Vol.2011-OS-118 No.14
2011/7/27

耐故障分散ロック機構の設計と検証

藤原祐二[†1] 藤田肇[†2] 石川裕[†1,†2]

並列アプリケーションにおいて，ユーザーは頻繁に共有リソースへアクセスする必要がある．分散環境では，共有リソースへのアクセスを管理するのに分散ロック機構が用いられるが，分散ロック機構はロックマネージャやロックを確保しているプロセスが故障するとシステム全体が止まってしまう耐故障性の問題を抱えている．本論文ではこれらの耐故障問題に対処するための新しい分散ロック機構のデザインを提案する．提案手法では，各マネージャは論理的に隣接しているマネージャのロック情報の複製を保持し，マネージャが故障した時に代替マネージャとしてロック管理を行う．さらにロックマネージャはロックを保持しているプロセスを監視し，プロセスが落ちていたらロックを回収する．また，PROMELA を使用した提案機構のモデル化手法，SPIN を用いた設計の正当性の検証法について論ずる．

The Design and Verification of Distributed Lock Manager with Fault Tolerance

Yuji Fujiwara,[†1] Hajime Fujita[†2]
and Yutaka Ishikawa[†1,†2]

In parallel applications, users often need to access shared resource. In distributed environment, distributed lock manager control accesses to shared resources, but there are fault tolerant problems that hole system may go down when a lock manager or a process keeping a lock fail. To deal with these problems, this paper proposes the novel design of a distributed lock manager. In proposed design, each manager has a duplication of logically contiguous manager's lock information and operate as an alternative node when the adjoining manager fails. Lock manager also monitor the process owning a lock and retrieve the lock when the process has failed. Furthermore, this paper describe the technique of modeling proposed algorithm that uses PROMELA and how to verify the validity of the design with SPIN model checker.

1. はじめに

並列アプリケーションを実行する際，共有リソースへのアクセス，書き込みは頻繁に発生する．そのため，分散環境では共有リソースへのアクセスの競合を防ぐために分散ロックマネージャを利用しているが，分散ロック機構を設計するにあたって考慮すべき項目がいくつかある．ロック管理の負荷分散の問題，パフォーマンスの向上の問題，さらにロック機構の耐故障性の問題である．これらの問題に対してロックの受け渡しをクライアントプロセス同士で行うことによる負荷分散[5] や InfiniBand[2] を用いたパフォーマンスの向上[5)9)] 等様々な設計が提案されている．

本論文では，分散ロック機構の耐故障性問題に焦点を当てて論じていく．分散ロック機構の耐故障性問題は主に二つのケースが存在する．一つ目のケースは動作中に複数のロックマネージャのうちのひとつが落ちてしまうケースである．このケースでは，ロックマネージャの故障とともに，管理しているロック情報も失われてしまうため，ただ単に他のノードが代わりに管理を行うだけでは解決できない．もう一つのケースはある共有リソースへのロックを保持しているプロセスがそのロックを開放しないまま突然落ちてしまうケースである．ロックが開放されないため，他のプロセスがそのリソースにアクセスできなくなってしまう．いずれのケースでもクラスタ内の一つのノードが故障するだけで，システム全体が止まってしまう危険性をはらんでいる．

この二つの耐故障性問題に対応できる機構として，DFLM 機構 (Distributed Fault Tolerant Lock Manager) を提案する．DFLM 機構では，前者の問題を解決するために，ロックマネージャが落ちたときにそのロック管理を代行する代替マネージャを用意している．さらに，代替マネージャにはロック情報の複製を持たせておくことによって，ロックマネージャ故障後も問題なくロック管理を行うことができるようにしておく．また，後者の問題に対応するために，ロックマネージャはロックを保持しているプロセスを定期的に監視し，プロセスが落ちていたらロックを回収する．

さらに，本論文では提案したアルゴリズムを分散並列アルゴリズムのモデルを記述できる言語である PROMELA を使ってどのようにモデル化するかを論ずる．そして，SPIN モデ

†1 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
†2 東京大学情報基盤センター
Information Technorogy Center, The University of Tokyo

