広域分散環境における分散共有メモリの実現とその性能評価

斎藤彰一[†] 國枝義敏[††,☆] 大久保英嗣[†††]

分散共有メモリは，プロセス間通信の手段として，主に小規模分散システムにおいて使用されてきた．また，共有データの格納庫として，計算機クラスタ上での分散並列処理に使用されている．本論文では，これらの適用分野に加えて，インターネットを基本とした広域分散環境においても，データ共有の基盤として分散共有メモリが有効であることを示す．このために，我々は，新しいメモリ一貫性制御方式である *Cluster-based Release Consistency* 方式を，我々が開発を進めている分散並列処理のプラットフォーム *Lemuria* 上に実装した．本方式の特徴は，計算機クラスタを階層化し，それに対応した一貫性制御を行うことによって，広域分散環境においても分散共有メモリを実用可能としている点にある．本論文では，広域分散環境のための *Lemuria* における機能と処理方式について述べる．さらに，インターネット上での性能評価の結果を示す．性能評価の結果，他の分散共有メモリシステムと比較して数倍の速度向上を得た．

An Implementation of Distributed Shared Memory in Wide Area Network and Its Performance Evaluation

SHOICHI SAITO,[†] YOSHITOSHI KUNIEDA[††,☆] and EIJI OKUBO[†††]

A distributed shared memory (DSM) has been used mainly in small-scale distributed systems as a way of inter-process communications. Furthermore, it has been used as a repository for shared data in distributed parallel processing on computer clusters. In addition to these applications, in this paper, it is confirmed that DSM can be effectively used as foundation of data sharing in a wide-area network based on the Internet. For that purpose, we have implemented a new memory consistency protocol called "*Cluster-based Release Consistency*" (CRC) on *Lemuria*: a platform of distributed parallel processing which has been now developing. CRC can be applied to layered computer clusters and can make it possible to practically use DSM in wide-area distributed environments. In this paper, the functions and processing schemes in *Lemuria* for the wide area distributed environments are described. Furthermore, the performance evaluation in the Internet is presented. As a result of evaluation, *Lemuria* has been able to run at higher speed several times as much as other DSM systems.

1. はじめに

分散共有メモリ[1),2)]は，プロセス間通信の手段として，主に小規模分散システムにおいて使用されてきた．また，共有データの格納庫として，計算機クラスタ上での分散並列処理に使用されている．特に，並列処理ではプログラムの高速実行を主目的としているために，分散共有メモリを使用する場合，負荷となる通信回数と通信量を削減する一貫性制御方式が主に研究されてきた[3)~9)]．これらの研究の結果，情報共有のための基盤技術として分散共有メモリが十分に利用可能な段階にまで来ていると考えられる．しかし，インターネットや組織間ネットワークなどの広域分散環境の基盤としては，依然として利用されていないのが現状である．これは，分散共有メモリが高速ネットワーク上でのみ利用可能な技術として，あるいは並列処理の基盤技術として扱われてきたためである．そこで，我々は，分散共有メモリの広域分散環境における情報共有の基盤としての可能性を検証するために，我々が開発している分散共有メモリシステム *Lemuria*[10)] 上にそのための機能を実装し，その評価を行った．

Lemuria は，大規模分散環境と広域分散環境で実際的に利用可能な分散共有メモリを提供することを目的

† 和歌山大学システム工学部
Faculty of Systems Engineering, Wakayama University
†† 和歌山大学システム情報学センター
Center for Information Science, Wakayama University
☆ 現在，和歌山大学システム工学部
Presently with Faculty of Systems Engineering, Wakayama University
††† 立命館大学理工学部
Faculty of Science and Engineering, Ritsumeikan University

とした分散並列処理のためのプラットフォームである．*Lemuria* の特徴は，一貫性制御方式として *Cluster-based Release Consistency*（以下，CRC と記す）を採用し，計算機クラスタの階層化を行っている点にある．CRC は，Release Consistency（以下，RC と記す）を階層化された計算機クラスタに再帰的に適用した一貫性制御方式である．本論文で提案する手法は，我々が論文10) で提案した実装方法を基本に，書き込み時の更新差分の取りまとめ機能と通信の中継の遅延評価機能を導入することによって，通信回数と通信量をさらに削減している．これらの新たな機能と CRC の実現によって，広域分散環境での利用が可能となっている．

Lemuria における計算機クラスタは 3 階層構成となっている．すなわち，下位の階層のクラスタを代表する計算機に Lemuriad と呼ばれるサーバを配置し，クラスタ内の通信を効率化している．さらに，中間の階層のクラスタを代表する計算機に Reflector と呼ばれるサーバを配置し，下位のクラスタ間の通信の効率化を図っている．上位の階層は，すべての Reflector によって構成され，使用可能な帯域が狭いネットワークで接続されたクラスタ間の通信の効率化を図っている．このように，計算機クラスタを階層化することによって，システムを構成する計算機数の増加によって発生する一貫性制御のための通信回数と通信量を削減し，さらに計算機間の通信遅延をも低減している．*Lemuria* における計算機クラスタの階層化による利点としては，以下のものがある．

