システムソフトウェアと
オペレーティング・システム 72－6
(1996. 5. 15)

分散並列処理のためのプラットフォーム *Lemuria* の構成

斎藤 彰一† 大久保 英嗣††

†立命館大学大学院理工学研究科
††立命館大学理工学部情報学科

滋賀県草津市野路町 1916 Tel:0775-66-1111(Ext.8863)
†saito@sol.cs.ritsumei.ac.jp ††okubo@cs.ritsumei.ac.jp

あらまし 我々は, ワークステーションクラスタを使用した分散並列処理のためのプラットフォーム *Lemuria* を開発している. *Lemuria* は, 分散共有メモリを実現するための機能を中心としたライブラリと, デーモンとリフレクタと呼ばれるネットワーク管理のための 2 種類のサーバをユーザに提供している. デーモンは, ネットワークを構成する各マシンに配置され, クラスタ内の共有メモリの一貫性制御を行う. リフレクタは, 各クラスタに配置され, クラスタ間の共有メモリの一貫性制御を行う. *Lemuria* では, これらのサーバによって, スケーラブルな分散共有メモリの実現を可能としている. また, サーバ間の通信の並列化, プロセスの複製による負荷分散, 分散共有メモリを使用した入出力の仮想化などの機能も実装中である. 本論文では, 以上の特徴を持つ *Lemuria* のソフトウェアアーキテクチャについて述べる.

キーワード 分散並列処理, 分散共有メモリ, ワークステーションクラスタ, メモリコンシステンシ

Lemuria: A Platform for Distributed Parallel Processing

Shoichi Saito† Eiji Okubo††

†Graduate School of Science and Engineering, Ritsumeikan University
††Department of Computer Science,
Faculty of Science and Engineering, Ritsumeikan University

1916 Noji, Kusatsu, Shiga 525, Japan Tel:+81-775-66-1111(Ext.8863)
†saito@sol.cs.ritsumei.ac.jp ††okubo@cs.ritsumei.ac.jp

Abstract We have been developing a platform called *Lemuria* for distributed/parallel processing by using workstation clusters. *Lemuria* provides users with a library which is mainly concerned with realizing the distributed shared memory, and also provides with two types of servers for the network management: daemon and reflector. *Lemuria* daemon is placed on each machine and controls the intra-cluster memory consistency. *Lemuria* reflector is placed on each cluster and controls the inter-cluster memory consistency. By these severs, *Lemuria* can make it easy to implement the scalable distributed shared memory. Furthermore functions for parallel communications between servers, load distributions by the process replication, and the virtual I/O with the distributed shared memory are now being implemented. In this paper, the software architecture of *Lemuria* is described.

key words distributed/parallel processing, distributed shared memory, workstation cluster, memory consistency

1 はじめに

我々は, 分散環境上で並列処理を可能にするためのプラットフォームである *Lemuria* を開発している. *Lemuria* は, 我々が Mach オペレーティングシステム上で実現した分散共有メモリサーバ [1][2] を基に構築している. 分散共有メモリ [3] は, メモリオブジェクト (プロセスの仮想空間やファイルなど) を複数のマシンで共有するための機構であり, そのためにメモリオブジェクトの複製を作成して, それらへの読み出しと書き込みの一貫性制御を行わなければならない. *Lemuria* は, UNIX 上で動作するより汎用的なシステムを目標としており, 分散共有メモリ機能を中心に, プロセスの複製による負荷分散, 共有メモリを使用した仮想入出力の機能などを提供する.

Lemuria は, ワークステーションクラスタを使用した分散環境を想定しており, 分散並列処理機能の実現とともに, データ量及びマシン数の増加に対応したスケーラビリティの評価, ATM などの高速ネットワークにおける分散共有メモリの利用可能性の評価も目的としている. 以下, これらについて述べる.

