システムソフトウェアと
オペレーティング・システム 89－１０
システム評価 2－１０
（２００２．２．１５）

Tender においてデータの永続化を高信頼化する手法

稲本 慎司† 谷口 秀夫‡

†九州大学大学院システム情報科学府 ‡九州大学大学院システム情報科学研究院

計算機では、データを再利用するために、データを外部記憶装置に格納し、データを永続化する。しかし、データの永続化処理が途中で異常終了した場合、データを再利用できない状態に陥る可能性がある。このような状態に陥らないために、データの永続化を高信頼化する必要がある。そこで、本論文では、*Tender* オペレーティングシステムにおいてデータの永続化を高信頼化する手法について述べる。具体的には、メモリ領域の管理とデータを永続化するための管理単位である永続ユニットを説明し、永続ユニットへの書き出し処理を高信頼化する手法について述べる。さらに、実装と評価により、提案手法の性能を示す。

Method which Persist Data Certainly on *Tender*

Shinji INAMOTO and Hideo TANIGUCHI

Graduate School of Information Science and Electrical Engineering, Kyushu University

Computer keeps data in nonvolatile storage to persist it. But, if processing which persist data ends in failure, computer cannot reuse it. To prevent such thing, computer needs to persist data more certainly. So, we propose the method which persist data more certainly on *Tender* (The ENduring operating system for Distributed EnviRonment) Operating System. In the concrete we describe memory control and "persistent unit" which is control unit for persisting data. And, we describe the method which keeps data in "persistent unit" more certainly. By implementation and evaluation, we show performance of the method furthermore.

1 はじめに

計算機では、データを処理して得た結果を電子化データとして再利用するため、従来からデータの永続化機能を実現している。データを永続化するためには、データを不揮発性記憶媒体である外部記憶装置に格納する必要がある。この格納されたデータは、明示的に削除されない限り、外部記憶装置上に存在し、再利用可能でなければならない。外部記憶装置上に存在するデータを再利用するためには、データの実体を外部記憶装置に格納するだけでなく、データに付与された名前やデータの格納位置などの管理情報も外部記憶装置に格納し、管理する必要がある。このため、データを永続化する場合、外部記憶装置への出力処理を複数回行う必要がある。これらのデータの実体や管理情報を更新している際に、システムが異常終了した場合、永続化したはずのデータを失う可能性がある。例えば、データの実体の一部が破壊されたり、部分的に更新された場合、データの実体を再利用できな

い状態に陥る可能性がある。また、データの管理情報が破壊された場合、当該データを再利用できなくなるだけでなく、他のデータも再利用できなくなる可能性がある。

そこで、***Tender*** オペレーティングシステム[1]においてデータの永続化を高信頼化する手法を提案する。この手法は、データの実体の更新中にシステムが異常終了した場合、外部記憶装置上のデータを永続化処理開始前の状態にし、データの再利用を可能にする。また、データの管理情報の更新中にシステムが異常終了した場合でも、システム再起動時にデータの管理情報を復元する。

本論文では、***Tender*** オペレーティングシステムにおけるメモリの領域と外部記憶装置の領域の管理について述べ、データの永続化の信頼性を向上させる手法について述べる。また、提案手法がデータの永続化処理性能に及ぼす影響を測定し、評価する。

2 *Tender* オペレーティングシステム

2.1 概要

Tender オペレーティングシステム (以降、***Tender*** と略す) では、OS が制御し管理する対象の単位を資源と呼び、全ての資源に資源識別子を付与する。

Tender におけるメモリ関連資源について、図 1 に示す。資源「仮想空間」とは、特定のアドレス領域を持つ仮想的な空間であり、アドレス変換表に相当する。資源「仮想ユーザ空間」は、メモリイメージを仮想化した領域である資源「仮想領域」をユーザ空間用の資源「仮想空間」に貼り付けることで作成できる。資源「仮想領域」の実体は、実メモリまたは外部記憶装置上に存在する。「貼り付ける」とは、仮想アドレスを実アドレスに対応付けすることであり、具体的には、当該の仮想アドレスに対応するアドレス変換表のエントリに、実アドレスまたは外部記憶装置のアドレスを設定する。資源「仮想カーネル空間」は、資源「仮想領域」を OS 用の資源「仮想空間」に貼り付けることにより作成される。資源「プレート」[2]とは、資源「仮想カーネル空間」や資源「仮想ユーザ空間」上のデータを永続的に保持する機能を持つ資源である。図 1 に示した例では、資源「プレート」

