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. ユーザ空間ファイルシステム上でアプリケーションを書き換えずに動作させるためには POSIX インタフェースへの対応が必要であるが, 既存手法の FUSE やプリロードライブラリには様々な問題がある. 本研究では, アプリケーションが呼び出すシステムコールをユーザ空間ファイルシステムの API 呼び出しに置き換えるシステムコールフックライブラリを設計する. ライブラリの実装において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. 実装したシステムコールフックライブラリに対する評価実験の結果, ユーザ空間ファイルシステムの API を直接呼び出した場合と比較して性能がほぼ低下しないこと, 既存手法の FUSE を使用した場合と比較して大幅に性能が向上することを確認した.

目 次

図 目 次

第1章　序論

第2章　背景

2.1　ファイルシステム

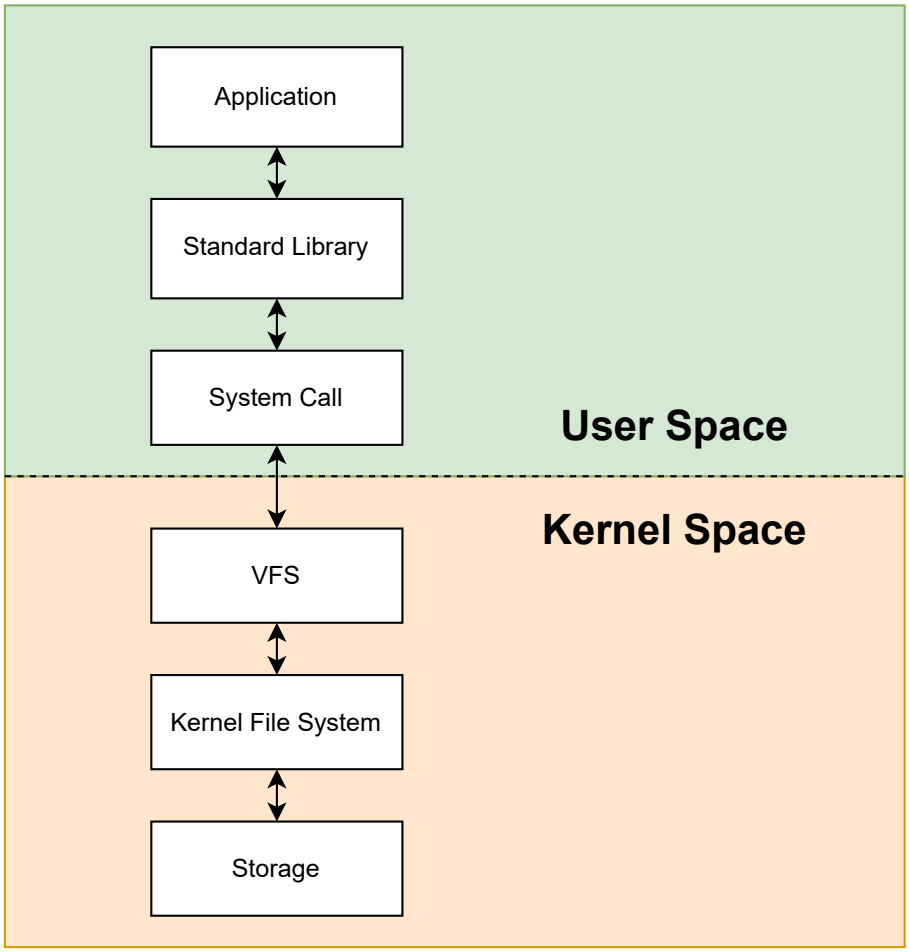

図 2.1: ファイルシステム概念図

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.

