I/O Viewの一貫性を保証する高速ストレージ向けチェックポインティング

小林 直登[1,a)]　山田 浩史[1,b)]

概要：チェックポイント・リスタートは，アプリケーションの進行状況を後から再開可能なメモリイメージとして保存することで，耐障害性向上を達成する．近年の PCIe フラッシュや不揮発性メモリなどの高速ストレージの登場は，従来のチェックポイント・リスタート機構を見直す機会を与える．こうした高速ストレージを用いれば，従来の低速なディスクを仮定している機構に比べ，より高頻度にかつよりリッチな情報を組み込んだチェックポイントの実現が期待できる．本研究では，高速ストレージの読み書き性能を活用したチェックポイント・リスタート機構を提案する．提案機構では，アプリケーションの改変をすることなく，I/O View の一貫性を保証しながらチェックポイントを取得する．提案手法を Linux 3.14.9 上に実装した．MySQL や memcached に提案機構を適用し，透過的なチェックポイントの取得とメモリイメージから実行を再開することに成功した．

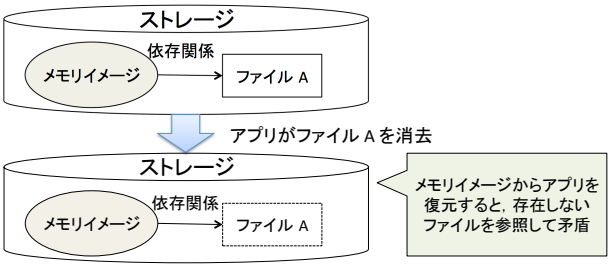

図 1　メモリイメージに矛盾が起きる例

1. はじめに

チェックポイント・リスタートは，アプリケーションの進行状況を後から再開可能なデータ（メモリイメージ）として保存する技術であり，サービスの信頼性の向上に役立つ [4], [6], [7], [9], [16], [17]．定期的にメモリイメージをストレージに保存することで，電源切断などの障害が起きた時に直前のチェックポイントからアプリケーションを再開できる．単純なアプリケーションの再起動に比べると，チェックポイントを取得した時点の途中状態からの再開となるため，復旧が迅速に行えサービスの全体の性能を高く保つことができる．メモリイメージの内容には，ハードウェアコンテキスト（例：インストラクションポインタ，スタックポインタ）やユーザ空間のメモリの内容，展開中のファイル，ネットワークに関する情報などが含まれる．

[1] 東京農工大学
[a)] kobayashi@asg.cs.tuat.ac.jp
[b)] hiroshiy@cc.tuat.ac.jp

近年の PCIe フラッシュや，Phase Change Memory（PCM）[20]，NVDIMM[8] といった不揮発性メモリなどの高速ストレージの登場は，従来のチェックポイント・リスタート機構を見直す機会を与える．従来の機構は，ディスクドライブが他の計算装置に比べて極めて遅いことを前提として設計されており，オーバヘッドを隠蔽するために，低い頻度でチェックポイントを取得し，より粗い処理単位ごとにメモリイメージを更新することが多い．高速ストレージを用いれば，より高頻度にかつよりリッチな情報を組み込んだチェックポイントの実現が期待できる．実際，不揮発性メモリに特化したチェックポイント・リスタート機構が提案されている [2], [10], [21], [29]．

本研究では，高速ストレージの読み書き性能を活用したチェックポイント・リスタート機構を提案する．提案機構では，アプリケーションの改変をすることなく，I/O View の一貫性を保証しながらチェックポイントを取得する．I/O View とはアプリケーションからのファイルシステムやネットワークの見え方を意味する．障害の前後で I/O View に齟齬があると，再開した時に誤動作を起こす可能性がある．誤動作を起こす例を図 1 に示す．図のように，特定のファイルに依存したメモリイメージから復元する場合，アプリケーションのファイル操作が適切に反映されていないと，矛盾を招く．提案機構では，アプリケーションが発行したシステムコールを監視し，ファイルシステムおよびネットワークサブシステムと協調して，I/O View が一貫するタイミングでチェックポイントを取得する．

表 1　記憶装置の比較

種類	write bandwidth	access latency	IOPS
HDD[24]	204MB/s per drive	2,000µs	185
SSD[22]	460MB/s per drive	56µs	32k
PCIe Flash[23]	1.7GB/s per drive	15µs	375k
PCM	100MB/s per die[11]	150ns[11]	-
STT-MRAM	1GB/s per die[31]	20ns[11]	-
DRAM	1GB/s per die[31]	10ns[11]	-

本論文の貢献は次のとおりである．

- I/O View の一貫性を保証するチェックポイント・リスタート機構を提案する．既存手法と比較すると，提案機構は次の特徴を有する．第一に，提案方式はアプリケーションを改変することなくチェックポイントを取得可能であるため，様々なアプリケーションに対して適用可能である．次に，提案機構は自動的に I/O View を保証しながらチェックポイントを取得するため，ユーザはチェックポイント取得のポリシー等を記述する必要はない．また，PCIe フラッシュや不揮発メモリなど，特定の高速ストレージに依存せずに稼動できる (2，3 章)．
- 提案機構を実現するメカニズムの詳細について述べる．提案機構では，アプリケーションのシステムコールを発行するタイミングでチェックポイントを取得する．I/O View を保証するために，I/O に関連するシステムコールが発行された場合には，システムコールの結果がチェックポイント再開時と一貫するよう I/O の発行を操作する．また，チェックポイント取得のオーバヘッドを削減する最適化手法についても示す (4，5 章)．
- 提案機構を実装し，実験結果を示す．提案方式のプロトタイプは Linux 3.14.9 上で稼働し，これを用いて実験を行なった．7 種類の Real-world なアプリケーションをプロトタイプ上で稼動させたところ，10 から 89%のオーバヘッドが発生するものの，いずれも I/O View の一貫性を保持したままチェックポイントからの再開に成功した．電源障害を模した実験では，プロトタイプによるチェックポイント取得によって，障害復帰後にアプリケーションの挙動が乱れることなく再開できることがわかった (6 章)．

本研究で対象とする障害は，電源障害などの突然のマシン停止である．電源障害は依然としてサービスの信頼性を脅かす一因となっている [12], [13], [25], [30]．ソフトウェアバグやビットフリップといったハードウェアの一時的な故障による障害は本研究の対象外とする．

