Cloning IO Intensive Workloads Using Synthetic Benchmark

Dheeraj Chahal Manoj Nambiar

{d.chahal|m.nambiar}@tcs.com
TCS Innovation Labs
Mumbai, India

ABSTRACT

Performance evaluation of an enterprise application on multiple storage systems of interest called target systems, is a time consuming and costly process. Moreover, it is increasingly challenging to predict the performance at higher concurrencies (no. of users) on target systems when the application is migrated from the low performance source system where the application is currently deployed.

In this paper, we present a methodology to generate equivalent synthetic benchmark workloads for IO intensive applications. These synthetic benchmarks are easy to deploy and recreate the application behavior faithfully. We essentially extract the temporal and spatial characteristics of the application workload traces on the source system and replay those characteristics using synthetic benchmark on the target system. Also, we extrapolate these characteristics to predict the performance at higher concurrencies using synthetic benchmark without generating traces for those concurrencies.

To verify the efficacy of our methodology, we have tested our approach successfully using two different types of workloads namely TPCC and JPetStore. We present our extrapolation results on the target storage system with an error bound of less than 20% for concurrencies up to an order of the magnitude of the source system.

Keywords

Performance prediction; IO traces; extrapolation

1. INTRODUCTION

When the resource utilization or latency of an IO intensive application increases, enterprises are interested in migrating their data from a system using slow performing storage devices (e.g. low-end HDDs) to a system with low latency devices like high-end hard disk drives (HDDs) or solid-state-drives (SSDs). SSDs are becoming commonplace and practitioner in academia and industry are giving more emphasis on their performance studies [6] as SSDs offer dual advantage of better performance and energy efficiency.

Unfortunately, studying the performance of an application with multiple types of storage devices and varying number of cores is a time consuming and daunting task. It either requires deploying the application and migrating the data to each target system or a priori knowledge of the system with different concurrency levels (no. of users). IO trace replay is one technique that can be used to reproduce the application characteristics without deploying the application on the target system. Trace collection at large workload results in time dilation and replaying such traces results in incorrect performance estimation. Hence, trace replay based techniques are not advisable for large workloads.

Synthetic benchmarks like IOZone [3], IOmeter [8] overcome many drawbacks of trace and replay when configured appropriately for representing the application workloads.

Our scheme presented in this paper allows system administrators to experiment easily with multiple systems with minimal efforts. We capture IO traces of the application of interest at low concurrencies and extract important properties. We feed these properties to synthetic benchmark and play them on the target system and thus accurately creating the same behavior of the application. Also, we use statistical methods to extrapolate these features to predict the performance at higher concurrency levels without generating the traces at those concurrency levels.

This approach does not require deploying real database on the target system because performance of the system is dependent on the access pattern instead of the actual data. We are recreating the same access pattern using only replica of database files of the same size as in the real database. Also, our approach is independent of operating systems or schedulers as we capture application characteristics in the traces which does not change across operating systems or schedulers.

The objective of this research work is two-fold:

1) Generate representative workloads for IO intensive enterprise applications using synthetic benchmarks that can be used with ease across multiple platforms with different storage systems.

2) Extrapolate features of the application that represent the workload of the application at higher concurrency levels.

We make an assumption that there are no software bottlenecks present in the application and bottlenecks are resulted only due to hardware resources at higher concurrencies.

The rest of the paper is structured as follows: Section 2 describes related work and state of the art. In section 3, we

ICPE'17, April 22-26, 2017, L'Aquila, Italy

DOI: http://dx.doi.org/10.1145/3030207.3030238

discuss our approach and implementation in detail. We discuss experimental setup in section 4. Results are discussed in section 5 followed by conclusion in section 6.

2. RELATED WORK

Trace and replay is a well known technique used for debugging and profiling applications. There is a large body of work available for trace replay based mechanisms for storage system evaluation[5][10]. Tak et al. developed a trace based technique called PseudoApp[9] to estimate the performance and resource consumption of the application. ROOT[11] is similar work in this area to capture the traces at each resource and replay them in the same order on the target system. These methods either replay traces as it is on the potential target systems or develop representative workloads using traces without extracting application features.
In another work, Carrington et al. [4] used statistical methods for extrapolating the trace characteristic of the application for inferring the large scale computation behavior of an application. While this work is applied to compute intensive applications, our research is focused on IO intensive applications where application features differ from compute-intensive applications.
We extend the recent work of Carrington et al. for IO intensive applications by introducing novel method for extrapolating the application features and predict the scaling behavior. To the best of our knowledge there is no work available for extrapolating the IO workloads using application features from IO traces.

