Yizheng Xie

yizheng_xie@brown.edu | 857-498-3205 | yizhengx.github.io | Providence, RI

Education

Brown University, Providence, RI, USA — Sept 2023 – Expected May 2028
Ph.D. in Computer Science (GPA: 4.0/4.0, Distributed System, Operating System, Cloud Computing)

Boston University, Boston, MA, USA — Sept 2021 – May 2023
M.S. in Computer Science (GPA: 3.9/4.0)

Chinese University of Hong Kong, Shenzhen, Shenzhen, Guangdong, China — Sept 2017 – May 2021
B.E. in Electronic Information Engineering

Working Experience

Meta Platforms, Menlo Park, CA — May 2022 – Aug 2022
Production Engineer Intern in Kernel Team (Python, Fedora Linux, Upstream Rust Packaging, Linux Userspace)

- Integrate Fedora and CentOS Stream forges and issue trackers (Pagure, GitLab, Bugzilla) into Meta Internal Task tool.
- Collaborate with Fedora engineers to fix and improve external tools like fixing Bugzilla authentication issues and adding Copr build integration in fedora-create-review tool.
- Implement a CLI tool to automate the process of updating Rust RPM packages by recursively updating dependencies and packaging missing dependencies, analyzing parallel chain-build order and checking compatibility for the current update set or any Bodhi rust update set. (Pagure Repository: pagure.io/fedora-rust/rust-update-set)
- Integration between Meta and Upstream aims to help Meta employees contribute to Fedora and Stream, rust packaging tool aims to increase efficiency and safety for updating a large set of Rust packages for the whole Fedora community.

Research

Slowpoke: End-to-end Throughput Optimization Modeling for Microservices — Sept 2023 - Apr 2025
Project Lead, under submission of NSDI'26 (Distributed profiling, Tracing, Kubernetes, Istio, Docker, Golang, C++)

- Introduce the first profiling system for accurate what-if analysis of throughput optimizations in complex microservices, e.g., "what end-to-end throughput could be if I optimize the cart service from 10K req/s to 20K req/s?", achieving RMSE of 2.1% for 4 real benchmarks and 1.9% for 108 synthetic benchmarks covering comprehensive characteristics.
- Design a performance model to infer bottleneck equivalence by selectively slowing down non-target services.

λEASH: Scaling Out Shell Scripts with Recoverable Serverless Computing — Sept 2023 - Present
Project Lead (Serverless, Shell, AWS Lambda, PL, Python, C, Rust)

- Introduce a system automatically scale out unmodified shell scripts correctly, achieving significant speedups over Bash (avg: 15.2×, max: 215.8×) with minimal cost increases (avg: $0.11)—in some cases being both faster and cheaper.
- Design a recoverability protocol to allow Lambda run longer than 15 minutes—limit imposed by AWS—while preserving correctness and Unix pipe-like streaming efficiency with minimal overhead (avg: 0.03× slower).
- Schedule computations smartly across EC2 and Lambda to overcome limited network bandwidth, allow direct TCP communication between Lambdas using hole-punching technique, and automatically resolve software dependencies.

Per-key Watermark for Apache Flink — Sept 2022 – Jan 2023
Project Lead (Apache Flink, Java, Source Code Improvement, Stream Processing)

- Modify Apache Flink source codes to support Per-key Watermark propagation, processing and API, generate testing sources with different number of keys, portion of delayed keys and delayed time interval.
- Reduce latency of stream processing with 10-second tumbling window by 30% given 40000 keys with 50% delayed by 50ms, and the latency decreases more as the number of keys, portion of delays and delayed time increase.

SSD-Concurrent Tree Traversal/Searching Algorithm — Jan 2022 – May 2022
Project Assistant (C++, SSD Read Parallelism, Parallel BFS/DFS/IDDFS, Concurrent I/O, Direct I/O, OpenMP)

- Implement memory-mapped file structure for tree traversal with direct I/O to avoid OS cache, implement parallel BFS/IDDFS and unordered parallel DFS using OpenMP library, tune hyperparameter to split searching stack for DFS.
- For 8-CPU and 32-thread settings, parallel DFS speeds up 7.5-12.8× given 20000 to 80000 nodes and 2 to 8 branches, parallel BFS speeds up 13.2-14.3× while the SSD Read Parallelism is around 14 under random key access testing.

Technologies

Python, C++, Golang, Java, Rust, Shell, JavaScript, SQL, Kubernetes, Docker, AWS, Django, Apache Flink, Istio, Linux