2. 背景

2.1 高速ストレージ

近年，PCIe フラッシュや不揮発性メモリなどの，読み書き速度の高いストレージが登場している．これらの記憶装置は，従来のストレージである HDD や SSD より高速にデータを転送できる．表 1 に，それぞれのストレージの性能を比較して示す．PCIe フラッシュは，SSD と同じくフラッシュメモリにデータを保存するが，SATA や USB よりも高速なインタフェース PCIe を用いてコンピュータに接続した機器である．不揮発性メモリは，データが揮発せず，バイト単位でアクセス可能な記憶装置の総称で，CPU のロード・ストア命令で読み書きが行える．例えば Phase Change Memory（PCM）や Resistive RAM（ReRAM），Magnetoresistive RAM（MRAM）などがあり，他のストレージよりもアクセスレイテンシが短い．

アプリケーションの入出力処理はストレージや NIC の性能に律速するので，PCIe フラッシュなどにより高速化したチェックポイント取得のオーバヘッドは隠蔽される．例えば HDD にファイルを保存する場合，表 1 の通り書き込み帯域幅は約 200MB/s であり，PCIe フラッシュの 1.7GB/s と比べると 1/8 以下である．また NIC についても，現在ギガビット・イーサネットに対応したものが普及しているが，通信プロトコルの処理などにより 1Gbps 未満の性能になる場合が多い．クラスター上のネットワークの性能を調べた [5] では，ギガビット・イーサネットで接続したノード間の TCP による通信のスループットを計測した結果，350Mbps 以下となった．これらのデバイスへの入出力の合間に，PCIe フラシュや不揮発性メモリにメモリイメージを保存すれば，オーバヘッドを顕在化させることなくチェックポイントを取得できる．

2.2 既存手法

提案機構は，主に電源障害を対象として，チェックポイント・リスタートによる信頼性向上を達成する．同じように，アプリケーションに耐障害性を付与する手法が過去に提案されている [2], [4], [7], [9], [10], [15], [16], [17], [19], [21], [26], [27], [29], [32]．これらの従来手法の比較を表 2 に示す．提案機構と異なり，いずれの手法もアプリケーションに対する透過性と I/O View の一貫性の保証を両立していない．

[4], [9], [16], [17] は，外部のコマンドから指定したアプリケーションについてチェックポイント・リスタートを行う．メモリイメージは，アプリケーションをシグナルなどで停止させてから保存する．アプリケーションの動作によらず，ユーザがコマンドを発行した時や定期的にチェックポイントを取るため，I/O View の一貫性は保証されない．

表 2 チェックポイント・リスタート機構の比較

	アプリケーション透過性	電源障害への耐障害性	I/O View の保証	様々な高速ストレージへの適用性
Linux-CR[9]，Zap[16]	✓	✓	×	✓
Mnemosyne[29]，NV-Heaps[2]	×	✓	✓	×(NVRAM only)
LMC[27]，Rx[19]	×	×	✓	×(NVRAM only)
WSP[15]，ThyNVM[21]	✓	✓	×	×(NVRAM only)
提案機構	✓	✓	✓	✓

利用者が指定したユーザ空間の領域を永続化する手法もある．Mnemosyne[29] や NV-Heaps[2]，Mojim[32] は，API によって指定されたアプリケーションのデータ構造に不揮発性メモリを割り当てる．障害が起きてデータ構造の更新処理が中断すると，CPU キャッシュの消失によって一貫性を損なう可能性がある．これを防ぐために，更新中は Undo ログを書き出し．障害時にロールバックする．NV-process[10] は，ユーザ空間の全領域とカーネル空間内のアプリケーションに関連するデータ構造に不揮発性メモリを割り当てて，耐障害性を実現する．チェックポイントの一貫性のため，プログラミングモデルが限定されている．SoftPM[7] は，アプリケーションのデータ構造について，指定したタイミングでチェックポイントを取る．チェックポイント取得が発行されると，SoftPM が指定されたルートポインタから自動的にデータ構造をたどり，ストレージに書き出す．これらの手法は，永続化するデータの選択やチェックポイント取得の発行などのためにソースコードを改変する必要があり，アプリケーションに対する透過性を持たない．

LMC[27]，Flashback[26]，Rx[19] は，ストレージではなくメインメモリにメモリイメージを保存する手法で，ビット反転などによるソフトウェアエラーへの耐性を実現する．データの揮発する DRAM に保存するため，再起動を必要とする電源切断のような障害には対応していない．また，チェックポイントの取得は API によって発行するので，アプリケーションの改変を要求する．

ThyNVM[21] は，不揮発性メモリを用いたチェックポインティング手法で，メモリコントローラに変更を加えて実現した．定期的にメモリコントローラがユーザ空間のページテーブルを確認，汚れたメモリページを不揮発性メモリにコピーしてチェックポイントを取る．同時に，CPU に通知してレジスタの内容などをも保存する．ハードウェアによって耐障害性を達成するので，アプリケーション透過に適用できる．一方で，ネットワーク I/O などの監視はせず，I/O View の一貫性を考慮していない．

WSP[15] は，電源障害からシステム全体を保護する．メインメモリを全て不揮発性メモリで構成することで，アプリケーションやカーネルのデータを永続的に残す．障害時には，システムに残された電力で CPU レジスタやキャッシュ，デバイスの状態などを不揮発性メモリに退避して，再起動後に復旧する．しかしながら，障害時の残りの電力ではデバイスの状態を保存するのに不十分であることが実験で確認された．従って，復旧時にネットワーク I/O などの整合性は保証されない．

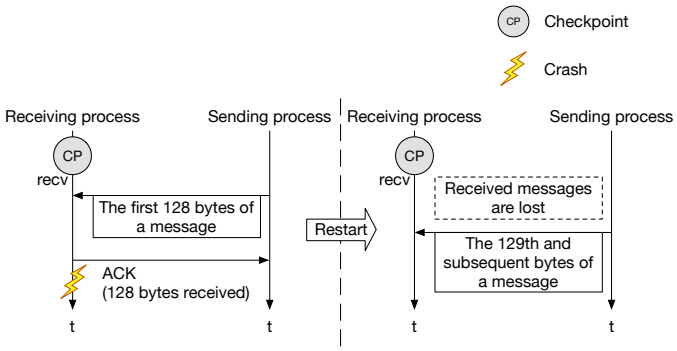

図 2 ネットワーク I/O の矛盾が起きる例

3. 提案機構

提案機構は，次の 3 つを目的とする．

- アプリケーションのソースコードを修正せずに導入できるようにする
- 電源障害や再起動の際，システムダウンの前の状態から実行を再開する能力をアプリケーションに付与する
- 実行再開の前後で，アプリケーション視点でのファイル I/O やネットワーク I/O の整合性を保つ
- PCIe フラッシュや不揮発性メモリといった様々な高速ストレージに適用可能な手法とする．

