center systems. Bighouse uses a combination of queuing theory and stochastic modeling to derive workload inter-arrival and service time distributions. Juan *et al.* [17] investigated the inter-arrivals of job requests in an industrial data centers. In [17], the authors found that the majority of job requests show a regular periodicity with log-logistic noise, power-law-like distribution.

Despite some recent studies [7, 36, 37] that characterized

[1] iGen Project - https://github.com/renzj/iGen

Table I
AN OVERVIEW OF BIG DATA BENCHMARKING EFFORTS.

Benchmarks	Workload Types	Target Systems	Arrival Pattern
CloudBench[11] CloudBurst[12]	VM instances deployment and applications running	IaaS Clouds, *e.g.*, Amazon EC2, Emulab, Openstack	N/A
GridMix2[13] HiBench, SWIM[14]	MapReduce jobs	MapReduce-style systems	Unconsidered or Poisson arrival process
BigDataBench[15][16]	Offline analysis, online services, realtime analytics	Hadoop, Spark, MySQL, NoSQL, MPI, Impala, etc.	N/A
HiBM [17]	A mix of MapReduce jobs and high-performance (or throughput) jobs	Resource schedulers in data centers.	Power-law-like distribution
CloudSuite[18] DCBench[19]	Data analytics, data caching, data serving, media streaming, Web search, Web serving	NoSQL, DBMS, Hadoop, etc.	N/A
Rain[20], CloudStone[21]	A mix of HTTP requests	Web 2.0 applications	N/A
COSBench[22]	Object operations, Restful API (PUT, GET, DELETE)	Object storage systems, such as Amazon S3, Openstack Swift, MOS	N/A
YCSB[23] YCSB++ [24]	Relational operations, record insert, update, scan, read	NoSQL, key-value store	Constant rate
Pitchumani et al. [25]	Relational operations, records insert, update, scan, read	NoSQL, key-value store	Poisson, self similar or Envelope-guide process
TPC-C TCP-W	Relational operations	Databases	Poisson arrival process.
MimesisBench [26] Abad *et al.*[27]	File meta-data operations	HDFS	Delayed renewal process, clustered renewal process
Delimitrou et al.[28]	Block-level operations, random/sequential reads/writes blocks	Block-level storage systems.	Inter-arrival times follow one of four distributions: normal, exponential, poisson and gamma.
PostMark[8], SPC, Filebench [29] CodeMRI [30],	File and directory operations, random/sequential reads/writes	(Distributed) File systems	Poisson process or unconsidered
iGen (proposed in this paper)	File and directory operations, random/sequential reads/writes	Cloud file systems	Five alternatives: log-normal, normal, Poisson, *etc.*

the workloads on cloud storage systems, it is far from enough to know how to mimic a realistic workload for benchmarking cloud file systems. The lack of understanding realistic request arrivals motivates our work in this paper.

III. REQUEST ARRIVAL PROCESS MODELS

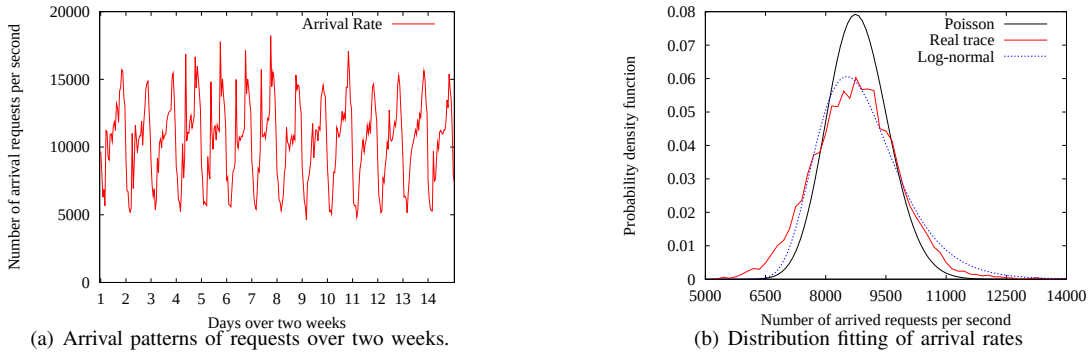
Request arrival process is a stochastic process. The distribution of Inter-Arrival Time (IAT for short) and arrival rate (the number of Arrived Requests Per Second, ARPS for short) are key attributes for profiling the pattern of arrival process. A commonly-used method for modeling request arrivals is *Poisson* process. In a Poisson process, the request arrivals are independent events, with an average rate of λ. Meanwhile, the inter-arrival times follow an exponential distribution.

