計算機アーキテクチャ 106－7
（１９９４．６．１３）

プロセス間のアクセス競合を低減する並列２次記憶システムの構想

大上靖弘　北村徹　大西一正　清水雅久

RWCP[1]超並列三洋研究室[2]

超並列計算機のための並列二次記憶システムについて考える。超並列計算機では、マルチユーザ環境が強く求められるため、並列二次記憶に対するプロセス間のアクセス競合を低減することが重要である。本稿では、並列二次記憶システムの要件を検討し、ファイル用ディスクとページング用ディスクを分離することによってプロセスのローカルなアクセスにおける干渉を排除する並列二次記憶の構成を述べる。また、データの更新を追加の形態で行なうことにより、データ書き込み位置の自由度を高め、二次記憶への各プロセスからのグローバルなアクセス競合を低減した並列ファイルシステムを提案し、その構成と特徴について述べる。

A Scheme for Reducing Access Conflicts on the Parallel Secondary Storage System

Yasuhiro OUE, Toru KITAMURA, Kazumasa OHNISHI and Masahisa SHIMIZU

Massively Parallel Systems Sanyo Laboratory, RWCP

3-10-15 Hongo, Bunkyo-ku, Tokyo 113, Japan

This paper describes a secondary storage system for a massively-parallel computer. In a multi-user, multi-process environment, it is essential that the processes do not disturb each other. This means that the secondary storage system must be designed to reduce any file access conflicts between the processes. We propose a secondary storage system that eliminates data access conflicts by isolating the disk system containing the file data from the local disk for process paging. Then, we propose a parallel file system that reduces data access conflicts by enhancing the degree of flexibility in positioning the write data.

[1]RWCP:Real World Computing Partnership(新情報処理開発機構)

[2]三洋電機 (株) 東京情報通信研究所内

1 はじめに

超並列計算機は、大規模なシミュレーションや音声・画像認識などのさまざまな分野で利用されると考えられる。それらのアプリケーションが扱うデータは膨大であり、この大量のデータを二次記憶に蓄え高速に効率良く管理する新たな手法を構築することが重要である。

一方、大量のプロセッサを有する超並列計算機では、その大規模な演算資源を複数のユーザが効率的に利用できることが要求されるが、複数のユーザが同時にさまざまな目的で二次記憶を利用した場合、同一ディスクへのアクセスの競合が各プロセスの進行を阻害し、超並列計算機の実行効率を低下させる原因となる。

本稿では、まず並列二次記憶に求められる要件と並列二次記憶システムの基本構成について述べる。また、これらを効率的に管理するファイルシステムとして、データの更新を追加の形態で行なうことにより、データ書き込み位置の自由度を高め、二次記憶への各プロセスからのアクセス競合を低減した並列ファイルシステムを提案し、その構成と特徴について述べる。

2 並列二次記憶の構成

2.1 想定する超並列計算機モデル

まず、並列二次記憶の検討に先立ち想定した超並列計算機の構成について述べる。

膨大な資源を有する超並列計算機では、その資源を複数のユーザが有効に活用できることが重要である。複数ユーザの複数プロセスが、超並列計算機上で実行される環境を実現するための具体的な方式としては、複数のプロセスがプロセッサ空間を分割使用する方式やプロセッサを時分割で使用する方式、及びそれらを融合した方式が考えられる。特に空間分割して実行される際には、各プロセスが分割領域ごとに独立して動作し、他のプロセスの実行を阻害しないことが重要である。我々は、これらの方式を実装できる超並列計算機として、プロセッサがいくつかのプロセッサグループ(クラスタ)に分割され、各プロセスがこのクラスタを境界として実行されるアーキテクチャを持つものを想定した[1]。

また、結合網については、入出力データが結合網を占有し命令実行系のデータ転送を阻害することを避けるために、入出力のための結合網が命令実行系の結合網から独立した形態をとるものを想定した[2]。

図1は、想定した超並列計算機のモデルを示したものである。超並列計算機は1000台以上のプロセッサを有し、少数のプロセッサごとにクラスタに分割されている。また、命令実行系のプロセッサ間結合網とは別に、入出力のための結合網を備える。