不整合の例として，アプリケーションが TCP 通信を行っている時に，受信応答したメッセージが電源障害後によって消失してしまうと，再開後に矛盾を生む（図 2）．受信応答を受け取っている通信相手はメッセージを再送しないので，メッセージの完全性は失われて TCP に違反する．提案機構では，このような不整合が起きないようにする．

以上の目的を達成するために，提案機構は以下の条件でチェックポインティングしなければならない．まず，チェックポイントを取っている間は，プロセスコンテキストの更新が停止している必要がある．そうしなければ，メモリイメージに矛盾が生じる．次に，アプリケーションのデータ I/O を監視して適切にメモリイメージに反映されること

を保証する．メモリイメージが保持している進行状況と，ファイルの更新やネットワークメッセージの送受信との整合性を維持する．最後に，チェックポイントの取得や，プロセスコンテキストの停止，データ I/O の監視は，全てアプリケーション透過に行うことが求められる．

そこで提案機構では，アプリケーションが呼ぶシステムコールを監視して適宜チェックポイントを取るようにする．システムコールを発行した時，アプリケーションのプロセスはカーネルモードに移行するので，ユーザ空間の更新が停止した状態でのチェックポイント取得が可能である．また，システムコールの引数からアプリケーションのデータ入出力を監視できる．引数を調べて，メモリイメージへの反映の必要性を判断する．システムコールの種類によっては，実際の関数内だけではなく，別のタイミング・機構でデータ I/O を発生させるものもある．例えば，accept() システムコールは通信相手から接続を受けて，ソケットを割り当てた後ユーザモードに戻るが，それ以降もメッセージ受信やそれに対する応答が実行される．こうしたデータ I/O についても監視を行う．システムコールの監視などの提案機構の機能は，全てカーネル層で動作するのでアプリケーションに対する透過性を持つ．

提案機構を適用したアプリケーションは，以下の手順でシステムコールを実行する．

(1) システムコールを発行して，カーネルモードに移行
(2) システムコールの引数からデータ I/O を監視，チェックポイントの必要性を判断
(3) 必要性があればチェックポイントを取得
(4) システムコールを実行後，ユーザモードに復帰

メモリイメージは，不揮発性メモリかブロックストレージ（PCIe Flash）に保存される．どちらのメモリイメージも内容は同じで，ユーザ空間のメモリマップやデータ，展開しているファイル，プロセス間の親子関係，ネットワークの状態などを保存する．不揮発性メモリに保存する場合は，これらのデータを 1 つのデータ構造として保持する．そのデータ構造のルートポインタを不揮発性メモリ内の基底から始まるリスト構造に書き込み，復元する際はリストからメモリイメージを参照する．PCIe フラッシュに保存する場合，メモリイメージのデータをシーケンシャル化したファイルとして保存する．

4. I/O View の一貫性の保証

ファイル I/O とネットワーク I/O の整合性維持について述べる．

4.1 ファイルシステムの一貫性

ファイルシステムとメモリイメージとが不整合になりうるファイルシステムの操作として，ファイル参照，ディレクトリ操作，ファイルのメモリマッピングの 3 つがある．

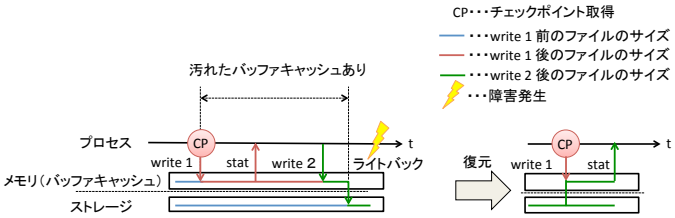

図 3 ファイルの参照に関する矛盾が起きる例

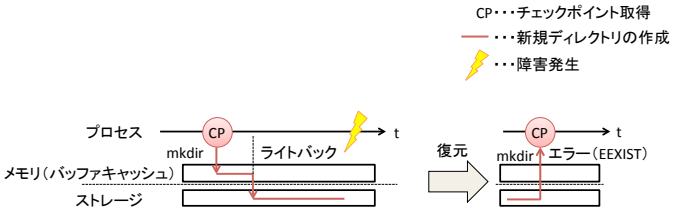

図 4 ディレクトリのに操作に関する矛盾が起きる例

提案機構では，これらに関連するシステムコールやカーネルの機能を監視して，適宜チェックポイントを取得，メモリイメージの整合性を保つ．それぞれの場合について，起きうる矛盾と対応策を述べる．

ファイル操作：問題

stat() システムコールや read() システムコールでファイルを参照した後に write() システムコールなどで同ファイルを更新した時に，いずれのシステムコールでもチェックポイントを取らなかった場合，齟齬が生じる可能性がある．矛盾が起きる例を図 3 に示す．stat() で参照したファイルのサイズが後続の write() で変化し，ストレージに反映された場合，このプロセスを復元して stat() を発行すると障害発生前とは異なるサイズが返される．

ファイル操作：方策

ファイルを参照した後の，同じファイルに書き込むシステムコール発行時にチェックポイントを取れば整合性を維持できる．図 3 の write2 がそれに該当する．この条件を満たすシステムコールの発行を検出するために，ファイルのタイムスタンプを利用する．ファイルのタイムスタンプには read() などで更新される最終アクセス時刻が記録されており，いつプロセスがファイルを参照したのかがわかる．write() や truncate() で変更するファイルの最終アクセス時刻が，最後にチェックポインティングした時刻より後であれば，矛盾を起こしうるシステムコールとみなす．この場合必ずチェックポイントを取り，齟齬が生じるのを防ぐ．

ディレクトリ操作：問題

障害発生前と復元時に重複してディレクトリの操作を行うシステムコールを実行すると，矛盾が発生する可能性がある．例えば図 4 のように，mkdir() を発行してディレクトリを作成し，ストレージにライトバックされたのち障害が発生した場合，プロセスを復元して mkdir() を再度実行するとエラー値 EEXIST が戻される．EEXIST は作成し

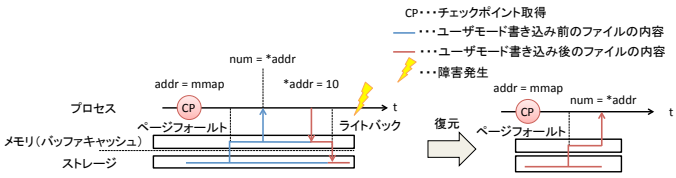

図 5 メモリマップトファイルに関する矛盾が起きる例

ようとしたディレクトリが既に存在することを意味する．mkdir() の他，rename() や unlink() なども重複したときにエラー値を返す．これらのディレクトリを操作する処理が重複したときに出力されるエラー値は，システムコールの種類によって決まっている．