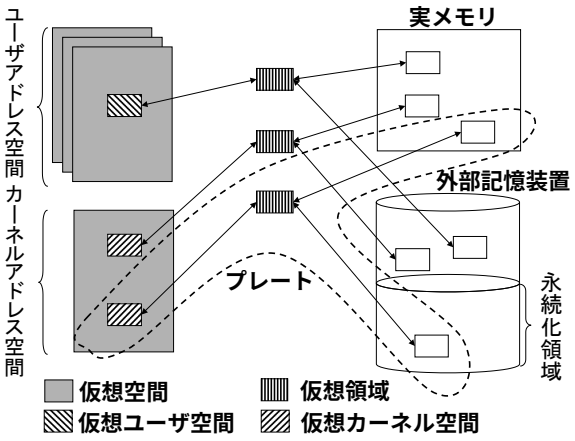

図 1 メモリ関連資源の関係

は資源「仮想カーネル空間」上のデータを永続化している。データは、外部記憶装置上の永続化領域を用いて永続化される。この永続化領域は、一時的なデータの格納ではなく、システム再起動後もデータを再利用できるようにデータを格納する領域である。永続化領域上のデータは、永続ユニットと名付けた単位で管理する。

2.2 永続ユニット

永続ユニットは、外部記憶装置の永続化領域上に存在し、データを外部記憶装置に格納して永続化する際の管理単位である。また、永続ユニットは資源「プレート」と一対一で対応し、資源「プレート」のデータを永続化する。

Tender では、既存 OS とのデータ再利用を可能にするため、永続ユニットのデータの格納形式を既存 OS のファイルシステムとした。現在の ***Tender*** では、UNIX ファイルシステムを利用している。したがって、永続ユニットは UNIX ファイルシステム上のファイルと同様に操作でき、各永続ユニットは、データの実体として一つ以上のブロックと、データの管理情報として一つの i ノードから構成される。

ブロックは、外部記憶装置上に存在する一定の大きさを持つ領域で、ブロック番号によってブロックの位置を特定できる。一方、i ノードは、永続ユニットのサイズや、永続ユニットを構成するブロックのブロック番号などを管理しており、i ノード番号によって一意に識別される。したがって、ブロックに格納している情報が全て正しい場合でも、i ノードの情報に誤りがあれば、永続ユニットを正しく使用することができない。また、i ノードのサイズは 128bytes で、i ノード

番号が連続する四つのiノードが一つのセクタ(512bytes)に格納されている。このため、セクタ単位で入出力を行っている***Tender***では、任意のiノードの更新を外部記憶装置に反映した場合、同一セクタに存在するiノードの更新も同時に反映することになる。

永続ユニットを構成するiノードとブロックは別のセクタに存在するため、永続ユニットへの書き出しでは、iノードとブロックの更新処理を逐次的に行う。ただし、処理の効率化のために、永続ユニットに対応するメモリ領域の更新状況を把握し、更新する必要のあるブロックのみ更新する。また、永続ユニットの書き出し処理中は、対応するメモリ領域の更新を禁止する。