Although many big data benchmarks generate the requests using a Poisson process [25][14], it is not always true in cloud environments. Our previous study on Pangu file system shows that the ARPS follows a log-normal-like distribution [6]. The details of the request arrival pattern are depicted in Figure 1. Motivated by this observation, we argue that using a Poisson process is not precise enough to model the real request streams. In the remainder of this section, the models for generating realistic request arrivals will be depicted in detail.

A. The ARPS Generation Model

The distribution of ARPS reflects the fluctuation of request arrival rates. In the iGen, the ARPS is constructed by two steps.

1) First, generate a group of arrival rate values that follow a log-normal distribution. The parameters of mean and standard deviation can be specified by the users. For most users, the optimal values for the parameters should be derived by a quantitative workload analysis of their own I/O traces. The arrivals may be a mixture of multiple kinds of distributions, and it is difficult to decompose workload and figure out which the best fitting distribution is and what the optimal values of the corresponding parameters are. To address this problem,



(a) Arrival patterns of requests over two weeks.
(b) Distribution fitting of arrival rates

Figure 1. Patterns of request arrivals observed in the Pangu file system [6].

the users are allowed to set a sample of arrival rate values, and the iGen creates the complete group of arrival rates by fitting the sample values.

2) Second, the periodicity attribute of workload is introduced to regulate the arrival rate in a longer-time scale. In the iGen, users are allowed to set multiple periods for a specific workload. The multiple periods schedule the fluctuation arrival rates over time.

The distribution and periodicity determine the stream of request arrival events of a simulated I/O workload. At this state, the number of arrival request for a given time interval has been determined. However, the generation of request arrival is not completed yet, as the intervals of two subsequent requests (IAT) are still undetermined. Supposing the arrival rate is 10 req/s during one second, these ten requests within the one-second interval do not arrive with a constant rate. Therefore, the IATs should be generated to form the arrival patterns for every second.

B. The IAT Generation Model

If the requests arrivals are independent, they will form a Poisson process and the inter-arrival times follow an exponential distribution. However, our previous study demonstrates that inter-arrival times fail to follow an exponential distribution, but show a Pareto-like distribution. This observation can be explained by that the request arrivals are not independent. Therefore, in the iGen, Pareto distribution is used to model inter-arrival times.

1) Pareto Distribution: The Pareto distribution is a kind of heavy-tail and power-law distribution. The survival function of Pareto distribution is:

$$P(X > x) = \begin{cases} 1 & x < x_t \\ (\frac{x_t}{x})^{\alpha} & x >= x_t \end{cases} \quad (1)$$

The probability function of the Pareto distribution is governed by shape parameter α and scale parameter x_t. A large-scale parameter gives the minimum value of X while shape parameter, also called *tail index*, has an effect on its skewness.

The expectation of the Pareto distribution is calculated as follows:

$$E(X) = \frac{\alpha x_t}{\alpha - 1} \quad (2)$$

The Pareto distribution is common in the field of computer systems. For instance, the file sizes followed a Pareto-like distribution [38]. In the field of computer network, the best fitting distribution of the IATs of network requests is the Pareto distribution, rather than an exponential distribution. In addition, multiple Pareto distributions with different shape parameters, are mixed to make up all of the IATs. This conclusion implies IATs should be modeled by multiple probability functions with different parameter values.

2) Generating IAT with Pareto Distribution: The ARPS and IAT are two key attributes of request arrival pattern. For a Poisson process, it is easy to generate a group of IATs that accord with an exponential distribution. As discussed in Subsection III-A, the ARPS varies over time and it is likely to follow a log-normal distribution. A realistic workload should fit both of these properties. We use the Pareto distribution to generate some inter- arrival times to form a time interval. Table II lists a set of symbols used in the following description and their meanings.