(1) 通信の高速化が可能である．
複数の計算機がメモリを共有している場合，1 回目の共有メモリアクセス時のメモリの内容をクラスタを代表するサーバ（Lemuriad と Reflector）においてキャッシュしておくことによって，2 回目以降のアクセス要求に対して高速に応答することが可能となる．さらに，同一内容の共有メモリがネットワーク上を何度も通過することを防ぎ，結果としてネットワークの利用効率が向上する．

(2) 通信量を削減できる．さらに，各計算機の受信負荷を軽減できる．
Lemuriad と Reflector は，共有メモリの更新時におけるコピーの更新のためのデータの送受信は行わない．各々の計算機からメモリアクセス要求が発行されるまでは実際にデータを送信しない．したがって，当該計算機に不要なデータが送信されることはない．さらに，受信側の計算機では，受信にともなう処理も必要なくなる．

(3) 特定の通信プロトコルに依存しない．
Lemuria が扱うデータは，単なるバイト列であり，特定の通信プロトコルやデータ構造に依存しない．したがって，個別に開発されている各種アプリケーションのプロキシサーバの共通の通信基盤となりうる．結果的に，これらのシステムの開発が容易となる．

(4) 多階層構成
従来の分散共有メモリシステムでは，複数のノードへの通信を個別に行っていた．しかし，通信先のノード数が多くなると，通信回数と通信量が増加し実行速度を低下させる．また，計算機間の接続に低速ネットワークを使用する場合，低速ネットワークの両端で *Lemuria* が持つ通信の取りまとめ機能やキャッシュ機能を使用することで，低速ネットワークを通過する通信回数と通信量を軽減することが可能となる．これらの問題を解決するためには，2 階層ではおのずと限界がある．したがって，*Lemuria* では多階層構造を採用している．なお，現在の実装では 3 階層であるが，これは容易に 4 階層以上に拡張可能である．

以下，本論文では，2 章で広域分散環境における分散共有メモリに関してメッセージ通信と比較しながら述べる．3 章では，*Lemuria* の全体構成，特にクラスタの構成と階層化を中心に述べる．4 章では，本論文で提案する一貫性制御方式 CRC について述べ，5 章では，それを *Lemuria* の一貫性制御方式として実装する手法について述べる．6 章では，他の分散共有メモリシステムとの性能の比較評価について述べ，広域分散環境おける *Lemuria* と CRC の有効性について議論する．

2. 広域分散環境における分散共有メモリ

本章では，分散共有メモリシステムと Unix における socket などによるメッセージ通信に関して，データの共有方法と通信方法について利害得失を比較しながら述べる．さらに，広域分散環境において分散共有メモリを使用することの利点と，その利用分野についても述べる．

2.1 メッセージ通信と分散共有メモリとの比較

2.1.1 データの共有方法

メッセージ通信によるデータの共有と，分散共有メモリによるデータの共有（特に，*Lemuria* を用いた場合）を比較すると**表 1** のようになる．表 1 から，複製

表1 データ共有に関するメッセージ通信と分散共有メモリ（*Lemuria*）の比較
Table 1 Comparison for data sharing between message passing and distributed shared memory (*Lemuria*).

比較項目	メッセージ通信	分散共有メモリ
利用環境	ほとんどのオペレーティングシステムで標準装備.	別途，専用システムが必要.
プログラミング	socket を用いたプログラミングが必要.	並列処理を意識したプログラミングが必要.
共有（位置）管理	ユーザがプログラム内に記述する.	共有メモリシステム側で行う.
キャッシュ	ない.	*Lemuria* で実現している.

の位置管理が不要な，決まったクライアントに決まったデータを提供するサービスにはメッセージ通信が有利であるといえる．一方，不特定多数のクライアントに様々なデータを提供するサービスには，複製の位置管理をシステム側で行う分散共有メモリが優れているといえる．さらに，広域分散環境においては，*Lemuria* が提供する分散共有メモリのキャッシュ機能によって，メッセージ通信と比較して高速な共有が可能となる．

2.1.2 通信方法

メッセージ通信によるアプリケーションの実装方法は，サーバ/クライアントモデルによるものが一般的である．この場合，クライアント間の情報共有は，サーバによって集中管理される．すなわち，クライアントどうしの通信はサーバを経由することになる．したがって，サーバとクライアント間のネットワークが高速な場合に主に使用される．一方，広域分散環境におけるように，サーバとクライアント間のネットワークが低速である場合には，クライアント間の直接通信による情報共有の方が通信時間において有利であるといえる．この点に関して，分散共有メモリは，クライアント間の直接通信が基本であり，しかも隣接クライアント間の情報共有はメモリを直接使用するためメッセージ通信よりも低コストで実現可能である．

次に，通信の形態について考察する．現在の通信には，一対一の通信（ユニキャスト），一対すべての通信（ブロードキャスト），一対多の通信（マルチキャスト）がある．しかし，多対一の通信はない．これは，現在のメッセージ通信には，複数の計算機から送信されるデータを取りまとめる機能がないことによる．そのため，複数のクライアントにサービスを提供するサーバは，各クライアントごとに個別に通信のための処理を行わなければならない．この代表的な例が WWW のプロキシ（代理）サーバである．また，*Lemuria* 以外の分散共有メモリシステムも通信を取りまとめる機能はなく，個別の通信が必要である．一方，*Lemuria* には，一貫性制御のための通信を1つに取りまとめる機能がある．これによって，複数の計算機からの要求をまとめて受信することが可能となり，受信コストを削減することが可能となっている．さらに，同一要求を1つにまとめることも可能であり，同一要求を複数回処理するといった無駄も削減できる．

