階層的構造によるPCクラスタ内P2Pシステムの構築

上川　純一 ††　廣安　知之†
三木　光範 †　谷村　勇輔 ††

本研究では，PC クラスタ内における階層的な構造を持つ P2P 的システムを提案する．このシステムでは高い耐障害性と，効率の良い通信を両立することを目標とする．本システムは，通信量の多い数値計算クラスタ等の計算のためのデータ通信の枠組として利用されることを想定している．このシステムは，階層的な構造を取りリレー方式で情報をやりとりする．さらに，木構造を動的に変化させて，耐障害性を実現している．また，PC クラスタには代表的ノードが存在するという性質を応用して効率の良い通信を実現しようとする．

P2P System Within PC Cluster Using Hierarchical Architecture

Junichi Uekawa,†† Tomoyuki Hiroyasu,† Mitsunori Miki†
and Yusuke Tanimura ††

In this paper a P2P-type system implementation with a hierarchical structure inside PC Cluster is proposed. This is a system which has a high tolerence against failure and has a high efficiency in communication. This system aims to be used as a framework for intercommunication used in numeric computation clusters, where intensive communication is done. This new system applies a hierarchical relaying scheme, and uses a dynamical tree reconstruction in order to achieve the fault tolerency. PC Cluster systems tend to have a representative node, and this system aims at taking advantage of that characteristic for efficient communication.

1. はじめに

P2P システムは，従来の High Performance Computing (HPC) の分野において今後普及すると考えられる耐障害性を提供するシステムである．P2P では，メッセージが集中的に管理されることなく目的地に到達する．例えば freenet[1)]，gnutella[2)]，napster[3)] 等はファイル交換を目的とするが，代表的な P2P のシステムである．これらは，各ノードがクライアントであると同時に，ファイルサーバとしての機能を果たしている．P2P では単一サーバに負荷が集中せずサーバを増強することなくより大規模なシステムに対応できる．しかし，P2P には実装しているアプリケーションの動作が遅いという問題がある．例えば，gnutella[2)] ではデータの検索に長時間を要する．アプリケーションのアイドル時でもネットワークの維持をするためだけに，ネットワークの帯域を大量に消費する．

† 同志社大学工学部
Faculty of Engineering, Doshisha University
†† 同志社大学大学院工学研究科
Graduate School of Engineering, Doshisha University

PC クラスタは一般的な PC をネットワーク接続することにより構築する並列計算機である，近年のコモディティハードウェアの性能の飛躍的な向上とコストパフォーマンスの良さにより，急速に利用が広まっている．また，さらに規模の大きな PC クラスタも多く構築されるようになってきた．

規模の大きな PC クラスタを利用する際に問題となるのは，多ノードへのユーザのジョブの割り当てと頻繁に発生するノードの故障に対する対処である．PC クラスタ上で稼働するアプリケーションには静的なノード数が必要なものだけでなく，ノードの負荷状況や故障状態に応じて柔軟に使用するノード数を変更することが可能なものも存在する．そのようなアプリケーションに対応できるミドルウェアとしては Condor[4)] があげられる．しかしながら Condor はより広いネットワークでの使用を想定するミドルウェアであり，本研究で対象とするアプリケーションを稼働するには大きくかつオーバヘッドが大きくなる．

そこで本研究では，P2P の特徴をもったシステムを PC クラスタ内に実装する手法を提案する．提案システムでは，クラスタ内で**図 1** のようなツリー形式の

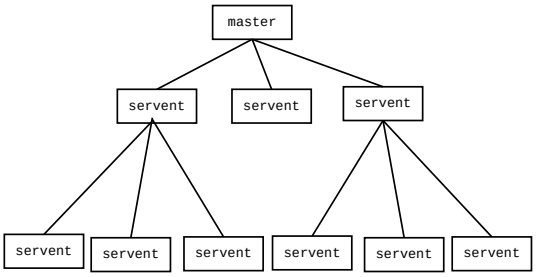

図 1 Tree structure within the cluster

トポロジを利用して情報を伝達することとする．これは，従来の P2P システムのノードがグラフを構成しているトポロジとは大きく異なる．しかし PC クラスタでは必ずマスターノードが存在するため，ツリー構造を利用することは有効である．

