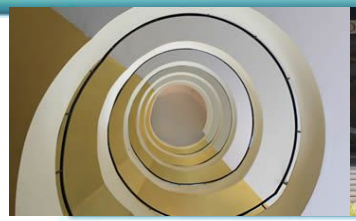
Mini-Project Approval

Project Title	:	Tailbench
Project Guide	:	Dr K.V. Subramaniam
Project Team	:	Supreeth G Kurpad - PES1UG19CS520 Abhishek Aditya BS - PES1UG19CS019 Vishal R - PES1UG19CS571

PES UNIVERSITY






Problem Statement

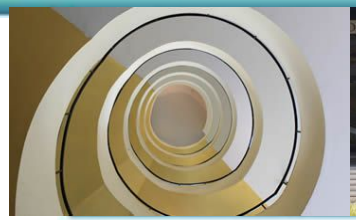
1. Latency-critical applications, common in data centers, must achieve small and predictable tail (e.g., 95th or 99th percentile) latencies. Their strict performance requirements limit utilization and efficiency in current data centers.

2. These problems have sparked research in hardware and software techniques that target tail latency. However, research in this area is hampered by the lack of a comprehensive suite of latency-critical benchmarks.

3. TailBench is a benchmark suite and evaluation methodology that makes latency-critical workloads easy to run and characterize as conventional, throughput-oriented ones. TailBench includes eight applications that span a wide range of latency requirements and domains, and a harness that implements a robust and statistically sound load-testing methodology.

4. Our goal is to find out the underlying causes that are responsible for tail latencies in data centers and construct benchmarks to profile the applications in terms of their latencies.

5. Once the profiling of the latiencies is done, the final goal is to minimize the tail latencies of the applications.






Literature Survey

- Kasture, Harshad, and Daniel Sanchez. "Tailbench: a benchmark suite and evaluation methodology for latency-critical applications." In *2016 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 1-10. IEEE, 2016.

  Link : https://ieeexplore.ieee.org/abstract/document/7581261

- Xiong, Xingwang, Lei Wang, Wanling Gao, Rui Ren, Ke Liu, Chen Zheng, Yu Wen, and Yi Liang. "DCMIX: generating mixed workloads for the cloud data center." In *International Symposium on Benchmarking, Measuring and Optimization*, pp. 105-117. Springer, Cham, 2018.

  Link : https://link.springer.com/chapter/10.1007/978-3-030-32813-9_10

- Kasture, Harshad, Xu Ji, Nosayba El-Sayed, Nathan Beckmann, Xiaosong Ma, and Daniel Sanchez. "POSTER: improving datacenter efficiency through partitioning-aware scheduling." In *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 134-135. IEEE, 2017.

  Link : https://ieeexplore.ieee.org/abstract/document/8091227

- K. -. Lee, H. -. Hon and R. Reddy, "An overview of the SPHINX speech recognition system," in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 1, pp. 35-45, Jan. 1990, doi: 10.1109/29.45616.

  Link : https://ieeexplore.ieee.org/abstract/document/45616

- Tu, Stephen, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. "Speedy transactions in multicore in-memory databases." In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pp. 18-32. 2013.

  Link : https://dl.acm.org/doi/abs/10.1145/2517349.2522713

Literature Survey (contd)

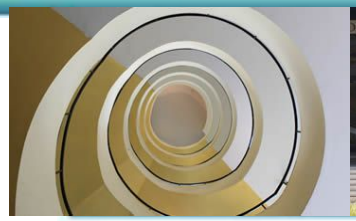
1. Tailbench is modular - Majority of the applications on Tailbench are independent of one another.

2. A common application “harness” provides a variety of load-testing scenarios in order to benchmark and validate the applications.

3. After evaluating the applications on different workloads, it was found that compared with the Service-Standalone mode (only executing the online service workload), 99th percentile latency of the service workload under the Mixed mode (workloads mix execution) increased 3.5 times, and the node resource utilization under that mode increased 10 times.