ディレクトリ操作：方策

再開時に誤ったエラー値が戻るのを防ぐため，ディレクトリに書き込むシステムコールを発行するときは，必ずチェックポイントを取り，メモリイメージの一部として戻り値の情報も保存する．記録する情報は，戻り値が重複時のエラー値になるか否かであり，システムコールの引数から調査する。プロセスがディレクトリを操作するシステムコールから再開するとき，戻り値が重複時のエラー値になったならばメモリイメージの記録と照合する。もしメモリイメージに記録された戻り値が成功を示すなら，代わりに成功を返す。

メモリマップトファイル：問題

ファイルをユーザ空間にメモリマップすると，ユーザモードから直接読み書きできる。図 5 のように，ファイルに書き込んだ後に復元して書き込む前の命令に戻ると矛盾が生じる。しかし，図で行っている読み書きはすべてユーザモードでの処理なので，システムコールは介さない。システムコールとは別の，カーネルが介入できるタイミングで整合性を確保する必要がある。

メモリマップトファイル：方策

齟齬の発生を回避するため，デマンドページング方式でファイルがマップされたメモリ領域にページを割り当てたときにチェックポイントを取得する。チェックポイントを取るのは，ページが割り当てられてファイルのデータがページにコピーし終わった後である。当該ページもメモリイメージの一部として保存する。プロセスの復元時，ファイルがマップされていたメモリ領域については再度ファイルをメモリマップし，メモリイメージからチェックポインティングしたときのデータを復帰させる。この操作によって，障害発生前に行われた書き込みをなかったことにできる。

4.2 ネットワークの一貫性

チェックポイント・リスタートに伴うネットワークの問題と，提案機構で講じた対応策を説明する．

システムコール発行時のチェックポイントだけでは，復元後のネットワーク I/O に矛盾が起きる可能性がある．例えば図 2 のような矛盾を引き起こす受信応答の制御は，アプリケーションを介さない．提案機構では，カーネルのネットワーク機構も監視して，メモリイメージの整合性を維持する．

本研究では，多くのネットワークで稼働する TCP/IP に焦点を当てて提案機構を設計する．TCP/IP は 4 つのレイヤで構成され，上から順にアプリケーション層，トランスポート層，ネットワーク層，リンク層と呼ぶ．上位層は下位層がプロトコル通りに動作することを期待する．TCP/IP のうち，アプリケーションが観測するのはアプリケーション層と，トランスポート層への要求・応答のみである．従って，トランスポート層でプロトコルに則したサービスを保証すれば，アプリケーションを正確に復元できる．

トランスポート層の通信プロトコルとしては，UDP と TCP が普及しており．多くのアプリケーションは 2 つのうちどちらかを使う．UDP は応答速度を重視しており，転送データの信頼性や順番を保証しない．欠陥のあるデータに対する対応はアプリケーションに求めれらる．よって，チェックポイント・リスタートによって，メッセージの重複や欠損が起きてもプロトコルには反さず，復元の正確性は保たれる．一方で TCP はメッセージの信頼性，順番を保証するプロトコルなので，復元の前後でメッセージの送受信について一貫性を維持しなければならない．

そこで提案機構では，システムコールに加えて，トランスポート層の TCP を遂行するカーネルの処理についても，振る舞いを監視して適宜チェックポイントを取る．以降，復元時に起こりうる TCP での誤作動と，それを回避する方法について説明する．

ネットワーク受信：問題

TCP のメッセージ内のデータには，1 バイトごとに通し番号（シーケンス番号）が割り振られ，受信側がどこまでのシーケンス番号のデータを受け取ったかを示す ACK を返すことで，メッセージの信頼性，順序性を保証する．例えば受信側がシーケンス番号 128 を返すことは，128 バイトまでのデータを受け取れたことを意味する．また送信側も，シーケンス番号を用いて送るデータが何バイト目かを知らせる．具体的には，メッセージはセグメントと呼ばれる単位に分割して送られるので，各セグメントのヘッダにデータの先頭バイトのシーケンス番号を記述する．

セグメントの受信を復元するとき，矛盾を起こす可能性がある．図 2 を例にして説明する．図のように，セグメントを受け取り，対応するシーケンス番号を ACK で返した後に障害が発生すると，メモリ上のセグメントは消失する．しかし送信側からは，ACK によって相手の受け取りが完了したと解釈するので，それ以降のデータをセグメントに載せて送る．結果として，復元時に受信側でメッセージの欠落が生じる．

ネットワーク受信：方策

セグメントを受け取った時，ACK を返す前にチェックポイントを取ることで，受け取り完了を通知したメッセージが揮発しないようにする．メモリイメージを保存するときに，セグメントの内容，シーケンス番号も含めて不揮発性メモリに書き込む．チェックポイント取得完了後，もしくは復旧後，ACK の返信からプロセスを再開する．ACK 内のシーケンス番号までのメッセージは必ず永続するので，受信プロトコルの一貫性を維持できる．

ネットワーク送信

データのシーケンス番号が復元の前後で一貫していれば，正確にプロセスを再開できる．同一のセグメントを重複して送っても，受信側がシーケンス番号から重複を検知して破棄するので，アプリケーションの動作に矛盾は起きない．従って，トランスポート層の処理を監視する必要はなく，システムコール発行時などのチェックポイントを取得する際に TCP に関連するデータ構造を保存すれば良い．保存するべき情報としては，データに割り振るシーケンス番号などが該当する．

5. 最適化

この章では，提案機構に施した最適化について説明する．最適化によって，アプリケーションにチェックポインティングを適用した時に発生する，性能に対するオーバヘッドを軽減した．

提案機構導入によって，アプリケーションがチェックポイントを取得する時にダウンタイムが生じる．ダウンタイムの要因は，アプリケーションの停止とメモリコピーの 2 つである．上述したように，チェックポイントを取る時，アプリケーション中の全てのプロセスを停止する．プロセスの停止は，システムコールなどのチェックポイント取得を必要とする動作をしたプロセスから，シグナルなどを用いて要求する．しかし，即時に停止要求が処理されるとは限らず，各プロセスが要求に応じて止まるまでの間，ダウンタイムが発生する．例えば，ブロック I/O の完了を待つプロセスは，該当するブロック I/O が終わって初めて，停止要求に従う．もう 1 つのオーバヘッドの原因であるメモリコピーは，停止した後，メモリイメージを保存するときに行われる．アプリケーションを正確に再現するために，全プロセスのユーザ空間をストレージに保存するため，アプリケーションのメモリ消費量に比例してコピーにかかる時間が大きくなる．