ルチェッカ[3)6)] を用い，アルゴリズムの正当性を検証する．

2. 関 連 研 究

分散ロック機構では，一つのノードがすべてのロックを管理するのではなく，複数のノード（基本的にクラスタ内のすべてのノード）がロックマネージャをデーモンとして動かして，ロックを分散管理している．そのため，ロック管理にかかる負荷がクラスタ全体に分散され，スケーラビリティが向上するとともに，単一故障点を持たなくなるという利点がある．さらに，各々のマネージャはロックリクエストを送ってきたプロセスの情報を FIFO キューを用いて管理している[12)]．分散ロック機構は実際のシステムに用いられており，例えば HP 社の VMS クラスタ[1)] には共有リソースへのアクセス管理に分散ロック機構が用いられている．

共有リソースへのアクセスは頻繁に発生するため，分散ロックマネージャの性能はシステム全体の性能に大きな影響を与える．そのため分散ロックマネージャのパフォーマンスを向上させる研究は数多い．例として，InfiniBand の RDMA(Remote Direct Memory Access) を利用した分散ロック機構[9)5)] について紹介する．RDMA を用いることにより，リモートのメモリにアクセスする際，CPU へ割り込みをすることなくアクセスでき，パフォーマンスの向上が期待できる．この機構では，RDMA を利用するために，分散 FIFO キュー機構（Distributed FIFO Queue mechanism）という機構を取り入れている．この機構では，ロックマネージャはロックを管理しているキューの末尾を表している 64 ビットの値のみを保持している．ロックを要求するプロセスは RDMA の CS(Compare and Swap) オペレーションを用いて自身をキューの末尾に加え，さらにマネージャが管理している値を更新する．この分散キューでは，直接的なロックのやりとりをロックマネージャではなくロックを要求しているプロセス同士が行うため，パフォーマンスの向上だけでなく，ロック管理の負荷分散にも貢献している．

この他にもロック情報とロック管理権限の移譲によりロック管理をできるだけローカルで行う設計[7)] や書き込み時にリソースの複製を用意することでロックを取得しやすくする設計[4)] 等様々な方針で[8)] パフォーマンスを向上させている．しかし，これらの設計では，パフォーマンスの向上やより良い負荷分散を実現しているが，分散ロック機構の耐故障性の問題については言及していない．

3. 設　　計

3.1 ノードの配置

DFLM では，分散ハッシュテーブルのアルゴリズムの一つである Chord[10)13)] と同じように，すべてのノードをハッシュ関数により得られた数値順に円形に配置し，一つのリングを形成する．ノードの追加削除が頻繁に発生する P2P システムの Chord アルゴリズム[11)] は，ノードの故障と復帰を想定している DFLM 機構の要求にマッチしているが，未知のノードの追加は想定していない．

図 1 の例で具体的に説明する．図中の 100 や 200 といった数値はそのノードやリソースのハッシュ値である．ファイル等のリソースもノードと同様にハッシュ関数によりリング上にマップされている．このとき，各ロックマネージャは，自身のハッシュ値と前ノードのハッシュ値との間にマップされたファイルのロックを管理する．図 1 では，ハッシュ値 200 のマネージャはハッシュ値 150 のリソースを，ハッシュ値 300 のマネージャはハッシュ値 250 のリソースをそれぞれ管理する．この様にリソースのロック管理を割り振ることで，ロック管理にかかる負荷を全体に分散させると同時にロックマネージャがどの範囲のリソースのロックを管理するのかが明確になる．

3.2 ロックマネージャの構成

各ロックマネージャは以下のものを構成要素として持っている．

- 自身が管理しているロック情報（FIFO キュー）
- 前ノードのマネージャが管理しているロック情報の複製
- クラスタ内の全てのノードの位置情報とハッシュ値を対応付けているテーブル
- 使用している全てのノードの位置情報とハッシュ値を対応付けているテーブル

ユーザーが分散並列プログラムを動かす際，クラスタ内のすべてのノードではなく，一部のノードを使用するのが一般的である．そのため，ロックマネージャは起動時にクラスタ内すべての位置情報を記録しているテーブルから，実際に使用しているノードの情報のみを抽出したテーブルを作成する．プロセスが共有リソースへアクセスしようとするとき，この新しく作ったテーブルからロックリクエストの送り先の情報を入手する．以下，この起動時に作ったテーブルを実行時テーブルと呼ぶことにする．