さらに、マルチユーザの利用環境では、プロセスのプロセッサ空間への割り当てについて自由度の高いシステムであることが重要であり、論理的には、各プロセッサが一様な記憶資源を共有していることが望ましい。従って、各プロセッサに分散されたメモリが論理的には一つの共有メモリとして扱われるような大域的仮想記憶の構成をとることを想定した。

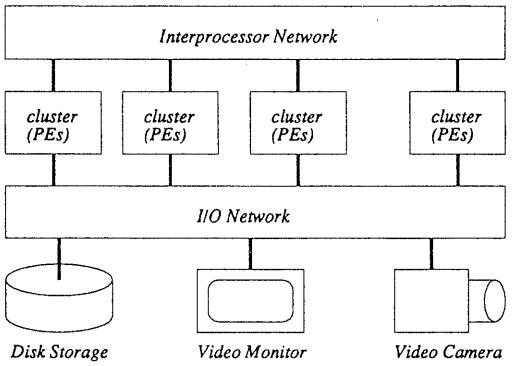

図 1: 想定する超並列計算機モデル

2.2 並列二次記憶の要件

以上のような超並列計算機を前提として、並列二次記憶に対して求められる要件を抽出した。以下にその要件を述べる。

アクセスに関するプロセス間での相互干渉が少ないこと

マルチユーザの利用環境において、我々は超並列計算機上で様々な特性や要求を持ったアプリケーションが同時に実行されることを想定している。このような利用環境においては、実行中の各プロセスが他のプロセスの影響を受けないような構成を考える必要がある。同様に二次記憶に対するアクセスについてもプロセス間の相互干渉を排除できることが重要である。

スケーラビリティを持つこと

超並列計算機は扱うデータも膨大であり、この大量のデータを蓄えるために大規模な二次記憶を必要とする。このため、二次記憶が必要に応じて柔軟に

拡張でき、かつ高速性を保つためにスケーラブルであることも重要である。

ファイルシステムが各プロセッサから一様に見えること

二次記憶上に蓄えられるファイルデータは、超並列計算機上の各プロセスに共有される記憶資源である。プロセスがどのプロセッサに割り当てられても、ファイルシステムの構造やアクセス手順のみならず、アクセスに要する時間等も同様であることが望まれる。

十分な信頼性を有すること

大規模な二次記憶を実現する方法として、一般的には大量のディスクデバイスで構成することが考えられる。しかし、ディスク台数の増加は二次記憶システム全体としての信頼性の低下を引き起こすため、十分な信頼性を維持するための手段が必要である。

2.3 並列二次記憶システムの構成

並列二次記憶に対する要件のうち、特にプロセス間の相互干渉の低減を目的として、並列二次記憶システムの検討を行なった[3]。図2に並列二次記憶システムの構成を示す。この並列二次記憶システムに後で述べる並列ファイルシステムを実装し、アクセスに関するプロセス間の相互干渉を大幅に低減する。

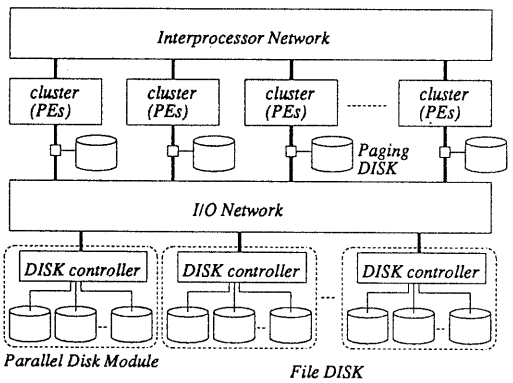

図 2: 並列二次記憶システムの構成

(1) ファイル用ディスクとページング用ディスクの分離

図2に示すように、各プロセスに共有されるファイルデータを蓄えるディスク(ファイル用ディスク)を、各クラスタ内でのページングなどに利用するローカルなディスク(ページング用ディスク)から分離する構成をとる。ページング用ディスクが各クラスタごとに接続されるのに対し、ファイル用ディスクは、各クラスタから独立し入出力のための結合網を介してプロセッサと接続される。

本並列二次記憶システムでは、ファイルアクセスと他プロセスのローカルなアクセスが、アクセスの経路上においても、またディスクに対しても競合することがない。このため、ファイルアクセスが他プロセスのページング等のローカルなアクセスを阻害しないシステムを実現できる。また、ファイルアクセスに関しても後述の並列ファイルシステムにより相互干渉を低減することが可能であるため、「二次記憶に関するプロセス間のアクセス競合」を大幅に低減することが可能になる。

