

2024 年度

筑波大学情報学群情報科学類

卒業研究論文

題目

ユーザ空間並列ファイルシステムのための
システムコールフックライブラリの設計と評価

主専攻 情報システム主専攻

著者 宮内 遥楓

指導教員 建部 修見

要旨

ユーザー空間並列ファイルシステムは、ストレージシステムの性能を向上させるために開発されてきた。一方、POSIX インターフェースは、標準として長い間アプリケーションに使用されてきた。多くのアプリケーションを動作させるためには POSIX インタフェースのサポートが必要であるが、FUSE やシステムコールインターセプションライブラリなどの既存の手法には様々な問題がある。本研究では、バイナリ書き換えに基づくシステムコールフック機構である `zpoline` を利用することでこの問題を解決し、その性能結果を示す。

目次

第 1 章	序論	1
第 2 章	背景	2
2.1	ファイルシステム	2
2.2	ユーザ空間ファイルシステム	3
2.3	並列ファイルシステム	3
第 3 章	既存手法	4
3.1	FUSE	4
3.2	プリロードライブラリ	6
第 4 章	提案手法	7
4.1	システムコールフックライブラリの設計	7
4.2	zpoline	7
4.3	CHFS	7
第 5 章	評価実験	10
5.1	予備実験: FUSE 性能評価	10
5.2	実験環境	10
5.3	実験方法	10
5.3.1	ネイティブ API	10
5.3.2	FUSE	11
5.3.3	システムコールフック (提案手法)	11
5.4	実験設定	11
5.5	結果	11
第 6 章	終論	13
	謝辞	14
	参考文献	15

図 目 次

2.1	ファイルシステム概念図	2
3.1	FUSE アーキテクチャ	5
4.1	システムコールフックによるファイルシステムアクセス	8
4.2	zpoline メカニズム	9
5.1	IOR 読み込み性能	12
5.2	IOR 書き込み性能	12