構築したシステムの有効性を検討するためにクラスタモニタシステムを実装した．そこでは複数のコンピュータを servent とし，相互に接続した．servent からの情報は最終的にはクラスタシステムのマスターノードに到達するように設計している．マスターノードの情報は，クライアントシステムから取得され，利用者が利用する．

その結果，提案システムが実際に動的にネットワークを再構築することができることが示され，耐障害性をもつことが明らかとなった．

2. P2P の目的と形態

既存の「P2P」システムの実装は，その目的などの点で，次のような特徴を有する．

- 耐障害性: 一台の停止は全体に影響しない．
- 動的な接続制御: ユーザはネットワークトポロジを指定しない．
- マシン間のロードバランスの維持: サーバとしての仕事を多くのホストが分担する．
- servent の存在: サーバでありクライアントであるノードがある．

Windows のネットワークで利用されている SMB プロトコル[5)6)] は P2P システムであり，SMB プロトコルでは複数のホストがサーバとしての動作を行い，上記の目的を達成していると考えられる．

純粋な P2P システムは，中央サーバを持たないシステムである．そのようなネットワークにおいては，各ノードがサーバとして動作することができ，ネットワークの動作のために必ず存在しなければならないシステムというものが無い．gnutella[2)] や，IRC サーバのプロトコル[7)] はこの特徴を有する．

逆に，ハイブリッドな P2P システムとは，中央サーバを必要とする P2P システムである．ノード間の直接の通信は行われているが，中央サーバが存在しなければ他のノードの情報が入らない．直接のデータ交換を可能にしているが，情報の検索をする際には，中央のサーバを必要とする Napster[3)] はこの代表である．

本研究では，PC クラスタシステムのローカルなネットワークに一台だけゲートウェイとして動くホストがあることを想定している．本論文で提示するシステムは中央サーバを要求するために，ハイブリッド P2P システムと呼ぶべきである．もしくは，動的な階層型 C/S システムと呼ぶことが可能である．システムの特性を検討するために情報伝達ネットワークの構造の情報管理に焦点をあてて議論を進める．

3. P2P 的階層的構造システムの提案

3.1 背　　景

多くのネットワークアプリケーションではクライアントサーバシステムが広く用いられているが，サーバに負荷が集中する．また，キャッシュ等で情報をリレーして負荷分散すると，キャッシュサーバが停止した時に情報の流れが途絶えることになる．

P2P システムはこのような問題の一つの解決法である．各クライアントは各自で情報を有することができ，また情報の伝達経路も動的に決定される．これは，動的なプロキシメカニズムである．このような考えに基づいてプロキシシステムを構築する場合，理想的なトポロジは木構造である．グラフ構造をとるシステムでは，情報のループなどが生まれるため効率が悪化する．DNS 等のシステムでは木構造を有効に利用してシステムを構築している．それらで利用されている階層的 C/S ネットワークの構成を動的にすることにより P2P 的性質が得られる．そこで本研究では動的に生成される木構造を有するクラスタ内 P2P 的階層型システムを提案する．

3.2 階層的構造

階層構造を作るには，まず，事前に作成された静的な階層構造を利用できる．そのような中間層にあるノードの障害に対して弱い階層構造に対して，停止したノードを削除して動的に階層構造を作りなおすようなシステムを構築すれば，ノードの障害に耐えられるシステムを構築することが可能である．

まず，PC クラスタ内のローカルなシステムを考慮すると，以下が本システムの前提条件となる．

- 「マスターノード」が唯一システムの外部から見えるノードであり，不変である．

- 「クライアント」はシステムの外部にある

この二つの前提により，マスターシステムからクライアントシステムが C/S 的に情報を取得できることになる．情報の検索に必要な時間は，$O(1)$ になる．他の手法と比較すると，例えば chord[8] では，$O(logN)$ である．リアルタイムなアプリケーションには P2P の手法は遅すぎるという場面が多いため，提案する手法は必要な折衷案である．

システムを階層的にすることにより，完全に分散したシステムと比べて利点が生まれる．もっとも顕著なものとしては，効率の良い情報の複製が行われる[9] ということがある．これは，階層的なシステムでは，単一のサーバが複数のノードによりサーバとして利用されるという構造自体の特性による