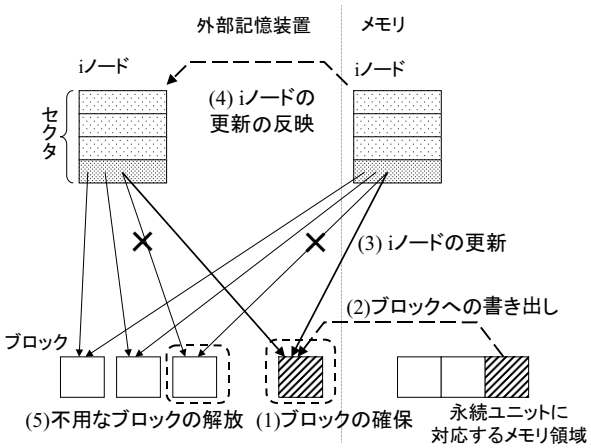

図2　ブロックを保護できる永続ユニットへの書き出し処理

3　データの永続化を高信頼化する手法

3.1　要求条件

データの永続化を高信頼化する手法に対する要求条件を以下に挙げる。

(要求条件1) データの実体の更新処理中にシステムが異常終了した場合、当該永続ユニットを構成する全てのブロックを更新前の状態に復元できる。

(要求条件2) iノードの更新処理中にシステムが異常終了し、iノードの情報が破壊された場合でも、iノードの情報を復元できる。

(要求条件1)を満たすことにより、永続ユニットの実体の破壊や、部分的な更新に起因する整合性の欠如から、永続ユニットを保護できる。また、(要求条件2)を満たすことにより、永続ユニットの管理情報の破壊に伴って永続ユニットを利用できない状態に陥ることを防げる。

3.2　データの永続化を高信頼化する手法の提案

3.2.1　永続ユニットを構成するブロックの保護

(要求条件1)を満足するため、当該ブロックに対して直接更新を行う代わりに、新たにブロックを確保し、確保したブロックに対して更新を行う手法を提案する。

提案手法において、ブロックを保護する場合の永続ユニットへの書き出し処理を図2に示し、以下に説明する。

(1) 書き出しを行う必要があるブロックの数だけ、新たにブロックを確保する。

(2) 確保したブロックに、永続ユニットに対応するメモリ領域の内容を書き出す。

(3) 当該永続ユニットを構成していたブロックの代わりに、確保したブロックを使用するように、iノード(メモリ上)を更新する。

(4) iノード(メモリ上)の更新を外部記憶装置に反映する。

(5) 不用になったブロックを解放する。

上記の処理(1)から処理(4)の間に、システムが異常終了した場合、使用していたブロックは解放されていないので、ブロックの状態を、永続ユニットへの書き出し処理開始前と同じ状態にすることができる。また、処理(5)において、システムが異常終了した場合、不用なブロックが解放されていないだけで、ブロックの更新処理は完了している。この際には、使用されていないブロックが確保されたままの状態になり、ファイルシステムに矛盾が生じる。この矛盾によって、外部記憶装置上の一部の領域は無駄になるが、当該永続ユニットや他の永続ユニットに影響を与えることはない。このような使用されていないブロックは、fsckのようなファイルシステム修復ツールによって容易に解放できる。

このように、提案手法は、永続ユニットへの書き出し処理中にシステムが異常終了した場合でも、当該永続ユニットのデータの実体を更新前、もしくは更新後の状態にできる。つまり、提案手法は、永続ユニットのデータを完全に更新するか、全く更新しないかのいずれかであり、ブロックが破壊されたために、永続ユニットを利用できない状態に陥ることはない。

3.2.2　永続ユニットを構成するiノードの保護<基本方式>

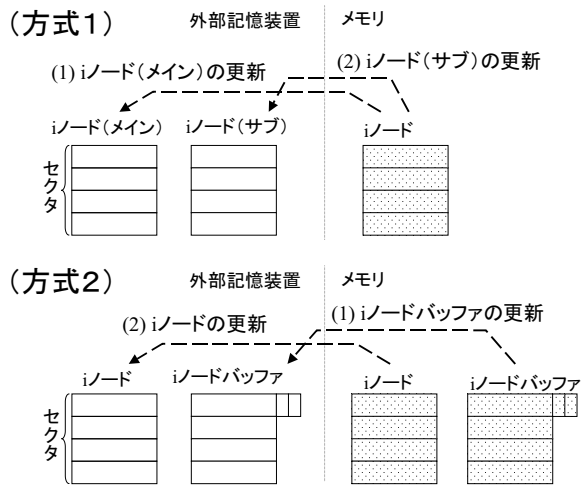

図 3 i ノードを保護する方式

(要求条件 2) を満足するために、i ノードに格納されている情報 (以降、i ノード情報とする) を複数箇所で保持し、i ノードを保護する。この時、i ノードを保護する方式として、以下の二つの方式がある。二つの方式の様子を図 3 に示す。