(1) スケーラブルな分散共有メモリ

分散共有メモリを利用する場合, データ量及びマシン数の増加に伴い, 一貫性制御やメモリイメージの転送に要するコストが増加し, スケラービリティの実現が困難になる. これを解決するために, *Lemuria* では, ネットワークを介して接続されたワークステーション群を複数のクラスタに分割し, クラスタを単位として管理することによって, ネットワーク通信及び管理のためのコストを低減させ, スケーラビリティを実現する.

(2) 高速ネットワークにおける分散共有メモリの利用

ATM を始めとした高速ネットワーク環境においては, 従来から指摘されている分散共有メモリを利用することによるネットワーク通信のコストや入出力のコストを低減することが可能となることが予想される. 例えば, 共有メモリの使用によって発生するページングのための 2 次記憶へのアクセスを, ネットワークを介した他のマシンの主記憶アクセスに置き換えることによって, ページング自体を高速することができる. また, 高速ネットワークを利用することにより, オーバヘッドの少ない一貫性制御プロトコルを実現することも可能となる.

(3) プロセスの複製による負荷分散

複数のプロセスが同一のメモリオブジェクトをアクセスした場合, それらのプロセス間でメモリオブジェクトの移動が頻繁に発生し, いわゆるネットワークスラッシング状態となる. 特に, スラッシングに関与するプロセスが異なるネットワークセグメントに存在する場合は, ネットワークの通信コストがさらに増大することになる. *Lemuria* では, 特定のマシンあるいはセグメントにプロセスの複製を生成することによって, このようなネットワークスラッシングを回避する機能を実現する.

以下, 本論文では, 2章で *Lemuria* の全体構成について述べ, 3章で *Lemuria* を構成するユーザライブラリとサーバの機能について述べる. さらに, 4章でクラスタを導入した場合の通信コスト, 5章で *Lemuria* の基本機能である分散共有メモリの処理方式について述べる.

2 全体構成

Lemuria は分散共有メモリに基づいた分散並列処理のためのプラットフォームである. 分散共有メモリを用いて, メモリオブジェクトをプロセス間で共有し, 分散環境での並列処理および負荷分散を実現する. *Lemuria* の構成は, 分散共有メモリを実現する機能の上にプロセスの複製を生成する機能や, クラスタの管理機能を追加したものとなっている. 本章では分散共有メモリ機構に基づく *Lemuria* の全体構成について述べる.

2.1 階層構造

Lemuria は, マシン, クラスタ, クラスタ間の 3 つの階層によって分散共有メモリの制御を行う. 全体の概要を図 1に示す. マシン内の一貫性制御は, *Lemuria Daemon*(以下 Lemuriad) によって行われる. クラスタ内の一貫性制御は, 各マシンの Lemuriad の協調によって行われる. 複数のクラスタに属するメモリオブジェクトの一貫性制御は, クラスタ内のでの一貫性制御を行った上で, *Lemuria Reflector* (以下 Reflector) によってクラスタ間の制御を行う.

2.2 Lemuria Cluster

Lemuria では, *Lemuria* が動作するネットワークの物理セグメントを *Lemuria Cluster*(以下 Cluster) と呼んでいる. Cluster を構成するには, 2 つの目的がある. 第 1 に, ネットワークを構成するマシンを複数のセグメントに分割配置することで, 通信の並列性を高めることである. 第 2 に, ブロードキャストとマルチキャストをルータに