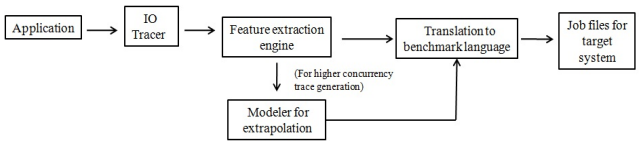


Figure 1: Framework

3. OUR APPROACH

The framework we developed is as shown in Figure 1. Our approach requires running the application workload at different levels of concurrencies and recording the IO traces at the system call level of the database server. All significant features are extracted from the traces by the feature extraction engine. These features are extrapolated by modeler using the statistical models. Extrapolated feature values are translated into job file for synthetic benchmarks. Synthetic benchmark is run with the extrapolated values to study the performance of the new device at larger concurrencies. Detailed description of each component is as below:

3.1 IO Trace Recording

Two popular tools for tracing I/O activity of an application are *strace* and *blktrace*. *strace* records system calls of the application and blktrace records I/O activity of the device. We choose *strace* as it is a better representation of application behavior which can be easily reproduced unlike blktrace which captures kernel activity related to a device. To capture the trace at database server we first track all the threads spawned by MySQL and then attach *strace* to each of those threads. The *strace* induces some overhead, which increases with an increase in workload due to the fact that *strace* pauses application twice for each system call, i.e. when a system call begins and when system call is stopped. We capture only IO related system calls to mitigate the strace overhead and size of the trace file as well. Moreover, we collect traces at low concurrencies only and extrapolate for higher. The IO calls that we capture are *read()*, *write()*, *pread()*, *pwrite()*, *lseek()*, *fsync()*, *open()*, *close()*. Our trace file contains system calls with thread ids, timestamp value, size of data read or written, offset address and system call execution time.

3.2 Feature Extraction

Feature extraction is the most important step in this work. The challenge lies in selecting all the important features that represent the temporal and spatial characteristics of an application and then extracting judiciously from the traces. We observe that there is a *n:m* relationship between threads and database files, i.e. a thread can access multiple files in the database and multiple threads can access the same file as well. In a worst case scenario, a thread can access all m files in the database. Hence, for an application running n threads would require $n*m$ jobs in the benchmark. We overcome this limitation by grouping threads in the trace file based on file descriptor (FD) such that for each FD we create two trace files,one containing read IO threads and other containing write IO operations threads. All features are extracted from read and write files of each FD. There are two benchmark jobs created (read and write ops) for each FD representing all features corresponding to that FD. Since all database files are not opened in RW mode i.e. some are read only and others write only, hence total number of benchmark jobs created is equal to or less than twice the number of unique FDs (Figure 2).

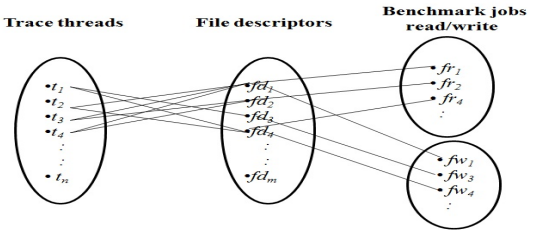


Figure 2: Thread (tn), file descriptor(fd) and benchmark jobs mapping

The important features we captured that represent the application characteristics and also affect the device performance significantly are as follows [7]:
Randomness:The performance of a storage device is significantly affected by the retrieval and the storage patterns. The sequential read and write operations are performed faster as compared to random read and write operations.
Thinktime: Time duration for which a job is stalled after an IO has completed and before next IO operation is issued.
Blocksize: The distribution of chunk sizes to issue IO.
IOPS: Total number of IO (read and write) operations performed per second.

3.3 Feature Extrapolation

In this work we propose use of a statistical method based approach to predict the performance of the application at

No.of Users	iops(writes/s)							
	customer.ibd (fd1)	orders.ibd (fd2)	stock.ibd (fd3)	order_line.ibd (fd4)	ibdata1.ibd (fd4)	ib_logfile0(fd5)	.....	10.ibd
10	1.67	2.2	8.1	2.6	1.7	2	.....	1.8
15	2.5	3.1	12.14	3.3	1.6	3.5	.....	2.5
20	3.2	4.1	16	4.7	2.2	5.4	.....	3.3
.....	.....	.....	.....	.....	.....	.....	.....	.....
200(extrapolated)	**30.70 (linear)**	**31.9(power)**	**154(power)**	**42.3(linear)**	**4.4(linear)**	**64.7(power)**	**.....**	**24.23(power)**