第1章 序論

第2章 背景

2.1 ファイルシステム

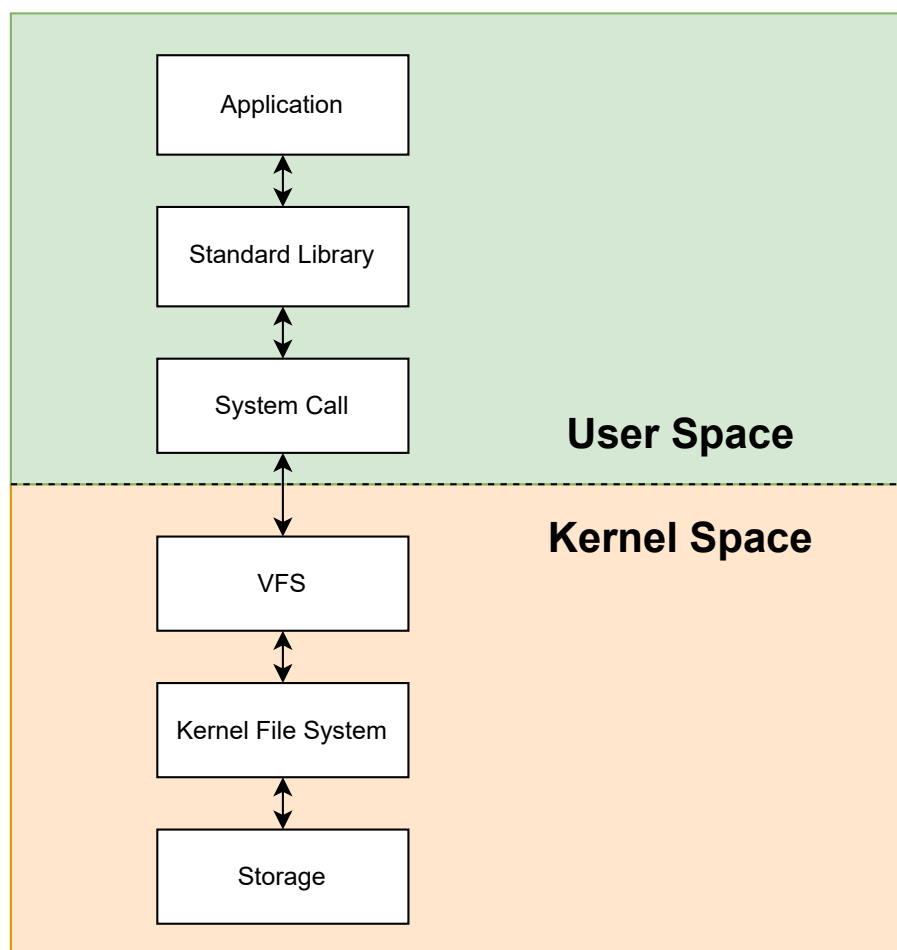


図 2.1: ファイルシステム概念図

2.2 ユーザ空間ファイルシステム

2.3 並列ファイルシステム

第3章 既存手法

アプリケーションの書き換えなしにユーザ空間ファイルシステムを利用する方法は大きく分けて2つある.

3.1 FUSE

FUSE(Filesystem in Userspace) はユーザ空間ファイルシステムを開発するときに最も使用されているフレームワークである. FUSE を利用することで容易にユーザ空間ファイルシステムを POSIX インターフェースに対応させることができるため, SSHFS [1], GlusterFS [2], ZFS [3] など数多くの FUSE ベースのファイルシステムが開発されている.

しかしながら, アプリケーションと FUSE プロセスとの通信コストや, コンテキストスイッチによるオーバーヘッドが原因の性能低下により, 特にパフォーマンスが求められる高性能計算分野での利用は適していないと論じられている [4]. そのため FUSE から派生したより効率的なユーザ空間ファイルシステムフレームワークの研究も多くされている [5, 6].

アプリケーションが FUSE を介してユーザ空間ファイルシステムにアクセスする過程を図 3.1 に示す. アプリケーションが標準ライブラリを通じてシステムコールを発行し, VFS に到達するまでは通常のファイルシステムアクセスと同じである. VFS には FUSE ドライバが登録されており, このドライバとユーザ空間ファイルシステムが `/dev/fuse` ブロックデバイスを介してやり取りをする.

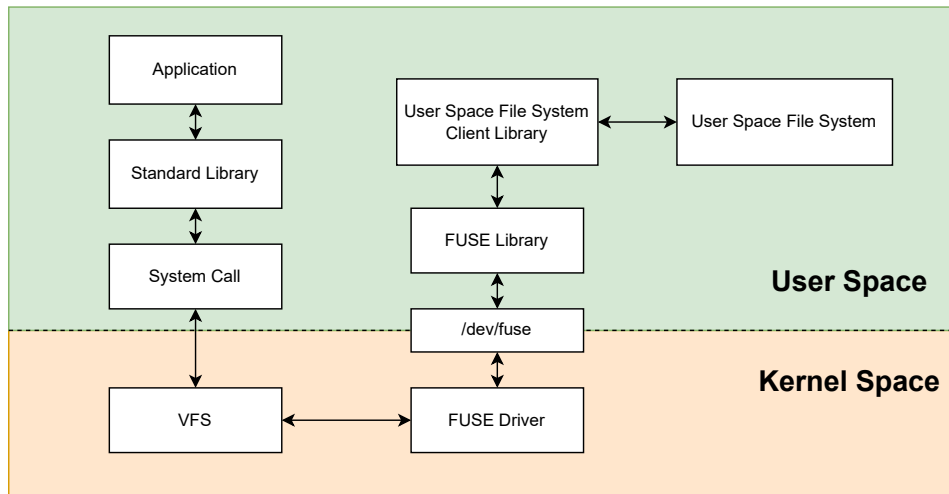


図 3.1: FUSE アーキテクチャ

3.2 プリロードライブラリ

Linux には共有ライブラリの関数をフックできる `LD_PRELOAD` 環境変数が存在する。 `LD_PRELOAD` に共有ライブラリのパスを指定すると、プログラム起動時に動的リンクが対象ライブラリを優先的にリンクする。これにより、ある関数が `LD_PRELOAD` に指定したライブラリに存在する場合、他のライブラリの実装より優先して呼び出されるため、既存関数の実装差し替えや、`main` 関数実行前に任意の関数を実行することが可能になる。各ユーザ空間ファイルシステムが提供するプリロードライブラリを `LD_PRELOAD` で指定してアプリケーションを実行することで、ソースコードの変更や再コンパイルなしにユーザ空間ファイルシステムを利用することが可能になる。しかしアプリケーションが呼び出す I/O 関数を網羅的にフックすることは容易ではないため、プリロードライブラリの作成には次に示すライブラリが使用されることが多い。

