ヘテロジニアスシステム向けリモートプロセス管理機能

清水 正明[†1,†2] 小笠原 克久[†3]
舩生 真紀子[†4] 米澤 明憲[†2]

異なるプロセッサアーキテクチャの計算機ノードを多数組み合わせたヘテロジニアスシステムにおいて，利用者がプロセッサの相違やノード数を意識することなく効率的に並列プログラムを実行できるリモートプロセス管理アーキテクチャを提案する．ペタスケールシステムの性能要求および制限を満たすためには，目的に最適なプロセッサで最適な機能を実行する必要があるが，OS の機能を分散することとヘテロジニアスシステムを一元的に扱うことには複数の課題がある．本稿では，ヘテロジニアスノード上の Linux 間で OS 機能を分散し，またヘテロジニアスノード間のユーザプロセスを一元管理するシングルシステムアーキテクチャについて述べる．また，本機能を IA-32 ノードと複数の SH-4 ノードで構成するクラスタシステム上に実装した結果および機能評価について述べる．

Remote Process Management for the Heterogeneous System

Masaaki Shimizu,[†1,†2] Katsuhisa Ogasawara,[†3] Makiko Funyu[†4]
and Akinori Yonezawa[†2]

In this paper we propose a remote process management architecture for execution of parallel programs on a large heterogeneous system without requiring consideration of the large number of nodes or of the different processor architectures used in the nodes of the heterogeneous system. In the field of Petascale supercomputing, to satisfy the demand for performance and limitation, it is necessary for the best suited processor to execute the best function, but the problem is how to manage the many nodes and processor architectures in a uniform way. This paper describes a single system architecture that enables distribute OS function and remote process management between the Linux OSes running on each of the heterogeneous nodes, and describes implementation and evaluation on a cluster system of one IA-32 node and three SH-4 nodes.

1. はじめに

スーパーコンピュータのシミュレーションを用いた研究開発手法は，従来からある実験や理論と並ぶ重要な技術となってきている．このシミュレーションの利用分野と質を高めるために，年率 1.8 倍程度で要求性能が伸びており，スーパーコンピュータランキング Top500[1] の性能変遷から外挿すると 2011 年には 10 PetaFlops もの性能が必要になる．2007 年 7 月現在の世界最大性能のスーパーコンピュータは IBM 社 BlueGene/L[2] の実効 280 TeraFlops であるが，2011 年に実効 10 PetaFlops を達成するためには，さらに 36 倍もの性能向上が必要である．そのため，日米でペタスケールの性能を実現するプロジェクトが進行中であり，米国では DARPA が推進する HPCS（High Productivity Computing Systems）プログラムとして IBM 社と Cray 社が 2010 年に 1 PetaFlops を超える性能のコンピュータを実現する開発を行っている．また，日本では文部科学省および理化学研究所が 10 PetaFlops を実現する「次世代スーパーコンピュータ」プロジェクトを推進している．

このスーパーコンピュータへの要求性能の伸びに対して，従来はプロセッサの高速化で対応してきたが，近年ではプロセッサコア単体の性能向上の伸び悩みがあり，目的の総合演算性能を達成するためにはプロセッサコアを数万から数十万個用いたシステムを実現する必要がある．現在の PC クラスタのように汎用の計算機ノードを多数結合するアーキテクチャを採用し

†1 株式会社日立製作所中央研究所
Central Research Laboratory, Hitachi, Ltd.
†2 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†3 株式会社日立製作所システム開発研究所
System Development Laboratory, Hitachi, Ltd.
†4 株式会社日立超 LSI システムズ
Hitachi ULSI Systems Co., Ltd.

た場合には，4〜8 コアのプロセッサを 2〜4 個搭載した 16waySMP 程度のノードを数万個接続して実現することになる．

この規模のシステムになると，演算性能の達成ばかりでなく，消費電力や冷却そして設置面積が大きな問題になることから，見かけ上の性能だけでなく，アプリケーションレベルで高い実行効率を達成していることが必須となる．そのためには，すべてのノードで同種類の汎用プロセッサを搭載するホモジニアス構成とするのではなく，演算を行うユーザプログラムは演算に特化したプロセッサもしくは演算加速器を利用して実行し，システム管理や入出力（I/O）処理にはそれぞれの用途に向いたプロセッサを用いるヘテロジニアス構成のシステムにより，全体の実行効率を高める手法が必要になると考えている．

我々は，電力・冷却・設置面積の要求を満たしつつ 2011 年に 10 PetaFlops 級のスーパーコンピュータを実現するには，前述のとおり演算に特化したノードを持つヘテロジニアス構成が必須と考え，汎用プロセッサノードと演算専用のプロセッサノードを持つシステムを想定して，システムソフトウェアアーキテクチャを検討した．

本稿では，異なるプロセッサアーキテクチャのノードを多数組み合わせたヘテロジニアスシステムにおいて，利用者がシングルシステムイメージ（Single System Image）で並列プログラムを管理し，また高い実行効率で実行することを可能とするリモートプロセス管理方法を示す．このリモートプロセス機能はヘテロジニアスシステム上の並列プロセスを一元管理し，複数のヘテロジニアスノード間でシングルプロセス名空間を実現する．利用者は従来の単一ノード用 OS と同じインタフェースを利用してヘテロジニアスクラスタシステム上で並列プログラムを管理・実行可能である．また本機能により演算ノードの機能をユーザプログラムの実行に必要な最低限に削減することが可能となり，OS による外乱を減らし，演算ノードのプロセッサの効率を高めることが可能となる．このリモートプロセス機能を Intel IA-32[3] ノードと複数の RENESAS SH-4 (SuperH RISC engine)[4] ノードからなるクラスタに実装した結果および機能評価について述べる．