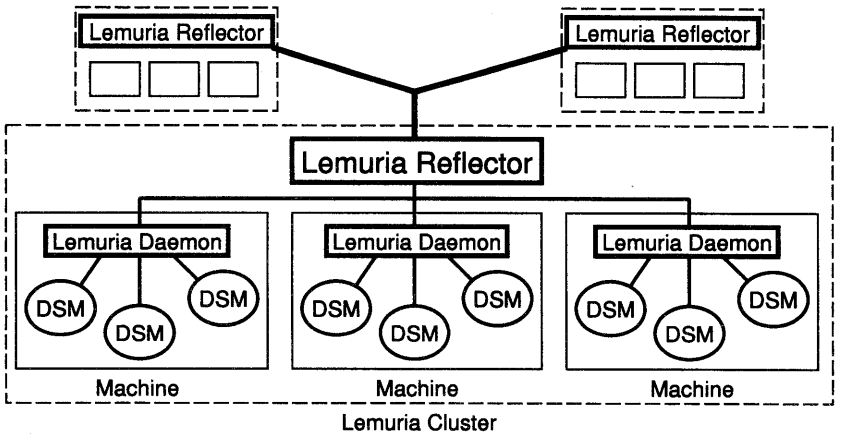

図 1 *Lemuria* の全体構成

依存せずに全体のマシンに届かせるためである. 本節ではそれぞれについて述べる.

2.2.1 並列配送

現在多くの LAN は, Ethernet によって構成されている. Ethernet はバス型接続の形態をとるので, 1 本の Ethernet(Cluster) をそれに接続されたすべてのマシンで共有することになる. 当然ながら, 接続するマシンの数が増えれば, 全体の通信量が増えるので, バス型接続の通信にはより多くの時間が必要となる. *Lemuria* では, ネットワークを構成するマシンを複数セグメントに分散配置することで, 1 つの Cluster を共有するマシンの数を小さくしている. また, 複数の Cluster を使用することで, ページの配送をそれぞれの Cluser 内で並列に行うことが可能になる. これらによって, 通信時間の短縮が可能となる.

しかし, 異なる Cluster 間の通信では, 中継するシステムが必要となる. この場合には, 中継を行うためのコストが余分に必要となる. *Lemuria* では, 複数の Cluster の接続に Reflector を用いる. Reflector では, Cluster 間に発生する通信の監視を行い, 通信内容によって中継するか否かの判断を行う. また, 複数の Cluster が接続されている場合には, 中継すべき Cluster の選択を行う. このように中継の制御を行うことで, Cluster の内外に送出する通信の削減を行う. なお, Reflector については 3.3節で, 中継コストについては 4章で述べる.

2.2.2 ブロードキャスト

Lemuriad は, Cluster 内の他の Lemuriad との通信にブロードキャストとマルチキャストを使用する. これらの通信は, 一般にルータを通過することができない. 従って, 通過させるためにカスタマイズされた OS が必要となる. しかし, *Lemuria* では, Cluster 間の通信において Reflector で通信内容の解析を行うので, ルータで物理セグメントを通過する必要はない. このことからも, 物理セグメントを単位として Cluster を構成している.

3 システム構成

Lemuria は 2 つのサーバとユーザアプリケーションにリンクするライブラリ (以下 libLemuria) によって実現される. サーバには, アプリケーションが動作するマシン上で一貫性制御やメモリオブジェクト管理を行う Lemuriad と, Cluster 間の制御を行う Reflector がある. 以下, これらのライブラリとサーバが実現する機能について述べる.

3.1 Lemuria Library

Lemuria を使用するユーザアプリケーションにリンクするライブラリである. 現在は C 言語用を実装している.

libLemuria では, UNIX のシグナルを用いて分散共有メモリへのアクセス (ページフォルト) の検出を行い, mmap() システムコールでプロセス空間への分散共有メ

モリの割り当て, mprotect() システムコールで分散共有メモリへのアクセス許可の制御を行う.

libLemuria は, ページフォルトによりページの移動が必要になった場合には, Lemuriad に当該ページを取得するように要求を行う. 逆に, Lemuriad からのページの解放要求に基づいて, ページを Lemuriad に移動させる. なお, libLemuria では, ページのアクセス制御を行うのみで一貫性制御は行わない. 一貫性制御を必要とするメモリオブジェクトの位置関係が同一マシン内またはマシン間となる場合を問わずに, 一貫性制御は Lemuriad で行われる. これは, 同一マシン内で共有可能な情報を共有することで, マシン内の一貫性制御コストを軽減するためである. さらに, 個々のプロセスが直接他のプロセスと一貫性制御を行わないことで, 同一マシンからの要求を Lemuriad で制御し, Cluster 内での通信量を軽減するためである.

