システムソフトウェアと
オペレーティング・システム 73－3
（１９９６．８．２６）

ワークステーションクラスタに用いる回復可能分散共有記憶システム

大澤 範高　　弓場敏嗣

電気通信大学大学院情報システム学研究科
〒182 東京都調布市調布ヶ丘 1-5-1
{osawa,yuba}@is.uec.ac.jp

ワークステーションを組み合わせてクラスタとして利用する場合には、個々のワークステーションの停止を管理することが必ずしもできず、耐故障性が重要である。また、単一プロセッサおよび共有記憶型マルチプロセッサで動作していたマルチスレッドプログラムをクラスタ上で動作させるためには、分散共有記憶が重要である。状態回復の管理と分散共有記憶の管理を統合して行う回復可能分散共有記憶システムを提案する。複数の読み手を許容する分散共有記憶において複製されたデータを状態回復のために利用することによって、状態保存・回復を効率よく行うことができる。さらに、回復可能性を実現するために必要なデータの複製を遅延させることによって、状態保存のためのデータの移動を分散共有記憶のためのデータの移動で隠蔽できる。遅延複製は、複製処理の時間的分散にも役立つ。また、UNIX 上に構築するユーザレベル回復可能分散共有記憶システムの設計について考察する。

A Recoverable Distributed Shared Memory System for Workstation Clusters

Noritaka OSAWA　Toshitsugu YUBA

Graduate School of Information Systems
The University of Electro-Communications
Chofugaoka 1-5-1, Chofu, Tokyo 182, JAPAN
{osawa,yuba}@is.uec.ac.jp

When a cluster that is composed of idle workstations is used, fault-tolerance is important because a workstation may be stopped or rebooted without notice by an administrator who is not a user. A distributed shared memory (DSM) is also important because it allows a multi-threaded program that is developed on a single processor workstation or a shared-memory type multiprocessor workstation to be executed on the workstation cluster with little modification. This paper proposes a recoverable distributed shared memory (RDSM) that integrates coherency control of distributed shared memory with control of checkpoint/recovery. Copies in a DSM system that allows multiple readers to access the same data simultaneously are used as replicas for recovery. This reduces data transfers for checkpoint/recovery and improves efficiency of the system. Furthermore, replication of data that does not have a copy is performed lazily because a future access to the data may make a copy and hide the overhead of replication of the data for recovery. Lazy replication distributes the load of processing over time. Design of a user-level RDSM system on a cluster of UNIX workstations is also discussed.

1. はじめに

現在では、多数のワークステーションが研究室などに散在しており、常時通電されている場合が多い。しかし、それらのワークステーションがユーザによって実際に利用されていない時間の割合が高いものも少なくない。負荷が低い状態(アイドル状態)の複数のワークステーションによって、特別なコストなしに並列計算を行うためのワークステーションクラスタを構成することができる。分散記憶型であるワークステーションクラスタで単一プロセッサや共有記憶型マルチプロセッサを持つワークステーションで開発したマルチスレッドプログラムをできるだけ容易に実行するためには、分散共有記憶が有効である。

ところで、コストなしに利用可能なワークステーションは、一般に集中管理がなされておらず、個々のワークステーション管理者の都合でシステムの停止やリブートが行われる可能性がある。クラスタを構成するすべてのワークステーション(ノード)が一斉に停止する可能性(停電など)は無視できるが、いずれかのノードが停止する可能性は必ずしも低くなく、プログラム実行に長時間かかる場合には無視できない。したがって、ノードの停止(故障)に対処するために、耐故障性が必要である。

耐故障性の実現には、プログラマが明示的に実行状態のスナップショットをとり、回復を行えるようにする方法がある。しかし、この方法では、スナップショットを採るコードを新たに挿入したり、スナップショットを採るタイミングを明示的にプログラミングする必要がある。これらのプログラミングを不要にするには、分散共有記憶を回復可能にし、状態の保存、回復処理をシステムに任せる方法が適当である。