2.2 広域分散共有メモリの利用分野

当然のことながら，前節までに述べたように，メッセージ通信と分散共有メモリのそれぞれに利点がある．それにもかかわらず，分散共有メモリが利用される場面は少ない．特に，広域分散環境での利用は皆無である．以下に，広域分散環境において分散共有メモリ（特に，*Lemuria*）を使用することによって，システム開発および利用が促進される分野を示す．

(1) 利用されていない計算機の遠隔利用や広域分散並列処理
(2) 分散協調作業（CAD など）
(3) ネットワークマルチプレイヤゲーム
(4) イントラネットにおける情報共有のための基盤システム

(1)から(3)は，クライアントどうしの情報共有が頻繁に行われ，しかも広域環境における利用が多く，分散共有メモリの利点を活かせる分野である．また，今後多くの利用が見込まれる分野でもある．(4)の分野として，今後の大規模化が予想されるイントラネット内において，様々な情報共有システムが利用されると考えられる．それらの共通の基盤となる情報共有システムとして，複製の位置管理と一貫性の保証が可能であり，特定のプロトコルやデータ構造に依存しない分散共有メモリが有効であると考えられる．

2.3 他の分散共有メモリシステム

Lemuria 以外の分散共有メモリシステムは，LAN（Local Area Network）もしくは高速ネットワーク上での使用を前提としており，広域分散環境をその対象とはしていない．そのために，*Lemuria* が備えている通信のキャッシュや取りまとめ機構を備えていない．一方，*Lemuria* における CRC では，Entry Consistency[5]（以下，EC と記す）や Scope Consistency[9]（以下，SC と記す）におけるような同期変数と共有変数の関連付けによる一貫性制御のコスト削減機能は実現していない．このように，CRC は広域分散環境に

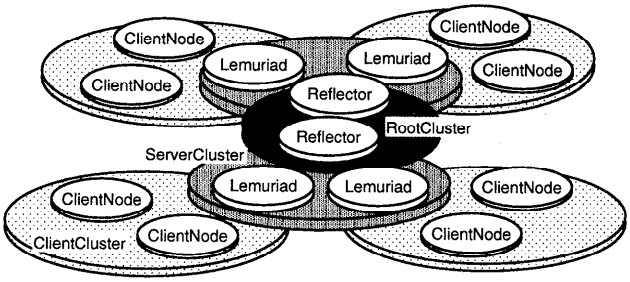


図1 *Lemuria*の全体構成
Fig. 1 Structure of *Lemuria*.

向けたコスト削減を，ECとSCはLANにおけるコスト削減を図っている点で，*Lemuria*と他のシステムが異なっているといえよう．

しかし，CRCの基本的な発想，すなわち多階層化とクラスタごとの一貫性制御は，特定の一貫性制御方式に依存しない．したがって，他のシステムが実現している各種一貫性制御方式（Lazy Release Consistency[8]，EC，SC）にも応用可能である．*Lemuria*の各種機能を利用してこれらを実現することで，広域分散環境における分散共有メモリの有効利用範囲がさらに広がると考えられる．

3. *Lemuria* の構成

本章では，*Lemuria*の構成について述べる．*Lemuria*を構成する各プロセスはノードと呼ばれる．ノードには，ReflectorとLemuriadの2つのサーバと，ユーザプロセスであるクライアントノードとがある．LemuriadとReflectorは，同様の機能を提供するが，システム中の位置が異なることから，排他制御管理，同期管理，通信の中継ポリシの3つの処理方式に違いがある．クライアントノードは，Lemuria Libraryと呼ばれるライブラリをリンクしたユーザプログラムである．

*Lemuria*の全体構成を図1に示す．*Lemuria*では，下位のクラスタをクライアントクラスタ，中間のクラスタをサーバクラスタ，上位のクラスタをルートクラスタと呼んでいる．

クライアントクラスタは，複数のクライアントノードとそれらを管理する1つのLemuriadとで構成される．クライアントクラスタは，高速ネットワークで接続されたノード，たとえば同じLANに接続されているノード群で構成される．サーバクラスタは，複数のLemuriadとそれらを管理する1つのReflectorとで構成される．たとえば，1つの組織内の部門ごとにLemuriadを配置し，それらの全体を管理するために1つのReflectorを配置することが考えられる．サーバクラスタは，クライアントノードの数が多く，それらを複数のクラスタに分割する場合に配置される．ルートクラスタは，複数のReflectorによって構成される．本クラスタは，サーバクラスタ間の接続が低速ネットワークの場合や，サーバクラスタの数が多い場合に配置される．*Lemuria*におけるクラスタの最小構成は，クライアントクラスタとサーバクラスタが各々1つの場合である．Reflectorはシステム全体で1つあればよいため，ルートクラスタが配置されない構成もある．