Gotcha [7] は任意の関数をラップするライブラリで、`LD_PRELOAD` に似ているが、プログラム可能な API を介して動作する。UnifyFS [8] は Gotcha を利用して `libc` (C 標準ライブラリ) の I/O 関連の関数をフックし、UnifyFS 固有の API に置き換えることで POSIX インタフェースの対応を行っている。

`syscall_intercept` [9] は、メモリにロードされた `glibc` のテキストセグメント中の `syscall` 命令を検知し、`jmp` 命令に置き換えることで任意の関数呼び出しを可能にするライブラリである。`syscall_intercept` を使用して POSIX インタフェース対応をしているユーザ空間ファイルシステムには GekkoFS [10] がある。`glibc` の関数をフックする場合と比較して、少数のシステムコールフックを定義するだけで済むため、ファイルシステムが提供するプリロードライブラリの実装は相対的に容易である。しかし `syscall_intercept` は Linux の `glibc` にしか対応していないため、`glibc` を使わないアプリケーションは動かない可能性がある。

第4章 提案手法

本研究では, ユーザ空間ファイルシステムにおける POSIX インタフェース対応のために, 前章で既存手法として挙げたプリロードライブラリの新しい設計を提案する. 具体的には, ユーザ空間ファイルシステム上で動作するアプリケーションによって呼び出されるシステムコールをフックするライブラリを設計する. また提案手法の評価のため, ユーザ空間並列ファイルシステムである CHFS [11] の POSIX インタフェース対応を目的としたシステムコールフックライブラリを実装する. ライブラリの実装においては, システムコールフックメカニズムの `zpoline` [12] を利用する.