さらに、各プロセスの共有資源であるファイル用ディスクをクラスタから独立させ、結合網を介してホモジニアスに接続しているため、ファイルシステムの一様性とスケーラビリティを実現するのも容易である。

(2) ファイル用ディスクの構成

次に、ファイル用ディスクの構成について述べる。ファイル用ディスクは、複数の並列ディスクモジュール(PDM)から構成される。各PDMは、複数の高信頼性ディスクとディスクコントローラから成り、ディスクコントローラが入出力結合網に並列に接続される。ディスクコントローラには複数のディスクが接続され、各ディスクごとにバッファ、DMAコントローラ、SCSIインタフェース等を備えることにより、ディスクの高速な並列制御を実現する。

(3) 入出力結合網

結合網は、プロセッサを含む複数のクラスタとファイルデータが蓄えられた複数のPDMを相互に接続する。結合網の規模は超並列計算機のクラスタ数にも依存するが、128 × 128ノード、転送速度は各ノード当たり$75MB/sec$程度を想定している。

ファイルアクセスに関するプロセス間の相互干渉を低減するためには、結合網についてもデータ転送の相互干渉を生じないことが望ましい。具体的には、クロスバー網などの非閉塞網を用いる予定であるが、結合網の規模を考えた場合、実装面での課題

が残る。

3 並列ファイルシステム

並列計算機のファイルアクセスにおいては、論理的なファイルを複数のディスクに対してどのように分散格納するか、また複数の計算ノードに対してどのように分散配置するかが重要な課題となる。

従来の並列ファイルシステムでは、ユーザ(アプリケーション)またはファイルシステムがこれらのマッピングを別々に定義している[4, 5]。従って、2種類のマッピングの間での変換が必要となる。

たとえば行列データが行単位でディスクにマッピングされている場合に計算ノードへのマッピングも行単位であれば、分離された通信チャネルを使って、計算ノードが独立に、同時に複数のディスクにアクセスできるので高速なアクセスが可能になる。しかし、計算ノードへのマッピングが列単位であれば、全ての計算ノードが全てのディスクにアクセスしないとならないので大きなオーバーヘッドが生じる。

これを解決するために、ディスクに対するマッピングに応じて読み出した後、計算ノードに対するマッピングに応じてデータを交換する手法が提案されている[7]。データ交換のために余分な処理が必要になるが、これは計算ノードの相互結合網を使用するため、ディスクからの読み出しに比べてかかる時間のオーダーが小さく、ディスクからの読み出しを高速に行なうことが可能になることで、全体の性能は上がる。

この手法は、画像データや行列計算のようにファイル全体を一度にアクセスする場合には有効であるが、ファイルデータを時間的・空間的にランダムにアクセスする場合には必ずしも有効とは言えない。

計算ノードへのマッピングが一意に決定されず、プログラムの進行状況に応じて動的に決定されていくようなアクセス環境においては、ファイルシステムがディスクへのマッピングを管理して、計算ノード側のアクセスに対応して動的に変更していく必要がある。

また、超並列計算機は大規模かつ高価であり、マルチユーザ、マルチプロセスでの使用が不可欠となる。プロセス間のアクセス競合が発生することによって、個々のアプリケーションに対する最適化は効果を失う。さまざまなプロセス(例えば、ファイルを連続的にアクセスするプロセスと、ランダムにアクセスするプロセス)が混在する環境では、プロセス間の競合を低減して、システム全体の負荷を分散する機能が求められる。

以上の要件を満たすファイルシステムとして、我々はloggingの概念を導入した並列ファイルシステムParallel Logging FileSystem(PLFS)を提案する。loggingを使用することによって、データの書き込み場所に関する自由度が高くなるので、ファイルデータのディスクへのマッピングを動的に管理することが可能であり、また書き込み先のディスクを動的に選択することで、書き込み動作におけるプロセス間の相互干渉を低減することができる。さらに、負荷の低いディスクに対して選択的に書き込みを行なうことにより、二次記憶システム全体に負荷を均等に分散し、ディスク台数に応じた性能向上が期待できる。