4. Cluster-based Release Consistency

本章では，*Lemuria*を広域分散環境に対応させるために新たに開発したメモリの一貫性制御方式であるCRCについて述べる．

4.1 階層化されたクラスタにおける一貫性制御方式

CRCは，RCに基づく方式であり，階層化された計算機クラスタに適合するように拡張したものである．従来のRCでは，臨界領域を出た後とバリア同期を行うときに，すべての計算機が一貫性制御の対象となる．一方，CRCでは，クラスタごとに一貫性制御を行い，クラスタ外の計算機がその対象となることはない．以下に，*Lemuria*におけるCRCの処理方式について述べる．

4.1.1 クライアントクラスタにおける一貫性制御

最下層のクライアントクラスタの一貫性制御方式としては，従来のRCに基づく分散共有メモリと同様の方式を実現している．すなわち，あるノードで更新されたメモリの内容は，クラスタ内の他のすべてのノードに転送され，クラスタ内の共有メモリの一貫性が保証される．

4.1.2 サーバクラスタにおける一貫性制御

クライアントクラスタの上位の階層であるサーバクラスタでは，各々のLemuriadがメモリのアクセス要求を発行し，クラスタ内の他のLemuriadと協調して

一貫性制御を行う．ただし，Lemuriadは，実際には分散共有メモリをアクセスしない．これは，Lemuriadからのメモリのアクセス要求は，下位の階層にあるクライアントクラスタ内のノードからの要求の中継であることによる．Lemuriadは，クライアントノードからの要求が，当該ノードが所属するクラスタ内で完了しない場合にのみ，その要求をサーバクラスタに中継する．

サーバクラスタにおいては，下位の階層にあるクライアントクラスタ内の一貫性は保証されているので，クライアントノードからの要求をLemuriadからの要求として扱えばよい．すなわち，Lemuriadはマルチスレッドのクライアントノードと等価であるとして扱う．このようにすることによって，サーバクラスタ内の一貫性制御においても，通常のRCと同様の処理が可能となる．

一方，サーバクラスタからの要求がクライアントクラスタ内のノードに中継される場合も，同様のことがいえる．すなわち，Lemuriadによってクライアントノードに中継するサーバクラスタからの要求は，クライアントクラスタから見るとLemuriadからの要求と見なすことができる．サーバクラスタ内では一貫性が保証されているので，Lemuriadはマルチスレッドのクライアントノードと等価であるとして扱えばよい．

4.1.3 ルートクラスタにおける一貫性制御

階層の最上位層であるルートクラスタにおける一貫性制御は，サーバクラスタの場合と同様の方法で実現することができる．すなわち，サーバクラスタとクライアントクラスタ間の関係を，ルートクラスタとサーバクラスタ間の関係に単に置き換えればよい．ともに，それぞれの階層で一貫性制御が行われており，下位クラスタからの要求を当該中継ノードからの要求として見なすことが可能である．

以上のことから，3階層各々のクラスタ内でRCによって一貫性を保証することによって，システム全体としての一貫性が保証されることになる．このように，CRCは階層化された計算機クラスタにRCを再帰的に適用した方式であるといえる．

4.2 スケーラビリティ

CRCは，前節で述べた処理方式からも分かるように，クラスタごとの一貫性制御によって，システム全体における一貫性を保証している．これは，各ノードの一貫性制御に要する通信コストが，システム全体のノード数ではなくクラスタ内のノード数によって決まることを意味している．すなわち，CRCでは，システム全体のノード数の増加が，各ノードの通信コストの増加に直接的には結び付かないことを意味するものであり，CRCの利点となっている．これは，我々がすでに論文10)で示した結果からも明らかである．以上のことから，CRCは，従来の一貫性制御方式と比較したとき，より多くのノードから構成されるシステムに適応可能であるといえる．

さらに，*Lemuria*では，クラスタ間の一貫性制御において，そのための各種情報の転送を中継ノード（LemuriadとReflector）によって取りまとめることで通信回数と通信量を可能な限り削減し，キャッシュによって通信遅延による実行速度低下を最小限に抑えている．したがって，回線速度が低速なネットワークの場合においても，従来の分散共有メモリに比べて*Lemuria*は，実用的なアプリケーションの領域を広げることが可能になると考えられる．ただし，低速ネットワーク上での利用には，さらなる検討が必要であると考えている．特に，大きな通信遅延によるスケーラビリティの低下の抑制は，今後の検討課題である．

5. *Lemuria*の処理方式

*Lemuria*を構成する各ノードには，分散共有メモリを実現するための各種機能が実装されている（詳細については論文10)を参照）．本章では，論文10)において課題となっていた，書き込み時に生成される更新差分の取りまとめ機構の実現方法と，新たに導入した遅延評価について述べる．

5.1 更新差分の取りまとめ

LemuriadとReflectorによる更新差分の取りまとめは，同一ページ☆の更新差分のマージとして行われる．特にバリア同期の場合には，すべてのノードが更新差分を上位のLemuriadとReflectorに送る．この場合，クライアントノードで生成された更新差分を，LemuriadとReflectorがそのまま中継すると，同一ページの更新差分が複数中継されることになる．これは，通信回数と通信量の増加を招き，さらに，更新差分を複製に反映させるコストも増加させる．