本研究では次の 2 つのアプローチからオーバヘッドの軽減を図る．1 つはメモリコピー高速化，もう 1 つはアプリケーションの停止回数の低減である．

5.1 メモリコピーの高速化

メモリコピーの高速化としては以下の 2 つを行った．

- メモリのコピー量削減
- メモリコピーの並列化

メモリ上のデータをコピーする量を減らすよう，対象のアプリケーションのユーザ空間について，前回のチェックポイントとの差分のみを保存する．具体的には，前回のチェックポイント以降に汚れたメモリページを抽出して，ストレージに書き出す．これを実現するために，ユーザ空間に一時的な書き込み保護を適用して，アプリケーションのメモリ書き込みを追跡する．メモリページ上のデータを上書きする時，プロテクションフォールトが発生するので，対象をページ番号を記録してから書き込み保護を解除する．次回のチェックポイント取得時に，記録されたメモリページをメモリイメージの一部としてコピーすることで，差分の保存を達成する．

プロセスのユーザ空間のチェックポイントについて，単独のスレッドがコピーを行うと，プロセス数に比例してコピー時間が増大する．これを防ぐために，なるべくメモリコピーを並列化する．アプリケーションがチェックポイントを取得する時，それぞれのプロセスに自身のコンテキストのコピーをさせる．メモリイメージの保存の手順を以下に示す．

(1) チェックポイントを要する処理を行ったプロセス A が他のプロセスをカーネルモードで停止
(2) プロセス A は，（A も含めて）各プロセスに自身のコンテキストを保存するように指示
(3) プロセス A は，他のプロセスのコンテキスト保存が終了するまで待機
(4) 各プロセスのコンテキストを一括して 1 つのメモリイメージとして保存
(5) プロセス A が，各プロセスに再開信号を送信

5.2 アプリケーションの停止回数削減

アプリケーションの停止回数を減らすために，以下のことを行った．

- 監視するシステムコールを限定的にする
- TCP における ACK の返信処理について，アプリケーションを停止せずにメモリイメージの整合性を維持する
- ディレクトリ操作の結果のキャッシング

監視するシステムコールの限定

提案機構では，アプリケーションが発行するシステムコールなどを監視してメモリイメージの整合性を維持する．ここで，監視が不要なシステムコールについては，アプリケーションを停止せずに実行させることで，オーバヘッドの発生を避ける．監視が不要なものとは，データ I/O を伴わないシステムコールである．その処理内容にかかわらず，I/O View の整合性に影響しない．これはシステムコールの作用が，障害発生と共に消失するためである．例

えば，mprotect() システムコールによってメモリ領域のアクセス権限が変わっても，ファイルシステムなどの永続的なデータには反映されないので，これに応じてメモリイメージを更新しなくても良い．

TCP の非同期チェックポインティング

TCP の ACK の返信処理について，アプリケーションの停止を伴わなずにメモリイメージの整合性を維持することで，停止回数を減らす．上述したように，ACK を返信する時に，受信バッファを保存しないと矛盾が起きる可能性がある．しかし，返信処理を行うトランスポート層はアプリケーションから観測されないので，アプリケーションの進行状態を保持するメモリイメージとは独立したデータとして，受信バッファの内容をストレージに保存しても，整合性は失われない．そこで TCP の ACK 返信処理では，アプリケーションを停止せずに，受信バッファのみをメモリイメージとして保存後，返信するようにする．

ディレクトリ操作の結果のキャッシング

アプリケーションを停止せずにディレクトリ操作の戻り値をメモリイメージに反映させる．4.1 章で述べたように，ディレクトリ操作を矛盾なく再開させるために，戻り値が重複時のエラー値に一致するか確認する．この際に最新のチェックポイントを取得する代わりに，調べた値を前回のチェックポイントで保存したメモリイメージに追記することで，アプリケーションの停止を防ぐ．アプリケーションが前回のチェックポイントから再開する場合，ディレクトリ操作を行う時にメモリイメージを確認する．戻り値が記録されていれば，それに従ってアプリケーションに値を返す．

6. 評価

提案機構を評価するため，提案機構のプロトタイプに対して実験を行った．最初にマイクロベンチマークを実行して，ファイル I/O やネットワーク I/O に対するチェックポインティングのオーバヘッドを計測する．次に，memcached などのアプリケーションに提案機構を適用して，性能に与える影響を調べる．

6.1 実験環境・方法

この章で述べる実験は，全て同じ計算機上で実施した．計算機の OS は，Linux 3.14.9 である．また，CPU の Intex Xeon E3-1270（3.4GHz，4 コア，8 スレッド，キャッシュ 8MB）を 8 個，メモリを 32GB 搭載している．ローカルディスクとして 500GB の HDD を備える．この環境に提案機構のプロトタイプを実装し，各種のベンチマークを行う．

提案機構では NVRAM を利用するが，現在市販されている製品はない．そこで実験では，計算機の DRAM を不揮発性メモリとみなしてチェックポインティングに用いる．不揮発性メモリへの書き込みは，DRAM と比べて遅い．チェックポイント取得時には不揮発性メモリへのメモリコピー時に遅延命令を挿入し，書き込み速度をエミュレートする．遅延の時間は，表 1 に記載する各記憶装置の書き込み帯域幅に従う．

6.2 マイクロベンチマーク

ファイル I/O やネットワーク I/O を実行した時の，提案機構のオーバヘッドを調べる．提案機構のチェックポイント取得回数は，ファイル書き込みや TCP セグメント受信の頻度によって変わる．マイクロベンチマークでは，これらのパラメータに対する変化の度合いも測る．

write() システムコールによってファイル書き込みを繰り返すマイクロベンチマークを実行し，オーバヘッドを調べた．複数のスレッドで，それぞれ別のファイルに追記する．全スレッドの合計書き込み量は 4GB である．実験では，スレッド数と書き込み回数の両方の変化に対する性能の変化を調べた．ファイル更新に伴うチェックポイント取得では，アプリケーション全体を停止させるため，スレッド数が多いほど停止の影響が大きくなる．