3.2 Lemuria Daemon

Lemuriad は, 各マシン上に 1 つずつ動作する *Lemuria* システムサーバである. Lemuriad は, マシン内のメモリオブジェクトの割り当てと解放に関する管理と, Cluster 内の他の Lemuriad との間で分散共有メモリの一貫性制御を行う. なお, 一貫性制御については 5章で詳しく述べる. また, *Lemuria* のアプリケーションをリモートマシン上に生成する機能を持ち, それら *Lemuria* プロセスの管理を行う.

Lemuriad は, 要求を受信した時にメッセージのキューイングを行う. キューイングにより, 複数の要求を受信した場合には, Lemuriad は要求内容の先読みを行う. 先読みによって, 複数のマシンから同一ページへの要求を受信している場合には, ページの配送をマルチキャストによって行う. また, 隣接するページへの要求を受信している場合には要求を取りまとめて, 1 回の通信でページの配送を行う. さらに, libLemuria からの要求についても, 同様に要求のキューイングと取りまとめを行う.

3.3 Lemuria Reflector

Reflector は, 1 つの Cluster に最低 1 つ動作する *Lemuria* システムサーバである. Reflector の目的は, Cluster 間の通信の中継と調整であり, 主な役割を以下に示す.

- 通信内容のキャッシュ
- Cluster 間通信の中継および内容解析
- Cluster 内のメモリオブジェクトの割り当て情報の収集と, 他の Cluster への提供
- 内容解析の結果に基づく通信の取りまとめ
- 内容解析の結果に基づくメモリオブジェクトおよびプロセスの Cluster 間移動

一貫性制御の結果, Cluster を越えてページが配送されるときの通信も Reflector を通過する. Reflector は, 当該ページをキャッシュすることで, これ以後に発生する当該ページへの配送要求に応じる. これにより, Cluster 間のページ配送量を削減する. なお, 当然ながら, キャッシュされたページも一貫性制御の対象となる.

内容解析は, 通信の中継時に行われる. 解析の対象となる情報は, 1) 要求種別, 2) メモリオブジェクトの ID, 3) ページ番号とページ数, 4) ページの最新内容を所有するマシン (または Cluster) である. 1) の要求種別には, 大別してページの転送要求及び同期要求と, メモリオブジェクトの割り当てに関する要求がある. 割り当てに関する要求は, 当該情報を Reflector 内で利用するために記録される. その他の要求と 2)〜4) の通信は, 要求の取りまとめに使用する. 要求の取りまとめとは, 2)〜4) の情報に基づいて通信にマルチキャストを用いたり, 複数要求を 1 つの要求に集約することである. また, 先だって要求されたページへの, 同一の要求を行わないことをいう. これにより, 通信量の削減を行う.

また, Reflector では中継を行った時刻についても記録する. 同一ページが複数の Cluster 間で共有され, 同時に書き込みが発生した場合などに, ページの激しい移動が発生する場合がある. これをネットワークスラッシングという. Reflector は, 中継時刻を記録することで, ネットワークスラッシングの発生の監視を行う. 発生した場合には, スラッシングの発生源となっているメモリオブジェクトやプロセスを, 特定の Cluster (またはマシン) に移動させる. これによって Cluster 間の通信量を抑制する.

4 Cluster 間通信

Lemuria では, Reflector を介して Cluster 間通信を行う. この場合には, 当然ながら, 直接通信を行う場合と比べて中継に伴うコストが余分にかかる. しかし, 分散共有メモリでは, 1 対 1 の通信ではなく同時に多数のマシンが通信を行う. このような場合には, Cluster にマシンを分散配置して, 通信を並列に行う方がより高速な通信が可能となる.

以下の4種類の通信において, 単一 Cluster 構成の場合と複数 Cluster 構成の場合に, それぞれに必要なページの配送時間について述べる.