(方式 1) 一つの永続ユニットが二つの i ノードを保持する

一つの永続ユニットが二つの i ノードを保持し、永続ユニットへの書き出し時に、メインの i ノード、サブの i ノードの順に更新する。また、システム起動時、もしくは、i ノード使用時に、メインの i ノードを調べ、壊れている場合のみ、サブの i ノードを使用する。

(方式 2) i ノードの他に i ノードバッファを設け、i ノード情報を保持する

i ノード情報と i ノードの復元に必要な情報を格納する領域を外部記憶装置上に設け、この領域を i ノードバッファと名付ける。永続ユニットへの書き出し時には、i ノードバッファ、i ノードの順に更新する。i ノードバッファは、永続ユニットへの書き出しが完了した時点で、不用になるので、他の永続ユニットへの書き出しに使用できるように、i ノードバッファに対応するメモリ領域を初期化する。システム起動時には、i ノードバッファの情報により、i ノードが壊れているかを調べ、i ノードが壊れている場合、i ノードバッファの i ノード情報をもとに i ノードを復元する。

(方式 1) は、i ノード情報が正しいか否かを i ノード情報のみから判断しなければならず、この処理は困難である。一方、(方式 2) は、i ノード情報の正否を判断するための情報を i ノードの復元に必要な情報として、i ノードバッファに格納しておくことで、容易に i ノード情報の正否を判断できる。また、(方式 1) では、全ての永続ユニットに対して、二つの i ノードの領域を外部記憶装置上に確保しなければならない。一方、(方式 2) では、i ノードバッファを永続ユニットと同数設ける必要はないため、必要となる外部記憶装置上の領域が小さい。以上のことより、(方式 2) を採用する。

(方式 2) を実現するために、i ノードバッファについて以下の内容を検討する。

(1) i ノードバッファに格納する情報
(2) i ノードバッファの数
(3) i ノードバッファの格納形式と位置の特定
(4) i ノード情報の管理
(5) 同一 i ノードバッファへの複数の処理要求

各項目についての検討を以下に示す。

(1) i ノードバッファに格納する情報

先に述べたように、i ノードバッファは、i ノード情報を格納する。これに加え、i ノードバッファは i ノードの復元に必要な情報として、当該 i ノード情報がどの i ノードの情報であるかを把握するために、i ノード番号を格納し、また、i ノード情報の正否を判断するために、チェックサムを格納する必要がある。

ただし、任意の i ノードの更新を外部記憶装置に反映した場合、同一セクタに存在する四つの i ノードの更新も同時に反映するため、一つの i ノードバッファは四つの i ノード情報を格納する必要がある。

(2) i ノードバッファの数

同時に任意の複数の永続ユニットへの書き出しを可能にするには、複数の i ノードバッファが必要である。ただし、i ノードバッファは、永続ユニットへの書き出しが完了した時点で、、他の永続ユニットへの書き出しに使用できる。したがって、i ノードバッファの数は、同時に永続ユニットへの書き出しを行う数よりも多ければよい。

(3) i ノードバッファの格納形式と位置の特定

i ノードバッファの内容を外部記憶装置に格納する形式として、通常ファイル、特殊ファイル、外部記憶装置上の特定位置の三つが考えられる。

特殊ファイルは、i ノードのみを持ち、ブロックを持たないため、i ノードバッファの i ノード番号とチェックサムを特殊ファイルに格納することができない。一方、外部記憶装置上の特定位置を使用すると、i ノードバッファの数を動的に増やす必要が生じた場合、新たな領域の確保が困難である。よって、i ノードバッファの格納形式として、通常ファイルを採用する。

また、i ノードバッファの格納位置は、システム再起動時に特定可能である必要がある。そこで、i ノードバッファを格納する通常ファイル (永続ユニット) の名前を固定にすることにより、システム再起動時に i ノードバッファの格納位置を特定する。

(4) i ノード情報の管理

同一 i ノード情報が複数の i ノードバッファに存在する場合、永続ユニットの復元処理時にどの i ノードバッファを使用すべきかを判断することが困難である。このため、同一 i ノード情報が複数の i ノードバッファに存在しないように対処する必要がある。