以下 2 章では想定しているヘテロジニアスシステムの課題について述べ，課題解決に向けたアプローチを検討する．3 章でヘテロジニアスシステム向けリモートプロセス管理機能の概要と IA-32+SH-4 環境における実装について述べ，続く 4 章で実装した機能の評価を行う．5 章で関連研究について述べ，最後に 6 章で本稿をまとめる．

2. ヘテロジニアスシステムの課題

本稿で想定するヘテロジニアスシステムを**図 1** に示す．システム管理および I/O を行う複数の制御ノードと，ユーザプログラムの実行に特化した複数の演算ノードで構成している．これらのノード間はメモリ共有をせず高速ノード間ネットワークで接続している．

2.1 課　　題

ヘテロジニアスシステムで 10 PetaFlops 級の大規模システムを実現するには，次の課題がある．

(A) ヘテロジニアスシステムの一元管理機能の実現

一般に，プロセッサアーキテクチャが異なる計算機システムでは，各アーキテクチャ専用の OS がそれぞれの計算機を管理する．たとえば同じ Linux という名前であっても IA-32 用と SH-4 用では各プロセッサ向けに専用実装された別の OS イメージが実行される．同様にユーザプログラムも各アーキテクチャ専用の実行イメージとなる．しかし，はじめに述べたように最適なプロセッサで最適な機能を実行するためには，ヘテロジニアスなシステムを統合して管理し，使いやすく提供する必要がある．具体的には，ヘテロジニアスシステムをシングルシステムイメージで管理する OS 機能が必要である．

(B) 演算ノードの実行効率向上

システムの実行効率を高めるためには，ユーザプログラムを実行する演算プロセッサおよびノードの性能を最大限発揮できるシステム構成にする必要がある．

特に OS の機能が割込みやシステムコール契機で呼び出されると，プロセッサ時間の消費，キャッシュ汚染，メモリアクセス帯域・ファイルキャッシュ領域の消費等でユーザプログラム実効性能の低下が発生する．マルチプロセッサ環境でスレッド並列プログラムを実

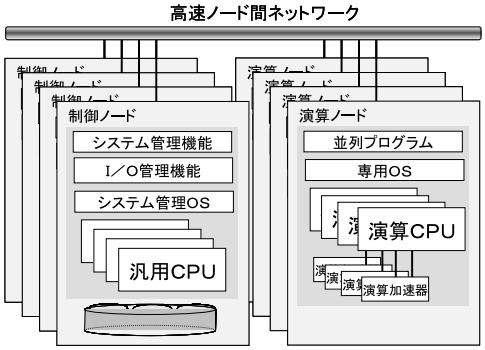

図 1 ヘテロジニアスクラスタシステム
Fig. 1 Heterogeneous cluster system.

行している場合には，OS 処理を割込み等で実行したプロセッサが上記の理由で遅れるとプロセッサ間同期の際に他のプロセッサを待たせることになるため[5]，この問題はより顕著になる．よって，OS 処理による外乱を減らす機構が必要である．

(C) 大規模システムの信頼性向上

10 PetaFlops を実現するためには数十万個のプロセッサ，数万もの計算機ノードを使用することになる．単体のプロセッサおよびノードとしての信頼性は十分であったとしても，シングルポイントの障害が全系の障害につながる場合には，システムトータルとしての信頼性は数万〜数十万分の一になってしまう．したがって，ハードウェアとしての十分な信頼性向上とともに，部分的なハードウェア障害があってもシステム全体としてはサービスを継続できる OS 機能・構造が必要である．

(D) 大規模システムの性能向上

数万計算機ノードのシステムにおいてプロセスを管理するノードが 1 台であったり，I/O を行うノードが 1 台では，性能ボトルネックが発生してノード数分の性能を発揮することはできない．信頼性確保とともに性能のスケーラビリティを発揮できる OS 機能・構造が必要である．

(E) 発展性・拡張性の実現

演算ノードの実行効率向上，ヘテロジニアスシステムのシングルシステムイメージ化，信頼性向上等を実現するためには，専用のシステムソフトウェアが必要な部分もある．しかし，Linux 等のデファクトインタフェースとの互換性や，最新の技術に対応して機能の向上を継続的に図っていくという要求もあり，オープンソース技術等を取り込みやすい構造である必要がある．

2.2 課題解決に向けたアプローチ

2.1 節にあげた課題を解決するために，次のアプローチをとる．提案するソフトウェア構成を**図 2** に示す．

汎用プロセッサの制御ノード 1 台に対して演算専用プロセッサの演算ノード 8〜16 台を対応させ，制御ノードにおいて演算ノードのプロセスおよび I/O を管理する．制御ノードでは汎用 Linux を実行してシステムを管理し，演算ノードではユーザプログラム実行のための最小限の機能を持つ OS とユーザプログラムのみを実行する．システムの大規模化は，上述した制御ノードと演算ノードの組合せを基本単位として，この単位を並列に並べることで実現する．

以下，提案するソフトウェア構成の特長をまとめる．

(1) ヘテロジニアスノード間でリモートプロセス実現

並列プログラムのプロセス管理は制御ノードで行い，

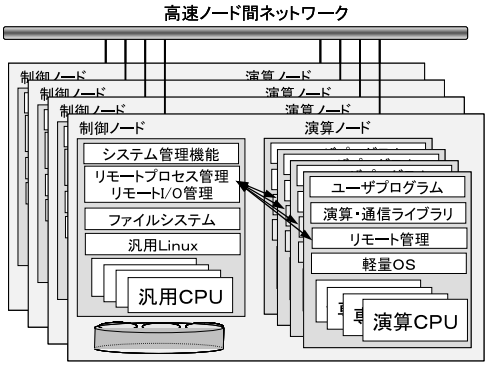

図 2 提案するソフトウェア構成
Fig. 2 Software overview of the heterogeneous system.