3.1 Log-structured ファイルシステム

Log-structured ファイルシステムはRosenblumらによって開発された、逐次型の計算機におけるUNIX用のファイルシステムであり、ディスクへの全ての書き込みをログ構造体に連続的に行なうことによって、ファイルの書き込みとクラッシュからの復旧を高速化する[6]。

Log-structured ファイルシステムは、ファイルデータやディレクトリの変更、inodeなどのメタデータの修正などの全ての書き込みをいったんファイルキャッシュ上に蓄え、それらをまとめてログとしてディスクへ書き込む。多数の小さなアクセスが単一の大きなアクセスに変換されるため、ディスクのシーク操作が削減され、書き込み性能を向上することができる。

更新データを元の位置に上書きせず、追加書き込みを行なうため、最新のファイルデータを示すためのインデックスとしてinodeとinode mapを使用する。メモリ上にキャッシュされたinode mapが最新のinodeを管理するため、最新のファイルデータの位置を知ることができる(図3)。

また、ログの連続的な性質によって、クラッシュからの非常に高速な復帰も可能になる。システムクラッシュ後のファイルシステムの一貫性を修復するためには、log-structured ファイルシステムでは直前に書き込まれたログを検査するだけでよい。さらに、ファイルシステムが矛盾のない状態にあるログ中の位置(チェックポイント)をディスクの特定の領域(チェックポイントリージョン)に定期的に記録し、クラッシュ後は直前のチェックポイント以降の操作結果を捨てることによって高速な復旧が可能となる。

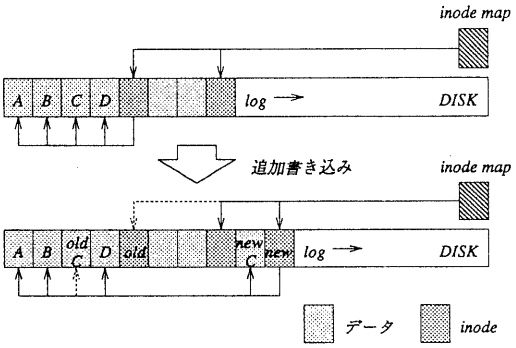

図 3: Log-structured ファイルシステム

しかしながら、データに対する修正がログへの追加の形で行なわれるので、データの更新や削除によって、ログは有効なデータと無効なデータが混在した状態となる。log-structured ファイルシステムが効率的に動作するためには、新しく書き込むデータのための領域が常に確保されなければならない。そこで、ディスクをセグメントと呼ばれる固定サイズの領域に分割し、無効なデータが多数存在するセグメントから有効なデータのみを取り出す。有効なデータを別のセグメントに移動することによって新しいログを書き込むための連続的な領域を確保するクリーニング操作が必要となる。

3.2 PLFS

我々の提案する並列ファイルシステム PLFS は、logging の概念を導入することによって、以下のような特徴を有する。

動的な負荷分散

書き込みに関して、ディスク上の書き込み位置の自由度が高いので、書き込み動作におけるプロセス間の相互干渉の低減と、二次記憶システムの稼働状況に動的に対応した負荷分散が可能である。

高速な復旧

チェックポイント機能を実現することによって、システムクラッシュ後の再起動の際に、ファイルシステムの状態を高速に復旧することができる。

PLFS は図4のように、各プロセッサ上で独立に動作するFサーバと、ディスクコントローラ上のDサーバによって構成する。Fサーバはアプリケーションプログラムからの要求に応じて任意のDサーバとの通信を行ない、Dサーバは実際のディスクアクセスと、複数台のディスクの管理を行なう。

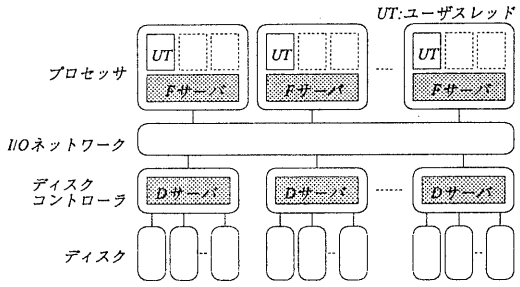

図 4: 並列ファイルシステムの構成

以下に PLFS でのファイルアクセスにおける動作を述べる。

アプリケーションはアクセス要求をFサーバに対して行ない、データの書き込みはFサーバによってプロセッサ毎にバッファリングされる。データが一定量に達するか、もしくは前回の書き込みから一定時間を経過すると、Fサーバはアクセス負荷が分散するようにDサーバを選択し、ログ単位での書き込み要求を発行する。