Given an arrival rate N in an interval T, the interval T needs to be divided into N subsequent IATs. At first, we know that

$$T = \sum_{i=1}^{N} X_i \quad (3)$$

and let the Pareto distribution's expectation be equal to the average of N IATs,

$$E(X) = \frac{\alpha x_t}{\alpha - 1} = \frac{T}{N} \quad (4)$$

For a given α, the scale parameter can be calculated from Equation 2 and 4, thus

$$x_t = \frac{T(\alpha - 1)}{N\alpha} \quad (5)$$

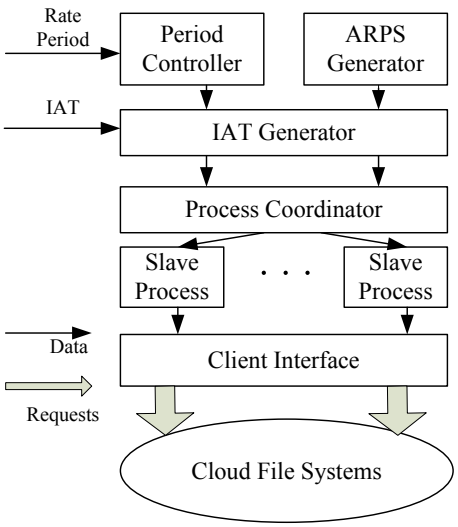


Figure 2. The architecture of the iGen.

Then, the set of IAT values can be produced using the parameters x_t and α.

Table II
SYMBOLS AND THEIR DESCRIPTIONS USED IN THE WORK.

Symbols	Descriptions
T	a time interval in resolution of second
N	number of request(s) arrived in T
X	IAT series
x_t	scale parameter of Pareto distribution
E(X)	expectation of Pareto distribution
α	Parameter of iGen, defining shape parameter of Pareto distribution
γ	Parameter of iGen, defining array of ARPS distributions
ρ	Parameter of iGen, defining array of periods

IV. DESIGN AND IMPLEMENTATION OF THE IGEN

The iGen simulates I/O requests and submits the request stream to target file systems. It mimics realistic I/O requests by three statistical properties, including request arrival rate, inter-arrival time distribution and request periodicity. Figure 2 demonstrates the architecture of the iGen in detail.

Based on our previous research findings, in cloud infrastructure different services have various ARPS distributions and periods. Even for a same cloud service, the APRS distribution varies over time. Therefore, to achieve high flexibility, the iGen is designed to be easily configured with multiple ARPS distributions and periods, which are integrated to generate hybrid I/O workloads. The iGen contains a *Client Interface* module that is responsible for invoking common I/O operations described as follows.

- *Write*. Write data into an existing file.
- *Read*. Read data from an existing file.
- *Create*. Create a new file.
- *Delete*. Delete an existing file.
- *Rename*. Rename an existing file.
- *CreateDirectory*. Create a new directory.
- *DeleteDirectory*. Delete an existing directory.
- *RenameDirectory*. Rename an existing file.
- *List*. List all files of a directory.

In the iGen, the request arrivals obey an open model [39]. In an open model, new requests arrive independently of request completions. The number of requests in the system is indefinite at any time. Every request is arrived asynchronously and thread pool is applied to achieve this feature. Once a request is generated, it will be put into the thread pool. If there are some threads available, the request will be fetched and executed. Otherwise, the request will be blocked and kept waiting. To enhance the scalability, the iGen adopts one-master/multiple-slave architecture. The *Master Process* is responsible for generating requests following some arrival patterns and distributing them to *Slave Process*, while the *Slave Process* receives and triggers request submission.

The *Master Process* is composed by four modules, which are described as follows.