*Lemuria*では，LemuriadとReflectorが同一ページへの更新差分を受信した場合に，これらをマージする．これによって，複数のクライアントノードで生成された同一ページの更新差分が，1つの更新差分に集約される．これは，false sharingによって発生する，サイズは小さいが数が多い更新差分の数を削減することになる．結果として，各ノードにおける更新差分の

☆ 分散共有メモリは，仮想記憶のページの集合体である．一貫性制御は，ページを単位に行われる．

送受信の回数が削減される．さらに，副次的な効果として，クライアントノードにおけるユーザプログラムの実行を干渉する回数が減少し，実効速度を向上させることができる．

5.2 遅延評価

*Lemuria*では，以下の各処理を遅延評価によって行い，更新差分の配送に要する時間を短縮している．

- 更新差分の生成
- 同期変数へのアクセス権の移動
- Lemuriad と Reflector における更新差分の中継
- Lemuriad と Reflector におけるバリア同期要求の中継

以下，これらについて説明する．

5.2.1 更新差分の生成とアクセス権の移動

Release Consistency では，各々の臨界領域の直後に一貫性制御のための処理を行い，他のノードにある複製との間の一貫性を保証しなければならない．しかし，更新されたデータが，その直後に他のノードで利用されるか否かは知ることができない．また，臨界領域の前後で，他のノードから要求がないにもかかわらず，臨界領域を保護する同期変数へのアクセス権の確保と解放を繰り返すことは，通信コストにおいてもクライアントノードの実行速度の観点からも無駄となる．これは，並列処理によく見られる臨界領域を含んだループにおいて，大きな実行速度低下の原因となっている．これらを防止するために，*Lemuria*では，更新差分の生成と同期変数へのアクセス権の解放を，他のノードからの要求を受けた後に行っている．この遅延評価により，クライアントノードの実行速度の低下を防いでいる．

5.2.2 Lemuriad と Reflector による中継

Lemuriad と Reflector は，それぞれ下位クラスタ内の各ノードから送られた更新差分を，受信直後には上位クラスタのノードに中継しない．更新差分の中継は，同期変数へのアクセス権が上位クラスタに移動するときか，または下位クラスタ内でバリア同期が完了するときに，それぞれの処理に先立って行われる．さらに，下位クラスタ内のノードからのバリア同期要求も，受信直後には上位クラスタに中継しない．これらの中継は，当該バリア同期を行う下位クラスタに所属するすべてのノードからの要求の受信後に行われる（**図2** 参照）．

6. 評　　価

本章では，広域分散環境における性能評価について述べる．本評価では，和歌山大学と立命館大学に，それぞれクライアントノードを4台ずつ配置し，2クラスタ（合計8台）の計算機による並列処理を行った．評価アプリケーションには，Splash2[11]を用いた．また，従来の分散共有メモリとの比較には，Utah 大学で開発された Quarks[6]を用いた．

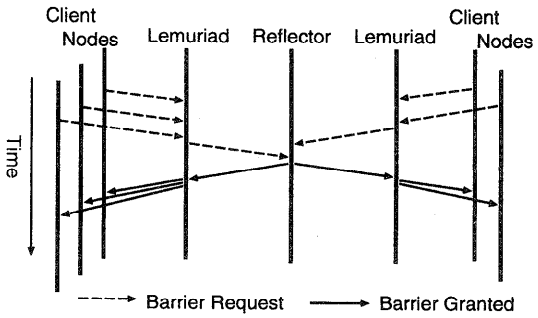


図2 バリア同期における通信
Fig. 2 Communications for barrier synchronization.

6.1 評価環境

評価に使用した計算機は，Solaris 2.5 を搭載した Sun Microsystems 社製のワークステーション8台☆とPC-AT 互換機☆☆3台である．

和歌山大学と立命館大学の接続には，それぞれの大学が日常的に使用している対外線を使用した．それぞれ地域大学間ネットワーク組織である ORIONS[12]と NCA5[13]から，SINET[14]を経由して接続している．ping コマンドを用いて測定したこれらの組織間の RTT（Round Trip Time）は約80 msである．また，ftp による転送速度は約90 KB/sである．**図3** にノードの接続状況を示す．2つのクライアントクラスタと1つのサーバクラスタで構成されている．クライアントノードはすべて Sun ワークステーションを使用し，Lemuriad と Reflector には，PC-AT 互換機を使用した．これらの各ノードを接続するネットワークには，スイッチングハブを用いた100 Mbps のイーサネットを使用している．

6.2 評価アプリケーション

Splash2 のアプリケーションから FFT，LU 分解，OCEAN の3種類を用いて評価を行った．2章でも述べたように，広域分散環境における *Lemuria* の利用は並列処理だけには限らない．しかし，従来の分散共

☆ Sparc Station 5 が3台（Processor: SPARC 70 MHz, 70 MHz, 110 MHz), Sparc Station LX (SPARC 50 MHz), 20 (SuperSPARC 50 MHz×2), Sparc Ultra1 (UltraSPARC 143 MHz), Ultra2 (UltraSPARC 200 MHz), Enterprise 3000 (UltraSPARC 168 MHz×4) が各1台．

☆☆ Processor: Pentium II 266 MHz, OS: Solaris 2.6.