- スレッド数についての計測：1 回の write() システムコールで要求する書き込み量を 4KB に固定して，スレッド数を 1，2，4，8 に変えてそれぞれ計測
- 書き込み回数についての計測：4 スレッドで実行し，1 回の write() システムコールで要求する書き込み量を 1KB，2KB，4KB，8KB に変えてそれぞれ計測

ネットワーク I/O のチェックポインティングには，recv システムコールを用いてオーバヘッドを計測する．TCP セグメント受信時にセグメントバッファを永続化するので，受信サイズに比例してオーバヘッドが増す．受信サイズを 256MB，512MB，1GB，2GB にした時のそれぞれの実行時間を測った

以上の計測を，メモリイメージの保存先として NVRAM，PCIe Flash にした場合それぞれについて行う．NVRAM は，PCM，ReRAM，MRAM の 3 つをエミュレートする．

図 6 から 7 にそれぞれのベンチマークの実行時間を示す．両方とも，チェックポイントを取らない場合の実行時間で正規化した．スレッド数，書き込み回数，メッセージ受信量に比例してオーバヘッドが大きくなり，最大で 89%のオーバヘッドが発生した．スレッド数と書き込み回数に比べると，メッセージ受信量に対するオーバヘッドの増分が小さい．これはメッセージ受信に対するチェックポインティングは非同期に行うため，ベンチマーク内の処理と並列化したからである．

6.3 マクロベンチマーク

memcached や apache などのアプリケーションに提案機構を適用した時のオーバヘッドを計測する．次の 7 つ

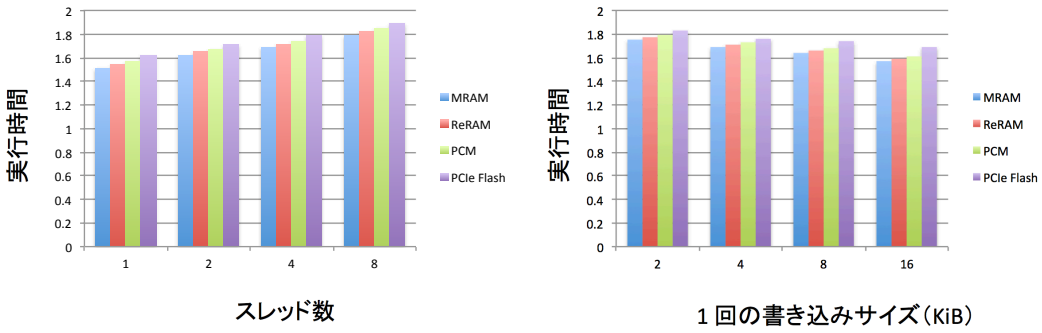

図 6　スレッド数と書き込みサイズに対するファイル書き込みの実行時間

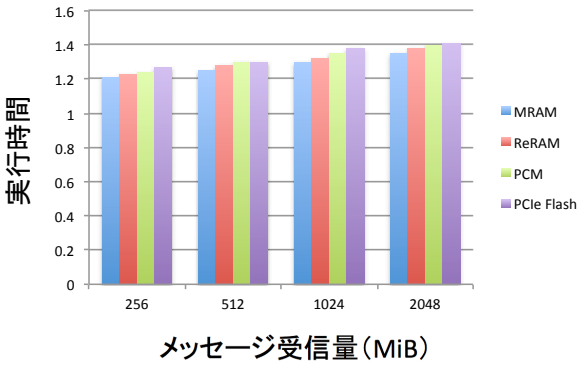

図 7　受信セグメント量に対するエコーサーバの実行時間

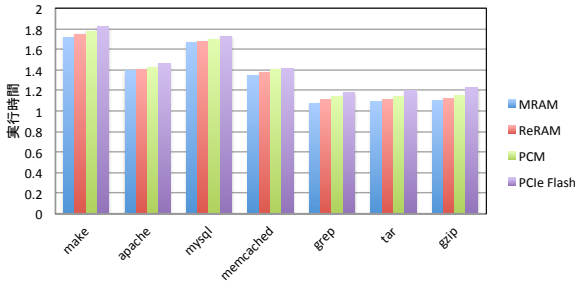

図 8　提案機構を導入したアプリケーションの実行時間

のワークロードを実行した．**Make** は，Linux カーネルのコンパイルである．**Apache** では，Apache Bench により Apache に同時接続数 4 で合計 1 万のリクエストを発行する．**MySQL** では，SysBench[1] を用いて，500MB のデータテーブルに対して合計 1 万のトランザクションを実行する．**Memcached** では，memaslap[14] によるワークロードを行う．4 スレッドで get と set を比率 3:7 で合計 1 万回実行する．キーとバリューのサイズはそれぞれ 128B，4KB である．**grep**，**gzip**，**tar** はそれぞれ 1 スレッドで 4GB のファイルに対してコマンドを実行する．

図 8 にそれぞれのベンチマークの実行時間を示す．いずれの値もデフォルトの実行時間で正規化されたものである．また，MySQL と Apache でのチェックポイント取得回数を，取得要因別に計測したもの表 3，4 にのせる．それぞれのベンチマークで 10〜82%のオーバヘッドが見れ

表 3　mysql のチェックポインティング回数

	回数
ファイル書き込み	4213
ページフォールト	10
TCP	2324
合計	6547

表 4　apache のチェックポインティング回数

	回数
ファイル書き込み	24
ページフォールト	9
TCP	1941
合計	1974

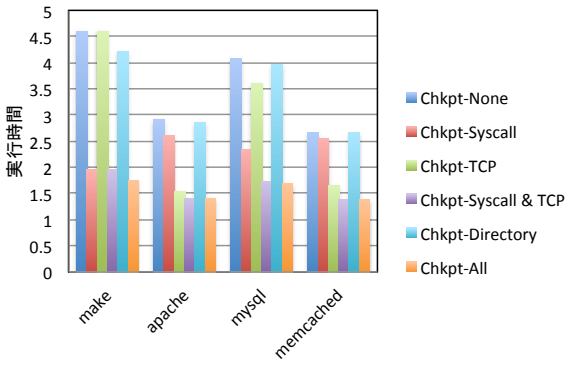

図 9　提案機構の実行時間

らた．grep，tar および gzip は単一のスレッドで実行したため，チェックポインティングの同期に伴うダウンタイムが短く，他に 4 つに比べてオーバヘッドが小さい．表 4 のように memcached や apache は複数スレッドで動作するものの，チェックポイント取得の主な要因はネットワークのメッセージ受信であり，非同期にメモリイメージを更新するので make や mysql より性能劣化が小さい．一方で make や mysql はファイル書き込みを多く発行するため，大きいオーバヘッドが発生した．