1) *Period Controller*. This module regulates the periods of ARPS. In computer, the same random seed determines the same random number. Thus, every time *Period Controller* resets the random seed, the same ARPS series are generated. The accuracy of period depends on efficiency and precision of clock.
   It will register the alarms in the interrupt vector table and provide resolution in millisecond. A linked list of period values is maintained. Every time the alarm comes, it checks the linked list whether there is a period ending or not.
2) *ARPS Generator*. This component generates ARPSs. It contains an array of threads, corresponding to parameter γ. Each thread generates one of the ARPS distributions. Except these threads, the generator has a queue which preserves the sum of arrival rate array. Every thread puts ARPS into it. Serious preemption will happen if using *mutex* to synchronize thread. To improve efficiency, atomic operations like *fetch_add* is adopted. This component itself is producer-consumer. It generates ARPSs and passes them to *IAT Generator* to produce IATs.
3) *IAT Generator*. For every second, this component receives ARPS from *ARPS Generator*. Then, it generates IAT using Pareto distribution.
4) *Process Coordinator*. To generate higher loads, multiple slaves are set to send I/O requests conforming to the same configuration. This component is responsible for distributing requests generated by master to slaves. In order to decrease frequency of sending requests in each slave node, load balancing is applied. Its principle is,

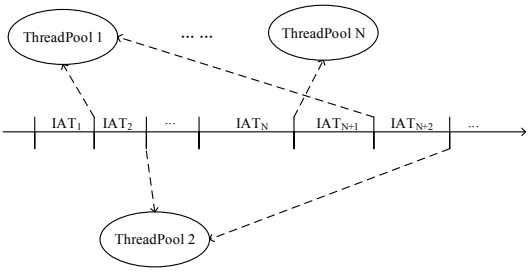


Figure 3. Coordination of the slave processes.

for N slave nodes, M time-sequential requests and their IATs IAT_1, IAT_2, ..., IAT_M, assigning requests one by one, circularly. The i-th slave gets i-th, i+N-th, i+$2N$-th $\cdots$ requests, as shown in Figure 3. Thus, in the i-th slave, its new IATs for requests are generated using

$$IATs = (\sum_{j=1}^{i} IAT_j, \sum_{j=i+1}^{i+N} IAT_j, \sum_{j=i+N+1}^{i+2N} IAT_j, ...) \quad (6)$$

The interval times between requests become larger and action of sending is less frequent. So each slave gets enough time to schedule.

The *Slave Process* is responsible for receiving IATs from master and sending requests to the target systems. Each slave process has a timer for deciding when to transmit requests according to the corresponding IATs. Once an IAT ends, one request will be selected and put into one of the thread pools.

V. Evaluation

In this section, we conducted a group of experiments to validate the accuracy of the iGen framework. The trace used in these experiments is same as the one in [6].

A. Periodicity

The feature of periodicity is common in various workloads, *e.g.,* daily, weekly period, or even mixture of multiple periods. Figure 4 presents the periodic behavior of ARPS in two hours. A period with 30 seconds is pre-defined by users. Figure 4 shows that the arrival rate has a period around 30 seconds. The method of discrete fourier transform is used to detect the periodicity. Figure 5 depicts the amplitudes of each discrete frequency and high power-spectrum amplitude at 30-seconds is consistent with user configuration.

In addition, we conducted another experiment of validate the effectiveness by configuring iGen with multiple periods. More specifically, the periods of 30-second and 50-second work together, while the corresponding ARPS is a log-normal distribution with the mean value of 6.5 and the standard deviation value of 0.8. According to the spectrum in Figure 6, the highest amplitude represent 26-second period, and the closest value to the expectation is 31-seconds and 52-seconds period. This small deviation is caused by the iGen's request scheduling operations.

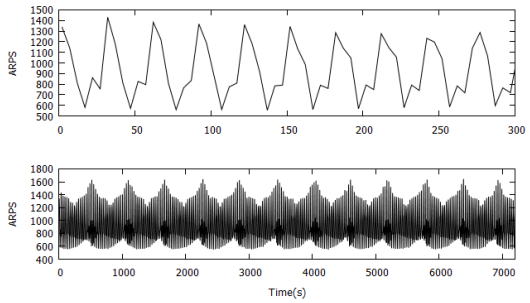


Figure 4. Periodic request arrivals, with a configuration of 30-seconds period.

