B-001

近似コンピューティング向け分散インメモリ・キャッシュの実現

齊藤 七菜[1)]　河野 健二[1)]
Nana Saito　Kenji Kono

1　はじめに

大規模分散ストレージでは，すべてのファイルに対して高い堅牢性を保証することを念頭に設計されている．しかし，画像の機械学習など，用途によってはファイルの一部に欠損が生じてもよい場合がある．本研究ではこのような用途を念頭に，ファイルの一部の欠損を許容する分散インメモリ・キャッシュを提案する．ファイルの欠損を許容するため，冗長性の維持や欠損したキャッシュの回復が不要となり，軽量かつスケーラブルな分散キャッシュとして機能する．実験では，erasure coding による誤り訂正を行う場合と比較して，open から close までの所要時間を平均 56% 削減することができた．また Pascal VOC データセット [1] を用いた画像分類の実験にて，mean Average Precision(mAP) への影響を調べた結果，最大 9.3% のサーバの障害を許容できることを確認した．

2　背景

2.1　大規模分散ファイルシステム

クラウド環境では，大規模分散ファイルシステムを使用する．大規模分散ファイルシステムでは，堅牢性を保証するために，データの冗長化と分散配置，及び障害時の自動リカバリなどを行う．

障害発生時の回復手法は，様々なものが提案されており，有名な手法として erasure coding が挙げられる．Erasure coding は誤り訂正符号の一種で，レプリケーションに比べ，少ない冗長性で高度な信頼性を保証する [2]．そのため，Microsoft Azure, Google など様々なサービスで利用される [3]．しかし，大規模分散ファイルシステムは，信頼性保証のための処理を行う一方で，処理速度が遅いという問題点がある．

この低速性を補うために，X や Facebook などでは分散インメモリ・キャッシュを導入している [4]．ファイルアクセスに先立って，あらかじめインメモリ・キャッシュにファイルデータを配置しておくことで，ユーザのデータ取得時間を短縮することができる．

しかし，インメモリ・キャッシュを導入しても，遅延や障害が生じた場合は，ユーザプロセスは回復処理完了まで待たされてしまう．実際に，大規模モデルの学習を計算機クラスタ上で行う際には，メモリエラーやネットワーク障害，GPU のクラッシュなどの障害が生じることが報告されており，マシン 1 台あたりの年間障害発生率は 1.3% に達するという調査結果もある [5]．

2.2　近似コンピューティング

近似コンピューティングとは，計算の精密性を必ずしも要求しないアプリケーションにおいて，ある程度の誤差を許容する代わりに，処理性能やエネルギー効率の向上を図る考え方である [6]．具体的なアプリケーションの例としては，機械学習，パターン情報処理，メディア処理，信号処理などがあり，ハードウェアから，ソフトウェアやアルゴリズムまで，幅広いレイヤで導入されている．実際に，一部の Google の TPU や IBM の AI 向けアクセラレータは，近似コンピューティングに基づいて設計されている [7]．

3　提案

本研究では，近似コンピューティングのための，ファイルの一部の欠損を許容する分散インメモリ・キャッシュを提案する．ファイルの欠損を許容するため，冗長性の維持や回復処理，データの再送処理などが不要となり，高速な分散キャッシュとして機能する．

提案手法では，通常時には正常なデータを返し，障害や遅延が生じた場合には，回復処理を待たず取得できるデータを用いてデータを返す．この時，各用途の欠損許容の有無や欠損データの補完方法は，モジュールとしてプラグインで追加できるようにする．例えば画像の欠損補完方法として，opencv の inpaint[8] や LaMa[9] などの手法が挙げられる．

提案システムは，ユーザ空間ファイルシステムとして Filesystem in UserSpace(FUSE)[10] を用いて実装する．FUSE を使うことで，カーネルの修正を伴わずに，軽量なシステムとして動作する．また，クライアントとキャッシュサーバ間の通信は gRPC[11] を用いる．gRPC はシリアライズ形式として Protocol Buffer を採用しているため，シリアライズ，デシリアライズを意識せず容易に通信を行うことができる．