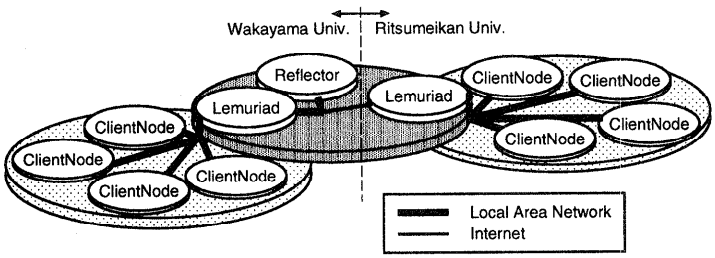


図3 評価環境
Fig. 3 Network for performance evaluation.

有メモリシステムと性能を比較するために，実行時間が明確であり，かつ多くの分散共有メモリシステムで動作実績のある Splash2 を評価アプリケーションとして選択した．以下に，評価に使用した3種類のアプリケーションの諸元を示す．

(1) **FFT**
評価に用いたデータ数は，2^{14} 個，2^{16} 個，2^{18} 個の3種類である．

(2) **LU 分解**
評価に用いた行列の大きさは，256×256，512×512，1024×1024 の3種類である．行列を分割するブロックの大きさは，すべての場合で 16×16 である．なお，行列の分散共有メモリへの割当てには，Non contiguous block allocation バージョン☆を使用した．

(3) **OCEAN**
評価に用いた海洋シミュレーションの海の大きさは，130×130 グリッドと 258×258 グリッドの2種類である．グリッド間の距離は 20,000 m，計算の許容誤差は 1^{-5} である．なお，海の各グリッドの分散共有メモリへの割当てには，Contiguous partition allocation バージョン☆☆を使用した．

6.3 実行結果

Lemuria と Quarks の，ノード数4台（両大学とも2台ずつ）と8台（両大学とも4台ずつ）による評価結果として，FFT を**表2**に，LU 分解を**表3**に，OCEAN を**表4**にそれぞれ示す．さらに，各アプリケーションの通信回数を**図4**に，通信量を**図5**に，実行時間に占める同期の時間（Lock と Barrier．ただし，これらの同期時間には，更新差分の配布と反映の時間が含まれる）の割合を**図6**にそれぞれ示す．以下，アプリケーションごとに *Lemuria* と Quarks の実行結果について比較する．

表2 FFT の実行結果（単位：秒）
Table 2 Execution time of FFT (sec).

Size	2^{14}		2^{16}		2^{18}	
Nodes	4	8	4	8	4	8
Lemuria	62	79	151	143	794	1,144
Quarks	113	157	459	439	966	1,571
Speed up	1.82	1.98	3.03	3.07	1.21	1.37

表3 LU 分解の実行結果（単位：秒）
Table 3 Execution time of LU factorization (sec).

Size	256 × 256		512 × 512		1024 × 1024	
Nodes	4	8	4	8	4	8
Lemuria	19	26	92	141	1,199	1,823
Quarks	124	392	562	1,012	5,486	9,079
Speed up	6.52	15.07	6.10	7.17	4.57	4.98

表4 OCEAN の実行結果（単位：秒）
Table 4 Execution time of OCEAN (sec).

Size	130 × 130		258 × 258	
Nodes	4	8	4	8
Lemuria	931	1,559	4,259	3,738
Quarks	1,208	2,008	3,957	4,441
Speed up	1.29	1.29	0.93	1.19

FFT では，*Lemuria* は，Quarks の約 1.2 倍から約3倍の実行速度となった．これは，*Lemuria* における共有メモリのキャッシュと中継時の各種情報の取りまとめ機能によって，通信回数と通信量が削減されたためである．図4と図5から，*Lemuria* の通信回数と通信量は4台の結果と8台の結果がほぼ同じである．これは，通信回数と通信量の変化が，台数に無関係であることを示している．すなわち，両大学間の通信を中継している Lemuriad の取りまとめとキャッシュによって，同一ページがインターネット上を複数回移動することがなくなったためである．一方，Quarks は，4台から8台へ台数が増加したことによって，通信回数と通信量がともに約2倍になっている．これは，インターネットを経由して通信するノードが2倍になった

☆ プログラミングは容易であるが，メモリ参照の局所性が少ないことが特徴である．
☆☆ メモリ参照の局所性にすぐれたデータ構造が特徴である．

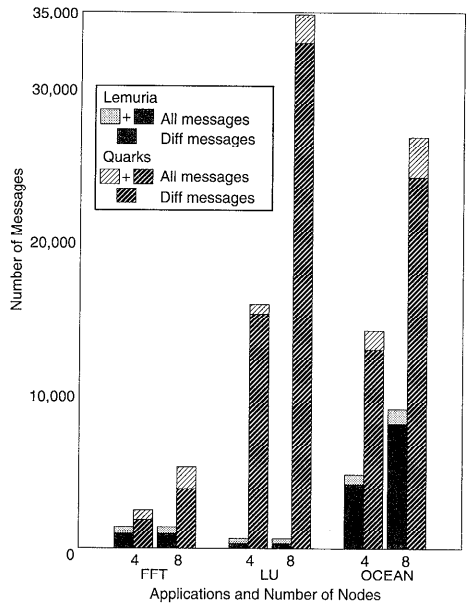


図4 クライアントクラスタ間の通信回数
Fig. 4 Number of communications between client-clusters.