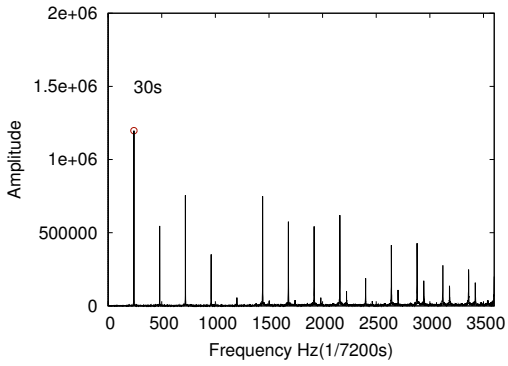


Figure 5. The spectrum of arrivals events. The 30-second period is configured and the highest power spectrum amplitude is located at 30-second.

B. IAT Generation Model

The ARPS is configured to follow a log-normal distribution with mean 6.5 and standard deviation 0.8, while IAT's shape parameter is set to 5. Note that the IAT distribution is not static. That is to say, the Pareto distribution's scale parameter is changing over time. This feature is also confirmed in [6]. Therefore, the probability density decreases as its value increases, forming a long tail behavior. The

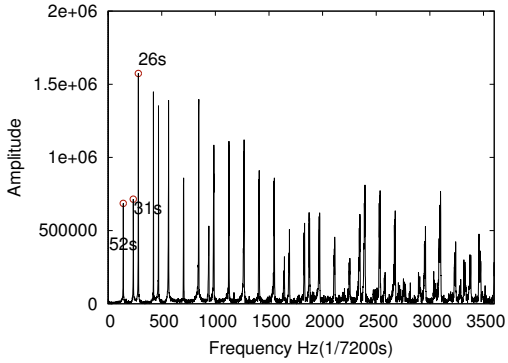


Figure 6. The spectrum of arrivals events. Two periods of 30-second and 50-second are configured.

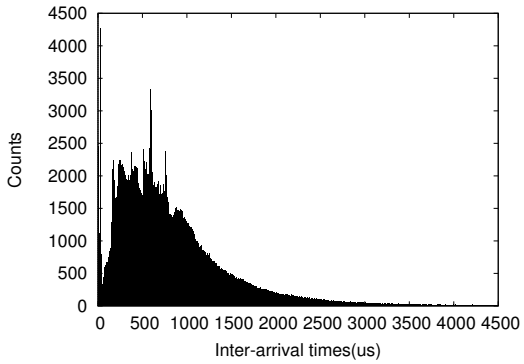


Figure 7. The shape parameter of inter-arrival time distribution is 5. The probability density function is formed by multiple Pareto distributions with different scale parameters.

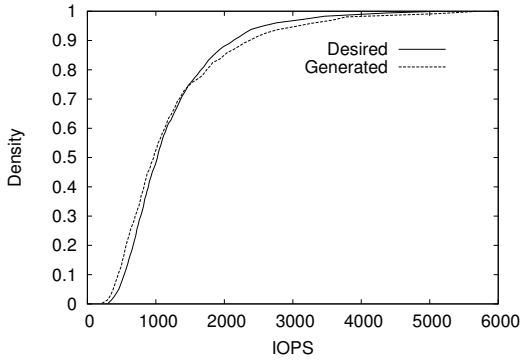


Figure 8. The arrival rates follows a log-normal distribution with the mean value of 6.5 and the standard deviation value of 0.8. The cumulative distribution function of the generated arrival rate fits well with the desired rates.

probability density function in Figure 7 is not smooth, but with several fluctuations. This is caused by overlap of many IATs following different scale parameters. There are several waves in the range of between 0 and 100. This can be explained by that ARPS with a log-normal distribution has burstiness, generating a few high rates at some time.

This phenomenon is also shown in the cumulative distribution function of ARPS in Figure 8. About one percent of ARPS is higher than 5,000 which determines the significant difference. In the plot, the generated ARPS fits well with the desired values. Figure 8 demonstrates that the generation model adopted by the iGen meet the requirements for the workload simulation. Based on the observations in [6], the arrival rates follows a log-normal distribution. Here we set the distribution parameters with the mean value of 6.5 and the standard deviation value of 0.8. As shown in Figure 8, the cumulative distribution function of the generated arrival rate fits well with the desired rates.