4.1 システムコールフックライブラリの設計

4.2 `zpoline`

4.3 CHFS

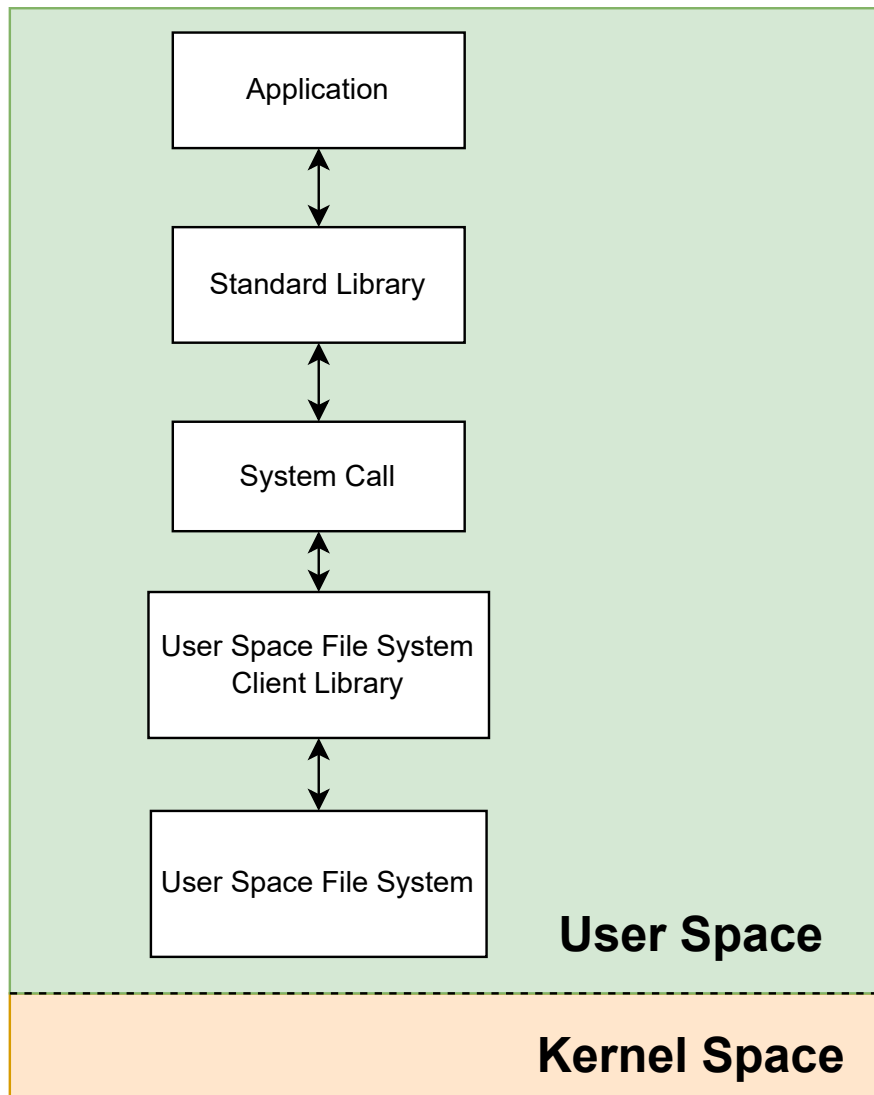


図 4.1: システムコールフックによるファイルシステムアクセス

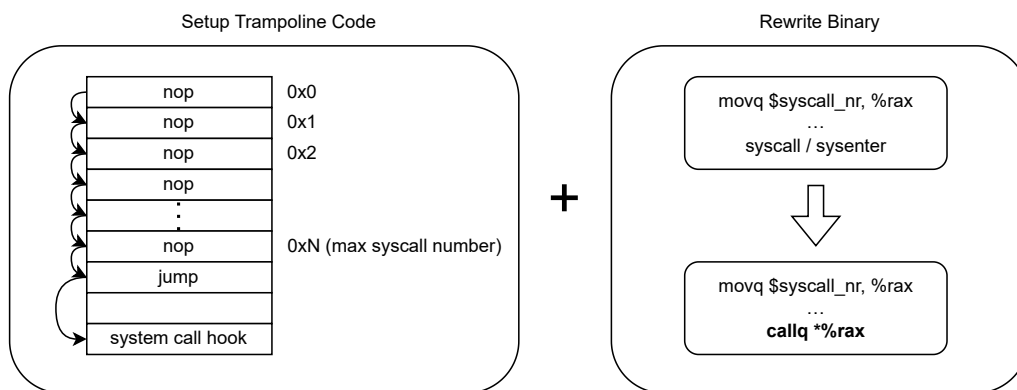


図 4.2: zpoline メカニズム

第5章 評価実験

前章で実装したシステムコールフックライブラリの評価実験を行う。本実験の目的としては、ユーザ空間ファイルシステムが提供するクライアントライブラリを使用した場合と比べてどの程度ストレージアクセス性能が保たれるか評価すること、また既存手法の中で最もよく用いられる FUSE との性能を比較することである。

5.1 予備実験: FUSE 性能評価

TODO: 余裕があればやる

5.2 実験環境

実験環境として筑波大学計算科学研究センターが運用する Pegasus スーパーコンピュータを利用する。Pegasus は各計算ノードに専用のストレージを保持しており、SSD と PMEM を利用できる。本実験では PMEM を devdax モードで利用する。計算ノード間は InfiniBand NDR 200 で接続されており、帯域幅は 200Gbps である。

5.3 実験方法

本実験では IOR ベンチマーク [13] を実行して CHFS ファイルシステムに対するファイル読み込み/書き込み帯域幅を測定する。提案手法を含む以下の 3 種類の条件下でベンチマークを実行し、その性能を比較する。

5.3.1 ネイティブ API

ファイルシステムが提供するクライアントライブラリの API を利用してファイルにアクセスする。本実験では IOR ベンチマークのソースコードを変更し、CHFS クライアントライブラリの関数を呼び出すことで CHFS 上のファイルへの読み書きを行う。

5.3.2 FUSE

CHFS サーバを起動した後、`chfuse` コマンドを使用して指定のディレクトリにマウントする。IOR ベンチマークのファイル読み込み・書き込みはマウントしたディレクトリに対して実行する。

5.3.3 システムコールフック (提案手法)

前章で実装したシステムコールフックライブラリを利用する。仮想的なマウントディレクトリ/`chfs` に対して IOR ベンチマークのファイル読み込み・書き込みを行う。

5.4 実験設定

Pegasus システムにおいて、計算ノード 10 台を CHFS サーバとして割り当て、さらに 1 台を IOR ベンチマークの実行に使用し、合計で 11 台の計算ノードを専有する。