Table 1: Extrapolation of TPCC feature iops (writes/s) for each file descriptor

higher concurrency on the target system by extrapolating the application features.
Each feature might exhibit a different scaling behavior. Some features scale linearly as the workload or concurrency increases while others might scale differently (e.g. power or logarithmic). Various canonical functions can be fitted in to different features [4]. We used four canonical forms in this study namely linear, logarithmic, power and exponential. In order to decide the right statistical method for extrapolating each feature for each FD, we take traces at three different workloads and extract all features corresponding to each FD. We fit all four statistical methods in each feature for all FDs and then choose the one that has largest coefficient of determination (R^2) value. In Table 1 we show extrapolation of one application feature (iops). Using the feature data for 10,15,20 user workloads we extrapolate the write iops for 200 users for each FD. Likewise, we can extrapolate read iops for each FD. Other features described in section 3.2 are also extrapolated from both read and write trace files for required concurrencies.

3.4 Trace Feature Replay

In addition to defining all the features that are mentioned in the section 3.2, we also define some additional application and system configuration specific parameters as below:
ioengine: Defines how IO is delivered to the kernel.
buffered: Set for buffered IO.
fsync: Set to sync dirty data when writing to a file.
time_based: Time based criterion for a benchmark run.
runtime: Duration of the job run.
filesize: File size for each job to perform r/w ops.
filename: Name of the database file.
Benchmark jobfile is prepared by defining all the parameters or features defined above and in section 3.2 for each FD. This file can be migrated to any storage system where benchmark is installed and run with ease.

4. EXPERIMENTAL SETUP

We have validated our approach using a well know TPC-C benchmark and web based application JPetStore [1]. TPC-C is a benchmark for comparing online transaction processing (OLTP) performance on various software and hardware configuration. IO intensive TPC-C has a complex database and consists of five concurrent complex instructions. JPetStore is an eCommerce J2EE application benchmark which emulates an online pet store. For replaying the application characteristics on target systems we use flexible IO (FIO) [2] tester synthetic benchmark due to flexibilty it provides for detailed workload setup. MySQL is used as a back-end for both the benchmarks. The storage systems used in this study are HDD and SSD (Table 2). IO traces are captured at multiple concurrency levels on the database server for 20 minutes including 3 minutes warm-up time.

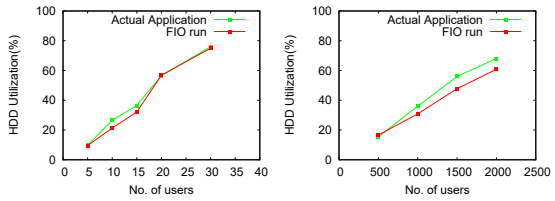


Figure 3: FIO vs application run on HDD for (a) TPCC and (b) JPetStore

5. RESULTS

We performed tests to validate our thesis for replicating the application behavior using synthetic benchmark FIO and also extrapolating the features for higher concurrency levels. The resemblance in storage device utilization is used as an evaluation criterion.

5.1 Feature Replay on HDD

Our first objective is to generate representative workloads for IO intensive applications. We show that using application feature with the FIO synthetic benchmark, we are able to recreate the application behavior. Results for two applications that we used are as below:

5.1.1 TPCC on HDD

We run TPCC at multiple concurrency levels 5, 10, 15, 20 and 30. Trace files are captured for all these concurrencies. We also record the disk utilization for each of these concurrencies when strace is not running. These trace files are supplied to feature extraction component. The output of feature extraction component is a jobfile for FIO. We run FIO with each of these files on the same storage device and record resource utilization. As shown in Figure 3a, the FIO run results in resource utilization comparable to actual application run for most of the concurrencies.

5.1.2 JPetStore on HDD

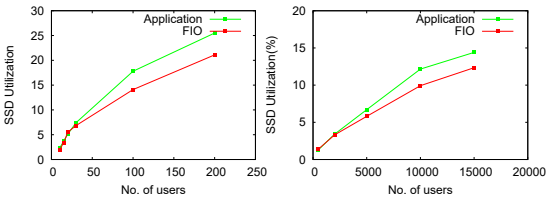
Similar behavior is seen with the JPetStore application as well. We collect traces for 500, 1K, 1500 and 2K users. After extracting features from these traces, we feed in FIO jobfile and run for same duration (Figure 3b). We see some difference in the device utilization only at higher concurrency i.e. 1500 and 2K. This can be attributed to the *strace* overhead while collecting traces as discussed earlier.