そこで、i ノードバッファに格納している i ノード情報の i ノード番号を常に把握し、当該 i ノード情報が既に i ノードバッファに格納されている場合、必ずその i ノードバッファを使用する。

(5) 同一 i ノードバッファへの複数の処理要求

書き出すように要求された複数の永続ユニットの i ノードが同一セクタに存在する場合、同一 i ノードバッファに対して複数の処理が要求される。このとき、一つの処理が完了するまで、他の処理を行うプロセスを WAIT 状態にする。

＜処理の流れ＞

以上の検討を踏まえ、i ノードバッファを使用して i ノードを保護する場合の i ノード更新処理を図 4 に示し、以下に説明する。

(1) i ノードバッファを確保する。このとき、全ての i ノードバッファの i ノード番号と当該 i ノードの i ノード番号を比較する。
  (1-1) 等しい場合、必ず当該 i ノードバッファを使用する。
  (1-2) 異なる場合、空き i ノードバッファのいずれかを使用する。
(2) i ノード (メモリ上) を更新する。
(3) i ノード (メモリ上) の内容を i ノードバッファ(メモリ上) に複写する。

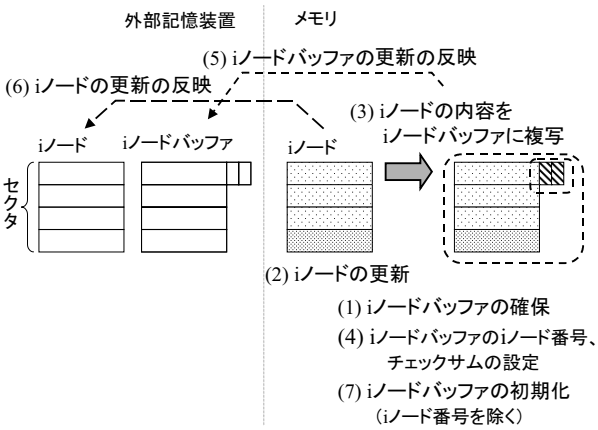

図 4 i ノードを保護する場合の i ノード更新処理

(4) チェックサムを算出し、i ノードバッファ (メモリ上) に i ノード番号とチェックサムを格納する。
(5) i ノードバッファ(メモリ上) の更新内容を外部記憶装置に反映する。
(6) i ノード (メモリ上) の更新内容を外部記憶装置に反映する。
(7) 使用した i ノードバッファを初期化する。ただし、i ノードバッファの使用状況を把握するために、i ノード番号の初期化は行わない。

上記の処理 (6) を行っている間は、システムが異常終了した場合、i ノードの情報は破壊される可能性がある。しかし、処理 (5) で、正しい i ノード情報を i ノードバッファに格納しているので、壊れた i ノードの情報を復元することができる。一方、処理 (6) 以外の処理を行っている間は、外部記憶装置上の i ノードを更新していないため、システムが異常終了した場合でも、i ノードの情報は破壊されない。

＜復元処理＞

永続ユニットへの書き出し処理中のどのタイミングでシステムが異常終了するかによって、i ノードと i ノードバッファが保持する情報は異なり、i ノードの復元処理の内容も異なる。そこで、先に述べた i ノードを保護する場合の i ノード更新処理を場合分けし、システム停止のタイミングと復元処理内容の関係を表 1 に示す。また、各タイミングにおいて、外部記憶装置上の i ノードと i ノードバッファが保持する i ノード情報の状態についても表 1 に示す。

表 1 が示すように、処理 (6) の処理中にシス

表 1　システム停止のタイミングと復元処理内容の関係

システム停止のタイミング	i ノード	i ノードバッファ	復元処理内容
処理 (1) の開始直前～処理 (5) の開始直前	更新前	——	なし
処理 (5) の処理中	更新前	更新中	なし
処理 (5) の完了後～処理 (6) の開始直前	更新前	更新後	i ノードバッファの内容を i ノードに複写
処理 (6) の処理中	更新中	更新後	i ノードバッファの内容を i ノードに複写
処理 (6) の完了後～処理 (7) の完了後	更新後	更新後	なし