CHFS はチャンクサイズを 16MiB に設定し、各ノード 46 プロセスで CHFS サーバを起動する。通信プロトコルに `verbs`、バックエンドに `pmemkv` を利用する。

IOR はプロセス数を 1、2、4、…と 16 まで増やしながら実行する。IOR では `file-per-process` 方式でアクセスし、各プロセスが別々のファイルに対して 1TiB 読み書きする。この操作を 5 回行い、帯域幅の平均を求める。

5.5 結果

実験結果を図 5.1 と図 5.2 に示す。どの条件でもプロセス数を増やすにつれ帯域幅が増加していることがわかる。読み込みの 16 プロセスにおいて性能の増加がないのは、Pegasus 計算ノード間の通信帯域幅 200Gbps に到達していると考えられる。

基準であるネイティブ API を使用した場合の性能に対して、提案手法のシステムコールフックは読み込み・書き込みともに同程度の性能を示した。また FUSE を使用した場合と比較して提案手法は 5.3 倍から 6.4 倍高い性能を示した。

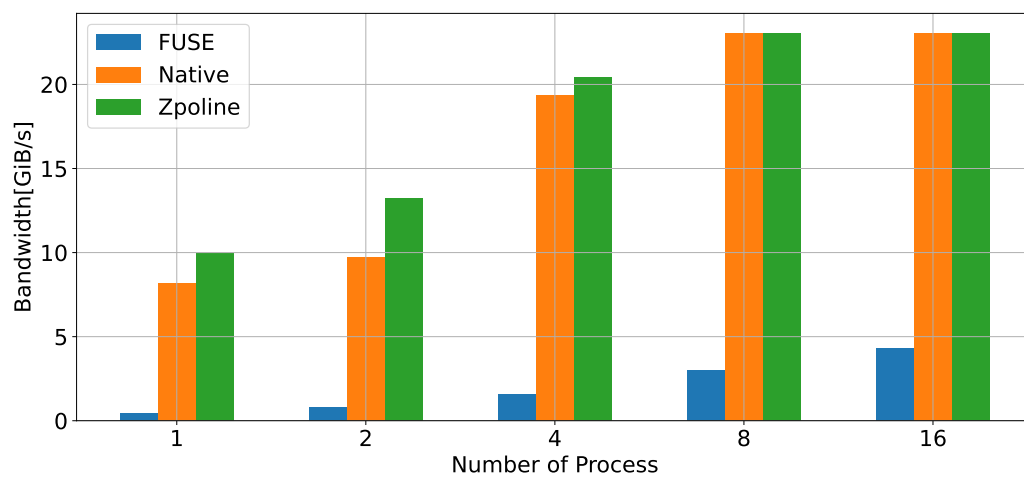


図 5.1: IOR 読み性能

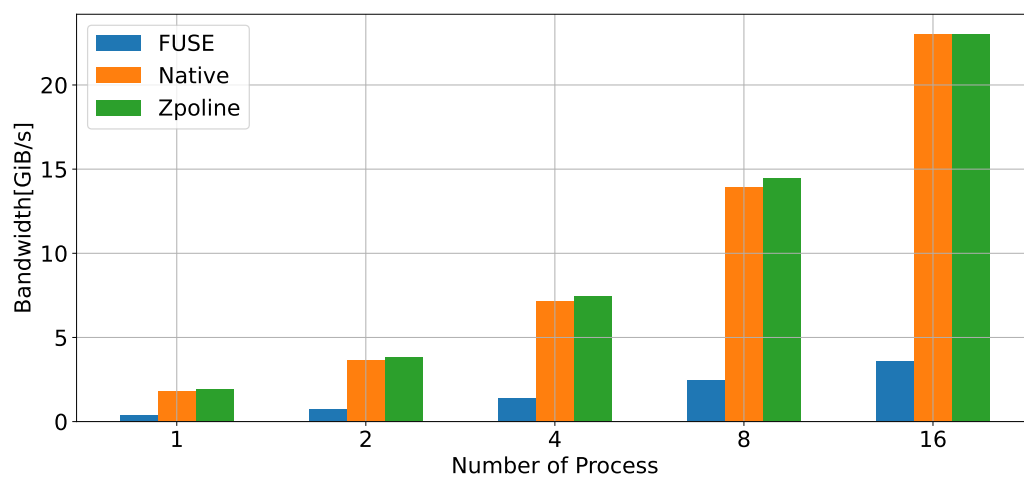


図 5.2: IOR 書き込み性能

第6章 終論

謝辞

1年間親身かつ丁寧に研究を指導してくださった筑波大学計算科学研究センターの建部修見教授に深く感謝申し上げます。加えて、研究テーマの相談段階から論文作成にいたるまで研究にご指導ご尽力いただき、また国際会議の発表において共著者としてご協力いただいたHPCS研究室の小山創平氏に大変感謝申し上げます。また、計算科学研究センターの研究員である平賀弘平氏、HPCS研究室の杉原航平氏にも研究を進めるうえで多大なアドバイスをいただきました。感謝申し上げます。同時に、計算科学研究センターの職員の皆様、特に同センター秘書の桑野洋子氏には事務的な面で研究をサポートしていただきありがとうございました。HPCS研究室の皆様には、研究を進めるにあたり議論を通じてご協力いただきました。そして共同研究先である富士通研究所の皆様には学外としての立場から大変ありがたいご支援をいただきました。ありがとうございました。最後に、これまで大学生活を共にした友人と、学生生活を支えてくれた家族に心から感謝致します。

参考文献

- [1] Matthew E Hoskins. Sshfs: super easy file access over ssh. Linux Journal, Vol. 2006, No. 146, p. 4, 2006.
- [2] Alex Davies and Alessandro Orsaria. Scale out with glusterfs. Linux Journal, Vol. 2013, No. 235, p. 1, 2013.
- [3] Ohad Rodeh and Avi Teperman. zfs-a scalable distributed file system using object disks. In 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings., pp. 207–218. IEEE, 2003.
- [4] André Brinkmann, Kathryn Mohror, Weikuan Yu, Philip Carns, Toni Cortes, Scott A Klasky, Alberto Miranda, Franz-Josef Pfreundt, Robert B Ross, and Marc-André Vef. Ad hoc file systems for high-performance computing. Journal of Computer Science and Technology, Vol. 35, pp. 4–26, 2020.