演算ノードで実行すべきプログラムは演算ノードに転送して実行する．このとき，演算ノードで実行されるプロセスも制御ノードのプロセスツリー上にマッピングして管理する．利用者からのインタフェースとしては制御ノードのプロセス名空間のみが見え，ユーザプログラムのオフロード先である演算ノードは隠蔽されることになる．たとえば並列プログラムの実行では，制御ノード上で複数の MPI プロセスが実行されているように見える．本リモートプロセス機能ではプログラム転送のほかに，ヘテロジニアス環境でリモートプロセスの実行に必要なシグナル・システムコールの転送，引数・エンディアンの変換およびデータ転送を行う．

(2) 演算ノードからプログラム実行機能以外を除外

演算ノードにおいて OS によるユーザプログラムに対する性能外乱を減らす手法としては，日立 SR8000[6] のようにユーザプログラムが使用するプロセッサ以外に OS 専用のプロセッサを持つ方法もある．しかし，演算ノードのコストと複雑さが増すことと，(5) で後述するように汎用機能・管理機能は極力制御ノードにオフロードする方針から，演算ノードではユーザプログラム実行に必要な機能以外を持たないことにする．具体的には，ユーザプログラム実体，演算ライブラリ，MPI 等の通信ライブラリ，ノード間ネットワーク用ドライバの最小限の OS 機能のみを持ち，制御ノードからのプログラム実行要求のみに応えることで演算プロセッサの高い実行効率を発揮させる．

ユーザ管理，システム管理等の機能，ファイルサーバ等のサービス機能は制御ノードが提供する．演算ノードのユーザプログラムのファイル I/O 要求は制御ノードに転送して実行する．したがって，制御ノードがファイルシステム，ディスクドライバ等を持つ．制御ノードは複数の演算ノードのプロセス管理および I/O 代行を行う．

(3) 演算ノードと制御ノードの障害アイソレーション

制御ノードと演算ノードの比率はリモートプロセスの管理，I/O 代行等の負荷を考慮して 1 対 8〜1 対 16 程度を想定している．したがって，単純にノード数から考えるとシステムノード数の 90%以上は演算ノードであり，演算ノードの障害やダウンによってシステム全体がダウンしないことは重要である．制御ノードと演算ノード間のリモートプロセス管理，リモート I/O 機能において，タイムアウト機構つきの通信や監視機能を利用することで演算ノードのダウンで制御ノードがダウンすることを防ぐ．

(4) 制御ノード単位のクラスタ化

(1)〜(3) の技術を実現することで 8〜16 個の演算ノードを含むヘテロジニアスノード群を 1 台の制御ノードとして見せることができる．この制御ノード（+8〜16 台の演算ノード）を数千ノードクラスタ化することで，システム全体としては数万〜数十万のヘテロジニアスプロセッサを含むノードを管理することができる．制御ノードをクラスタリングする技術としては PC クラスタ技術および，本稿で提案した制御ノード間・演算ノード間でプロセス名空間を統合する技術を拡張することを検討する．

(5) 汎用 OS と専用 OS の混在

汎用機能であるサーバ機能や最新デバイスに対応するデバイスドライバ等は，制御ノードに汎用 OS を用いることで最新技術との親和性を高くする．逆に演算ノード OS は，汎用 OS と比較すると機能を制限し，さらに部分的に専用開発を行って性能を追及する．このとき，演算ノードと制御ノード間のリモートファイル I/O インタフェースは，演算ノード側が制御ノード側で利用している個別のファイルシステムやデバイスの実装に依存しないようにすることで，制御ノード側の変更のみで最新の技術に追従できるようにする．

3. ヘテロジニアスシステム向けリモートプロセス管理機能

2 章で提案したアプローチを実現する，ヘテロジニアス向けリモートプロセス機能を IA-32 プロセッサの制御ノードと SH-4 プロセッサの演算ノードに実装した．ハードウェアおよびソフトウェアの概要を**表 1** に示す．

3.1 リモートプロセス管理機能の概要

図 3 に示すように，制御ノード上 Linux のプロセス名空間で管理されているユーザプロセスを演算ノードにオフロードして実行する．オフロードするか否かの判定は実行ファイルの ELF ヘッダのアーキテクチャ種類で行い，ヘテロ対応バイナリローダが転送および実行依頼を行う（**図 4**）．複数演算ノードにオフロードしているプロセスを制御ノードで一元管理することで，プロセス管理のシングルシステムイメージを提供する．また，ターゲットアーキテクチャごとにバイナリローダを用意することで，複数種類のアーキテクチャの演算ノードを混在管理することも可能である．制御ノードではオフロードしているプロセス以外の，たとえば演算ノードの init プロセス等は管理しない．

演算ノード上のリモートプロセスに対しては，**図 5** に示すように制御ノード側に，対応する pseudo ユーザプロセスを用意し，リモートプロセスのカーネルスレッド（delegate kernel thread）からのシステムコール依頼の処理，プロセス状態の同期，シグナル処理を行う．

現在の実装では，リモートプロセス実行時に制御

表 1 試作したヘテロジニアスシステム構成
Table 1 Configuration of the heterogeneous system.