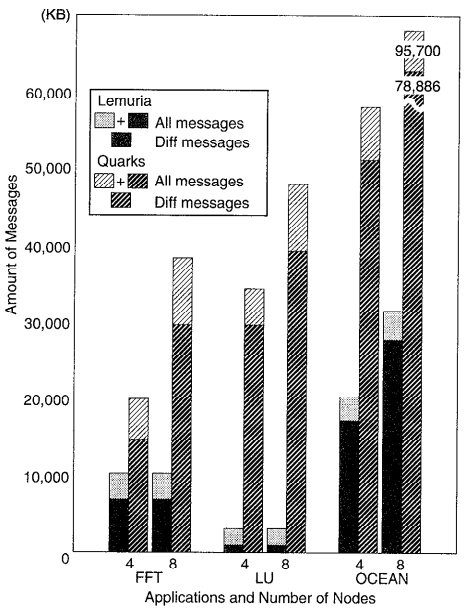


図5 クライアントクラスタ間の通信量
Fig. 5 Amount of communications between client-clusters.

ためである．すなわち，4台の場合で，インターネットを経由して通信するノードは2台であるが，8台の場合には，インターネット経由によるノードは4台と倍になり，通信相手の倍増とともに通信回数と通信量が比例して増加している．また，*Lemuria* と Quarks の

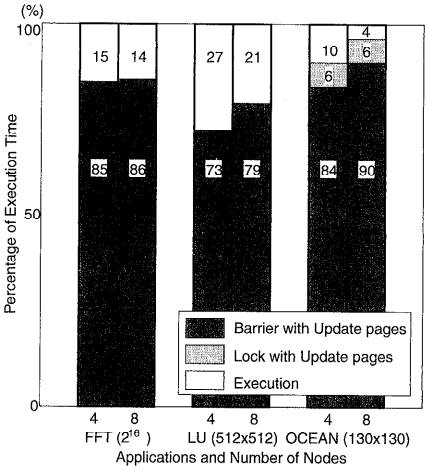


図6 実行時間中の各処理時間の割合
Fig. 6 Percentage of execution time.

通信回数と通信量を比較した場合においても，Quarks は4台の場合で *Lemuria* の約2倍，8台の場合で4倍になっている．

次に，LU 分解では，*Lemuria* は，Quarks の約 4.5 倍から 15 倍以上の速度向上を得た．これは，FFT の場合と同様に，*Lemuria* による通信回数と通信量の抑制の効果である．また，図4と図5から，通信回数と通信量の変化では，LU 分解の場合も FFT の結果とほぼ同様の傾向を示している．しかし，LU 分解では，更新差分の配送のための通信回数の削減効果が，通信回数と通信量の双方において *Lemuria* と Quarks の差となって現れている．今回，Non contiguous block allocation バージョンを使用したためにアクセスの局所性が少なく，多数の false sharing が発生した．*Lemuria* と Quarks は，ともに更新差分の生成と配布をページごとに行っている．Quarks は，各クライアントノードで生成された更新差分が，それぞれ個別に複製を持つノードに直接送られる．すなわち，同一ページの更新でも，それが複数のノードで更新される場合には，複数回の通信が必要となる．これによって発生した通信が，インターネットを経由することで実行速度を低下させた原因となっている．一方，*Lemuria* は，Lemuriad と Reflector の更新差分をマージする機能によって，複数のクライアントノードで生成された同一ページの更新差分を1つにまとめている．これによって，インターネットを経由する通信回数を削減し，実行速度を向上させている．

OCEAN では，*Lemuria* と Quarks の実行速度がほぼ同じになっている．図6からも分かるように，OCEAN では，実行時間中の同期に要する割合が他

のアプリケーションよりも多い．また，図4と図5から，通信回数と通信量の双方でDiff（更新差分）の占める割合が，他のアプリケーションと比較して多いことからも分かる．さらに，実行中の計算機の観測から，同期時にLemuriadを動作させている計算機がスラッシングを発生させていることが分かっている．これらから，同期時に，Lemuriadの負荷が更新差分を中継するために上昇し，これがボトルネックとなっていることが分かる．これは，Lemuriadが受信する回数を削減することで解消することが判明しており，現在修正を行っている．

以上をまとめると，*Lemuria*の実行速度は，LU分解とFFTにおいてQuarksの実行速度よりも高速であることが分かった．特にLU分解では，10倍以上の高速化が達成されていることが示された．OCEANにおいては，スラッシングの発生が認められたが同程度の実行速度となった．これらによって，広域分散環境におけるデータ共有のために今回*Lemuria*に実装した各種機能とCRCが有効に働くことが分かった．今後の課題としては，同期を頻繁に使用する場合の対処法があげられる．