各ロックマネージャは自身が管理しているロック情報の他に前ノードのマネージャが管理しているロック情報の複製を保持している．図 1 の例で言うなら，ハッシュ値 300 のマネージャはハッシュ値 150 のリソースのロック情報も複製として持っている．これは前ノードが

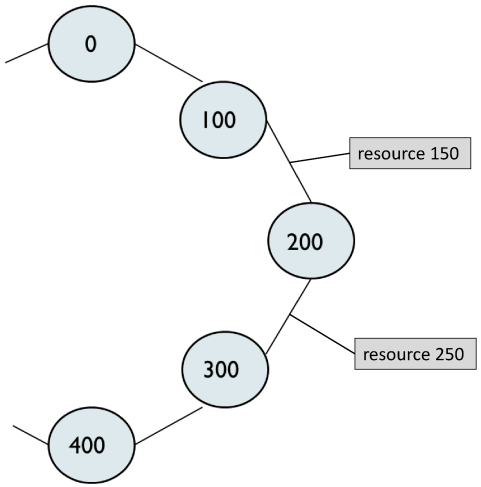

図 1 ノードを円形に配置する

故障したときに，代替ノードとして代わりにロック管理を行うためである．つまり，前ノードが故障したら，それまで自身が管理してたリソースに加え新たに前ノードが管理していたリソースのロックも管理する．管理するロック情報はリソースごとに FIFO キューを用いて管理する．キューの先頭にあるプロセスが現在ロックを所有しているプロセスであり，そのプロセスがロックを開放したら，そのプロセスをキューから取り除き，次のプロセスにロックを与える．

3.3 ロックリクエスト

単純に考えれば，プロセスのロック要求は，ロックマネージャにロックリクエストを送り，マネージャからリプライを受け取るという単純な通信で実現できる．しかし，DFLM ではロックリクエストの際に代替マネージャの持つロック情報の複製も同時に更新しなければ，突然のノードの故障に対処できない．そのためロックリクエストの手順がやや複雑になる．

クライアントプロセス（以下クライアント）は複製情報も同時に更新するために，要求するロックを管理しているロックマネージャ（以下オーナー）だけでなく，複製を持っているその次ノードのマネージャ（以下サブ）にもロックリクエストを送る．ロックリクエストの手順は**図 2** の通りである．クライアントプロセスはオーナーとサブ両方にロックリクエストを送信し，リプライを受け取る．そして両者からのリプライから状況を判断し，リプライバックを送信するというのが一連の流れである．

クライアントプロセスがリプライとして受け取るステータスは以下 4 つである．

- 即座にロックを取得できる状態であることを示す accept
- 他のプロセスが既にロックを保持していることを示す wait
- ロックリクエストを受理できない状態であることを示す block
- 一定時間内にリプライが返ってこなかったことを示す timeout

これらのリプライに応じてクライアントはリプライバックを送り返す．このとき，どちらか片方でも block もしくは timeout だった場合はそのロックリクエストは失敗となる．accept もしくは wait の場合はロックリクエストは成功し，リプライバック後にマネージャのロック情報に追加される．

リプライの受信が timeout し，そのマネージャが故障していることが分かるとクライアントはロックリクエストを再度やり直す．例えば図 1 で，ハッシュ値 150 のリソースに対するロックリクエストを行ったとする．このとき，ハッシュ値 200 のマネージャがオーナー，ハッシュ値 300 のマネージャがサブとなる．このケースにおいて，オーナーからのリプライ受信がタイムアウトし，実際にオーナーが故障していた場合，ハッシュ値 150 のリソースの新しいオーナーはハッシュ値 300 のマネージャになり，サブはハッシュ値 400 のマネージャになる．そして，新しいオーナーとサブに対してロックリクエストを再度おこなう．

3.4 ロックリリース

ロックリリースもロックリクエストと同様にオーナーとサブへリリース要求を送信し，両方からリプライを受け取る．このとき，リプライバックの送信は行わない．どちらかからのリプライがタイムアウトした場合，サブの次ノードのマネージャに同様のリリース要求を送信し（オーナーからのリプライがタイムアウトした場合でも，サブからのリプライがタイムアウトした場合でも，サブの次ノードマネージャが新しいサブになる），リプライを受け取る．