	制御ノード	演算ノード
計算機ノード	CPU:Intel Pentium4 1.6GHz メモリ:1GByte	CPU:Renesas SH-4(SH7751R) 266MHz メモリ:64MByte
Boot	Local HDD	Local HDD, or NetBoot
Network	100Mbit Ethernet	100Mbit Ethernet
OS	Linux 2.6.12(FC4)	Linux 2.6.12(Debian)
ノード数	1 台	3 台

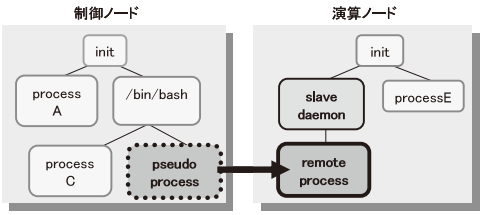

図 3 制御ノードと演算ノードのプロセスツリーの関係
Fig. 3 Process tree relationship of management node with processing node.

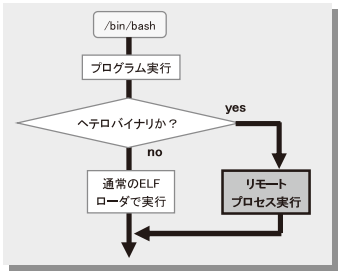

図 4 実行ファイルの判定と転送
Fig. 4 Selection of execution node.

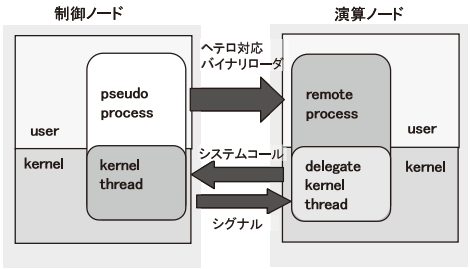

図 5 pseudo プロセスとリモートプロセス
Fig. 5 Relationship of pseudo process and remote process.

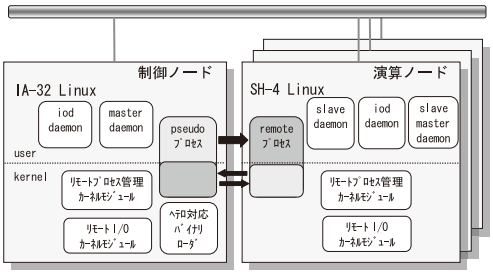

図 6 ソフトウェア構成
Fig. 6 Software configuration on management node and processing node.

ノードの親プロセスから演算ノードの子プロセスに継承するのは標準入出力のみであり，他のオープンファイルディスクリプタやソケット，アドレス空間は引き継がない．また，オフロード実行するユーザプログラムはターゲットアーキテクチャ上で動作する既存のオブジェクトを変更なしで実行可能である．

3.2 リモートプロセス管理機能を実現する構成

制御ノードと演算ノード上で実行するソフトウェア構成を**図 6** に示す．

本実装では制御ノードおよび演算ノードの OS は Linux ベースとすることでプロセス管理構造体，システムコール番号，ファイルシステム構造等極力互換性を高くした．また，リモートプロセス機能はホモジニアスクラスタ向け分散プロセス機能である BProc[7]（Beowulf Distributed Process Space）をベースとして，ヘテロジニアス対応機能を追加することで実現した．このヘテロジニアス対応リモートプロセス機能はリモートプロセス管理デーモン，ヘテロ対応バイナリローダ，リモート I/O 管理モジュール，リモートプロセス管理カーネルモジュール，ヘテロ対応プロセス構造体によって実現している．BProc はホモジニアスノード用のため，リモートプロセス空間生成時に VMADump 機能を使用して親プロセス空間をコピーしているが，本実装ではヘテロジニアスノード対応のために専用のヘテロ対応バイナリローダを使用している．また，リモート I/O 用機能は新規に設計実装した．BProc に対する本実装のソースコード差分を**表 2** に示す．

表 2 BProc とのソースコード差分
Table 2 Difference with BProc source code.

	BProc	ヘテロ対応リモートプロセス差分
カーネル変更・カーネルモジュール	9052 行	1446 行
VMADump	1525 行	—
ヘテロ対応バイナリローダ	—	2840 行
リモート I/O	—	9347 行
デーモン	4088 行	145 行
合計	14665 行	13778 行(追加差分)

(1) 制御ノード

制御ノードの構成モジュールを次に示す．

- **master daemon**
  演算ノード上の slave daemon と通信し，リモートプロセスの実行・管理に関するメッセージを交換する．
- **pseudo プロセス**
  演算ノード上で実行されるリモートプロセスに対して 1 対 1 に対応する．pseudo プロセス上ではユーザプログラムは実行せず，シグナル・システムコールの代行を行うカーネルスレッドとして機能する．
- **iod daemon**
  演算ノード上のユーザプロセスの標準入出力を制御ノード上の pseudo プロセスと接続するための通信路を提供する．
- **リモートプロセス管理カーネルモジュール**
  pseudo プロセスの CPU 時間・メモリ使用量等の状態管理，シグナル・システムコールの転送を行う．
- **ヘテロ対応バイナリローダ**
  演算ノード用の実行ファイルを判定すると，引数・環境変数を slave daemon に転送し，実行を依頼する．
- **リモート I/O カーネルモジュール**
  演算ノードから依頼されたリモート I/O システムコールを実行し，結果を演算ノードに返す．また，演算ノードで発生したメモリマップドファイルのページフォールトに対してデータ転送して解決する機能を提供する．

(2) 演算ノード

制御ノードの構成モジュールを次に示す．

- **slave master daemon**
  演算ノード上の複数の slave daemon を管理する親 daemon.

