1A-1

SCoreクラスタシステムにおけるスレッドチェックポイント実装

安井 隆　　清水 正明

株式会社 日立製作所 中央研究所

1. はじめに

近年，構築費用が安価でありコストパフォーマンスに優れることから，計算機センタにおいても PC サーバを構成要素とする大規模なクラスタシステムの導入が進んでいる。しかし，多数の計算機をネットワークで結合するクラスタシステムは構成要素が多いため故障率が高く，システムの信頼性が低下する問題がある。この問題に対し，システムの信頼性を向上させるソフトウェア技術として，プログラムの実行状態の退避/回復を実現するチェックポイント機能に関心が集まっている。

クラスタシステム用の並列プログラム実行環境として，SCore クラスタシステムソフトウェアが広く利用されている。SCore クラスタシステムは，チェックポイント機能を備えているが，本チェックポイント機能はマルチスレッド及びダイナミックリンクを使用したプログラムには対応していない。これに対し，科学技術計算においてスレッドレベルの並列化やライブラリの動的なリンクを行うプログラムは増加しており，SCore クラスタシステムのチェックポイント機能の拡張が望まれている。

そこで，SCore クラスタシステムに対しマルチスレッド及びダイナミックリンクに対応したチェックポイント機能を提供する SCore Checkpoint/Restart(SCCR)を検討した。本稿では SCCR プロトタイプについて述べる。

2. SCoreクラスタシステムの概要

SCore クラスタシステムソフトウェアは，PC クラスタコンソーシアムが開発/保守している高性能並列プログラム実行環境である。SCore クラスタシステムは，並列プログラム実行時に PMv2[1]と呼ばれる低レベルライブラリを使用し，TCP/IP を介さない高速なノード間通信を実現する。また，SCore-D[2]と呼ばれる並列プログラム実行環境によりプログラムを制御する。

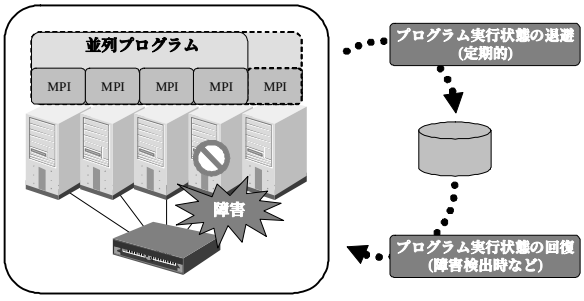

図 1: チェックポイント機能の概要

SCore クラスタシステムは大量のデータを数時間から数日かけて処理する科学技術計算で利用されている。故障率が高いクラスタシステム上で長時間に渡る大規模並列プログラムを実行可能とするために，SCore クラスタシステムは，プログラムの実行状態を定期的にディスクへ退避し，障害時や構成変更時に状態を回復して実行を再開するチェックポイント機能を備えている(図 1)。

次に SCore クラスタシステムにおけるチェックポイント機能の実現形態を図 2に示す。

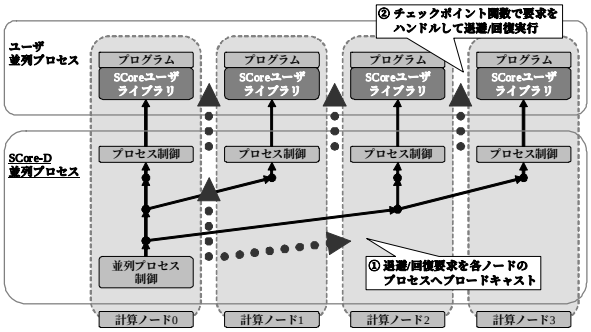

図 2: チェックポイント機能の実現形態

SCore クラスタシステムでは，並列プログラムに対してチェックポイント関数を含む SCore ユーザライブラリをリンクする。実行時には並列プログラムを各ノードにて SCore-D の子プロセスとして生成し，SCore-D が実行を制御する。SCore クラスタシステムのチェックポイント機能は，SCore-D から実行中の子プロセスに対して退避/回復処理への移行要求を発行し，リンクしたチェックポイント関数でこの要求をハンドルすることで実現している。ただし，ノード間通信の退避/復元は，SCore-D が PMv2 のプロトコルを利用して別途実現している。