7. おわりに

本論文では，広域分散環境における分散共有メモリの実現について述べた．さらに，新しいメモリ一貫性制御方式であるCRCについて述べ，*Lemuria*の構成と各種機能の処理方式について述べた．Splash2のアプリケーションを用いた広域分散環境での性能評価から，*Lemuria*とCRCが広域分散環境において有効であることが示された．また，それらが従来の分散共有メモリシステムを上回る性能を得ることも分かった．

今後は，CRCと*Lemuria*の性能向上を行うとともに，本論文で述べた分野も含めた様々な分野において分散共有メモリを適用する方法を検討していく予定である．

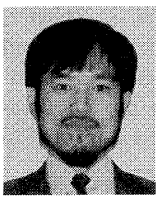
参考文献

1) Tam, M.C., Smith, J.M. and Farber, D.J.: A Taxonomy-Based Comparison of Several Distributed Shared Memory Systems, *Operating Systems Review*, Vol.24, No.3, pp.40–67 (1990).
2) Nitzberg, B. and Lo, V.: Distributed Shared Memory: A Survey of Issues and Algorithms, *IEEE Computer*, Vol.24, No.8, pp.5–60 (1990).
3) Li, K.: IVY: A Shared Virtual Memory System for Parallel Computing, *Proc. 1988 International Conference on Parallel Processing*, Vol.2, pp.94–101 (1988).
4) Carter, J.B., Bennett, J.K. and Zwaenepoel, W.: Implementation and Performance of Munin, *Proc. 13th ACM Symposium on Operating System Principles*, pp.152–164 (1991).
5) Bershad, B.N., Zekauskas, M.J. and Sawdon, W.A.: The Midway Distributed Shared Memory System, *Proc. COMPCON '93*, pp.528–537 (1993).
6) Khandekar, D.: Quarks: Portable DSM on Unix, Technical Report, Department of Computer Science Utah University, UT, USA (1992).
7) Amza, C., Cox, A.L., Dwarkadas, S., Keleher, P., Lu, H., Rajamony, R., Yu, W. and Zwaencpoel, W.: TreadMarks: Shared Memory Computing on Networks of Workstations, *IEEE Computer*, Vol.29, No.2, pp.18–28 (1996).
8) Keleher, P., Cox, A.L. and Zwaenepoel, W.: Lazy Release Consistency for Software Distributed Shared Memory, *Proc. 19th Symposium on Computer Architecture*, pp.13–21 (1992).
9) Iftode, L., Singh, J.P. and Li, K.: Scope Consistency: A Bridge between Release Consistency and Entry Consistency, *Proc. 8th ACM Symposium on Parallel Algorithms and Architectures*, pp.277–287 (1996).
10) 斎藤彰一，國枝義敏，大久保英嗣：分散並列処理のためのプラットホームLemuriaにおける分散共有メモリの性能評価，電子情報通信学会論文誌，Vol.J82-D-I, No.3, pp.457–466 (1999).
11) Woo, S.C., Ohara, M., Torrie, E., Singh, J.P. and Gupta, A.: The SPLASH-2 Programs: Characterization and Methodological Considerations, *Proc. 22nd International Symposium on Computer Architecture*, pp.24–36 (1995).
12) 大阪地域大学間ネットワーク：`http://www.orions.ad.jp/`.
13) 第5地区ネットワークコミュニティ：`http://www.nca5.ad.jp/`.
14) 学術情報ネットワーク：`http://www.sinet.ad.jp/`.

(平成10年12月1日受付)
(平成11年 4 月1日採録)

斎藤 彰一（正会員）

1969年生．1993年立命館大学理工学部情報工学科卒業．1995年同大学大学院博士前期課程修了．1998年同大学大学院博士後期課程単位取得満期退学．同年和歌山大学システム工学部情報通信システム学科助手，現在に至る．オペレーティングシステム，分散並列処理，インターネット等の研究に従事．日本ソフトウェア科学会，ACM，IEEE-CS各会員．



國枝 義敏（正会員）

1957年生．1980年京都大学工学部情報工学科卒業．1982年同大学大学院工学研究科修士課程修了．同年同大学工学部情報工学科助手．1991年同助教授．1996年和歌山大学システム工学部情報通信システム学科教授．1998年同大学システム情報学センター教授．1999年同大学システム工学部情報通信システム学科教授，現在に至る．並列処理，分散処理，特にその関連の基本ソフトウェアの研究に従事．京都大学工学博士．電子情報通信学会，ACM，IEEE-CS各会員．1992年情報処理学会論文賞受賞．

大久保英嗣（正会員）

1951年生．1974年北海道大学理学部数学科卒業．1977年同大学工学部情報工学科大学院修士課程修了．同年（株）日立製作所ソフトウェア工場に入所．主としてFORTRANコンパイラの開発に従事．1979年より京都大学工学部情報工学科助手．1985年同講師，1987年同助教授，1991年立命館大学理工学部情報学科教授，現在に至る．工学博士．オペレーティングシステム，データベースシステム，分散システム，実時間システム等の研究に従事．著書に「オペレーティングシステムの基礎」（単著，サイエンス社），「情報工学実験」，「情報処理入門」（いずれも共著，オーム社）がある．電子情報通信学会，日本ソフトウェア科学会，システム制御情報学会，ACM，IEEE-CS各会員．