- **slave daemon**
  制御ノード上の master daemon と通信し，リモートプロセスの実行・管理に関するメッセージを交換する．
- **リモートプロセス**
  制御ノードからリモート実行されるユーザプログラムを実行するプロセス．本プロセスのカーネルスレッドは制御ノードの pseudo プロセスカーネルスレッドと連携し，シグナル・システムコールを処理する．
- **iod daemon**
  演算ノード上のリモートプロセスの標準入出力を制御ノード上の pseudo プロセスと接続するための通信路を提供する．
- **リモートプロセス管理カーネルモジュール**
  リモートプロセス用カーネルスレッドの生成，シグナル・システムコールの転送を行う．
- **リモート I/O カーネルモジュール**
  リモートプロセスの発行するファイル I/O システムコールを制御ノードに転送し，結果をリモートプロセスに反映させる．また，メモリマップドファイルのページフォールトに対して制御ノードからデータを受け取って解決する機能を提供する．リモートプロセスの実行ファイルおよびダイナミックリンクライブラリも本機能を使用してロードする．

3.3 リモートプロセス管理機能の動作

3.3.1 起 動 処 理

制御ノードおよび演算ノードは各々local HDD よりOS を起動し，リモートプロセス管理カーネルモジュール，ヘテロ対応バイナリローダ，リモート I/O カーネルモジュールをロードし，master daemon，slave master daemon，slave daemon，iod daemon 等を起動して ready となる．演算ノードが Netboot する場合には対応する制御ノードが nfs root を提供する．

3.3.2 演算ノードの参加

演算ノードが ready となり制御ノードと接続する際のフローを**図 7** に示す．

(1) 演算ノードの起動時において slave master daemon は slave daemon を fork して生成し，以後 slave daemon が特定制御ノードとの通信を行う．

(2) slave daemon はリモートプロセス管理カーネルモジュールから version を取得し，制御ノードとの接続を試みる．

(3) 制御ノードの master daemon と socket 接続が成功すると version を送信し，制御ノードと

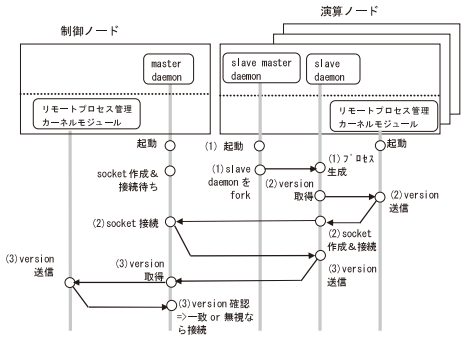

図 7 制御ノードと演算ノードの接続
Fig. 7 Connection flow of management node with processing node.

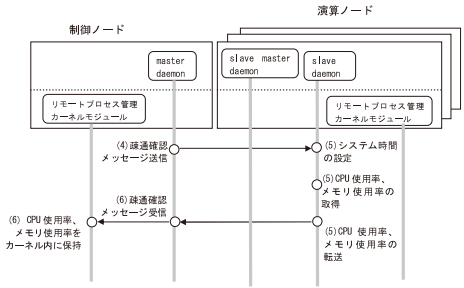

図 8 ハートビートの交換とノード状態の報告
Fig. 8 Exchange node status.

version が一致するとリモート接続可能となる．制御ノードの master daemon と演算ノードの slave daemon は定期的に（現在 15 sec 間隔）ハートビートメッセージを交換する．このとき演算ノードの CPU 利用率とメモリ利用率も制御ノードに渡される（**図 8**）．この情報はリモートプロセスを実行する演算ノードの選択の際に用いる．

(4) 制御ノードの master daemon は演算ノードの slave daemon に定期的に疎通確認メッセージを送信する．

(5) slave daemon は疎通確認メッセージを受け取ると CPU 利用率，メモリ利用率を取得し，疎通確認メッセージの返信として送信する．

(6) master daemon は疎通確認の返信を受け取ると，カーネルに CPU・メモリ使用率を送って保持させる．

3.3.3 リモートプロセスの実行

リモートプロセス実行時の動作を**図 9** に示す．

(1) 制御ノードにおいて shell からプログラムを実行すると fork，exec 処理の中で，起動時に登録したヘテロ対応バイナリローダが呼ばれる．へ

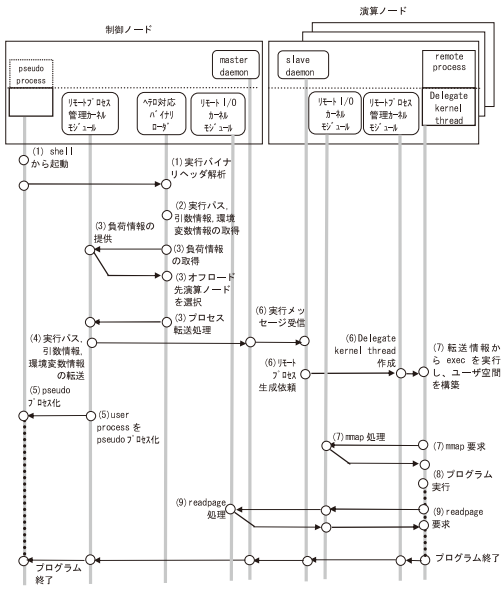

図 9 リモートプロセス実行時の動作
Fig. 9 Remote process execution.