(1) 1対1の通信

(2) マルチキャスト

(3) 1対Nの個別通信

(4) N組の1対1の通信

4.1 1対1の通信

それぞれのマシンが別々の Cluster に属していた場合には, Reflector による中継コストが, そのまま配送時間に付加される. 通信は, 発信元のマシンから2つの Reflector を経由する形で, 要求元のマシンに到達する. 従って, 計3回の通信が必要となる. そのため, Cluster 分割による利点はない.

4.2 マルチキャスト

あるマシンが, 同一 Cluster 内の N台のマシンにページの配送を行う場合には, マルチキャストを使用することで1回分の通信時間で終了する. 複数 Cluster に分割された場合には, 1対1の通信の場合と同様に, 中継コストが通信時間に付加されるために, 全体として3回の通信による配送時間が必要となる.

4.3 1対Nの個別通信

1個の Cluster 構成で, あるマシン (以下マシン α とする) が, N台のマシンからの要求に別々に応じる場合には, N回のページ配送時間が必要となる.

マシンを N/M台ずつ, M個の Cluster に分割した場合を考える. マシン α から異なる Cluster のマシンに1ページを配送するためには, 2つの Reflector を経由するために, 合計3回の通信が必要となる. したがって, 通信回数の合計は $N+2\cdot N/M$回となる. しかし, Reflector 間の通信と, Reflector と配送先マシンの間の通信は並列に行うことが可能である. したがって, 全体の通信時間は, マシン α から Reflector 間の通信の $N-N/M$回, Reflector 間の通信の $N-N/M$回, 最後のページが Cluster 内を通過する1回分を合計した $2\cdot(N-N/M)+1$ 回分の配送時間となる.

4.4 N組の1対1の通信

1対1の通信については先に述べたが, 本節ではこれを同時にN組がページ配送を行った場合について述べる.

1組のマシンで, 一方のマシンから他方のマシンに通信を行う場合を考える. 1個の Cluster では, N回の通信が必要となる. N/M台ずつ M個の Cluster に分割した場合を考える. 組をなすマシンが共に同一の Cluster に属している場合には, Cluster 間通信は発生しない. 従って, Cluster 内での通信が N/M 回となる. これは, Cluster 分割を行った場合の通信において, 最良の場合である. 次に, すべての組をなすマシンが異なる Cluster に属した場合の通信回数について考える. すべてのマシンが送信と受信を行うために, すべての Cluster で $2\cdot N/M$回の通信が発生する. さらに, すべての通信が Reflector 間を通過するので, この通信が合計 N回発生する. したがって, 通信回数の合計は $2\cdot N/M+N$である. しかし, 各 Cluster 内での通信は並列に行われるので, 全体の通信時間は Reflector 間に発生する N回の通信時間に, 最初と最後の通信がそれぞれの Cluster を通過する2回の通信時間を加えたものとなるので, 合計で $N+2$ 回分の通信時間となる.

4.5 並列配送のコスト

複数の Cluster にマシンを分散配置した場合には, Reflector による中継コストが必要になるので, 1回のページ配送を考えた場合には大きな負荷となる. しかし, 分散共有メモリの使用で, 特に大きなサイズのデータを扱う場合などには, 全体としての通信の配送コストを考えた方がよい. マシンの複数 Cluster への分散配置で通信を並列に行うことが可能になり, 結果的には, 1回のページ配送に見られるような大きな負荷にはならないと言える. また, Cluster 内で通信が完結するように, プロセスやメモリオブジェクトを配置あるいは移動することで, より一層の通信の並列化による効果が期待できる.

5 分散共有メモリ

5.1 一貫性制御方式

Lemuria では, 一貫性制御を行うレベルに3つのレベルがある. 各レベル毎に一貫性制御方式を変えることが可能である.