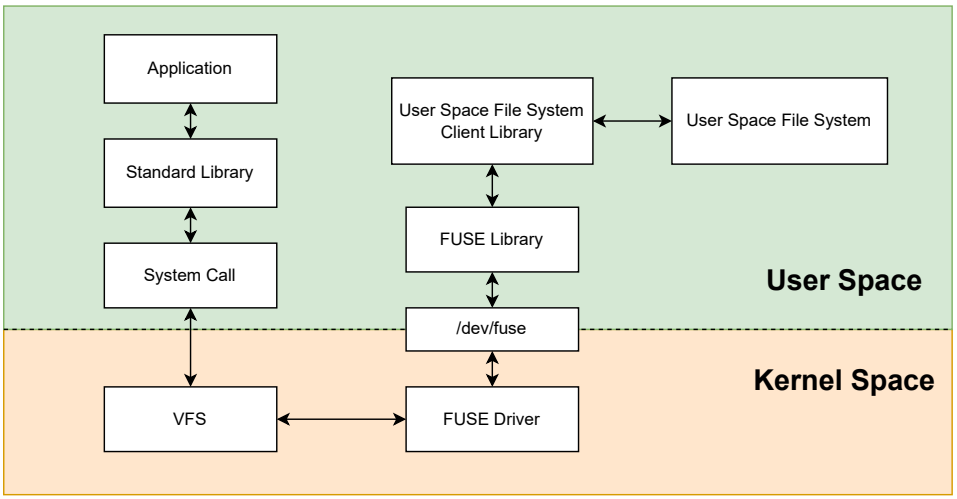

図 3.1: FUSE アーキテクチャ

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カ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 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

アプリケーションがユーザ空間ファイルシステムにアクセスするエントリポイントとして, 仮想的なマウントポイント/virtual_mount_point を仮定する. アプリケーションから呼び出されたシステムコールに含まれるパス名が/virtual_mount_point で始まる場合, そのファイルはユーザ空間ファイルシステムの管理下にあると判断し, システムコールの呼び出しをユーザ空間ファイルシステムのクライアントライブラリの API 呼び出しに置き換える. そうでない場合は本来のシステムコールを呼び, 通常通り VFS を経由してカーネル空間のファイルシステムにアクセスする.

システムコールがフックされた場合, 本来のシステムコールは呼ばれないためコンテキストスイッチが発生しない. そのため FUSE で発生しているようなコンテキストスイッチに起因するオーバヘッドを削減することができ, ユーザ空間ファイルシステムを効率的に利用することが可能になる.

4.2　zpoline

zpoline は安形らによって提案された, システムコールを高速にフックする手法である. 既存手法で例に挙げた syscall_intercept には glibc 内で呼び出されるシステムコールしかフックできないという問題があったが, zpoline ではバイナリ書き換えとトランポリンコードを組み合わせることでこの問題を解決し, 全てのシステムコールをフックすることを可能にした.

zpoline がシステムコールをフックする仕組みを図 4.2 を用いて説明する. zpoline は LD_PRELOAD でロードされることを想定したライブラリとして実装されており, アプリケーションの main 関数が実行されるよりも前にシステムコールフックの準備を行う.

プログラムのバイナリがメモリにロードされると, zpoline はまずロードされたバイナリの書き換えを実行する. ユーザ空間プログラムがカーネルの機能を利用する際には必ずシステムコールを発行する必要がある. そのために使われる syscall / sysenter 命令は 2byte の機械語命令であり, システムコールを他の関数呼び出しに置き換えるには syscall / sysenter 命令を別の機械語命令に書き換えてやればよい. しかし 3byte 以上の機械語命令で書き換えてしまうと syscall / sysenter 命令の次の命令を上書きしてしまい, プログラムが正常に動作しなくなるため, 2byte という非常に厳しい制約のもと命令を書き換える必要がある. zpoline は callq *%rax という 2byte の命令で syscall / sysenter 命令を書き換える. callq *%rax は%rax レジスタに入ったアドレスにジャンプし, ジャンプ元のアドレスをスタックへプッシュするという命令である. ここで Linux のシステムコールの呼出規約では%rax レジスタにシステムコール番号を入れておく必要がある. システムコール番号はカーネルの定義により高々500 に収まる数のため, callq *%rax が実行されるとアドレス 0 ~500 にジャンプする.