テムが異常終了した場合のみ、i ノードは正しい i ノード情報を保持できない。また、処理 (5) が完了してから処理 (6) が開始されるまでの間にシステムが異常終了した場合、i ノードバッファは、i ノードよりも、新しい i ノード情報を保持している。これらの場合のみ、復元処理として、i ノードバッファの内容を i ノードに複写し、それ以外の場合は、復元処理を行う必要はない。このように、i ノード更新処理中のどのタイミングでシステムが異常終了しても、i ノードの復元処理を行うことで、i ノードは正しい i ノード情報を保持できる。

3.3　データの永続化を高信頼化する手法の実現

3.3.1　永続ユニットへの書き出し処理

提案手法による永続ユニットへの書き出し処理の流れを以下に示す。

(1) i ノードバッファを確保する。
(2) 新たにブロックを確保する。
(3) 確保したブロックに、永続ユニットに対応するメモリ領域の内容を書き出す。
(4) 新たに確保したブロックを使用するように、i ノード (メモリ上) を更新する。
(5) i ノード (メモリ上) の内容を i ノードバッファ (メモリ上) に複写する。
(6) i ノードバッファ (メモリ上) に i ノード番号とチェックサムを格納する。
(7) i ノードバッファ (メモリ上) の更新内容を外部記憶装置に反映する。
(8) i ノード (メモリ上) の更新内容を外部記憶装置に反映する。
(9) 不用なブロックを解放する。
(10) 使用した i ノードバッファを初期化する。

3.3.2　永続ユニットの復元処理

永続ユニットの復元処理は、システム再起動時に、i ノードバッファをもとに i ノードを復元

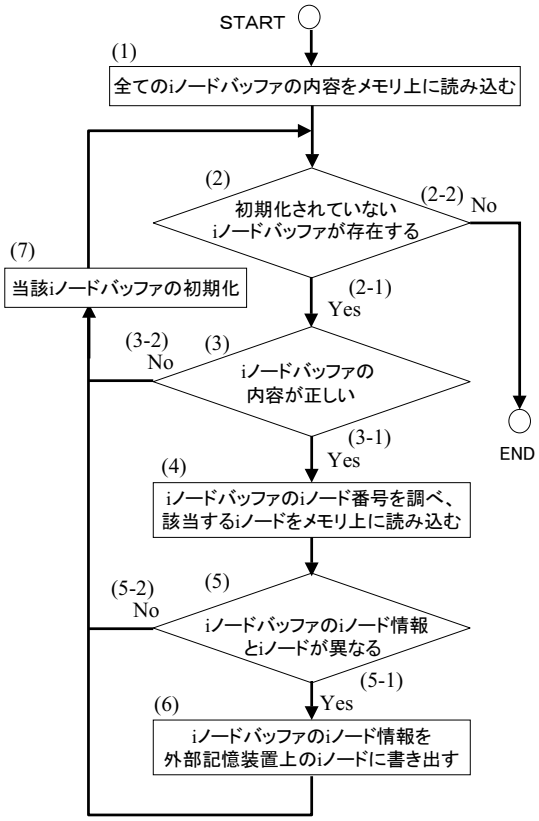

図 5　永続ユニットの復元処理の流れ

することによって行う。永続ユニットの復元処理の流れを図 5 に示し、以下に説明する。

(1) 外部記憶装置上の全 i ノードバッファの内容をメモリ上に読み込む。
(2) 初期化されていない i ノードバッファ (メモリ上) の有無を調べる。
　(2-1) 存在する場合、以降の処理を続ける。
　(2-2) 存在しない場合、復元処理を終える。
(3) i ノードバッファの内容が正しいか否かをチェックサムを用いて調べる。
　(3-1) 正しい場合、以降の処理を続ける。
　(3-2) 正しくない場合、(7) に進む。

(4) i ノードバッファの i ノード番号を調べ、該当する i ノードをメモリ上に読み込む。
(5) i ノードバッファの i ノード情報と i ノードを比較する。
(5-1) 異なる場合、以降の処理を続ける。
(5-2) 同じ場合、(7) に進む。
(6) i ノードバッファの i ノード情報を外部記憶装置上の当該 i ノードに書き出す。
(7) 当該 i ノードバッファ(メモリ上) を初期化し、(2) に戻る。ただし、i ノードバッファの使用状況を把握するために、i ノード番号の初期化は行わない。