独立したワークステーションから構成されるクラスタにおいては、故障(停止)せず、常時アクセス可能な独立した安定記憶装置の存在を仮定することは非現実的である。ワークステーションのディスク装置などを安定記憶として利用することは可能であるが、そのディスクが接続されているワークステーションが停止した場合には、処理が止まる。起動の権限が利用者にはない場合には、処理を再開することはできない。このことはシステムの可用性を低下させる。耐故障性を持った安定記憶システム(ディスクサーバ)を分散共有記憶システムとは別に構築し、その安定記憶システムを利用して回復可能にする方法も考えられるが、複雑になる。

本稿では、ワークステーションから構成されるクラスタのための回復可能分散共有記憶システムについて述べる。回復可能にするためのチェックポインティングや回復処理はユーザ透過に行われる。耐故障性のある安定記憶を前提としないで、状態回復の管理と分散共有記憶の管理を統合して行う回復可能分散共有記憶システムを提案し、その制御プロトコルを説明する。その後、UNIX ワークステーションで動作する回復可能分散共有記憶システムの設計について考察する。

2. 一貫性制御と回復可能性制御の統合

これまでに多くの分散共有記憶の研究がなされている[1]。それらの多くは複数読み手の存在を許容している。ここでは、複数の読み手の存在を許容する一貫性制御と回復可能性制御の統合を考える。複数読み手の存在を許容しているシステムでは、同一データの複製が存在しうる。複数の読み手を許すための複製(コピー)は、回復のための複製(レプリカ)として利用することができる。分散共有記憶における複製を利用することによって、回復のための状態保存時(チェックポイント時)のデータの移動量およびそのオーバーヘッドを少なくすることができる。この考え方に基づいた研究には[2]がある。[2]では、チェックポインティング開始時に複製の存在しないデータ(単独データ)は、その時点で複製を生成し、チェックポインティングを完了させる。

本稿では、複数読み手のための複製の利用に加えて、単独データの複製を遅延させる方法を提案する。複製を遅延させ、単独データが分散共有記憶のために複製もしくは移送される際に、回復のための複製を生成する(もしくは、コピーとレプリカを共用する)。これによって、状態保存のためのデータの移動を分散共有記憶のためのデータの移動で隠蔽できる。この方法を Integrated Lazy Replication (ILR)と呼ぶ。もちろん、一定時間内に複製の生成が行われない場合には、チェックポインティング完了のために複製を明示的に行う必要がある。

提案の方式によって、回復のための状態保存に必要なデータの移動量およびオーバーヘッドを、より軽減することが可能である。チェックポイント時に新たな複製が必要なデータは、前回のチェックポイントから変更が行われたデータである。これらの変更されたデータが、その後参照も変更もされないことは少ないと考えられる。もし、変更されたデータがそれ以降アクセスされないのであれば、一度だけ複製を行えばそれ以降は、回復のために新たに複製を生成する必要はない。繰返しチェックポインティングが行われる場合には、そのようなデータによるオーバーヘッドは無視できる。

変更・参照が単一ノードでのみ行われる場合には、回復のために他ノードに複製を生成しなければならない。そのような場合においても、複製を一括して行わず、適当な事象が発生するごとに複製することによって、複製のための処理、通信を時間的に分散させることができる。言い換えると、ワークステーションクラスタの他のタスクへの影響を少なくすることができる。

提案方式を[3]の集中マネージャ方式を基に説明する。 [3]と同様に分散マネージャ方式に拡張が可能である。

2.1. データ構造

クライアントはデータ単位毎に、mode、lock、dirty、version という情報を必要とする。dirty および version が、本方式で新たに必要になる情報である。mode は、アクセスモードおよびチェックポイント状態を表す。アクセスモードにチェックポイント状態を新たに追加した。lock は、事象処理や要求を同期させるために利用される。dirty は、直前のチェックポイントから変更が加えられたか否かを表す。version は、データのバージョンを表す。

マネージャは、クライアントの情報に加えて、owner、copyset、replicaset という情報を必要とする。owner は、データを所有しているクライアントを表す。すなわち、書出しの権利を有するクライアントである。copyset は、データの複製を持つすべてのクライアントを表す集合である。replicaset は、回復のためのスナップショットを構成するデータの有効な複製を含むすべてのクライアントを表す。これは、本方式で新たに必要になった情報である。