テロ対応バイナリローダは ELF ヘッダを解析し，リモート実行するプログラムかどうかを判定する．リモート実行の必要がなければ通常のバイナリローダが呼ばれ，ローカル実行となる．

(2) ヘテロ対応バイナリローダは実行パス，引数情報，環境変数を取得する．

(3) ヘテロ対応バイナリローダはリモートプロセス管理カーネルモジュールから各演算ノードの負荷情報を取得し，オフロード先演算ノードを決定する．

(4) バイナリローダは選択した演算ノードの slave daemon に対して，master daemon 経由で実行ファイルのパス，引数，環境変数等の情報をメッセージとして渡し，実行を依頼する．

(5) バイナリローダを実行していたユーザプロセスは演算ノードのリモートプロセスと連携する pseudo プロセスとなる．このプロセスはシグナル，システムコールの代行等を行うためにカーネル空間でブロックして待機する．

(6) 実行依頼のメッセージを受けた演算ノードの slave daemon は，リモートプロセス管理カーネルモジュールを呼び出して，リモートプロセス（このときは delegate kernel thread のみ）を生成する．このリモートプロセスは演算ノード上におけるプロセス ID だけでなく，制御ノードのプロセス名空間におけるプロセス ID も持っている．

(7) delegate kernel thread は受け取った実行ファイルのパス，引数，環境変数を利用して exec 処理を行い，リモートプロセスのユーザ空間を生成する．このとき，実行ファイル，ダイナミックリンクライブラリ等のファイルをリモートプロセス空間に mmap しておき，リモートページング機能を利用して解決する．

(8) リモートプロセスの実行環境が整うと，delegate kernel thread はユーザ空間に制御を移してリモートプロセスの実行を始める．

(9) リモートプロセスの実行により発生したページフォールトはリモート readpage 処理によって解決する．

3.3.4 シグナル・システムコールの動作

リモートプロセスが exit した際は delegate kernel thread 経由で，slave daemon，制御ノードの pseudo プロセスに通知され，pseudo プロセスも exit する．

リモートプロセスで発行されたプロセス処理・ファイル I/O 処理のシステムコールは pseudo プロセスに転送して実行する．ファイル I/O システムコールは，演算ノードと制御ノードのリモート I/O カーネルモジュールが連携することでリモート化を実現する．システムコールレベルで転送するため，演算ノードのリモート I/O 機能は制御ノード側の実際のファイルシステムおよびデバイスに依存しない．

また，制御ノード側からの pseudo プロセスに対するシグナルはリモートプロセスに転送して処理する．

3.3.5 リモートプロセスの管理

演算ノードで実行されているリモートプロセスは，制御ノード上の対応する pseudo プロセスを操作することで管理可能である．したがって，利用者および管理者は ps，top 等の標準コマンドを用いてシングルシステムイメージでプロセスを確認し，kill コマンド・シグナル等を用いて一元管理できる．

3.3.6 プロセス配布モード

ヘテロ対応バイナリローダは，リモートプロセス管理カーネルモジュールが保持する各演算ノードの CPU 負荷，メモリ利用率を利用して，プロセスを実行する演算ノードを決定する．プロセス配布モードには次の 2 つのモードがあり，動的に切替え可能である．

(1) 並列プログラム実行モード
MPI 等の並列プログラムの実行を想定しているモードである．1 演算ノードには同時に 1 リモートプロセスのみ実行可能である．

(2) 負荷分散モード
1 演算ノードに複数のリモートプロセスの実行

を許すモードである．次の優先度で実行する演算ノードを選択する．

- CPU 利用率の低い演算ノード
- メモリ利用率の少ない演算ノード
- 実行中のリモートプロセスの少ない演算ノード
- ノード番号の若い演算ノード

4. リモートプロセス管理機能の評価

表 1 の環境を用いてリモートプロセス管理機能の評価を行った．**図 10** に 3 台の演算ノードで並列にコッホ曲線を計算させている例を示す．左上のウィンドウが制御ノードの shell であり 3 つの koch プログラムを起動している．右上のウィンドウは制御ノード上においての top コマンド実行画面であり，演算ノードで実行している 3 つの koch プログラムに対する pseudo プロセスが見えている．右下のウィンドウは，演算ノードに login しての top コマンド実行画面である．1 つの koch プログラム実体がリモート実行されていることが分かる．左下のウィンドウは各演算ノードの計算結果を制御ノードが受け取ってリアルタイムでプロットしている．

4.1 機 能 評 価

設計実装したヘテロジニアスシステム向けリモートプロセス管理機能において実現できた機能を次に示す．

(1) ヘテロジニアスノード間でリモートプロセス実現

ヘテロジニアスなノードで構成している制御ノードと演算ノード間でプロセス管理を一元化した．**図 11** の例では SH-4 ノード上で 10 桁の π 計算を行い，その結果をパイプ機能を用いて IA-32 ノード上で二次処理をしている．図中に示しているようにそれぞれのプログラムは別のプロセッサ用の実行ファイルである．このように，異なるアーキテクチャの実行ファイルをシームレスに扱うことが可能である．

また，並列プログラム実行モードと負荷分散モードの 2 種類のプログラム配布モードを実現できた．この機能により MPI プログラムのような並列プログラムの実行と，パラメータサーチのように多数のシミュレーションプログラムを負荷分散実行する用途に対応可能であることを確認できた．

(2) 演算ノードからプログラム実行機能以外を除外

制御ノードと演算ノードでプロセス構造体等の OS 構造やシステムコールの内容等が近いと性能面や実装上有利なため，今回は演算ノードにも Linux を用いた．演算ノードではリモートプロセスの I/O を実行しないことで OS の軽量化を行うことができた．今後は 1 プロセス実行に特化した，さらに軽量なカーネルも検討する．