5.2 HDD to SSD Migration and Extrapolation

Another objective of this work is to predict the performance across platforms and extrapolate for higher concurrency levels. We run our applications on HDD first and replay the features using FIO on SSD. Since SSD shows significant improvement in resource utilization over HDD, we

Storage type	Disk Model	RPM	No. of Disks	IO Schedular	File System	Interface	System Config	Linux Kernel
High-end HDD	HP	10000	1	CGQ	ext4	Dual Port,SAS 6GB/s	16 Core Xeon CPU @ 2.4 GHz,12MB L2 cache	CentOS 6.6,2.6.32
SSD	Virident Systems Inc. FlashMAX Drive Micron-slc-32	-	1PCIe	Default	ext3	-	16 Core Xeon CPU @ 2.4 GHz,,12MB L2 cache	CentOS 6.6,2.6.32

Table 2: Storage systems used in our study



(a) **(b)**

Figure 4: FIO vs application run on SSD for (a) TPCC and (b) JPetStore

should be able to extrapolate for larger concurrencies.

5.2.1 TPCC on SSD

We take traces of TPCC application for three concurrency levels i.e. 10, 15, 20 on HDD. Features are extracted and extrapolated for 30, 100 and 200 and run with FIO on SSD. For concurrencies 10,15,20 (no extrapolation), we see same resource utilization by FIO and actual application (Figure 4a). There is some deviation from the actual resource utilization due to strace overhead amplification but within 20% error bound at large concurrencies.

5.2.2 JPetStore on SSD

For JPetStore application we collect IO traces for 1K,1500 and 2K users on HDD. All significant features are extrapolated for higher concurrencies 5K,10K and 15K. These features are replayed with FIO on SSD and results are as shown in Figure 4b. We observe a small difference in resource utilization only at large concurrencies due to strace overhead amplification.

6. CONCLUSION AND FUTURE WORK

We have presented a method that can be used for generating representative workloads for IO intensive applications by extracting features from IO traces. We also proposed a methodology for extrapolating the application features using canonical functions. Using synthetic benchmark FIO, we have faithfully replayed the representative workload of two applications on HDD. Also, we have successfully tested the application behavior on SSD for higher concurrencies using extrapolated features.

We have seen that the trace closely matches the application utilization behavior w.r.t concurrency. This means that the trace can be trusted to emulate the real application I/O behavior. The trace can then be used on any server with a different storage device without having to install the application or generate concurrent user workload. The utilization generated by the trace at a certain concurrency level can be divided by the throughput to derive the service demand of the application on the new storage device. Also, throughput(IOPS) can be easily converted to application transactions per second metric by comparing the application performance and trace data at the source system. This a can be fed along with other resource demands into an analytical or simulation model to predict performance of the application on the target system with a different storage device than the source while other things being the same. We intend to validate with experiments few more accuracy metrics (e.g. response time) using resource utilization and IOPS data.

7. REFERENCES

[1] iBATIS JPetStore. http://sourceforge.net/projects/ibatisjpetstore/.

[2] J. Axboe. FIO-Flexible I/O Tester. http://freshmeat.net/projects/fio/.

[3] D. Capps. Iozone file system benchmark. www.iozone.org.

[4] L. Carrington, M. A. Laurenzano, and A. Tiwari. Inferring large-scale computation behavior via trace extrapolation. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 1667–1674. IEEE, 2013.

[5] C. Delimitrou, S. Sankar, K. Vaid, and C. Kozyrakis. Accurate modeling and generation of storage i/o for datacenter workloads. In *Proceedings of the 2nd Workshop on Exascale Evaluation and Research Techniques, EXERT, Newport Beach, CA (March 2011)*, 2011.

[6] I. Jo, D.-H. Bae, A. S. Yoon, J.-U. Kang, S. Cho, D. D. G. Lee, and J. Jeong. Yoursql: A high-performance database system leveraging in-storage computing. *Proc. VLDB Endow.*, 9(12):924–935, Aug. 2016.

[7] Q. Noorshams, R. Reeb, A. Rentschler, S. Kounev, and R. Reussner. Enriching software architecture models with statistical models for performance prediction in modern storage environments. In *Proceedings of the 17th international ACM Sigsoft symposium on Component-based software engineering*, pages 45–54. ACM, 2014.

[8] OSDL. Iometer project. www.iometer.org.

[9] B. C. Tak, C. Tang, H. Huang, and L. Wang. Pseudoapp: Performance prediction for application migration to cloud. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 303–310, May 2013.

[10] M. Tarihi, H. Asadi, and H. Sarbazi-Azad. Diskaccel: Accelerating disk-based experiments by representative sampling. In *ACM SIGMETRICS Performance Evaluation Review*, volume 43, pages 297–308. ACM, 2015.

[11] Z. Weiss, T. Harter, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Root: Replaying multithreaded traces with resource-oriented ordering. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 373–387, New York, NY, USA, 2013. ACM.