2.2. クライアントの状態遷移

クライアントで発生する事象は大きく内部事象と外部事象に分類できる。内部事象に、Read Fault、Write Fault がある。Read Fault、Write Fault は、それぞれ、クライアント内部のデータアクセス時の読込み例外、書出し例外の発生を意味する。外部事象には、Read Request、Write Request、 Checkpoint、Completion がある。Read Request、Write Request は、リモートクライアントでの Read Fault、Write Fault によるデータ要求をそれぞれ意味する。 Write Request は、[3]における Invalidation に相当する。チェックポインティングは、2 相コミットと同様に 2 相に分けて行われる。Checkpoint は、チェックポインティングの開始を表し、Completion がチェックポインティングの終了を表す。

遅延を用いない方式[2]では、あるデータ(ページ)に対して Working、Checkpoint という 2 種類のバージョンが存在しうる。一方、ILR を用いる場合には、

Working と Checkpointing、Checkpoint の3種類のバージョンが存在しうる。このために、チェックポインティング中は、ILR の方がより多くの記憶領域が必要である。しかし、ワークステーションを用いるクラスタにおいては、十分な量のディスク装置(一時ファイル)を一般に用いることができるので、チェックポインティング中の必要記憶量の増加は問題にならない。

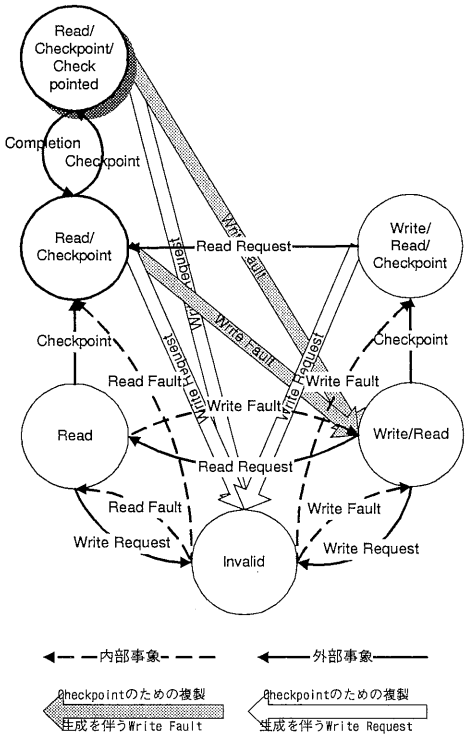

図 1　クライアントの状態遷移

提案の方式を用いた場合のクライアントの状態遷移図を図 1に示す。Working バージョンを含むデータの状態の遷移を表している。回復のためにのみ有効なデータの状態は紙面の都合から省略する。また、図が煩雑になるのを避けるために、発生しても状態が変わらない事象の遷移は示していない。遷移図のノード内ラベルは mode を表す。たとえば、Write/Read は、読み書き可能であることを表す。Read/Checkpoint は、読込み可能であり、そのデータが Working データ(コピー)としてだけではなく、Checkpointing データ(レプリカ)としても有効であることを表す。他も同様である。

ローカルに解決できない Read Fault、Write Fault が発生した場合には、その要求をマネージャに伝え、要求が満たされるのを待つ。状態 Invalid からの遷移先は、対象データの状態(mode)によって異なる。たとえば、状態が Read であった場合には、要求が満たされるクライアント(要求元)も Read となり、Read/Checkpoint であった場合には、要求元も Read/Checkpoint になる。

また、複製生成を伴う Write Fault の際に、ローカルにレプリカを独立させるだけでは、回復のための複製数が不足している(ローカルにのみレプリカが存在する)ならば、クライアントがマネージャにそのデータのレプリカを他のクライアントへ送ることを要求する。この複製は、マネージャが指示することも可能であるが、時間的な処理の分散を図るためには、アクセスを契機とするのが適当である。