Dサーバは、接続されたディスクの使用状況を把握しており、Fサーバからの要求を受けると、負荷の低いディスクに対して書き込みを行なう (図5)。

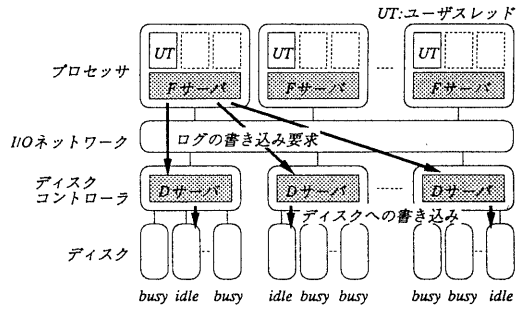

図 5: 書き込み先ディスクの選択

3.2.1 inode map の管理

ファイルデータのディスク上の位置は inode によって管理される。PLFS では、inode の位置も動的に変更されるので、inode map によって最新の inode の位置を管理する。

inode の最新の位置を高速に検索するために、inode map はメモリ上にキャッシュされていなければならな

い。また、inode map は全プロセスから一様に見える必要があるため、クラスタから独立しているディスクコントローラのメモリ空間上に配置する。一貫性のための通信の削除や、inode の検索による負荷が特定のディスクコントローラに偏ることを避けるために、inode map は全てのディスクコントローラに分散して配置する。

ファイルは生成時に、いずれかのディスクコントローラ上の inode map のエントリが割り当てられる。ファイルの inode の位置を管理するディスクコントローラは、ファイルが削除されるまで変更されない (図 6)。

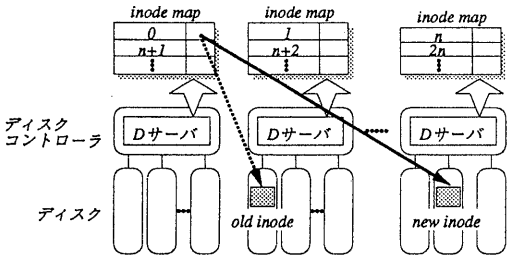

図 6: inode map

ファイル識別子は、ファイルの inode map のエントリが存在するディスクコントローラと、ディスクコントローラ内のローカルな番号で構成される。従って、ファイル識別子から inode の位置を管理しているディスクコントローラを一意に決定することができる。

例えば、ファイル読み出しの際にはファイル識別子から inode map のエントリが存在するディスクコントローラが決定され、そのエントリを調べることで inode のディスクアドレスがわかる。

3.2.2 クリーニング

クリーニング操作は、有効なデータと無効なデータが混在した状態となったディスクから、有効なデータを別の領域に移動することによって無効なデータが占めている領域を再利用可能とし、新しいログを書き込むための連続的な領域を確保する。

データブロックが有効であるかどうかを判定するためには、inode または間接ブロックからそのブロックが指されているかどうかを調べなければならない。しかし、PLFS ではログが並列ディスクに分散されているため、あるブロックを指している inode が別のディスクに存在することもある。従って、1 台のディスクのクリーニングのために、全ディスクからの inode の転送が必要となるケースも考えられる。

クリーニングをシステム全体で同期を取って行ない、クリーニングの度に inode の転送を行なうことは可能であるが、同期と inode の転送のためのオーバーヘッドが大きくなる。また、同期を取っている間、inode の更新ができなくなり、計算ノードに対する影響も生じる。それぞれの状況に応じたタイミングで、ディスクコントローラ毎に独自にクリーニングを行なうことが望ましい。

そこで、実際にオープンされているファイルの inode に関して inode の更新の履歴を取り、ファイルのクローズ時に新旧の inode の差分から無効なデータを判定してディスクに通知する手法を採用した (図 7)。

1. ファイルのオープン時に F サーバが inode を読み出し、データ更新の際に inode 更新の履歴を保存する。古い inode からリンクされているデータは、無効であり、ファイルのクローズ時にこれを D サーバに対して通知する。

