Load-Aware Replay of I/O Traces

Sankaran Sivathanu[1,*], Jinpyo Kim[2], Devaki Kulkarni[2], and Ling Liu[1]

[1]College of Computing, Georgia Institute of Technology
[2]VMware Inc., Palo Alto, CA
[*]Student author

Abstract

Trace replay is one of the commonly used techniques in benchmarking I/O systems. However it is still not obvious if trace replay faithfully preserves important characteristics of the original application. While this may not be a significant problem when traces are collected and replayed in a same or similar system environment, many issues arise when traces are replayed in a different kind of environment than the one used during the trace collection process.

In this work-in-progress, we present a novel mode of I/O trace replay called *Load-based replay* that aims to preserve the same I/O load pattern of original application traces irrespective of the system it is replayed.

1 Introduction

Benchmarking of I/O systems by subjecting them to realistic workloads is a common practise in the storage community and it is vital for bringing further improvements to the system. The most significant issue in benchmarking is maintaining realism with respect to the actual workload. I/O trace replay is one of the commonly used techniques for storage benchmarking. It strikes a good balance between realism with respect to actual application workload and usability.

Many issues need to be addressed when building an I/O trace replayer even if the trace collection environment and the trace replay environment are the same. For example, if trace replay is implemented in the file system layer, aging the file system is one issue. However in most cases, traces are collected in different setups and are maintained in centralized repositories like the SNIA IOTTA from where the users pick and use them in their own testing environment. This mismatch in I/O system environment between trace collection and replay introduces more issues that need to be carefully accounted for.

When traces are replayed in a different environment, they may no longer be representative of the real application because most applications tunes itself according to the I/O system in which it runs. For example, if a trace was collected in a high-performance storage system, and is replayed in a low-performance storage system, it may not be able to keep up with the request issue pace in the trace, thus leading to massive queueing and entirely different timing pattern when compared to running the application itself in the low-performance storage system. Traditionally this problem has been tackled by either accelerating the trace or decelerating the trace during replay based on the system capabilities. While this may be better than replaying the trace without any modifications at all, it may still not represent the behavior of the actual application as shown by our experiments. Also, the degree of acceleration or deceleration of traces during replay is still done in an ad-hoc manner without following any heuristics.

In this work-in-progress, we introduce a new mode of I/O trace replay called *Load-based replay*, where traces are replayed by adaptively varying the pace of request issues by matching the load profile of the original trace in the replay system.

2 Load-based Replay

Load-profile of the application on the I/O system in which traces are collected is often over-looked when they are replayed. This leads to inaccurate evaluation of target I/O system (including hardware and software at host side) because the load profile of the original application in a new system environment may be totally different. Experiments concerning load-balancing systems is a good example for this mismatch. Evaluation of a system for a particular application may be inaccurate because trace replay in the target I/O system may exhibit totally different load characteristics because of differences in system capabilities.

We capture a metric called 'No. of outstanding I/O requests' from the original trace using the start and end

times of each request to represent load-profile of a workload. We then dynamically adapt our replay algorithm to issue requests in a pace that matches original load profile of the trace. This way, when an I/O system is tested for a particular load behavior of an application, irrespective of how fast the I/O system services requests, overall number of outstanding I/Os remain the same.

The trace replay process starts in steady state – both original trace and the replay process starts with 0 outstanding I/Os. Subsequently, before issuing each request, the required outstanding I/O value is compared with the run-time value in the system. If the required number is less than the current number of outstanding I/Os in the system, the replayer waits till the number drops to the required value, which will happen as and when the requests get serviced by the system. As soon as the current value matches the required value, the next request is issued. As we start the trace replay in steady state there will rarely be a case when the number of I/Os outstanding in the system is less than the required number that was captured from the trace. It becomes tricky when the required number of outstanding I/O matches the current number of I/Os outstanding in the system. In this case, if original request issue timing from the trace is preserved, the number of outstanding I/Os in the system may drop before the next request issue. For e.g., consider a request R_1 that is issued at time t_1, and request R_2 issued at time t_2 with a required outstanding I/O count of 2. After the request R_1 is issued, the replayer checks the current number of outstanding I/Os in the system and if it is 2, it either issues the request R_2 immediately, in which case the load profile would match and the timing information is completely overlooked. Another alternative is to wait for time t_2 and then issue the request, in which case the number of outstanding I/Os in the system may drop to 1 or 0.

3 Evaluation

We evaluated our load-based replay accuracy using a Swingbench DSS workload trace which is a 'sales order' benchmark backed by Oracle 11gR1 database on RHEL 5.1 64-bit version. Figure 1 shows the distribution of outstanding I/Os for the original trace and for a variety of replay modes. The AS-IS mode tries to replay the trace by preserving original request issue timings without considering the load pattern. As the replay system has a poorly performing storage when compared to the trace collection system, requests get queued up because of increased service latencies of the storage system and the outstanding I/O is far more when compared to the original trace. We also followed the traditional approach of decelerating the trace replay by a factor of 2 and 4 which are also plotted in the figure. Interestingly, even slowing-down the trace by a factor of 2 lead to more outstanding I/Os during replay than the original trace, and when slowing-down by a factor of 4, makes the replay utilize the device far lesser than the original trace. Also, the load pattern of the AS-IS and the slowed-down version of replay is entirely different than the original. In the case of load-based replay, the distribution curve matches the original trace pretty closely. This is because rather than using a constant slow-down factor, our load-based replayer dynamically tunes itself so that it matches the load pattern of the original workload.

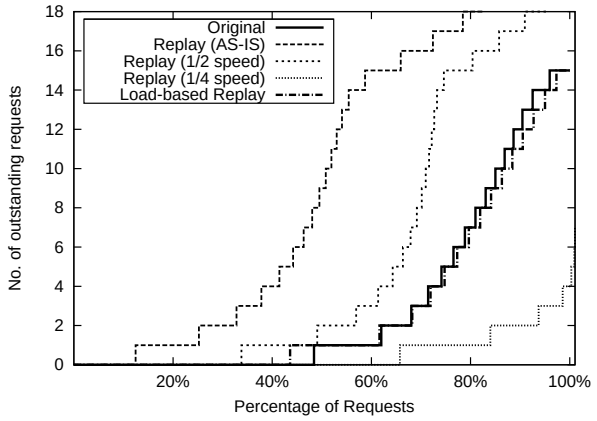

Figure 1: Outstanding I/O distribution for Swingbench DSS workload

4 Conclusion

We have introduced a new mode of I/O trace replay called the load-based replay with an overall goal of matching the load profile of original application/system with that of replay system. However, number of outstanding requests is just one metric to represent the load profile of a system and there may be other criteria that define load characteristics of a workload in a system. Also, moving towards the overall goal of making trace replay more representative of real applications, it is pertinent to study about adaptively matching more workload characteristics like accurate timing, access patterns, etc., along with load pattern of workloads, which we plan to work in future.