C. Flexibility of iGen

In the iGen, five probability distributions are provided, including *log-normal* distribution, *normal* distribution, *Poisson* distribution, *uniform* distribution, empirical distribution. To enhance the flexibility, the iGen supports to import historical workload trace and replays the trace using one of these distributions. In addition, the iGen allows users to easily add new probability distributions as plugins. In this experiment,

Table III
ACCURACY OF SIMULATING PROBABILITY DISTRIBUTIONS.

Distributions	Parameters	P value
Log-normal	mean is 8.05, deviation is 2.58	0.3344
Normal	mean is 5000, deviation is 2	0.9571
Poisson	λ is 10000	0.5145
Uniform	interval is [5000, 10000]	0.8621
Empirical	trace from Pangu file system	0.3470

K-S test is used to measure the accuracy of simulating these probability distributions. K-S test is a kind of hypothesis testing. The parameter settings of log-normal distribution and empirical distribution in Table III are derived from the Pangu. While, the parameters of the other three distributions are selected randomly. It is notable that the parameters of distributions do not affect the result of K-S test. The significance level P is 0.05. The results listed in Table III show that the simulation of each distribution is passed using K-S test, which proves the accuracy of distribution simulation.

VI. CONCLUSION

In this paper, we developed a framework iGen which model the requests with three statistics properties, request arrival rate, inter-arrival time distribution and request periodicity. We use dynamic Pareto distribution functions to generate IAT. We evaluated the efficiency and effectiveness of. In addition, we evaluated the characteristics of I/O requests the iGen generated, which is consistent with our expectations. We believe the iGen is helpful to deploy a realistic file system benchmarking for cloud infrastructures.

ACKNOWLEDGMENT

This work was partially completed during Biao Xu's internship at Alibaba. We would like to thank Pangu team at Alibaba. Zujie Ren is supported by NSF of China (No. 61300033). Weisong Shi is in part supported by the Introduction of Innovative R&D team program of Guangdong Province (No. 201001D0104726115) and Hangzhou Dianzi University. Zheng Ye is supported by NSF of China (No.61300117).

REFERENCES

[1] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *OSDI*, 2006, pp. 307–320.

[2] Glusterfs, http://www.gluster.org/.

[3] E. Stefanov, M. van Dijk, A. Juels, and A. Oprea, "Iris: A scalable cloud file system with efficient integrity checks," in *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 2012, pp. 229–238.

[4] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop YARN: Yet another resource negotiator," in *SoCC*. ACM, 2013.

[5] A. Traeger, E. Zadok, N. Joukov, and C. P. Wright, "A nine year study of file system and storage benchmarking," *ACM Transactions on Storage (TOS)*, vol. 4, no. 2, p. 5, 2008.

[6] Z. Ren, W. Shi, and J. Wan, "Towards realistic benchmarking for cloud file systems: Early experiences," in *IISWC*. IEEE, 2014, pp. 88–98.

[7] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *SoCC*, 2012, p. 7.

[8] J. Katcher, "Postmark: A new file system benchmark," Technical Report TR3022, Network Appliance, 1997. www. netapp. com/tech_library/3022. html, Tech. Rep., 1997.

[9] SPC, http://www.storageperformance.org/home/.

[10] D. G. Feitelson, *Workload modeling for computer systems performance evaluation*. Cambridge University Press, 2015.

[11] M. Silva, M. R. Hines, D. Gallo, Q. Liu, K. D. Ryu, and D. Da Silva, "Cloudbench: experiment automation for cloud environments," in *IC2E*, 2013, pp. 302–311.

[12] M. C. Schatz, "Cloudburst: highly sensitive read mapping with mapreduce," *Bioinformatics*, vol. 25, no. 11, pp. 1363–1369, 2009.

[13] GridMix2, https://hadoop.apache.org/docs/r1.2.1/gridmix.html.

[14] SWIM Project, https://github.com/SWIMProjectUCB/SWIM/wiki.

[15] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang *et al.*, "Bigdatabench: A big data benchmark suite from internet services," in *HPCA*. IEEE, 2014, pp. 488–499.