3. SCore Checkpoint/Restart(SCCR)の設計

SCore クラスタシステムのチェックポイント機能はユーザレベルで実現されている。マルチスレッド及びダイナミックリンクを使用したプログラムの退避/回復を実現するためには，カーネル内部に存在するスレッド ID やライブラリのメモリ配置などのカーネル内部に存在する情報へのアクセスが必要となる。そこで，SCore クラスタシステムに対してカーネル内部へのアクセス機能を提供する SCCR について検討した。

SCCR の実現に当たっては，2 つの設計方針を立てた。

(1) カーネル本体を再コンパイルすることなくカーネル内部の情報の退避/回復を実現し，商用 Linux ディストリビューションへの組み込みを可能とする。

(2) カーネル内部における退避/回復は，マルチスレッド及びダイナミックリンクに関する情報に限定し，従来の SCore クラスタシステムの構造を活用する。

プロトタイプ設計では，2 つの方針に基づいて SCCR をカーネルモジュール/システムライブラリ/ユーザライブラリの 3 つの要素で構成し，SCore クラスタシステム

Implementation of a Thread Checkpointing on SCore Cluster System,
Takashi YASUI and Masaaki SHIMIZU,
Hitachi, Ltd., Central Research Laboratory.

に組み込むことにした(図 3)。

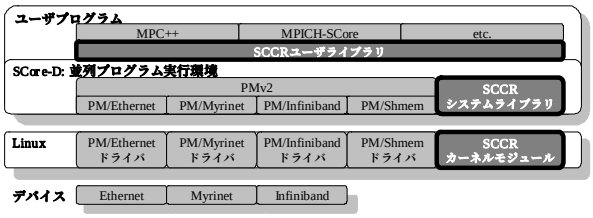

図 3: SCCR 実装時のソフトウェアレイヤ

SCCR カーネルモジュールは，SCCR のコアであり，スレッド ID を含むプロセス構造体やライブラリのメモリ配置を含むメモリ構造体へのアクセスを実現する。カーネルモジュールは，カーネルへの動的な挿抜が可能なオブジェクトであるため，1 つ目の方針を達成できる。

また，SCCR システムライブラリ/ユーザライブラリは，SCore-D/SCore ユーザライブラリと SCCR カーネルモジュールのインタフェース及びノード間の同期制御を実現する。SCCR カーネルモジュールを介してスレッド単位処理や ID 復元処理に対応しつつ SCore-D と連携したノード間の待ち合せを可能とし，2 つ目の方針を達成する。

SCore クラスタシステムと SCCR が連携してプログラムの実行状態を退避/回復する動作を図 4に示す。ネットワーク情報の退避/回復などの基本動作は従来の SCore クラスタシステムのフローを踏襲するが，実行停止処理や回復プロセス生成において SCCR が介在することで，マルチスレッド及びダイナミックリンクに対応している。