尚，モジュールの選択は，実行時にユーザがコマンドラインオプションを指定することで行う．

4　実験

本章では，データ欠損時におけるデータ取得時間短縮の有無と，近似コンピューティングによる性能への影響の 2 つについて検証を行う．

比較対象として，既存手法の erasure coding[12] を用いる．実験では，同一のファイルに対し，分散数及び欠損数を揃えた条件下で，open から close までの所要時間を測定する．また，erasure coding については，誤り訂正なし・ありのそれぞれにおける所要時間を計測する．ファイルは Open images Dataset V4 データセット [13] を用い，ファイル形式は BMP，ファイル枚数は 920 枚とする．実験は，AMD Ryzen 7 PRO 5750GE（8 コア），16GB メモリを搭載したマシン上で実施した．実験環境は，Ubuntu 22.04.2，Linux 5.15，FUSE 3.10.5 及び gRPC 1.60.0 である．

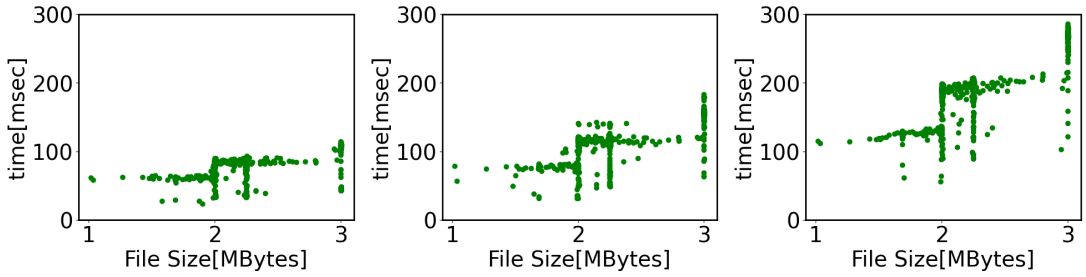
図 1 より，提案手法（図 1a）は，erasure coding で誤り訂正をする場合（図 1c）よりも所要時間が 200msec 程度短い．誤り訂正をしない場合（図 1b）と比較しても，所要時間は 80msec 程度短くなった．この結果は，提案手法が回復処理を省略することに起因する．

次に画像の機械学習を例として，近似コンピューティングによる性能への影響を調査する．

評価には，物体検出タスクの指標で用いられる mAP[14] を利用する．mAP は 0 から 100 で表され，値が大きい程よい．

実験では，Pascal VOC データセットを使用する．画

1)　慶應義塾大学



(a) 提案手法の所要時間　(b) Erasure coding（誤り訂正なし）　(c) Erasure coding（誤り訂正あり）

図 1: 各手法における所要時間の比較

像を縦横 6 ずつ分割し，指定したブロック数分だけランダムに欠損させ，学習をする．学習は yolo11[15] を用い，訓練画像枚数 1000 枚，検証画像 400 枚，テスト画像は 600 枚とする．欠損数は 1 から 10 とし，各条件で 5 回の学習を行い，平均と標準偏差を算出する．

先行研究 [16, 17] では，欠損を許容できる目安が mAP の低下が -2 程度とされている．図 2 より，画像の 9.3% が欠損しても，mAP の低下が欠損許容範囲に収まることがわかる．従って，最大で 9.3% のサーバで障害や遅延が生じた場合でも，データ取得を待機せずに学習を継続できると考えられる．

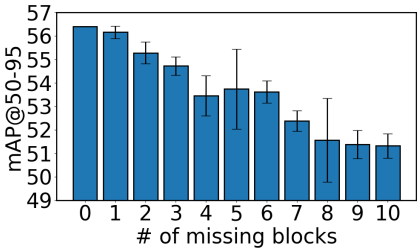


図 2: 画像の欠損による学習性能への影響

5　関連研究

AxNN[18] は，ニューラルネットワーク向けの近似コンピューティングに取り組み，損失への影響が小さいノードに関する演算を省略している．本研究と AxNN の相違点は計算量の多いアプリケーションに対して，そのアルゴリズムのレベルから計算量の削減を試みた点である．Rumba[19] は近似コンピューティングによる損失の影響を最小化する手法を提案している．近似コンピューティングにおける損失は決定木や線型モデルによって予測することができるため，この特性を利用し，損失検出を高速かつ効率的に行っている．