3.5 マネージャの故障への対処

あるマネージャが故障してしまったとき，周囲のマネージャ（故障したノードの前ノード，次ノード，さらにその次ノード）のロック情報，複製情報の再構成が必要になる．ロック情報と複製情報の再構成は各マネージャをブロックモードに移行して（ロックリクエストに対し，block ステータスを返す状態）から 3 つの操作を行う．

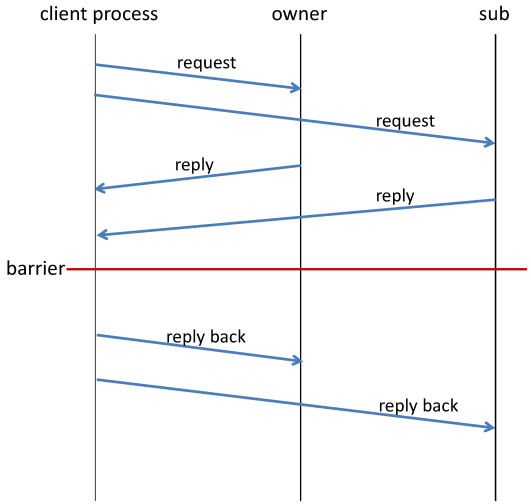

図 2 ロックリクエストの流れ

一つ目の操作として，故障したマネージャのサブが持っている複製情報をサブの管理するロック情報に統合する．マネージャが故障したとき，そのサブは代替マネージャであるので，故障したマネージャの管理していたロックを引き継がなければならない

2 つ目の操作として，故障したマネージャの前ノードが管理しているロック情報の複製を故障したマネージャのサブに持たせる．ロック情報の複製を持つべき対象は自ノードの前ノードになるのでそれに応じて複製も再構成しなければならない．

3 つ目の操作として，サブが新しく管理することになったロック情報の複製をサブの次ノードのマネージャに持たせる．これは，サブの管理するロック情報がマネージャの故障により増えたためそれに応じてサブのロック情報の複製も更新しなければならないからである．

以上の操作でロック情報，複製情報の再構成は完了し，各マネージャをブロックモードからもとに戻す．また，故障したマネージャを復帰させるには，ロック情報，複製ともに故障する前の割り当てに戻せば良い．

3.6 クライアントが落ちたときの対処

各マネージャは自身が管理しているリソースのロックを保持しているプロセス全てと定期的に通信を行い，ロックを保持したまま故障していないか確認する．通信時に一定時間返信がなくタイムアウトしてしまった場合，そのノードが故障しているかどうか確認し，故障している場合はロックを回収する．ロックを回収したら次に待っているプロセスにロックを渡す．

4. モデルの記述と検証

並列に動くシステムが本当に想定通り動くのかを検証する手法として，モデル検査がある．この章では，分散システムをモデル化する手法を簡単なモデルを交えて解説し，実際に提案手法をどのようにモデル化するかを論ずる

今回提案した DFLM 機構の耐故障性をモデル検査によって検証するに当たって以下のモデル化が必要となる．

- 計算ノードのモデル化
- 並列動作のモデル化
- ノード間通信のモデル化
- 突然発生するノードの故障のモデル化
- タイムアウトのモデル化

これらモデルを PROMELA 言語のどういった機能を用いて記述するのかを解説する．

4.1 計算ノードのモデル化

まず，計算ノードをどのようにモデル化していくかについて説明する．PROMELA 言語では，実行単位をプロセスとして記述する．DFML 機構をモデル化するに当たって，各計算ノードはロックマネージャと実際にロックの要求を行うクライアントプロセスを持っているので，1 ノードをモデル化するのに 2 つのプロセスを使って記述することになる．

Listing 1 のモデルは，2 つのクライアントプロセスが 2 つのマネージャに延々と整数値を送りつけ，マネージャはそれを延々と受信し続けるモデルである．実行される命令は，そのとき実行可能なプロセスが非決定的に選択されることになっており，Client_A ばかりが送信する例や，Manager_A ばかりに送信される例など，様々な実行結果が得られる．もちろん，SPIN によるモデル検査では非決定的に 1 パターンのみ実行するのではなく，到達可能なすべてのパターンを網羅する．