永続ユニットの復元処理中にシステムが異常終了した場合について述べる。永続ユニットの復元処理では、i ノード (外部記憶装置上) を更新する場合、i ノードバッファ(外部記憶装置上) が正しい i ノード情報を必ず保持しており、この i ノードバッファ(外部記憶装置上) は更新されない。このように、永続ユニットの復元処理に必要な i ノードバッファの情報が失われることはない。したがって、永続ユニットの復元処理中にシステムが異常終了した場合でも、システムを再起動し、再び永続ユニットの復元処理を行うことにより、永続ユニットを復元することができる。

3.4 実装と評価

提案手法を ***Tender*** に実装し、評価した。***Tender*** を Pentium プロセッサ (90MHz) の計算機で走行させ、永続ユニットへの書き出し処理時間と永続ユニットの復元処理時間を測定した。処理時間は、ハードウェアクロックカウンタを利用して計測した。

永続ユニットへの書き出し処理は、永続ユニットに対応するメモリ領域が更新されていない場合、実際にはメモリ領域の内容を外部記憶装置に書き出さない。そこで、永続ユニットへの書き出し処理として、提案手法を用いるか否か、永続ユニットに対応するメモリ領域を実際に書き出す必要があるか否かの組合せにより、四つの場合について測定し、処理時間を図 6 に示す。ただし、処理時間は実入出力に要する時間を除いた時間である。また、各処理時間 (t(μ sec)) と永続ユニットの大きさ (x(KB)) の関係を下式で示す。

(高信頼化:無, 書き出し:無) t=0.421x+24.3(μ sec)
(高信頼化:無, 書き出し:有) t=1.46x+143(μ sec)

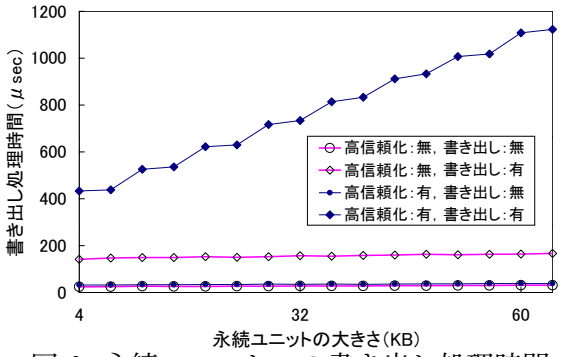

図 6 永続ユニットへの書き出し処理時間 (入出力時間を除く)

表 2 永続ユニットの復元処理時間

処理内容	処理時間 (実入出力時間なし)	処理時間 (実入出力時間あり)
復元なし	143(μ sec)	39.4(msec)
復元あり	187(μ sec)	62.1(msec)

(高信頼化:有, 書き出し:無) t=0.424x+31.7(μ sec)
(高信頼化:有, 書き出し:有) t=48.1x+365(μ sec)

図 6 より以下のことがわかる。

(1) メモリ領域を実際に書き出す必要がない場合、永続ユニットの大きさに関わらず、処理時間は数十μ秒と小さい。これは、外部記憶装置への書き出しを行う必要がなく、UNIX ファイルシステムの操作を行わないためである。
(2) 提案手法を用い、メモリ領域を実際に書き出す場合、他の場合よりも処理時間が大きい。これは、永続ユニットを構成するブロックを更新する処理に加え、ブロックの確保や解放、i ノードバッファを更新する処理を行う必要があるためである。

上記のことより、提案手法を用いる場合、永続ユニットへの書き出し処理性能は低下することがわかる。

永続ユニットの復元処理は、i ノードを復元する必要があるか否かによって処理が異なる。そこで、それぞれの場合について測定し、処理時間を表 2 に示す。また、処理時間は実入出力に要する時間を含めた時間と除いた時間の両方を示す。永続ユニットの復元処理時間は、いずれの場合も数十ミリ秒程度である。永続ユニットの復元処理はシステム再起動時に一度行うだけであるため、永続ユニットの復元処理時間は十分に小さいといえる。