- [5] Kyu-Jin Cho, Jaewon Choi, Hyungjoon Kwon, and Jin-Soo Kim. RFUSE: Modernizing userspace filesystem framework through scalable Kernel-Userspace communication. In 22nd USENIX Conference on File and Storage Technologies (FAST 24), pp. 141–157, Santa Clara, CA, February 2024. USENIX Association.
- [6] Yue Zhu, Teng Wang, Kathryn Mohror, Adam Moody, Kento Sato, Muhib Khan, and Weikuan Yu. Direct-fuse: Removing the middleman for high-performance fuse file system support. In Proceedings of the 8th International Workshop on Runtime and Operating Systems for Supercomputers, pp. 1–8, 2018.
- [7] LLNL. Gotcha. <https://github.com/LLNL/GOTCHA>.
- [8] Michael J. Brim, Adam T. Moody, Seung-Hwan Lim, Ross Miller, Swen Boehm, Cameron Stanavice, Kathryn M. Mohror, and Sarp Oral. Unifyfs: A user-level shared file system for unified access to distributed local storage. In 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 290–300, 2023.
- [9] Pmem. Syscall_intercept. https://github.com/pmem/syscall_intercept.
- [10] Marc-André Vef, Nafiseh Moti, Tim Süß, Tommaso Tocci, Ramon Nou, Alberto Miranda, Toni Cortes, and André Brinkmann. Gekko - a temporary distributed file system for hpc

- applications. In 2018 IEEE International Conference on Cluster Computing (CLUSTER), pp. 319–324, 2018.
- [11] Osamu Tatebe, Kazuki Obata, Kohei Hiraga, and Hiroki Ohtsuji. Chfs: Parallel consistent hashing file system for node-local persistent memory. In International Conference on High Performance Computing in Asia-Pacific Region, pp. 115–124, 2022.
- [12] Kenichi Yasukata, Hajime Tazaki, Pierre-Louis Aublin, and Kenta Ishiguro. zpoline: a system call hook mechanism based on binary rewriting. In 2023 USENIX Annual Technical Conference (USENIX ATC 23), pp. 293–300, Boston, MA, July 2023. USENIX Association.
- [13] hpc. Ior. <https://github.com/hpc/ior>.