2. D サーバが無効なデータを削除し、有効なデータを別の領域に詰め込むことによって新しいログを書き込むためのフリースペースを生成する。

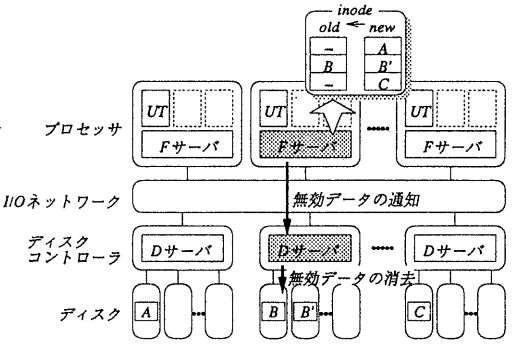

図 7: クリーニング

無効なブロックの判定は F サーバが行ない、有効なブロックの移動は D サーバが行なう。これらの二つの動作が時間的に連続する必要はなく、ディスク間での同期や無効なデータの判定のためのディスクアクセスが不要なので、少ないオーバーヘッドでクリーニングを行なうことが可能となる。

3.2.3 チェックポイント

超並列計算機は膨大なディスク容量を必要とするので、異常終了時のデータ構造の検査に要するコストが大きい。従って、超並列計算機のための並列ファイルシステムにおいては、クラッシュ後のファイルシステムの復旧は重要な課題である。

PLFSは、チェックポイント機構を提供する。チェックポイント機構によって、ファイルシステムの矛盾のない状態がディスクの特定の領域に定期的に記録され、クラッシュ後は直前のチェックポイント以降の操作結果を捨てることによって高速な復旧が可能である。

チェックポイントは、ディスク上に記憶されたファイルシステムの矛盾のない状態を示すものであり、本来はプロセッサとディスク間での同期を必要とする。全プロセッサのメモリ上の更新データ(ログ)をディスクに書き出し、それらが実際に書き込まれてチェックポイント操作が終了するまで、ディスクへの新たな書き込みは禁止されなければならない。しかし、メモリ上の更新データをディスクに書き出す時間はプロセッサ台数に応じて増大し、超並列計算機では、チェックポイント操作がプロセッサ上のプログラムの実行に及ぼす影響が問題となる。従って、プロセスの進行をできるだけ阻害しないように、チェックポイント操作を行なう必要がある。

われわれは、プロセッサとディスク間での非同期なチェックポイント操作を提案する。チェックポイント操作が起動されると、プロセッサはメモリ上のデータをログとして書き出すだけでよく、実際にディスクに書き込まれるまで待たずにプログラムの処理を続行できる。

チェックポイント操作を非同期に行なうためには、チェックポイント前にプロセッサが書き込み要求を発行したログと、チェックポイント後にプロセッサが書き込み要求を発行したログをディスク側が識別できる必要がある。そこで、ログに識別番号を持たせ、チェックポイント毎にインクリメントする(図8)。

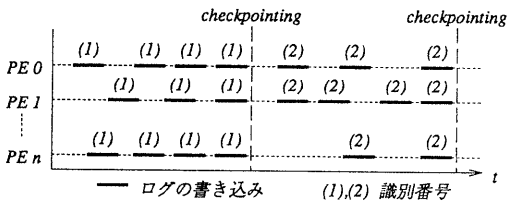

図 8: 識別番号

ディスク側は、同一の識別番号を持つログを全て書き込んだ後で、ディスクの特定の領域(チェックポイントリージョン)にinode mapブロックのアドレス、ログの再後尾のアドレス、チェックポイントを行なった時刻などの情報を書き込む。プロセッサは書き込み要求を行なったログの数をディスク毎に記憶しておき、ディスクコントローラが、実際にディスクに書き込まれたログの数と比較することによって、同一の識別番号を持つ全てのログを書き込んだかどうかの判定を行なう。全てのディスクが上記の操作を終了した時点で、チェックポイント操作は完了する。