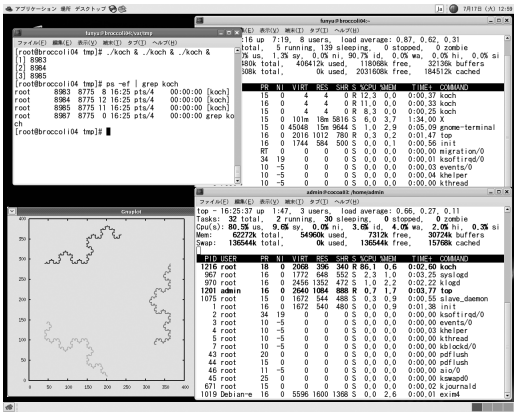

図 10 演算ノードへの計算のオフロード例
Fig. 10 Execution on processing node.

```
[shmz]# file /home/shmz/pi_test /usr/bin/tail
/home/shmz/pi_test: ELF 32-bit LSB executable, Hitachi SH, version
1, for GNU/Linux 2.3.99, dynamically linked (uses shared libs), not
stripped
/usr/bin/tail:      ELF 32-bit LSB executable, Intel 80386, version
1 (SYSV), for GNU/Linux 2.2.5, dynamically linked (uses shared libs),
stripped
[shmz]# /home/shmz/pi_test 10 | tail
3.141592653
```

図 11 ヘテロプログラム間パイプ実行例
Fig. 11 Binding heterogeneous processes with pipe.

(3) 制御ノードと演算ノードの障害アイソレーション

制御ノードと演算ノードは master daemon と slave daemon との通信のみで接続している．したがって，演算ノードがダウンしても，制御ノードの pseudo プロセスが異常終了するだけであり，システムとしては継続実行可能であり，目的を達成できた．また，動的に演算ノードを追加することも可能である．

(4) 制御ノード単位のクラスタ化

今回の実装では複数のヘテロジニアスノードを 1 台の制御ノードに隠蔽することができた．したがって，見かけ上のノード数を 8〜16 分の 1 に削減し，さらにヘテロジニアスシステムを 1 種類のアーキテクチャに見せることが可能なため，10 PetaFlops 級のシステムは，既存の PC クラスタ等で実現できている数千ノードのクラスタ管理技術を用いて達成可能である．しかし，より使いやすいシステムを実現するために，制御ノード間のプロセス名空間統合技術も検討していく．

(5) 汎用 OS と専用 OS の混在

(2) で述べたとおり演算ノードは軽量化した Linux を用い，制御ノードではファイルシステムやデバイスドライバ，ユーザ管理等の機能を提供する汎用 Linux を用いる混在構成を実現できた．また，演算ノードの

表 3 プロセス生成・終了時間
Table 3 Comparison of the time of process creation to exit.

	プロセス生成・終了時間
リモートプロセス (制御ノード→演算ノード)	全体 18.4msec (制御ノード処理 3msec) (演算ノード処理 10.1msec) (演算ノード実行 5.3msec)
制御ノードローカルプロセス	2.1msec
演算ノードローカルプロセス	5.3msec
rsh(制御ノード→演算ノード)	123msec

機能を変更せずに制御ノード側のみの変更で最新のファイルシステムやストレージデバイスに対応可能であることを確認できた．

4.2 性能評価

表 1 の環境において，単純に exit() システムコールのみを発行するプログラムを用いて，プロセスの生成・終了に要する時間を測定した．結果を**表 3** に示す．

評価環境においては，リモートプロセスの生成・終了速度は 18.4 msec であり，演算ノードで直接実行した 5.3 msec に対して 13.1 msec のオーバヘッドがある．この内訳は制御ノード側でのリモートプロセス開始準備・終了処理の 3 msec および，演算ノード側でのリモートプロセス開始準備・プログラム転送・終了処理の 10.1 msec である．今回の評価環境は，制御ノードのプロセッサ速度 1.6 GHz に対して演算ノードは 266 MHz と大きな性能差があり，またネットワークに 100 Mbit Ethernet を使用しているため演算ノード処理時間が大きく見えた．しかし，制御ノードと同程度のスカラ性能を持つ演算ノードおよび Gbit 級のネットワークを用いれば，このオーバヘッドを数分の 1 に削減可能であり，10 msec 程度の速度でリモートプロセスの生成・終了を行えると考える．また，同環境においての rsh によるリモートプロセス実行時間は 123 msec であり，現状においても 6.7 倍程高速化できている．

また，演算ノードのファイルサービス（NFS）に関するデーモンを削減することで，ユーザプロセス実行中のコンテキストスイッチ回数を 60 回/sec から 5 回/sec 程度の 1/10 以下に削減できていることを確認した．このとき，外部から NFS アクセスがある場合の割込み回数は 7,000 回/sec にもなり，通常のタイマ割込みのみの 100 回/sec と比較すると非常に外乱が多い．演算ノードにおいて，ユーザプログラムを実行しているバックグラウンドで前のジョブの計算結果のステージアウト，次のジョブのデータのステージインを行うと上記のような I/O が発生するが，提案方式のように制御ノードに I/O をオフロードしておけば演算ノードには外乱は発生しない．

5. 関連研究

OS 機能そのものをマルチノードに分散する研究としては，Mach [8]，Amoeba [9] があるが専用 OS であり，最新技術に対して独自開発でキャッチアップする必要がある．我々も独自 OS を採用する SR8000 の機能拡張において非常に苦労した．本研究では既存の Linux の実装との互換性を保ちつつ，ヘテロジニアスノードでプロセスを分散実行することを可能にした．

また，Linux の拡張として，プロセス管理を分散対応する Bproc，プロセスマイグレーションを可能にする OpenMosix [10] があるが，いずれもホモジニアス環境用であり，本リモートプロセス管理機能ではヘテロ対応バイナリローダ，リモート I/O 機能を実現することでヘテロジニアス環境に対応している点が異なっている．