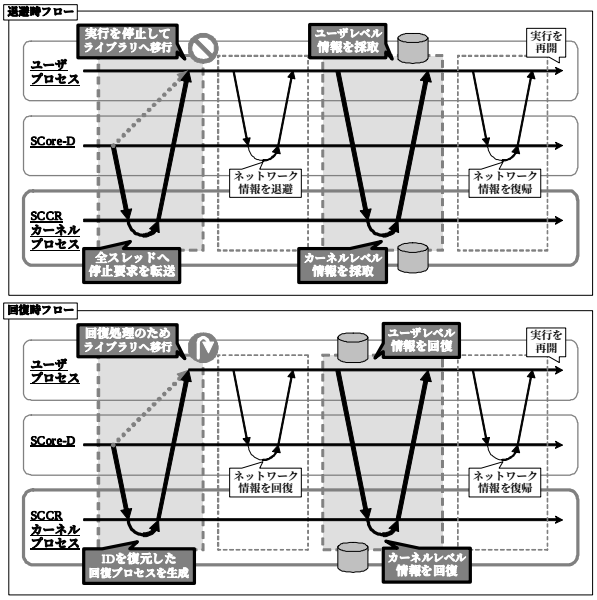

図 4: SCCR 実装時の実行状態退避/回復フロー

ここまでに示した設計にしたがって SCCR のプロトタイプを開発し，マルチスレッド及びダイナミックリンクに対応可能であることを確認できた。

4. 他のチェックポイント機能との比較

SCCR と同様のカーネルレベルのチェックポイント機能を持つものには Lawrence National Berkeley Laboratory の BLCR[3]や Los Alamos National Laboratory の TICK[4]がある。また，ユーザレベルのチェックポイント機能を持つものには Seoul National University の M^3[5]がある。これらと SCCR を実装した SCore クラスタシステムの比較結果を表 1に示す。

表 1: チェックポイント機能の比較

機能	対応状況 SCore/SCCR	BLCR	TICK	M^3
MPI並列プログラム	○	×	×	○
スレッド並列プログラム	○	○	×	×
ダイナミックリンク	○	○	×	×
障害検出/リカバリ	○	×	×	○
差分チェックポイント	×	×	○	×

SCCR を実装した SCore クラスタシステムのチェックポイント機能は，前回退避した実行状態からの差分を採取する差分チェックポイント機能はないが，マルチスレッド及びダイナミックリンクを使用したプログラムの退避/回復が可能であり，幅広いプログラムに対しシステムの信頼性を向上できる。

5. おわりに

SCore クラスタシステムのチェックポイント機能を拡張し，マルチスレッド及びダイナミックリンクを使用したプログラムの退避/回復を実現する SCCR のプロトタイプを開発した。SCCR のコアをカーネルモジュールとして実装することで，カーネルへの影響を最小限に抑えつつカーネル内部の情報へのアクセスを実現した。更に SCore クラスタシステム上に本プロトタイプを実装し，スレッドレベルの並列化が行われたプログラムにもチェックポイント機能を適用可能であることを確認した。

現在，SCore クラスタシステムソフトウェアのバージョン 6 への実装に向けた検証およびプログラムの実行性能に与える影響の評価を推進中である。

謝辞

PC クラスタコンソーシアム開発部会において，本研究に関する有益なご助言をいただいた，東京大学の石川 裕氏，Allinea Software Ltd.の堀 敦史氏，株式会社富士通研究所の住元 真司氏，株式会社 NEC 情報システムズの長谷川 篤史氏，PC クラスタコンソーシアムの亀山 豊久氏に心より感謝致します。

参考文献

[1] T. Takahashi, S. Sumimoto, A. Hori, H. Harada, and Y. Ishikawa. PM2: High Performance Communication Middleware for Heterogeneous Network Environments. In SC2000, Nov. 2000.

[2] A. Hori, H. Tezuka, and Y. Ishikawa. Highly Efficient Gang Scheduling Implementation. In SC'98, Nov. 1998.

[3] J. Duell. The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart. Technical Report of Lawrence National Berkeley Laboratory, 2003.

[4] R. Gioiosa, J. C. Sancho, S. Jiang, F. Petrini, and K. Davis. Transparent, Incremental Checkpointing at Kernel Level: a Foundation for Fault Tolerance for Parallel Computers. In SC|05, Nov. 2005.

[5] H. Jung, D. Shin, H. Han, J. W. Kim, H. Y. Yeom, and J. Lee. Design and Implementation of Multiple Fault-Tolerant MPI over Myrinet (M^3). In SC|05, Nov. 2005.