- 同一マシン内

- 同一 Cluster 内
- Cluster 間

一貫性制御は write-invalidate 方式によって行う. write-invalidate 方式は, 読み出しはコピーを生成することで複数のマシンで可能である. 書き込みは同時には1台のマシンでのみ可能であり, その時点で存在するメモリコピーを無効化することによって一貫性を保つ方式である.

5.2 一貫性制御処理の流れ

Lemuria を使用するアプリケーションは libLemuria をリンクすることによって, mmap() と mprotect() の2つのシステムコールと, シグナルによるプロセス内のページへのアクセス権の制御を行う (3.1節参照). ページフォルトによるシグナルが libLemuria で受け取られると, それはページ要求として Lemuriad に伝えられる. Lemuriad では当該ページの一貫性制御方式に基づいて Cluster 内での処理を行う. 当該メモリオブジェクトが他の Cluster と共有されており, 最新のページを所有するマシンが他の Cluster に所属していた場合には, Reflector に処理が要求される. 以下, Cluster 内での共有と Cluster をまたがった共有の場合について, それぞれの一貫性制御の手順を述べる.

5.2.1 Cluster 内のみの共有

ページ要求を受けた Lemuriad は, 当該ページへの書き込み権をもつマシン (以下**ページオーナ**) の Lemuriad にページの要求を行う. ページオーナが確定できない場合は, 当該ページを始めに作成したマシンか, 以前に当該ページを移動したマシンの Lemuriad にページの要求を行う. 要求を受けたページオーナの Lemuriad は, 当該ページを割り当てられたプロセスの libLemuria に, 当該ページのアクセス権の変更を要求する. さらに, 当該ページの内容が更新されていた場合には, 当該ページを要求元の Lemuriad を経由してプロセスに供給する. 最新内容のページの供給を受けたプロセスでは, libLemuria がページのアクセス権の設定を変更して, アクセス可能となる.

5.2.2 Cluster 間の共有

Reflector に伝えられた要求は, ページオーナが属する Cluster の Reflector に転送される. このときのページオーナの検索方法は, Cluster 内の場合と同様である. 異なる点は, すべて Reflector のみで処理が行われることである. Reflector のみで処理を行うためには, Cluster 内のページオーナを常に管理している必要があるが, Cluster 間の通信がすべて Reflector を経由して行われるので, これは容易に実現できる.

この後の Reflector から Cluster 内のページオーナへの要求は, 前節の Cluster 内の場合と同様の方法で行う. ページの転送が必要な場合は, 要求時とは逆方向に Reflector を経由して, 要求元のプロセスへページを配送する.

6 おわりに

本論文では, 分散並列処理のプラットフォーム *Lemuria* の構成について述べた. *Lemuria* は, 分散共有メモリに基づいて複数のマシンでメモリオブジェクトを共有し, Cluster 分割による通信の並列化と Relfector による通信量の抑制を行うシステムである.

Cluster 分割による通信の並列化を有効に機能させるために, 分散共有メモリの一貫性制御方式について検討を行う必要がある. 特に, 通信パケットの衝突を防ぐために, ページ配送の同期を行う必要がある. これは今後の課題である. また, メモリオブジェクトの移動やプロセスの複製の生成によるネットワークスラッシングの抑制方法について, 実行のタイミングや移動先の選定基準などが課題となっている.

参考文献

[1] 斎藤彰一, 中村素典, 大久保英嗣, 大野豊, 白川洋充: Mach の外部ページャを利用した分散共有メモリサーバの評価, 情報処理学会研究会報告 94-OS-65, Vol. 94, No. 64, pp. 145–152 (1994).

[2] 斎藤彰一, 中村素典, 大久保英嗣, 大野豊: 分散共有メモリサーバの大規模データ処理への適用と評価, 情報処理学会研究会報告 95-HPC-55, Vol. 95, No. 28, pp. 25–32 (1995).

[3] Ming-Chit Tam, Jonathan M. Smith, and David J. Farber: *A Taxonomy-Based Comparison of Several Distributed Shared Memory Systems*, Operating System Review, Vol. 24, No. 3, pp. 40–67 (1990).