4. A description is given of SPHINX, a system that demonstrates the feasibility of accurate, large-vocabulary, speaker-independent, continuous speech recognition. SPHINX is based on discrete hidden Markov models(HMMs) with LPC (linear-predictive-coding) derived parameters.

5. Silo is a new in-memory database that achieves excellent performance and scalability on modern multicore machines. Silo was designed from the ground up to use system memory and caches efficiently. For instance, it avoids all centralized contention points, including that of centralized transaction ID assignment.






Proposed Solution

1. Tailbench provides 8 different applications to perform benchmarking on different types of Latency critical applications.
2. The goal is to minimise the 95th, 99th percentile latencies for all applications.
3. The first approach is to fully maximize the resource utilisation of the system and calculate the
    a. Tail latencies - P_{95}, P_{99}, Mean (μ), Max
    b. Calculate the P_{95}/μ, P_{99}/μ ratios
    c. Major and Minor page faults,
    d. Voluntary and involuntary context switches
    e. Resident set size of the process
    f. Cpu Utilisation
    g. Memory Utilisation
4. Go through the applications to check how they are integrated with “harness” - the application that writes to the latency files.
5. Analyze the variation of tail latencies on changing different parameters - Queries per second (QPS), Max request sent, Warmup request of the applications.
6. Plot the tail latency graphs for ServiceTimes(svc), QueueTimes, SojournTimes(End2End) for all the given applications.
7. Use call-graph tracer on the applications to analyse the amount of time the process spends in different functions to find the bottleneck functions which affected the latencies the most.

Technologies / Methodologies

Operating System: Ubuntu 18.04

System Configuration : 2 x 8 core Intel Xeon 2.3Ghz, 64 GB DDR4 RAM, 1TB HDD

Languages used by the 8 applications in Tailbench:

- C
- C++
- Java
- HTML
- Python

Tools:

- Bash
- Perf
- Ftrace
- Valgrind
- Taskset
- Htop
- Matplotlib
- Pandas
- Scipy

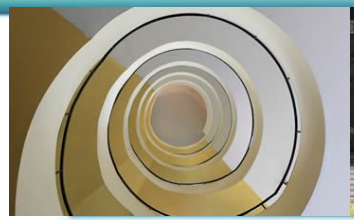
Project Timelines & Plan

What we have done so far:

- Compiled all 8 applications.
- Tabulated latency details for all applications.
- Made CPU and Memory Utilization Graphs.
- Analyzed if each application is CPU intensive/ Memory Intensive by observing the resident set size and the percentage CPU utilization for each of the 8 applications.
- Used perf and valgrind to generate function call graphs.

Next Steps:

- Drill down on Sphinx - A speech recognition application and create a benchmarking mechanism for it.
- Furthermore, profile Kaldi - An advanced speech recognition software - with the same metrics created for Sphinx and compare their performance and latencies under different workloads.






References

Link to GitHub Repository:
https://github.com/supreethkurpad/Tailbench

Progress Report 1:
https://github.com/supreethkurpad/Tailbench/blob/main/Tailbench-Report-1.pdf

Progress Report 2:
https://github.com/supreethkurpad/Tailbench/blob/main/Tailbench-Report-2.pdf

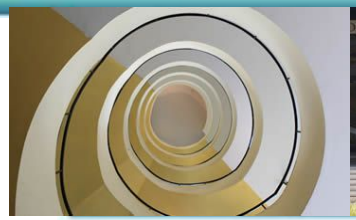
Contributions :

1. **Supreeth G Kurpad** : Analyzed how applications are integrated with Harness and how latency information propagates through all of them to generate the final latency information files. Checked if applications are memory intensive/ CPU intensive.

2. **Abhishek Aditya BS** : Ran all the applications with different input parameters, generated latencies and graphs of the tail latencies and summarised them in form of a table. Running the function graph-tracer using perf and valgrind on the applications to get a report of the most called functions in an application.

3. **Vishal R** : Compiled Tailbench. Created scripts for compiling tailbench. Setting process affinity, obtaining process runtime statistics like major page faults, context switches, resident set sizes, CPU and memory utilisation.

PES UNIVERSITY






THANK YOU