次に zpoline はメモリアドレス 0 から最大のシステムコール番号+1 の位置に, トランポリンコードと呼ばれる任意のフック関数にジャンプするためのコードを用意する. まずメモリアドレス 0 から最大のシステムコール番号までを nop 命令で埋める. nop 命令は何もせず次の命令を実行する命令である. その後最後の nop 命令の次の位置に, 任意のフック関数があるアドレスにジャンプするためのコードを埋め込む.

zpoline によるシステムコールフックの準備が完了すると, 通常通りプログラムの main 関数が実行される. プログラム内でシステムコールが呼ばれると, バイナリ書き換えにより syscall / sysenter 命令から置き換えられた callq *%rax 命令が実行される. callq *%rax 命令によりメモリ上ではシステムコール番号のアドレスに移動する. 移動先はトランポリンコードの設置により nop 命令で埋められており, nop 命令が繰り返し実行された後, 任意のシステムコールフック関数があるアドレスにジャンプする. これによりシステムコールに対し任意の関数呼び出しが実行される.

4.3 CHFS

CHFS は建部らの開発するスーパコンピュータ向けの並列ファイルシステムである. 近年のスーパコンピュータは各計算ノードにローカルストレージを持つことが多く, CHFS はそれらのストレージをまとめて 1 つのストレージであるかのように扱えることを目的としている. CHFS はスーパコンピュータ向けの I/O 性能の世界ランキング IO500 の 2023 年 6 月の 10 ノード研究部門において 21 位を記録しており, 高性能計算分野における今後の利用が期待される.

しかしながら IO500 で示された CHFS の性能は, ベンチマークプログラムのファイル I/O 部分の関数を CHFS の API に置き換えて実行することで測定されたものであり, 実際のアプリケーションの複雑なソースコードを同様に書き換えることは現実的ではない. そのためアプリケーションを CHFS 上で動作させるには FUSE を利用したマウントが必要になるが, 前章で述べた通り通信コストやコンテキストスイッチによる性能低下が予想される.

本研究では提案した設計に基づき, CHFS のシステムコールフックライブラリを実装する.

/chfs を仮想的なマウントディレクトリと設定し, /chfs から始まるパス名を対象とするシステムコールを, zpoline を利用して CHFS の API 呼び出しに置き換える. 次章の実験では IO500 で使用されるベンチマークプログラムである IOR [13] を用いて性能評価を行う. IOR ベンチマークが呼び出すシステムコールに対応するため, 実装するライブラリでは read, write, open, close, stat, lstat, lseek, pread64, pwrite64, openat, fsync, newfstatat システムコールをフックする.