6　まとめ

本論文では，近似コンピューティングが適用可能なアプリケーションを対象に，ファイルの一部の欠損を許容する分散インメモリ・キャッシュを提案する．欠損を許容することで，冗長性の維持や回復処理の待機が不要となり，軽量かつスケーラブルな分散キャッシュとして機能する．実験では，erasure coding による誤り訂正を行う場合と比較して，open から close までの所要時間を平均 56% 削減することができた．また Pascal VOC データセットを用いた画像分類の実験にて，mAP の低下しない範囲を調べた結果，最大 9.3% のサーバの障害を許容できることを確認した．

参考文献

[1] M. Everingham, S. M. Eslami, L. Gool, C. K. Williams, J. Winn, and A. Zisserman, The Pascal Visual Object Classes Challenge: A Retrospective, *Int. J. Comput. Vision*, vol. 111, no. 1, pp. 98–136, (2015).

[2] H. Weatherspoon and J. Kubiatowicz, Erasure coding vs. replication: a quantitative comparison, *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, (2002).

[3] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, Erasure coding in Windows Azure Storage, *USENIX Annual Technical Conference (USENIX ATC)*, pp. 15–26, (2012).

[4] J. Yang, Y. Yue, and K. V. Rashmi, A large scale analysis of hundreds of in-memory cache clusters at Twitter, *Proc. 14th USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, pp. 191–208, (2020).

[5] B. Schroeder, E. Pinheiro, and W.-D. Weber, DRAM errors in the wild: a large-scale field study, *Communications of the ACM*, vol. 54, no. 2, pp. 100–107, (2011).

[6] S. Mittal, A survey of techniques for approximate computing, *ACM Computing Surveys*, vol. 48, no. 4, (2016).

[7] W. Liu, F. Lombardi, and M. Schulte, Approximate computing: from circuits to applications [scanning the issue], *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2103–2107, (2020).

[8] OpenCV Developers, Inpainting with OpenCV, *https://docs.opencv.org/3.4/df/d3d/tutorial_py_inpainting.html*, (2024).

[9] Suvorov, I. et al., LaMa: Resolution-robust Large Mask Inpainting with Fourier Convolutions, *https://github.com/advimman/lama*, (2021).

[10] B. K. R. Vangoor, V. Tarasov, and E. Zadok, To FUSE or not to FUSE: Performance of user-space file systems, *Proc. 15th USENIX Conf. on File and Storage Technologies (FAST)*, pp. 59–72, (2017).

[11] G. Inc., gRPC: Bidirectional streaming RPC, *https://grpc.io/docs/guides/concepts/*, (2017).

[12] OpenStack Foundation, liberasurecode: Erasure Code Library, *https://github.com/openstack/liberasurecode*, (2020).

[13] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, T. Duerig, and V. Ferrari, The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale, *Int. J. of Computer Vision (IJCV)*, (2020).

[14] D. Reis, J. Kupec, J. Hong, and A. Daoudi, Real-time flying object detection with YOLOv8, *ArXiv*, abs/2305.09972, (2023).

[15] Ultralytics, Ultralytics GitHub Repository, *https://github.com/ultralytics/ultralytics*, (2024).

[16] M. Tan, R. Pang, and Q. V. Le, EfficientDet: Scalable and Efficient Object Detection, *Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 10778–10787, (2020).

[17] W. Jang, H. Jeong, K. Kang, N. Dutt, and J.-C. Kim, “R-TOD: Real-Time Object Detector with Minimized End-to-End Delay for Autonomous Driving, *arXiv preprint arXiv:2011.06372*, (2020).

[18] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, AxNN: Energy-efficient neuromorphic systems using approximate computing, *Proc. IEEE/ACM Int. Symp. on Low Power Electronics and Design (ISLPED)*, pp. 27–32, (2014).

[19] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, Rumba: An online quality management system for approximate computing, *Proc. 42nd ACM/IEEE Int. Symp. on Computer Architecture (ISCA)*, pp. 554–566, (2015).