2.3. マネージャ管理

マネージャは、分散共有記憶の一貫性制御およびチェックポイントの管理を行う。分散共有記憶の一貫性に関しては、従来と同様の制御を行う。チェックポイント管理は、一貫性のあるスナップショットの決定、回復のための複製の数の調整、チェックポインティング終了の検出からなる。スナップショットの決定は協調的アルゴリズムによって行う。複製数の調整は、複製が多くなる方向に行うのが基本である。マネージャが複雑になるが、必要数ちょうどに複製数を調整することも可能である。また、次のチェックポインティングを開始する前にチェックポインティングを完了させる必要がある。したがって、チェックポインティング開始後一定時間経過してもレプリカ数が十分にならない場合には、マネージャ

が明示的にスナップショットデータの複製を指示する。すべてのデータのスナップショットの複製数が2以上になった場合には、チェックポインティングを終了できる。Completion をクライアントに送ってチェックポインティングを完了させる。

通常の環境では単一ノード故障(停止)のみを考えるだけで十分と思われるが、多重故障への拡張も容易である。チェックポインティング終了に必要な複製数を変更するだけでよい。対応すべき故障の数をパラメータとして与えることも可能である。

3. 設計

一般的なワークステーションを活用できるよう回復可能分散共有記憶システムの移植性が高い必要がある。このために、ユーザレベルで回復可能分散共有記憶システムを構築する。カーネルを改造しない。UNIX ワークステーションで動作させることを考え、分散共有記憶の一貫性制御は、ページを単位とする。ローカルなページのマッピング管理にはシステムコール mmap、mprotect 等を利用する。SIGSEGV 等のシグナルによって、ページフォールトをシミュレートする。

3.1. プログラミングインタフェース

単一プロセッサおよび共有記憶型マルチプロセッサ用に作成した Portable Operating System Interface - POSIX.1c (IEEE 1003.1c) 互換のインタフェースを利用したユーザプログラムができるだけそのまま動作するシステムとする。スレッドの操作も POSIX.1c に準拠したライブラリを用意する。スレッドの生成は、負荷の低いノードにプロセスを生成することで実現する。

また、システムの環境に合わせた拡張を行う。たとえば、本稿の対象のクラスタでは、利用可能なノード数が変化することがある。探索問題などで任意の数のプロセッサを利用可能な場合には、利用可能ノード数に応じてスレッド数を調整することができ、調整した方が性能を高めることができる。この種の調整を可能とするために利用可能なノード数の増減によるシグナルを導入する。さらに、チェックポイントをプログラムから指示できるインタフェースを用意する。

3.2. 一貫性制御

単一プロセッサおよび共有記憶型マルチプロセッサで作成したユーザプログラムを一貫性制御のために変更することなく実行できるようにする。つまり、Sequential Consistency を一貫性制御の基本とする。また、プログラマが性能を向上させることができるように Entry Consistency を利用可能とする。プログラマは、領域を指定して Acquire/Release などのプリミティブをプログラムに追加することによって、性能向上を図ることができる。さらに、性能向上のための注釈(プリフェッチ等) [4] をユーザが付加できるようにする。

ワークステーションクラスタによる性能向上が期待できる探索問題での評価値管理などのために Reduce 演算を行う、比較的緩い一貫性を維持する領域も導入する。

3.3. 回復管理と I/O

ファイルディスクプリタの内部状態などの UNIX が管理する状態は、ユーザレベルでは完全に保存することができず、回復することもできない。また、I/O は実行してしまうと一般に回復できないので、回復可能 I/O ライブラリを用意する。

実際の I/O は、ユーザプログラムが起動されたノードで行われる。このため、起動したノードが停止し、I/O が行われようとした際には、プログラムの実行が停止することになる。このことは望ましいことではないが、異なるノードでの I/O の代替が必ずしも可能ではない。また、コンソールなどの回復できないデバイスへの出力中に障害が発生した場合には、その出力は回復されない。ただし、出力は、回復が不要になった時点(状態保存が行われた時点)で行われるので、矛盾や重複出力は発生しない。