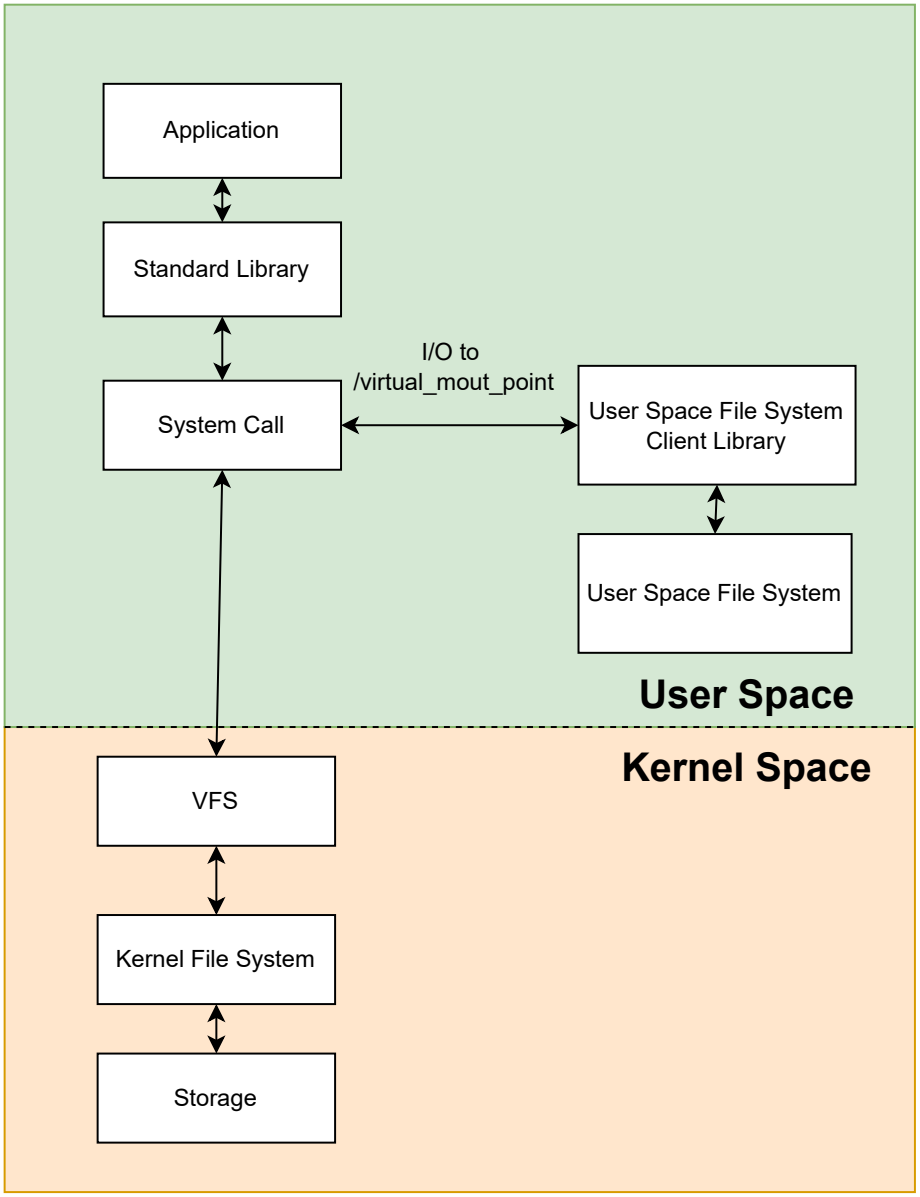

図 4.1: システムコールフックによるファイルシステムアクセス

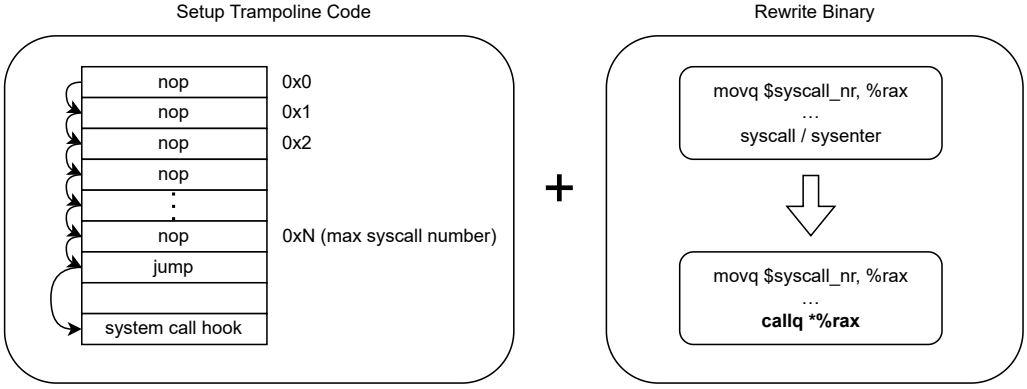

図 4.2: zpoline メカニズム

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 ベンチマークを実行して CHFS 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.