また，Compute Node では専用カーネルを実行し I/O ノードでは Linux を実行する IBM 社 BlueGene/L は，I/O ノードがリモートプログラムの管理と I/O の代行を行う点は本研究と同様のアプローチをとっている．しかし，BlueGene/L のリモートプロセスは，I/O ノードにおいて I/O ノードのプロセス名空間とは別の独立したプロセス名空間および専用コマンドで管理されるが，本研究のリモートプロセス管理機能では制御ノードのプロセス名空間でリモートプロセスを管理しており，複数ノードやヘテロジニアスノードを意識させないシングルシステムイメージを提供している点が異なっている．また，現在のところ BlueGene/L は，上記の BProc，OpenMosix と同様にホモジニアス環境のみをサポートしている点も異なっている．

6. おわりに

本稿では，ヘテロジニアスシステムにおいて，シングルシステムイメージでプロセスを管理できるリモートプロセス管理機能のアーキテクチャと実装と評価について述べた．ペタスケールの大規模スーパーコンピュータにおいては，最適なプロセッサで最適な機能を高効率で実行することが有効であり，このとき汎用性を保ちつつヘテロジニアスなノードを一元的に扱うことが重要な課題となる．この課題を解決するために，汎用プロセッサの制御ノードでシステム管理機能やプロセス管理機能，ファイルシステム機能を実現し，専用プロセッサの演算ノードではユーザプログラムの実行に特化するシステム構成を提案した．提案構成の実現に向けて，制御ノードと演算ノード間でプロセスを一元的に管理するヘテロジニアスシステム対応リモートプロセス管

理機能を開発した．本稿ではさらに，実装したリモートプロセス管理機能の評価を行い，目的の機能が実現できていることと，リモートプロセス生成・終了時間においても実用的な性能を達成できることを確認した．本機能はヘテロジニアスノードクラスタだけでなく，ヘテロジニアスマルチプロセッサにおけるマルチOS構成システムにも適用可能である．また，ハイエンドのスーパーコンピュータ用途だけでなく，マイコンを使用した小規模多数の端末を統合管理する用途への適用も可能である．

今後，まずは演算ノードとしてCell Broadband Engine[11]等の強力なプロセッサを用いたノードを用いて，実際の演算性能向上やリモートI/O性能の確認を行う．また，複数の制御ノードをクラスタリングして10 PetaFlops級のシステムを管理する方式の検討を行う予定である．

参 考 文 献

1) TOP500 Supercomputer sites. http://www.top500.org
2) IBM BG/L Team: An Overview of BlueGene/L Supercomputer, *Proc. ACM Supercomputing Conference* (2002).
3) Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture, Intel, SKU #253665 (2007).
4) SH-4 Core Architecture Manual Rev.2.4, Renesas (2006).
5) Jones, T., Dawson, S., Neely, R., Tuel Jr., W.G., Brenner, L., Fier, J., Blackmore, R., Caffrey, P., Maskell, B., Tomlinson, P. and Roberts, M.: Improving the Scalability of Parallel Jobs by adding Parallel Awareness to the Operating System, *SC'03*, Phoenix, Arizona, USA (Nov. 2003).
6) Tamaki, Y., Sukegawa, N., Ito, M., Tanaka, Y., Fukagawa, M., Sumimoto, T. and Ioki, N.: Node Architecture and Performance Evaluation of the Hitachi Super Technical Server SR8000, *Proc. 12th International Conference on Parallel and Distributed Computing Systems*, pp.487–493 (1999).
7) Hendriks, E.A.: BProc: The Beowulf Distributed Process Space, *16th Annual ACM International Conference on Supercomputing*, June 22–26 (2002).
8) Accetta, M., Baron, R., Bolosky, W., Golub, D., Rashid, R., Tevanian, A. and Young, M.: Mach: A New Kernel Foundation For UNIX Development, *USENIX summer'86*, pp.93–112 (1986).
9) Tanenbaum, A., van Renesse, R., van Staveren, H., Sharp, G.J., Mullender, S.J., Jansen, J. and van Rossum, G.: Experiences with the Amoeba Distributed Operating System, *Comm. ACM* (1990).
10) Moshe Bar and MAASK, "OpenMosix". http://openmosix.sourceforge.net/
11) Kahle, J.A., et al.: Introduction to the Cell Multiprocessor, *IBM J. Research and Development*, Vol.49, No.4/5, pp.589–604 (July 2005).

(平成19年 7 月23日受付)
(平成19年11月 8 日採録)

清水 正明（正会員）

1969年生．1993年東京農工大学大学院工学研究科電子情報工学専攻博士前期課程修了．同年（株）日立製作所中央研究所入社．スーパーコンピュータのOSの研究開発に従事．現在，東京大学大学院情報理工学系研究科コンピュータ科学専攻博士課程在学中．IEEE，ACM各会員．

小笠原克久

1979年生．2003年東北大学大学院工学研究科航空宇宙工学専攻博士課程前期修了．同年（株）日立製作所システム開発研究所入社．基本システムソフトウェアの研究開発に従事．

舩生真紀子

1975年生．1997年東京農工大学工学部物質生物工学科卒業．同年（株）日立超LSIシステムズ入社．Linuxクラスタリング技術の研究開発に従事．

米澤 明憲

1947年生．1977年MIT計算機科学科博士課程修了，MIT計算機科学研究所および人工知能研究所において並列・分散計算モデルの研究．「並列オブジェクト」概念のパイオニアとして知られる．帰国後，東京工業大学を経て，1988年から東京大学情報理工学研究教授．現在，東京大学情報基盤センター長．ACM TOPLAS副編集長，日本ソフトウェア会理事長，ドイツ国立情報科学技術研究所（GMD）科学顧問．ACMフェロー，日本ソフトウェア科学会フェロー．