情報は,「downlink」(マスターノードからの論理的距離が比較的遠いもの) のノードからより「uplink」(マスターノードへの論理的距離が比較的近いもの) のノードに対して定時間隔で，まとめて伝達されていく．downlink のノードからの情報を直接の uplink のノードはより uplink のノードに中継して行き，最終的に最上位にあるマスターノードに情報が到達する．また，各ノードが稼働しているのかいないのかという情報を得るために，定期的なデータの送受信が行われている．各ノードの状態が不明な定期的生存確認を行わないシステムでは，タイムアウトを設定して通信をするなどのオーバヘッドの大きい手法が必要になる．動的な木構造の再構築を効率良く行うために，常に存在を主張するための情報を送る手法をとる．このシステムでは，稼働確認メッセージに便乗して，システムにとって必要なルーティング情報と実際のデータパケットを交換するように設計されている．

3.3 ネットワーク負荷

論理的にネットワーク負荷を求めるために，C/S システムでの負荷を計算する．t がコミュニケーションの頻度であり，k がメッセージの単位量であり，n がノードの数であったとする．

C/S システムにおいては，データの転送量は式 1 になる．

$$\mathbf{I} = tkn \tag{1}$$

Gnutella 式のネットワークにおいて，各クライアントシステムは，$\mathbf{I} = tkn$ のメッセージを送出し，ネットワーク全体としては，メッセージのリレーを行うことにより式 2 に示す量になる．

$$\mathbf{I} = tkn^2 \tag{2}$$

階層的なシステムの場合，m という変数を各サーバントが直接通信している downlink の数として定義する．システムが全体としてバランスのとれた m 分木を構成している場合は，l 段のノードが $m^l kt$ のメッセージを送信しようとする．結果として，$\mathbf{I_l} = tlkm^l$ のデータ量を転送する．これは，n ノードシステムで，$n = m^l$ であるような場合，式 3 に示すような $\mathbf{I_d}$ のデータ通信を行うことになる．

$$\mathbf{I_d} = \sum_{l=1}^{log_m n} tklm^l \tag{3}$$

4. 提案システムの実装

本章では実装の方法について説明する．階層的な構造をもち，かつ動的に変化するようなシステムを構築するためには，初期的な構造を構築する方法と，その構造を有効に維持するための手法が必要となる．

4.1 ノード間の情報交換

まず，情報の伝達は基本的には一方向にしか行わないという前提を設ける．これは，システム全体の中に一台だけ全てを把握しているマスターシステムがあると想定している．主要な情報は外界から唯一アクセス可能なマスターノードのみもてば良い．また，対象とする環境において，物理的なルーティング情報はすでに分かっているものとする．PC クラスタは内部のネットワークとして構築されているため，この条件をみたすことができることが多い．

本システムは情報を一つのシステムから別のシステムに伝達する手法を実装している．本システムでは，「タグ」と「データ」のペアがマスタノードへ向かって転送される．このようにしておくことで，適当な情報が適当に追加拡張できる．データは，簡単に扱えるよう平文で送られる．形式は LDIF に似た形式である．各 downlink は uplink へ，メッセージを一定時間 (例 : 15 秒) に一回送信するように設定されている．各ノードのタイミングは非同期的である．

各ノードに対してのデータは,「:hostname:」のような形式の一行で始まる．それに続いて「tag: data」という形でデータとタグのペアが必要なだけ繰り返される．形式を**図 2** に示す．この情報単位には，後程説明する「Seen-By」という情報が必ず入っている．あるホストからの情報を**図 3** に例示する．

4.2 トポロジの維持情報

構造を維持するための仕組みについて説明する．システム情報には，Route-To，Seen-By，そして Data-Seen というタグが定義されている．

Route-To とは，uplink から downlink に伝達される情報である．uplink から master までの経路にあ

```
Info-packet :- meta-header each-host-info*
each-host-info: host-name host-info
host-name:- ":hostname:"
host-info:- tag-data*
tag-data :- "tag: data"
```

図 2 Information packet

```
:forte004-108:
machine: forte004-108
Seen-By: forte004-108,forte002-107,
 forte04,forte
load1: 1.030000
memtotal: 128307200
memused: 37777408
memfree: 90529792
topuser: ogu
time: 1003039183
```

