B-012

メモリ使用量削減を考慮したプロセスレプリケーション方式の設計

A Design of Process Replication Method to Reduce Memory Usage

菅原 智義†
SUGAWARA Tomoyoshi

1. はじめに

本稿では、プロセスレプリケーションの新方式である「複製ページング方式」を提案する。システム高信頼化手法の一つであるプロセスレプリケーションは、システムの突発的な障害に対し、短時間での復旧を可能にするが、メモリの使用量が多いという問題を有していた。提案方式では、プロセスの複製の保存先としてストレージを併用することにより、プロセスの複製による物理メモリの使用量を削減する。本方式を用いることで、従来はメモリの使用量が問題になり適用が難しかった、HPC 系アプリケーションや大規模ビジネスアプリケーションにもプロセスレプリケーションの適用範囲を広げることができる。

2. 従来のプロセスリカバリ技術と課題

電子政府やインターネット銀行など国民生活に深く関わる情報システムには、24 時間×365 日、サービスを提供し続けられる高信頼性が要求される。このような要求に応えるためには、障害や保守などによるサービスの停止時間を短くし、処理の失敗を少なくする必要がある。

サービスの停止は、定期保守など計画的な停止と予測困難なハードウェア故障など突然の停止に分けられる。計画的な停止に関して、我々はプロセスマイグレーション[1]を使って、サービスを提供するプロセスを予め退避させることにより、サービスの停止を利用者に隠蔽する解決策を示した。一方、予測困難なハードウェア故障など突然の停止の場合は、事前にプロセスを退避させることはできないため、停止から復旧までの時間を短縮する必要がある。

従来、システムのリカバリ方法としてはフェイルオーバが多く利用されてきた。しかし、フェイルオーバは復旧時に新たなプロセスを実行するだけで、元のプロセスの実行状態を引き継がないため、処理中のサービス要求の一部が失敗するという問題があった。

一方、プロセスの状態を引き継ぐことができるリカバリ方法として、チェックポインティング [2] とプロセスレプリケーション [3] が知られている。「チェックポインティング」は、ハードディスクなどの不揮発性記憶装置（ストレージ）に、プロセスのイメージを保存する方法である。ストレージへのアクセスを伴うため、障害からの復旧に時間がかかるという問題がある。「プロセスレプリケーション」は、他のノードにプロセスの複製を作成する方法である。障害からの復旧は極めて高速であるが、元のプロセスと同じサイズのプロセスを複製として作成するため、メモリの使用量が多いという問題がある。

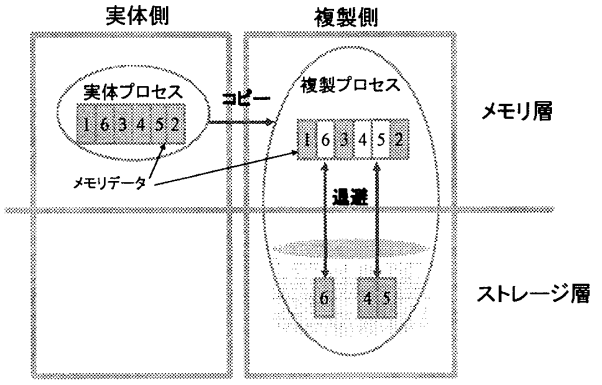
3. 複製ページング方式の提案

我々は、障害復旧の高速性とサービスの継続性という点から、上述の高信頼性を求められるシステムのリカバリ方法としては、プロセスレプリケーションが最適であると判断した。しかし、メモリ使用量の多さを考えると、従来のプロセスレプリケーションをそのまま実際のシステムに適用することは難しかった。

そこで、我々は、プロセスレプリケーションの特徴である障害復旧の高速性を損なうことなく、従来のプロセスレプリケーションに比べて、メモリ使用量を削減することが可能な「複製ページング方式」を提案する。

本方式の考案のポイントは以下の通りである

- 利用される可能性が低いプロセスの複製を保持するために多量のメモリを常に使用しないこと
- プロセスが使用するメモリ上のデータのうち、アクセス頻度が低い部分をストレージに保持すること
- ストレージへのアクセスが実行性能や復旧時間に直接影響させないこと



メモリデータ内の数字は最近、アクセスされた順番（1が一番最近アクセスされた）を示す。

図 1 複製ページング方式の概念図

図 1 に本方式の概念図を示す。本方式の特徴は、複製プロセスがメモリ層とストレージ層の 2 階層に分かれて存在することである。以下の本方式の仕組みについて簡単に説明する。

本方式でも、従来のプロセスレプリケーションと同様、複製元のプロセス（以下、実体プロセス）が使用するメモリ上のデータ（以下、メモリデータ）やカーネルが管理するデータを、定期的に複製先プロセス（以下、複製プロセス）にコピーする。

さらに、本方式では、実体プロセスにより、最近、アクセスされていないメモリデータに関して、それに対応する複製プロセスのメモリデータをストレージに退避させる。複製ページング方式という名前は、このストレージへの退避処理がページングに似ていることに由来する。

本方式では、複製プロセスのメモリデータをストレージに置くことにより、複製プロセスによるメモリ使用量を削減する。例えば、プロセスのメモリデータの全体に対する、最近、アクセスされたメモリデータ量の比率が 30%であっ

†NEC システムプラットフォーム研究所

たとすると、複製プロセスによるメモリの使用量は従来のプロセスレプリケーションに比べて 70%削減できる。最近、アクセスされたメモリデータの比率が小さいほど、本方式によるメモリ使用量削減の効果は大きくなる。