Listing 1　2 ノードによる送受信モデル

```
chan send_A = [4] of {byte};
chan send_B = [4] of {byte};

active proctype Client_A() {
  do
    :: send_A ! 1;
    :: send_B ! 1;
  od;
}

active proctype Client_B() {
  do
    :: send_A ! 2;
    :: send_B ! 2;
  od;
}

active proctype Manager_A() {
  byte temp;
  do
    :: send_A ? temp;
  od;
}

active proctype Manager_B() {
  byte temp;
  do
    :: send_B ? temp;
  od;
}
```

Listing 2　マネージャの故障モデル

```
chan send_A = [2] of {chan};
chan reply_A = [2] of {mtype};
mtype = {accept, fail};
bool kill = false;

active proctype Client_A()
{
  mtype temp;
  do
    :: send_A ! reply_A;
    :: reply_A ? temp;
  od;
}

active proctype Manager_B()
provided (!kill)
{
  chan reply_chan;
  do
    :: send_A ? reply_chan;
       reply_chan ! accept;
  od;
}

active proctype p_killer()
{
  chan reply_chan;
  kill = true;
  do
    :: send_A ? reply_chan;
       reply_chan ! fail;
  od;
}
```

4.2　並列動作のモデル化

計算ノードはプロセスレベルで記述することにより，並列に実行される動作をモデル化することができた．しかし，実際は一つのプロセスで表されているロックマネージャやクライアントプロセスも並列な動作を行う．例えば，ロックマネージャの場合は，ロックリクエストを受信して返信する，ロックリクエストのリプライバックを受信する，ロックリリースリクエストを受信して返信する等の操作を並列して行っている．そのため，プロセスレベルでの記述の他に，プロセス内部で並列動作を記述する必要がある．

Listing 1 では，各クライアントプロセスは Manager_A, Manager_B に対する送信を逐次的にではなく並列的動作にするために，PROMELA の do 文使っている．do 文では::で記述されるガード文のあとに並列動作させたい操作を記述する．ガード文が複数あり，さらに複数の操作が実行可能である場合は，実行可能な操作の中から非決定的に一つの操作が選択される．Listing 1 の例では，各クライアントプロセスは Manager_A と通信するのか Manager_B と通信するのかを非決定的に選択することにより，並列動作をモデル化している．

4.3　通信のモデル化

分散ロックマネージャのモデル化にあたって，ノード間通信のモデル化は必要不可欠である．PROMELA は通信をモデル化するたに通信チャネルというデータ構造を用意している．通信チャネルは FIFO キューで表現されており，チャネルへデータを送信するとキューにデータがポップされ，チャネルからデータを受信するとキューからデータをプッシュし，データを取り出す仕組みになっている．また，キューの先頭からプッシュするだけではなく，データのマッチングを用いてキューの途中からデータを取り出すことも可能となっている．

Listing 1 の例では，Manager_A と Manager_B へ整数値を送信するためのチャネル send_A, send_B を宣言している．そして，各クライアントはチャネルに対し整数値を送信し，各マネージャはチャネルにデータが入っていたらそのチャネルからデータを受信する．チャネルにデータが入っていないときは，受信文は実行可能にならない．

4.4　突然発生するノードの故障のモデル化

提案手法の耐故障性を検証するにあたって，ノードの故障もモデルの中に組み込んで置かなければならない．PROMELA ではプロセスに実行の可，不可を決定するブール値を指定することができ，それをによってノードの故障や割り込み操作をモデル化できる．また，任意のタイミングでの故障をモデル化するには，プロセスのブール値を変更する専用のプロセスによって外部から操作しなければならない．

Listing 2 は Client_A が Manager_A と通信し，Manager_A からリプライを受けるという動作を延々としている途中に Manager_A が故障するモデルである．Listing 2 では，ロックマネージャA の実行の可，不可を操作するブール値 kill を外部のプロセス p_killer が操作するようになっている．実行するプロセスの決定は非決定的に行われるため，任意のタイミングで p_killer プロセスが動作し，ロックマネージャA を実行不可能な状態にする．モデル検査ではすべてのパターンを網羅するので，起こりうる全てのタイミングでの故障をモデル化している．

4.5　タイムアウトのモデル化

実際の実装では，各クライアントプロセス，各マネージャは故障しているノードとの通信をタイムアウトによって途中で打ちきることができる．しかし，モデル検査では時間の経過によるタイムアウトそのものをモデル化するのは非常に困難である．そのため今回のモデルでは，通信先のプロセスが落ちていた場合，代替プロセスが fail のステータスを返すことに