4. 関連研究

前述のように [2]は、分散共有記憶のための複製をチェックポイントデータとして利用するものである。並列計算機(Intel Paragon)における実装において、この方式が有効であることを示している。また、回復のために複製したデータがプリフェッチの役割をして、回復機能を利用しない場合よりも利用した方が性能が高まる例が報告されている。

分散共有記憶と回復可能記憶の統合では、安定記憶を前提にするものがある[5,6,7] 。しかし、前述のように、ワークステーションを利用するクラスタでは、耐故障性を持った安定記憶システムの構築が、分散共有記憶や回復可能記憶システムの構築とは別に必要になり、システムが複雑になる。

2 次記憶装置を前提としない分散共有記憶に関する研究としては GVVM[8]や DVM[9]、Remote Caching Architecture[10] などがある。しかし、耐故障性については触れられていない。

5. おわりに

一般的なワークステーションによって構成されるクラスタにおいて、マルチスレッドプログラムを長時間実行するために重要な回復可能分散共有記憶システムの処理について説明した。分散共有記憶の複数の読み手のための複製(コピー)を回復のための複製(レプリカ)として利用し、コピーがない場合には遅延複製を行う点が特徴である。この特徴によって、回復可能にするためのオーバーヘッドを軽減でき、また、回復可能性実現のための処理の集中を避けることができる。

他ノードの記憶を利用して回復可能機能を実現する方式は、2 次記憶装置をすべてのノードに装着することが一般には難しい超並列計算機にも有効と考える。ただし、1 次記憶の有効利用のためにページアウト時の複製数調整が必要である。

考察したユーザレベル回復可能分散共有記憶システムの設計に基づいて UNIX ワークステーション上にシステムを実装し、その性能を評価する予定である。

参考文献

[1] Protic, Jelica, Milo Tomaševic and Veljko Milutinovic, "Distributed Shared Memory: Concepts and Systems," IEEE Parallel & Distributed Technology, Summer 1996, pp.63-79.

[2] Kermarrec, Anne-Marie, Gilbert Cabillic, Alain Gefflaut, Christine Morin, and I. Puaut, "A Recoverable Distributed Shared Memory Integrating Coherence and Recoverability," IEEE 25th Int'l Symp. on Fault-Tolerant Computing, pp.289-298, June 1995.

[3] Li, Kai and Paul Dudak, "Memory Coherence in Shared Virtual Memory Systems," ACM Trans. on Computer Systems, Vol.7, No.4, pp.321-359, Nov. 1989.

[4] 大澤 範高, 弓場 敏嗣, "ページ操作に対する注釈の低オーバヘッド実現機構," 信学技報 CPSY 96-34, pp.37-42, 1996.

[5] Wu, Kun-Lung and W. Kent Fuchs, "Recoverable Distributed Shared Virtual Memory," IEEE Trans. on Computers, Vol.39, No.4, pp.460-469, April 1990.

[6] Feeley, M. J., Jeffrey S. Chase, Vivek R. Narasayya, Henry M. Levy, "Integrating Coherency and Recovery in Distributed Systems," First Symposium on Operating Systems Design and Implementation (OSDI), pp.215-227, Nov., 1994.

[7] Carter, John B. et al., "Network Multicomputer Using Recoverable Distributed Shared Memory," COMPCON SPRING '93, pp. 63-75, 1993.

[8] 平野 聡, 田沼 均, 須崎 有康, 一杉 裕志, 塚本 亮治, "大域的仮想仮想記憶(GVVM)のマルチプロセス環境での評価", 並列処理シンポジウム JSPP '94, pp.365-372, 1994.

[9] Iftode, L., K. Li and K. Petersen, "Memory Servers for Multicomputers," Proc. of COMPCON Spring, pp.538-547, 1993.

[10] Leff, Avraham, Joel L. Wolf and Philip S. Yu, "Replication Algorithms in a Remote Caching Architecture," IEEE Trans. on Parallel and Distributed Systems, Vol.4, No.11, pp.1185-1204, Nov. 1993.