6.4　最適化の効用

提案機構で実施した以下の最適化について評価する．

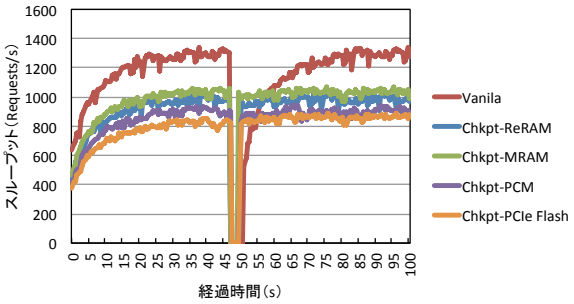

図 10 Web サーバのスループット

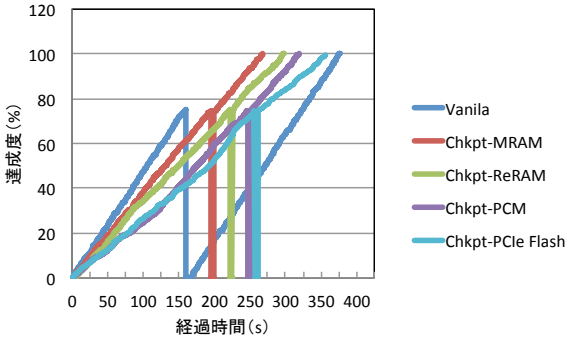

図 11 ディスクスクラッビングの達成度

(1) 追跡するシステムコールの限定
(2) TCP の非同期チェックポインティング
(3) ディレクトリ操作のメモライゼーション
これらを有効・無効にした次の 5 つの設定でベンチマークを実行し，性能を計測した．

- Chkpt-None：すべての最適化を無効
- Chkpt-Syscall：最適化 1 を有効
- Chkpt-TCP：最適化 2 を有効
- Chkpt-Syscall&TCP：最適化 1 と 2 を有効
- Chkpt-Directory：最適化 3 を有効
- Chkpt-All：全ての最適化を有効

ワークロードは，make，apache，mysql，memcached の 4 つを行う．

図 9 はそれぞれの設定のベンチマークの実行時間である．チェックポイントを取らずに実行した時の時間で正規化している．make や mysql のように，書き込みが多いワークロードでは，追跡するシステムコールの限定による効果が見られた．それぞれ，約 58%，42%のオーバヘッド削減を達成している．これは，それぞれの処理で write システムコールが多く発行されており，同期して監視する対象を絞ることによってオーバヘッドを低減している．一方で，memcached などの，ネットワーク I/O が多く，ファイル書き込みが少ないワークロードでは，この最適化の効用が小さい．これらのワークロードでは，TCP 受信時のチェックポイント取得を非同期に行う最適化による効用が大きく，50%近いオーバヘッドを削減した．

6.5 障害の挿入

障害が発生した時に，アプリケーションの性能に対して提案機構が与える影響を調べる．以下のベンチマークの実行中に，kexec による擬似的な障害を挿入し，スループットの変化を計測する．**Web サーバ**では，Apache サーバに用意した 8KB の Web ページを 10 万個用に対してランダムに HTTP リクエストを繰りかえす．Web ページはバッファサイズ 256MB の memcached によってキャッシングする．スループットがピークに達した後，kexec を実行してチェックポインティングの有無による性能の違いを比較する．**ディスクスクラッビング**では，ファイルシステムのスクラッビングを行う．スクラッビングはファイルシステム内の全データに対して整合性のチェックを行う処理である．経過時間ごとの達成度を計測し，達成度が 75%に達した時に kexec によって中断させる．

図 10, 11 はそれぞれのベンチマークの結果である．Web サーバではチェックポインティングにより，障害後の復旧時にデフォルトと比べて高速にスループットがピークに達している．提案機構では復元から即座にピーク性能に戻っているが，デフォルトではピーク性能に戻るのに 30 秒程度要している．これは，Web ページのキャッシュが保持されたためである．また，ディスクスクラッビングでは，障害による進行の損失を抑えて，デフォルトより早く処理を終えており，20〜100 秒の実行時間削減を達成した．しかしながら，チェックポインティングのオーバヘッドにより，Web サーバのピーク時のスループットや，スクラッビングの進行速度ではデフォルトに劣る．Web サーバのスループットでは，ピーク時を比較すると 23〜34%低下している．

7. 関連研究

7.1 フォールトトレランス

電源障害やソフトウェアエラーに対して耐障害性を付与する手法として，[2], [4], [9], [15], [16], [17], [19], [26], [27], [29], [32] が提案されている．これらの手法は提案機構と異なり，いずれの手法もアプリケーションに対する透過性と I/O View の一貫性の保証を両立していない．

7.2 高速ストレージの利用

チェックポインティング以外への高速ストレージの利用として，独自のファイルシステム BPFS[3] や Aerie[28], Pelley らのデータベース管理システム [18] も存在する．記憶装置としては DRAM と不揮発性メモリのみを使用し，前者をユーザ空間に，後者をファイルシステムやデータベース管理システムに割り当てる．アプリケーションは，不揮発性メモリ中のファイルやデータベースにバイト単位

でアクセスできるため，ストレージへのブロック単位の読み書きで生じていた余分なオーバヘッドを緩和できる．

8. 結論

本研究は高速ストレージを用いたチェックポイント取得手法を提案した．アプリケーション透過に I/O View の一貫性を保証する．ファイルシステムやネットワークに影響を与えるカーネル空間の処理を監視して，メモリイメージの整合性を保つようにした．計算機実験を行い，アプリケーションの性能に対する提案機構の影響を調査した．実験では，最大で約 90%のオーバヘッドでチェックポインティングできることを確認した．

参考文献

[1] akopytov: SysBench. https://github.com/akopytov/sysbench.

[2] Coburn, J., Caulfield, A. M., Akel, A., Gupta, R. K., Jhala, R. and Swanson, S.: NV-Heaps: Making Persistent Objects Fast and Safe with Next-Generation, Non-Volatile Memories, *Proceedings of the 16th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 105–118 (2011).

[3] Condit, J., Nightingale, E. B., Frost, C., Ipek, E., Lee, B., Burger, D. and Coetzee, D.: Better I/O Through Byte-Addressable, Persistent Memory, *Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, pp. 133–146 (2009).

[4] Emelyanov, P.: CRIU: Checkpoint/Restore In Userspace Main Page. http://www.criu.org.

