Microbenchmark Noise Reduction

Hai Huang

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation
Huang, Hai, "Microbenchmark Noise Reduction", Technical Disclosure Commons, (September 30, 2020)
https://www.tdcommons.org/dpubs_series/3648

This work is licensed under a Creative Commons Attribution 4.0 License.
This Article is brought to you for free and open access by Technical Disclosure Commons. It has been accepted for inclusion in Defensive Publications Series by an authorized administrator of Technical Disclosure Commons.
Microbenchmark Noise Reduction

ABSTRACT

Microbenchmarking is widely used to track performance regressions between versions of a software method, application programming interface (API), or component, and to detect and to prevent changes that negatively affect the performance or scalability. Microbenchmarking is vulnerable to run-to-run variance whereby multiple executions of a microbenchmark lead to different performance results. While larger numbers of iterations can be used to reduce variance, this imposes higher execution costs and can delay the release of software to production environments.

This disclosure describes a novel approach to execute microbenchmarks that utilizes the observation that a microbenchmark test usually includes multiple APIs. In the proposed approach, the iterations of each API are divided into multiple chunks. Each chunk is executed in one-shot. The execution of the chunks from one API is interleaved with that of chunks from one or more other APIs. The approach mimics real production environments, and can enable a user to provide input to control the number and order of chunks, e.g., based on production or synthetic data, and can produce multiple sample points for each API tested. The interleaving approach can reduce the variance in test results without increasing the execution cost.

KEYWORDS

- Microbenchmark
- Software testing
- Performance testing
- API testing
- Run-to-run variance
BACKGROUND

Load and performance testing is an important aspect of software development and testing. Microbenchmarking is a common approach to gauge the performance of a method, API, or a software component. For example, a microbenchmark result may be of the form “API method X took 324ms”. Microbenchmarking is widely used to track performance regressions between software versions and to detect and prevent changes that negatively affect the performance or scalability of a product.

Being a dynamic measurement, microbenchmarking is vulnerable to noise, or run-to-run variance. Even with identical software versions and testing environments, multiple executions of a microbenchmark can lead to different performance values. Such run-to-run variance is an important challenge since it affects reliable performance measurement and reduces the effectiveness of microbenchmarking as a way to track performance regressions.

One common approach to reduce variance is to run the method or API in many iterations and measure the average latency or throughput. Some microbenchmark frameworks implement mechanisms to dynamically determine the number of iterations such that the benchmarking can finish within a preset time limit. Some frameworks can also produce multiple sample points by repeating the benchmark N times. This increases the execution cost by N times as well.

While more iterations lead to reduced variances, this approach has two significant downsides. More iterations require a greater amount of time to run the test. This leads to an increase in the time required to validate changes in code, even such changes that do not significantly affect performance or scalability, thereby causing a delay before the code can be
released to production. Further, a greater number of iterations utilize more resources and therefore, this approach increases the hardware cost to validate the code change.

**DESCRIPTION**

This disclosure describes a novel approach to execute microbenchmarks that can reduce the variance in test results without increasing the execution cost. The approach utilizes the observation that a microbenchmark test usually includes multiple APIs. In the proposed approach, instead of executing multiple iterations for a single API in one-shot, the iterations are divided into multiple chunks. Each chunk is executed in one-shot. The execution of the chunks from one API is interleaved with that of chunks from one or more other APIs. The total number of iterations for each API can be kept the same and therefore, the proposed approach does not increase the execution cost for the microbenchmark. Fig. 1 provides a visual illustration of the approach.