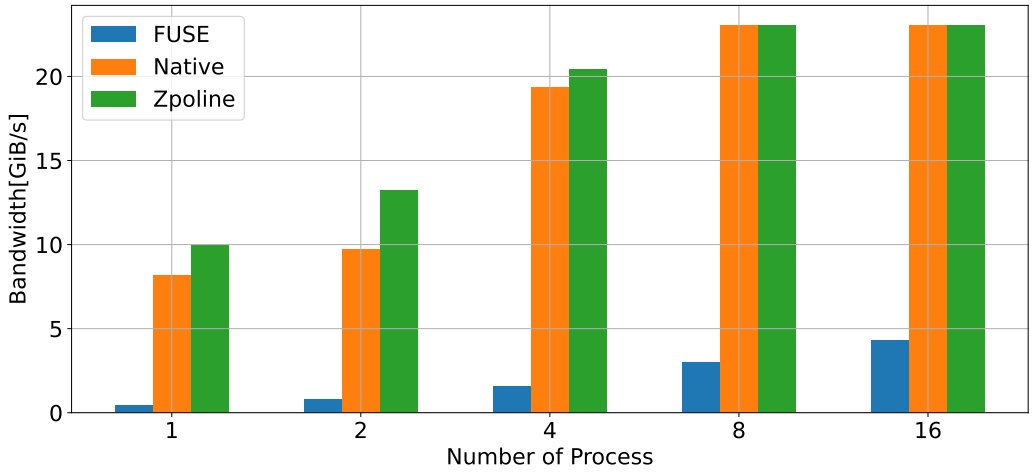

図 5.1: IOR 読込み性能

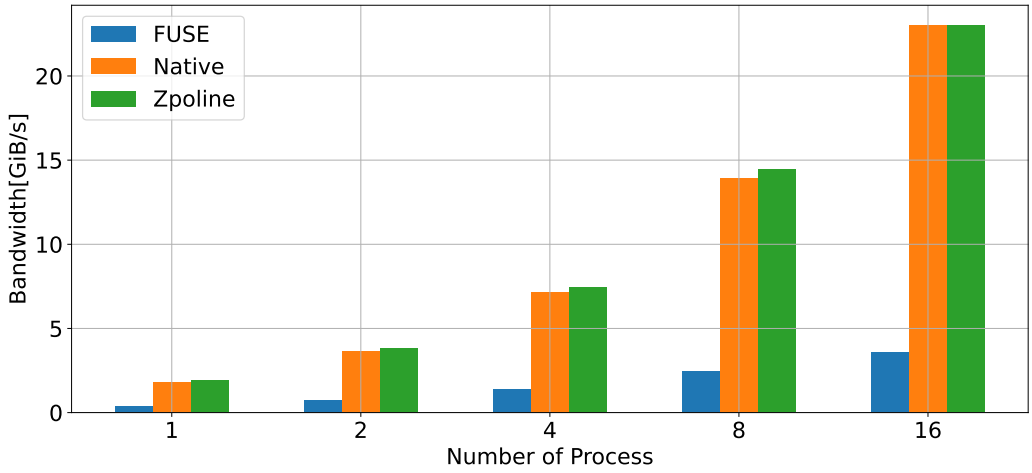

図 5.2: IOR 書込み性能

第6章　終論

6.1　まとめ

ユーザ空間ファイルシステムの POSIX インタフェース対応を目的とした手法はいくつか存在するが, 既存手法には性能や汎用性の面で問題があった. 本研究ではファイル操作に必要なシステムコールをユーザ空間ファイルシステムの API に置き換える, システムコールフックライブラリを設計した. 高速なシステムコールフック機構である zpoline を使用したシステムコールフックライブラリを実装し, 性能評価を行った. 性能評価においてはストレージ性能を評価する IOR ベンチマークを用いて評価を行い, ユーザ空間ファイルシステムの API を直接呼び出した場合と比較してほぼ性能が低下しないこと, 既存手法の FUSE を利用した場合と比較して 5.3 倍から 6.4 倍高い性能を示すことを確認した.

6.2　今後の展望

IOR ベンチマークを用いた性能評価により, 提案手法の有用性を示した. 今後は実アプリケーションの I/O 性能の向上に向け, より多くの実アプリケーションを動作させることを目指す. 実アプリケーションは今回使用した IOR ベンチマークよりも多くのシステムコールを使用するため, システムコールのフックを増やす必要がある. 現在はファイルディスクリプタを複製する dup・dup2, 子プロセスを生成する fork システムコール等の対応が完了している. その結果, Linux の ls, touch 等ファイルシステムに関連するコマンドや, 与えられた画像データセットをニューラルネットワークで分類する簡単な PyTorch プログラムを CHFS 上で動作させることに成功している.

今後はより大規模かつ実際に使用されているものに近いアプリケーションを動作させる. 背景の章で説明したように, 機械学習ワークロードでは大量のデータをストレージから読み込む必要があるため, CHFS を利用できれば学習をより短時間で完了できる可能性がある. そのため現在は機械学習ワークロード向けのストレージベンチマークである MLPerf Storage ベンチマーク [14] を動作させることを目標にシステムコールのフックを進めている.

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 Stanavige, 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. Gekkofs - 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`.

[14] mlcommons. Mlperf storage benchmark suite. `https://github.com/mlcommons/storage`.