よって，タイムアウトをモデル化する．

Listing 2 の例では，マネージャを故障させるプロセス p_killer がそのまま故障プロセスの代わりに通信を行い，fail ステータスを相手プロセスに送っている．

4.6 モデル検査

提案手法がロック管理を正しく行えているかどうかを検証するために，2 種類の検証を行う．一つは複数のプロセスが共有リソースに同時にアクセスする状況が発生しないか（競合状態），もう一つはクライアントプロセスが送ったロックリクエストが受理されたとき，必ずその後ロックを取得できるかどうかである．

4.6.1 競合状態の検証

競合状態の検証には PROMELA の assertion を用いる．SPIN はモデル検査中に assertion 違反が発生したら，その違反までの経路を示してくれる．アクセスの競合が起こらないことを検証するためには，プロセスがクリティカルセクションに入る前に，そのクリティカルセクションに他のプロセスが入っていないことを assert しておけば良い．

Listing 3 は，2 つのクライアントが 1 つのマネージャと通信し，リプライを受け取ったら即座にクリティカルセクションにアクセスするモデルである．このモデルでは，各クライアントはリプライのステータスを吟味せずにクリティカルセクションにアクセスしているのでもちろん競合状態に陥る．この様なモデルでクリティカルセクションへのアクセスの前に assert 文を入れておくと競合が起こることを検査時に報告してくれる．

4.6.2 ロック取得保証の検証

クライアントプロセスが送ったロックリクエストが受理されたとき，必ずその後ロックを取得できることを検証するのは競合状態のときのように assertion で簡単に検証することは難しい．そのためこの検証には線形時相論理による検証を用いる．クライアントプロセスが送ったリクエストが受理された状態を P，クライアントプロセスがそのロックを取得した状態を Q とすると検証したい内容は $\Box(P \rightarrow \Diamond Q)$ の時相論理式で表される．

今回は競合状態の検証のときと異なり，検証ようの変数は導入しない．その代わり，ロックの受理を記述している文と，ロックの取得を記述している文にラベルをつけ，それを用いて検証する．3 の例では簡単のために，ロック要求している文に P というラベルを，クリティカルセクションに入る文に Q というラベルをつけている．このラベルを用いて時相論理式を記述すれば，SPIN によりその要求が満たされているかの検証ができる．

4.7 DFLM 機構のモデル化

DFLM 機構のモデルを記述するにあたって，任意の N 個のノードによる動作モデルを記述するというのは現実的ではない．モデル検査は起こりうる全てのパターンを網羅して検証をおこなう．そのため，モデルを大きくしすぎると到達可能な状態数が爆発的に増加し，それに伴い必要メモリ量も増加してしまう．よって，要求する性質を検証するのに必要な分だけモデル化して検証することになる．

DFLM 機構で行われる，ロックリクエストやノード故障時の動作等をモデル化するには，最低でも 4 つのノードが必要となる．そのため，提案アルゴリズムの正当性の検証には，4 ノードで構成される DFLM 機構のモデルを記述することになる．

Listing 3　競合状態の検証モデル

```
chan send_A = [2] of {chan};
chan reply_A = [2] of {mtype};
chan reply_B = [2] of {mtype};
mtype = {client_A, client_B, no_process, accept};
mtype critical;

active proctype Client_A() {
  mtype stat;
  do
P:  :: send_A ! reply_A;
    :: reply_A ? stat;
       assert(critical == no_process);
Q:     critical = client_A;
  od;
}

active proctype Client_B() {
  mtype stat;
  do
    :: send_A ! reply_B;
    :: reply_B ? stat;
       assert(critical == no_process);
       critical = client_B
  od;
}

active proctype Manager_A() {
  chan reply_chan;
  do
    :: send_A ? reply_chan;
       reply_chan ! accept;
  od;
}
```

5. まとめと今後の課題

近年，様々な分野で分散並列コンピューティングへの要求が高まっている．分散並列コンピューティングには，共有リソースへのアクセスを管理する機構が必要不可欠であり，分散ロック機構がその役割を担っている．分散ロック機構ではあるノード単体ではなく，複数の