![Fig. 1: Microbenchmark with interleaved execution of multiple APIs](image)

The number of iterations for each API in the microbenchmark, e.g., API A, API B, and API C in the example of Fig. 1, is determined. The number of iterations is broken into multiple chunks which are then interleaved with chunks of the other APIs, as seen in Fig. 1. The approach takes into account that the initial state of the process can affect the performance of an API. By providing an execution sequence that interleaves chunks of different APIs, the microbenchmark is executed with changes in the initial state of the process for each chunk.
To better understand this, consider cache warm-up, new objects created in memory, new threads started by the scheduler, etc. Each of these conditions can change the state of the process and affect the performance. If all the chunks from the same API are run in a continuous sequence, only the first (or the initial few) chunks may exercise state changes. With interleaving, as shown in Fig. 1, execution of each chunk likely experiences state changes since a chunk of a different API is executed in between consecutive chunks of the same API.

In Fig. 1, the execution of chunks of APIs A, B, and C is interleaved. For ease of illustration, the state is shown only chunks of API A. As can be seen, the state is different for each chunk of API A, e.g., since Chunk 1 executes at the beginning of the test, Chunk 2 executes after a chunk of API B, and Chunk 3 executes after a chunk of API C. While Fig. 1 shows three APIs, any number of APIs (two or more) can be included in the microbenchmark test and chunks from the different APIs can be interleaved in any order.

It is worth noting that in production environments, it is unlikely that a single API is executed repeatedly in many iterations. Instead, multiple related APIs are executed in coordination to carry out a use case or use scenario. Interleaving chunks as described herein provides benchmark execution that mimics real production environments.

The interleaving of chunks can be done in a random fashion. Such randomization helps mimic a real production environment in which different use cases or use scenarios can happen at the same time and corresponding APIs may be executed in various different orders. Further, users are provided with the ability to control various parameters of the microbenchmark execution. For example, the user can select the number of chunks to balance between, and for a single chunk, select the amount of time spent to exercise state changes and the amount of time to run iterations and measure performance.
Since all iterations in prior microbenchmark frameworks are executed in one-shot, such microbenchmark frameworks only produced a single data point per API. With the interleaving approach, the microbenchmark can be executed to capture a respective performance data point for each chunk of an API, thereby providing multiple data points per API with the same overall number of iterations. This enables users to run A/B t-Test or u-Test, both of which require multiple sample points from both A and B tests, and gauge the statistical significance of performance differences. The multiple data points also expose the intra-run variance which is useful information for users to understand the nature of an API, e.g., the sensitivity of a given API to state changes, and may reveal optimization opportunities.

The described approach is suitable for use in any microbenchmark framework. The approach of generating multiple chunks of execution for each API to be tested and that interleaves the chunks with tests for other APIs, provides several benefits such as:

- Reduction in the variance of microbenchmarking without increasing the execution cost
- Enable a heuristic approach to interleave multiple microbenchmarking tests using random distributions to further remove the variance.
- Control over the number of chunks to balance between, and within a single chunk, the amount of time spent to exercise state changes and to run iterations and measure performance.
- Generates multiple sample points (one sample point per chunk) for each API without increasing the execution cost. Such sample points also enable comparison across multiple chunks which can help identify APIs that are sensitive to state changes and can reveal optimization opportunities.
The use of interleaved chunks allows execution of the benchmark in a manner that mimics real production environments

- Interleaving provides reduction in the repetitiveness of iterations within a test due to state changes between chunks
- Interleaving enables running APIs in random orders, which mirrors real production environments
- Allows the use of production data and/or synthetic data regarding an API, e.g., frequency of use, possible order, parallelism, possible preconditions, etc. to guide execution of the benchmarks that mimics a known production environment, or to simulate a future production environment

CONCLUSION

This disclosure describes a novel approach to execute microbenchmarks that utilizes the observation that a microbenchmark test usually includes multiple APIs. In the proposed approach, the iterations of each API are divided into multiple chunks. Each chunk is executed in one-shot. The execution of the chunks from one API is interleaved with that of chunks from one or more other APIs. The approach mimics real production environments, and can enable a user to provide input to control the number and order of chunks, e.g., based on production or synthetic data, and can produce multiple sample points for each API tested. The interleaving approach can reduce the variance in test results without increasing the execution cost.

REFERENCES

1. https://github.com/google/benchmark