4　関連研究

(1)soft update [3] [4]

外部記憶装置への書き出しを非同期に行う場合、一時的にファイルシステムが危険な状態になり、その時点でシステムが停止すると、ファイルシステムに深刻な矛盾が生じる。

そこで、soft updateは、各管理情報の書き込み依存関係を表現する構造を保持し、必要に応じて管理情報を一旦以前の状態に戻して外部記憶装置へ書き出す。管理情報の更新順序を制御することで、ファイルシステムの安全性を確保したまま、管理情報を非同期に外部記憶装置に書き出すことができる。

しかし、管理情報の更新中に障害が生じ、管理情報が破壊された場合、当該管理情報が管理するデータは失われてしまうという問題がある。

(2) ジャーナリング (メタデータロギング) [4]

ジャーナリングは、管理情報の更新に先立って管理情報の更新ログを作成して、ログ領域に管理情報を格納することで、安全に管理情報を非同期に外部記憶装置に書き出す。同じ管理情報を二度外部記憶装置に書き出すことになるが、ログ領域への書き出しは、通常の管理情報の書き出しよりも高速である。ただし、更新ログのために外部記憶装置上に領域を確保する必要がある。

(3)LFS(Log-structured File System) [5]

LFSは、ファイルシステムを構築するデータやiノード、間接ブロック等の構造を全てログ上に構築したものである。これらの情報を更新するときには、古いブロックに上書きするのではなく、いかなる場合でも新しいブロックをログに追記する。システムに障害が起きた場合、過去のログをもとに短時間でファイルシステムを復旧することができる。

しかし、ファイルシステムを使用していくうちに、ログとして再利用可能な領域を確保する処理が必要となる。この処理は高負荷であるため、ファイルシステムの性能が低下する。

5　おわりに

Tender においてデータの永続化を高信頼化する手法について述べた。さらに、実装と評価により、提案手法の性能を示した。

Tender では、データを永続化するための管理単位を永続ユニットと名付けた。この永続ユニットへの書き出しを行うことにより、データの永続化を行う。提案手法では、永続ユニットのデータの実体であるブロックを保護するために、ブロックに直接書き出すのではなく、新たに確保したブロックに書き出しを行う。また、永続ユニットの管理情報であるiノードを保護するために、iノード情報、iノード番号、チェックサムを格納するiノードバッファを外部記憶装置上に設けた。このiノードバッファは、永続ユニットへの書き出し時に更新され、システム再起動時に行う永続ユニットの復元処理に使用される。永続ユニットの復元処理時間は、数十ミリ秒で十分に小さい。一方、永続ユニットへの書き出し処理は、提案手法を用いることにより、性能が低下する。

残された課題として、永続ユニットへの書き出し処理性能の改善と、外部記憶装置への書き出しが非同期に行われた場合の対処がある。

謝辞　御指導いただきました九州大学大学院システム情報科学府の田端利宏氏に感謝します。

参考文献

[1] 谷口秀夫, 青木義則, 後藤真孝, 村上大介, 田端利宏, “ 資源の独立化機構による***Tender*** オペレーティングシステム ”, 情報処理学会論文誌,Vol.41,No.12,pp.3363-3374 (2000).

[2] 稲本慎司, 谷口秀夫, “ ***Tender*** におけるデータの永続化方式 ”, 情報研報, Vol.2000, No.75, pp.101-108 (2000).

[3] M. McKusick, G. Ganger, “ Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem ”, Proceedings of the Freenix Track at the 1999 Usenix Annual Technical Conference, pp.1-17 (1999).

[4] Margo I. Seltzer, Gregory R. Ganger, M. Kirk McKusick, Keith A. Smith, Craig A. N. Soules, Christopher A. Stein, “ Journaling versus Soft Updates: Asynchronous Meta-data Protection in File Systems ”, USENIX Annual Technical Conference, pp18-23 (2000).

[5] M. Rosenblum, J. Ousterhout, “ The Design and Implementation of a Log-Structured File System. ACM Trans. on Computer Systems ”, Vol.10, No.1, pp26-52 (1992).