ノードでロック管理をおこなうことにより，負荷を分散させているため，スケーラビリティの面でも優秀な機構である．

この論文では分散ロックマネージャが抱える耐故障性の問題を解決するために，DFLM 機構という新しいロックマネージャのデザインを提案した．DFLM 機構では，すべての計算ノードは Chord アルゴリズムと同じ手法を用いて円形に配置されている．すべてのマネージャは前ノードのマネージャが保持しているロック情報の複製を持っており，前ノードが故障した際は，代替マネージャとして代わりにロックリクエストを処理する．また，ロックリクエストを送るプロセスは，代替マネージャが持っている複製の情報も更新するために，目的のマネージャと同時に代替マネージャにもロックリクエストを送る．さらに，各ロックマネージャはロックを保持しているプロセスを定期的に監視し，プロセスがロックを保持したまま落ちてしまった場合はそのロックを回収する．これらの機構により，分散ロックマネージャの耐故障性を向上させた．

また，モデル検査によって本当に提案手法が耐故障性を実現で来ているが調べるための，モデルの記述の仕方について論じた．SPIN で利用されている PROMELA 言語は，分散並列システムを記述するのに適しており，複数ノードによる並列実行やノード間通信，突然のノードの故障などをモデル化することが可能である．さらに，assersion や線形時相論理を用いることにより，複数プロセスによる共有リソースへの競合が発生しないか，またプロセスがロックリクエストを送りそれが受理された場合，その後いつかは必ずロックを得ることできるか等の性質を検証できることを解説した．

今後の課題としては，実際の検証を通して提案手法の欠陥を発見，修正し，よりよい耐故障性能を実現したい．

謝辞　本研究の一部は，科学技術振興機構 戦略的創造研究推進事業 (CREST) (領域名：実用化を目指した組込みシステム用ディペンダブル・オペレーティングシステム) 技術課題：「高信頼組込みシングルシステムイメージ OS」による．

参 考 文 献

1) *HP OpenVMS Systems Services Reference Manual*.
2) *Infiniband Trade Association*. http://www.infinibandta.org/.
3) Mordechhai Ben-Ari. *SPIN* **モデル検査入門**. Ohmsha, 2010.
4) Sungchune Choi, Minseuk Choi, Chunkyeong Lee, and HeeYong Youn. Distributed lock manager for distributed file system in shared-disk environment. In *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology*, CIT '10, pages 2706–2713, Washington, DC, USA, 2010. IEEE Computer Society.
5) Ananth Devulapalli. Distributed queue-based locking using advanced network features. In *Proceedings of the 2005 International Conference on Parallel Processing*, pages 408–415, Washington, DC, USA, 2005. IEEE Computer Society.
6) Gerard J.Holzmann. *The SPIN Model Checker: Primer and Reference Manual*.
7) H.Kishida and H.Yamazaki. Ssdlm: architecture of a distributed lock manager with high degree of locality for clustered file systems. In *Communications, Computers and signal Processing, 2003. PACRIM. 2003 IEEE Pacific Rim Conference on*, pages 9–12. IEEE Computer Society, 2003.
8) Knottenbelt, W.J. Zertal, S. Harrison, P.G. Performance analysis of three implementation strategies for distributed lock management. In *Computers and Digital Techniques, IEE Proceedings*, pages 176–187. IEEE Computer Society, 2001.
9) S.Narravula, A.Marnidala, A.Vishnu, K.Vaidyanathan, and D.K. Panda. High performance distributed lock management services using network-based remote atomic operations. In *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '07, pages 583–590, Washington, DC, USA, 2007. IEEE Computer Society.
10) Parallel and Distributed OperatingSystems Group. *The Chord/DHash Project*. http://pdos.csail.mit.edu/chord/.
11) Simon Rieche, Klaus Wehrle, Olaf Landsiedel, Stefan Gotz, and Leo Petrak. Reliability of data in structured peer-to-peer systems. In *Proceedings of the 2004 International Workshop on Hot Topics in Peer-to-Peer Systems*, pages 108–113, Washington, DC, USA, 2004. IEEE Computer Society.
12) K.Thomas. *Programming Locking Applications*, 2001.
13) 江崎　浩. *PEER TO PEER TEXTBOOK*. インプレス R&D, 2007.