[16] R. Han, S. Zhan, C. Shao, J. Wang, J. Xu, L. K. John, L. Wang, and J. Zhan, "BigDataBench-MT: A Benchmark Tool for Generating Realistic Mixed Data Center Workloads," *arXiv preprint arXiv:1504.02205*, 2015.

[17] D.-C. Juan, L. Li, H.-K. Peng, D. Marculescu, and C. Faloutsos, "Beyond poisson: modeling inter-arrival time of requests in a datacenter," in *Advances in knowledge discovery and data mining*. Springer, 2014, pp. 198–209.

[18] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: a study of emerging scale-out workloads on modern hardware," in *ACM SIGPLAN Notices*, vol. 47, no. 4. ACM, 2012, pp. 37–48.

[19] Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo, "Characterizing data analysis workloads in data centers," in *IISWC*. IEEE, 2013, pp. 66–76.

[20] Rain Project, https://github.com/jacksonicson/rain.

[21] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, A. Fox, and D. Patterson, "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0," in *Proc. of CCA*, vol. 8, 2008.

[22] Q. Zheng, H. Chen, Y. Wang, J. Duan, and Z. Huang, "Cosbench: A benchmark tool for cloud object storage services," in *CLOUD*. IEEE, 2012, pp. 998–999.

[23] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *ACM Symposium on Cloud computing*. ACM, 2010, pp. 143–154.

[24] S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. López, G. Gibson, A. Fuchs, and B. Rinaldi, "YCSB++: benchmarking and performance debugging advanced features in scalable table stores," in *SOCC*. ACM, 2011, p. 9.

[25] R. Pitchumani, S. Frank, and E. L. Miller, "Realistic request arrival generation in storage benchmarks," in *Mass Storage Systems and Technologies (MSST), 2015 31st Symposium on*. IEEE, 2015, pp. 1–10.

[26] C. L. Abad, "Big data storage workload characterization, modeling and synthetic generation," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2014.

[27] C. L. Abad, M. Yuan, C. X. Cai, Y. Lu, N. Roberts, and R. H. Campbell, "Generating request streams on Big Data using clustered renewal processes," *Performance Evaluation*, vol. 70, no. 10, pp. 704–719, 2013.

[28] C. Delimitrou, S. Sankar, K. Vaid, and C. Kozyrakis, "Accurate modeling and generation of storage i/o for datacenter workloads," *EXERT*, 2011.

[29] R. McDougall, J. Crase, and S. Debnath, "Filebench: File system microbenchmarks," 2006.

[30] N. Agrawal, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Towards realistic file-system benchmarks with codemri," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 2, pp. 52–57, 2008.

[31] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *SoCC*, 2010, pp. 143–154.

[32] Y. Shi, X. Meng, J. Zhao, X. Hu, B. Liu, and H. Wang, "Benchmarking cloud-based data management systems," in *CloudDB*. ACM, 2010, pp. 47–54.

[33] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis," in *ICDEW*. IEEE, 2010, pp. 41–51.

[34] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," in *ISCA*. IEEE, 2015, pp. 158–169.

[35] D. Meisner, J. Wu, and T. F. Wenisch, "BigHouse: A simulation infrastructure for data center systems," in *ISPASS*, 2012, pp. 35–45.

[36] C. Abad, N. Roberts, Y. Lu, and R. Campbell, "A storage-centric analysis of mapreduce workloads: File popularity, temporal locality and arrival patterns," in *IISWC*, 2012, pp. 100–109.

[37] R. Gracia-Tinedo, D. Harnik, D. Naor, D. Sotnikov, S. Toledo, and A. Zuck, "SDGen: mimicking datasets for content generation in storage benchmarks," in *FAST*, 2015, pp. 317–330.

[38] J. R. Douceur and W. J. Bolosky, "A large-scale study of file-system contents," *ACM SIGMETRICS Performance Evaluation Review*, vol. 27, no. 1, pp. 59–70, 1999.

[39] B. Schroeder, A. Wierman, and M. Harchol-Balter, "Open versus closed: A cautionary tale," in *NSDI*, 2006, pp. 18–18.