一方で、複製プロセスのメモリデータのうち、最近、使用されていた領域はメモリ上に置かれるため、復旧時にも、ストレージへのアクセスなしで、所望のコードやデータにアクセスすることが可能である。また、実体プロセスに関してはページングする必要はないので、実体プロセスの実行速度は低下しない。

4. 設計

4.1 システム構成

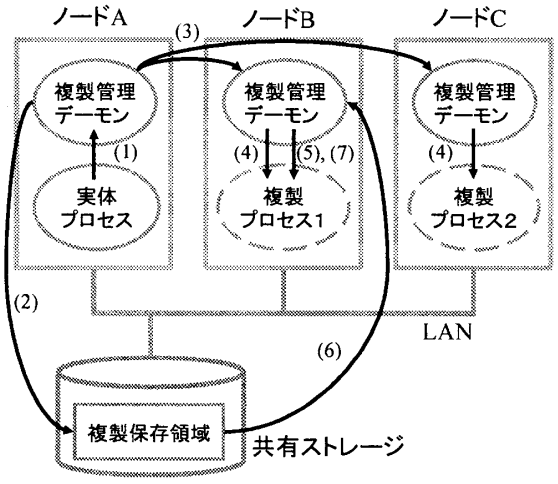


図 2 システム構成図

図 2 に複製ページング方式を実現するシステムの全体構成を示す。本設計では、複製ページング方式を OS レベルで実現する方針とした。このため、複製ページング方式の処理の主体を複製管理デーモンとして実現する。このデーモンはすべてのノードに配置され、LAN を介してお互いに通信する。

また、今回の設計では、ストレージ上のメモリデータへのアクセスの容易性から、メモリデータを保存するストレージとして LAN ですべてのノードに接続された共有ストレージを利用する。

4.2 複製管理デーモン

複製管理デーモンの主要な機能を以下に示す。

A) 複製プロセスの作成・削除
B) 複製プロセスの更新
C) 複製プロセスのページング（メモリデータの部分的な開放とストレージへの保存）
D) 複製プロセスを用いた復旧

複製管理デーモンは、上記の機能を実現するために、以下の 2 つのデータ構造を保持する。

- 複製管理テーブル
  実体プロセスとそれに対応する複製プロセスがどのノードに存在するかを管理する。また、プロセスを復旧する場合に、どの複製プロセスを利用するかを決めるための複製の順位付けも管理する。
- プロセス状態テーブル
  実体プロセスのメモリデータのうち、どの部分が更新されたか、どの部分が、最近、アクセスされていないかを保持する。複製プロセスの更新や複製プロセスのページングの時に用いられる。

複製管理デーモンの処理のうち本方式で特徴的な部分である、「C) 複製プロセスのページング」と「D) 複製プロセスを用いた復旧」について説明する。A)と B)については従来の方式とほぼ同様のため説明を省略する。

【C) 複製プロセスのページング】複製管理デーモンは、プロセス状態テーブルを参照し、実体プロセスのメモリデータのうち、最近、アクセスされていない領域を調べて（図中(1)）、共有ストレージに保存する（図中(2)）。同時に、複製管理テーブルを参照し、複製プロセスが存在するノードの複製管理デーモンに、複製プロセスの該当するメモリ領域の開放を要求する（図中(3),(4)）。

【D) 複製プロセスを用いた復旧】障害が発生し、実体プロセスが停止した場合、複製管理デーモンは、複製管理テーブルに含まれる順位に基づいて、複製プロセスの一つを選択し（図中、複製プロセス 1）、それを元の実体プロセスの代わりに実行状態に移す（図中(5)）。以下、このプロセスを復旧プロセスと呼ぶ。この状態では、まだ、復旧プロセスは一部のメモリデータしか持たずに動作する。復旧プロセスが実行されているバックグラウンドで、複製管理デーモンは共有ストレージからメモリデータを読み出して（図中(6)）、復旧プロセスのメモリデータをすべてメモリ上に復帰させる（図中(7)）。

なお、復旧プロセスの実行中にページフォールトが起こる可能性がある。その場合は従来のページフォールトと同様に、該当するページを共有ストレージから読み込んだ後で、復旧プロセスの実行を再開させる。

7. まとめ

本稿ではプロセスレプリケーションのメモリ使用量を削減する「複製ページング方式」を提案し、その設計について述べた。本方式を用いることで、プロセスレプリケーションの適用範囲が拡大すると期待できる。

今後は、本方式の実装、評価を進めるとともに、障害検出、障害箇所切離しの技術と組み合わせて、高速なリカバリシステムを構築する予定である。

謝辞

本研究は、独立行政法人 新エネルギー・産業技術開発機構 基盤技術研究促進事業(民間基盤技術研究支援制度)の一環として委託を受け実施している「大規模・高信頼サーバの研究」の成果である。

参考文献

[1] 高橋ほか, データセンタ環境に適した TCP 無切断プロセスマイグレーションの実現, 情報処理学会研究報告 2004-OS-96, pp.29-36, 2004.
[2] Condor Version 6.6.5 Manual, http://www.cs.wisc.edu/condor/manual/v6.6/4_2Condor_s_Checkpoint.html, 2004.
[3] Helal, et al., Replication Techniques in Distributed Systems, Kluwer Academic Publications, 1996.