具体的には、チェックポイント操作の際にチェックポイントプロセスが起動され、以下のように動作する(図9)。

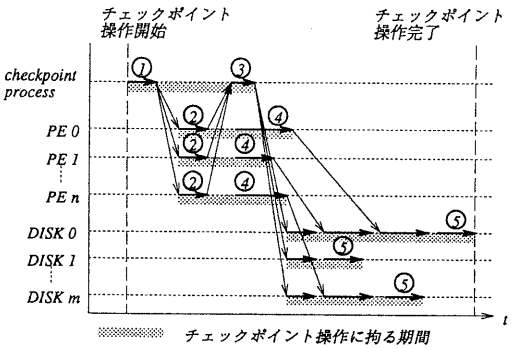

図 9: チェックポイント

1. チェックポイントプロセスは、各プロセッサ上のFサーバにチェックポイント操作の開始を通知する。
2. 各Fサーバから、前回のチェックポイント以降に書き込んだログのディスク毎の数がチェックポイントプロセスに返る。
3. チェックポイントプロセスは、前回のチェックポイント以降に書かれるべきログの数をディスク毎に集計し、各ディスクコントローラ上のDサーバに送信する。
4. Fサーバはメモリ上のデータをログとして書き出す。
5. Dサーバは実際に書き込んだログの数と書かれるべきログの数を比較し、一致するまで待ってから、ディスクの特定の領域(チェックポイントリージョン)に情報を書き込む。

項目3と項目4は時間的に前後してもよい。

上記のようなプロセッサとディスク間での非同期なチェックポイント操作によって、チェックポイント操作中にディスクへのアクセスを行なうことが可能となり、プログラムの実行に対する影響を抑えることができる。

4 おわりに

本稿では、まず並列二次記憶システムの要件と基本構成について述べた。本システムでは、ファイルデータを蓄えるファイル用ディスクをクラスタ内のローカルなアクセスを行なうページング用ディスクから分離することにより、ローカルなアクセスに関するプロセス間の相互干渉を排除する。次に、この並列二次記憶を効率的に管理する並列ファイルシステムとして、ログ構造の概念を導入した並列ファイルシステム PLFS を提案した。本ファイルシステムでは、データの更新もデータを追加する形態で行なうため、データの書き込み位置について自由度が高く、データの書き込み位置を動的に選択することにより、プロセス間のアクセス競合を大幅に低減することが可能である。また、書き込みの高速性、システムクラッシュからの復旧の高速性などの点でも有利であると考えられる。

今後は、並列二次記憶システムの詳細な方式設計を進めるとともに、シミュレーションにより並列ファイルシステムの予備的な性能評価を行ない、本並列ファイルシステムの有効性を検証していく予定である。

謝辞

本研究を遂行するにあたり、有益な御意見、御討論をいただいた(技組)新情報処理開発機構(RWCP)つくば研究センタ超並列アーキテクチャ研究室の各位に感謝いたします。

参考文献

[1] 坂井修一, 岡本一晃, 松岡浩司, 廣野英雄, 児玉祐悦, 佐藤三久, 横田隆史. 超並列計算機 RWC-1 の基本構想. 並列処理シンポジウム JSPP'93, pp. 87-94,(1993).

[2] 廣野英雄, 松岡浩司, 岡本一晃, 横田隆史, 堀敦史, 児玉祐悦, 佐藤三久, 坂井修一. 超並列計算機 RWC-1 における入出力機構. 情報処理学会研究報告 93-ARC-101, pp. 33-40,(1993).

[3] 大西一正, 北村徹, 大上靖弘, 清水雅久. 超並列計算機における並列2次記憶の基本アーキテクチャ. 情報処理学会第48回全国大会論文集, 3B-5, (1994).

[4] Rajesh R.Bordawekar, Alok N.Choudhary, Juan Miguel del Rosario *An Experimental Performance Evaluation of Touchstone Delta Concurrent File System.* 1993 International Conference on SUPERCOMPUTING, (1993).

[5] Peter F.Corbett, Dror G.Feitelson, Jean-Pierre Prost, Standra Johnson Baylor. *Parallel Access to Files in the Vesta File System.* Supercomputing, (1993).

[6] Mendel Rosenblum, John K.Ousterhout. *The Design and Implementation of a Log-Structured File System.* Proceedings of the 13th ACM Symposium on Operating Systems Principles, (1991).

[7] Juan Miguel del Rosario and Alok N.Choudhary. *High-Performance I/O for Massively Parallel Computers:Problems and Prospects.* Computer, Vol.27, No.3, (1994).