[5] Farrell, P. A. and Ong, H.: Communication performance over a gigabit Ethernet network, *Proceedings of the 19th IEEE INTERNATIONAL PERFORMANCE, COMPUTING, AND COMMUNICATIONS CONFERENCE*, pp. 181–189 (2000).

[6] Fiala, D., Mueller, F., Ferreira, K. and Engelmann, C.: Mini-Ckpts: Surviving OS Failures in Persistent Memory, *Proceedings of the 2016 International Conference on Supercomputing*, pp. 7:1–7:14 (2016).

[7] Guerra, J., Mrmol, L., Campello, D., Crespo, C., Rangaswami, R. and Wei, J.: The design and implementation of Zap: a system for migrating computing environments, *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pp. 361–376 (2002).

[8] Huang, H. F. and Jiang, T.: Design and implementation of flash based NVDIMM, *Nonvolatile Memory Systems and Applications Symposium*, pp. 1–6 (2014).

[9] Laadan, O. and Hallyn, S. E.: Linux-CR: Transparent Application Checkpoint-Restart in Linux, *Proceedings of the Linux Symposium*, pp. 159–172 (2010).

[10] Li, X., Lu, K., Wang, X. and Zhou, X.: NV-process: A Fault-Tolerance Process Model Based on Non-Volatile Memory, *Proceedings of the 3rd ACM SIGOPS Asia-Pacific conference on Systems*, pp. 1:1–1:6 (2011).

[11] Meena, J. S., Sze, S. M., Chand, U. and Tseng, T.-Y.: Overview of emerging nonvolatile memory technologies, *Nanoscale Research Letters*, Vol. 9, No. 1, p. 1 (2014).

[12] MILLER, R.: Data Center Outage Cited in Visa Downtime Across Canada (2013). http://www.datacenterknowledge.com/archives/2013/01/28/data-center-outage-cited-in-visa-downtime-across-canada/

[13] MILLER, R.: Power Outage Knocks DreamHost Customers Offline (2013). http://www.datacenterknowledge.com/archives/2013/03/20/power-outage-knocks-dreamhost-customers-offline/.

[14] Mingqiang Zhuang and Brian Aker: memaslap - Load testing and benchmarking a server. http://docs.libmemcached.org/bin/memaslap.html.

[15] Narayanan, D. and Hodson, O.: Whole-system persistence, *Proceedings of the 17th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 401–410 (2015).

[16] Osman, S., Subhraveti, D., Su, G. and Nieh, J.: The design and implementation of Zap: a system for migrating computing environments, *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pp. 361–376 (2002).

[17] Paul H Hargrove and Jason C Duell: Berkeley Lab Checkpoint/Restart (BLCR) for Linux Clusters, *Journal of Physics: Conference Series*, Vol. 46, No. 1, p. 494 (2006).

[18] Pelley, S., Wenisch, T. F., Gold, B. T. and Bridge, B.: Storage Management in the NVRAM Era, *Proceedings of the VLDB Endowment*, Vol. 7, No. 2, pp. 121–132 (2014).

[19] Qin, F., Tucek, J., Zhou, Y. and Sundaresan, J.: Rx: Treating bugs as allergies—a safe method to survive software failures, *Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, pp. 235–CRIU–248 (2005).

[20] Raoux, S., Burr, G. W., Breitwisch, M. J., Rettner, C. T., Chen, Y.-C., Shelby, R. M., Salinga, M., Krebs, D., Chen, S.-H., Lung, H.-L. and Lam, C. H.: Phase-Change Random Access Memory: A Scalable Technology, *IBM Journal Research and Device*, Vol. 52, No. 4.5, pp. 465–479 (2008).

[21] Ren, J., Zhao, J., Khan, S., Choi, J., Wu, Y. and Mutlu, O.: ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems, *Proceedings of the 48th International Symposium on Microarchitecture*, pp. 672–685 (2015).

[22] SanDisk Corporation: CloudSpeed Ultra Gen. II SATA SSD. https://www.sandisk.com/content/dam/sandisk-main/en_us/assets/resources/enterprise/data-sheets/cloudspeed-ultra-genII-sata-ssd-datasheet.pdf（2016 年 5 月 25 日閲覧）.

[23] SanDisk Corporation: Fusion ioMemory SX350 PCIe Application Accelerators. https://www.sandisk.com/content/dam/sandisk-main/en_us/assets/resources/enterprise/data-sheets/fusion-iomemory-sx350-pcie-application-accelerators-data pdf.

[24] Seagate Technology PLC: Cheetah 15K.7. http://www.seagate.com/files/docs/pdf/datasheet/disc/cheetah-15k.7-ds1677.3-1007us.pdf（2016 年 5 月 25 日閲覧）.

[25] VERGE, J.: Internap Data Center Outage Takes Down Livestream, StackExchange (2014). http://www.datacenterknowledge.com/archives/2014/05/16/internap-data-center-outage-takes-livestream-stackexchan

[26] Vogt, D., Giuffrida, C., Bos, H. and Tanenbaum, A. S.: Flashback: a lightweight extension for rollback and deterministic replay for software debugging, *Proceedings*

of the 2004 USENIX Annual Technical Conference, pp. 29–44 (2004).

[27] Vogt, D., Giuffrida, C., Bos, H. and Tanenbaum, A. S.: Lightweight Memory Checkpointing, *Proceedings of The 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 474–484 (2015).

[28] Volos, H., Nalli, S., Panneerselvam, S., Varadarajan, V., Saxena, P. and Swift, M. M.: Aerie: Flexible File-System Interfaces to Storage-Class Memory, *Proceedings of the 9th European Conference on Computer Systems*, pp. 14:1–14:14 (2014).

[29] Volos, H., Tack, A. J. and Swift, M. M.: Mnemosyne: Lightweight Persistent Memory, *Proceedings of the 16th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 91–104 (2011).

[30] WOLFFRADT, R. S. V.: Fire in your Data Center: No Power, No Access, Now What? (2014). `http://www.govtech.com/state/Fire-in-yourData-Center-No-Power-No-Access-Now-What.html`.

[31] Zhang, Y. and Swanson, S.: A Study of Application Performance with Non-Volatile Main Memory, *Proceedings of the 2015 IEEE Symposium on Mass Storage Systems and Technologies*, pp. 1–10 (2015).

[32] Zhang, Y., Yang, J., Memaripour, A. and Swanson, S.: Mojim: A Reliable and Highly-Available Non-Volatile Memory System, *Proceedings of the 20th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 3–18 (2015).