図 3 An example of information sent from a single host

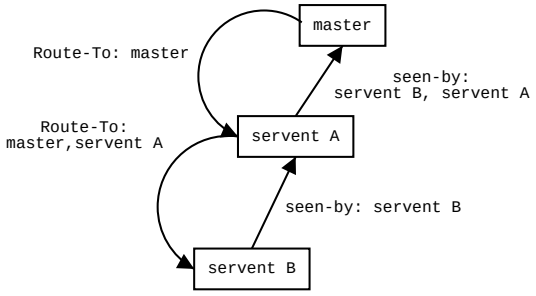

図 4 Information flow of Route-To: and Seen-By: message

る全ホストの名前の一覧となっている．各ホストは，downlink に対して，uplink から取得した Route-To 情報に自分のホスト名を追加したものを伝達する．この情報は，メタ情報であり，情報パケットの一番最初に現れる．

Seen-By は downlink から uplink に対して送信される情報で，このパケットをいままでに見たホストのリストになっている．各ホストは自分自身をパケットの Seen-By に追加する．

Data-Seen はある情報パケットが uplink に対して新しいパケットを送信する必要があるものなのか，それとも古いパケットで再送する必要が無いものなのかということを判断するための情報である．

以上の情報を利用すると，接続トポロジを動的に変

```
while link (car(Route-To)) fails do
  Route-To=cdr(Route-To)
```

図 5 Algorithm to relink to uplink, when uplink is lost

```
D=list of active downlinks
A=D[random]
B=D[random]
if A!=B then
  set Route-To[A]=Route-To[A]+B
```

図 6 Algorithm to relink on too many downlinks

化させることができる．この目的を達成するために二つの基本動作を定義する必要がある．そのアルゴリズムは以下に示す．

4.3 uplink 停止時の再接続のアルゴリズム

uplink の機能が停止しているとき，ホストは Route-To 情報を参考にして uplink の uplink がどれであるかということを発見しようとする．この相手をみつけて直接そのシステムを uplink として利用することにより，システムが uplink を失っている状態から脱出しようとする．さらに uplink の uplink にも障害がある場合，そのさらに uplink に接続を要求をする．この処理は再帰的に行われる．最終的にはマスターノードに到達する．仮にマスターに障害があり，接続できない場合には，システム全体が稼働できない (外部への情報の伝達の手法が存在しない) ので，そこで停止する．

このアルゴリズムは**図 5** に示したようになる．このアルゴリズムでは各ノードの同期をとる必要が無い．しかしながら，ループができる可能性があるので，同時に一つ以上の downlink が再接続をすることは無いものとする．

4.4 ホストに接続している downlink が多すぎる場合の再接続アルゴリズム

既定の上限より多くの downlink がホストに直接接続されている場合，ホストは，downlink を再構成しようとする．あまり多くのシステムが直接通信してきているのは好ましくないからである．現在の実装では，これは完全にランダムに行われている．任意の downlink ホストが選択され，他の downlink の downlink となるように再配置される．**図 6** に示すメタコードはこのアルゴリズムをまとめたものである．

4.5 膨大なノード数への対策

また，本システムでは，負荷を軽減するために，up-

link へ伝達するデータのリレー段数に制限をもうけることができるようにした．システムモニタとして利用される時には，情報が伝達される段数が制限されることは不適当である．しかし中継ノードが情報を処理して uplink のノードに伝達するようなシステムを構築すれば，全ノードの情報がマスターノードに集約される必要は無い．マスターノードや，中間ノードが処理できるノード情報の量には上限があると考えられる．これの制限を除去することにより，全体としてのノードの数がより大きくなった場合でも対応できるようにした．

多くのノードのデータの中継を行うにはメモリと帯域が必要となる．任意のシステムが接続できるようにした場合にはメモリの消費量等が予測できない．この機構を利用して，制限をもうけることにより，予測可能なリソースの消費におさえることができる．

4.6 複数ユーザの利用の衝突への対策

複数のユーザが PC クラスタを利用する際には，利用リソースの衝突が問題となる．これは，ジョブをどのノードに配置するかという問題である．提案システムでは負荷の高いノードに対してはプログラムを実行させない，という設定を行っている．クラスタの各ノードの負荷を常にモニタしているため，あるノードの負荷がユーザが事前に指定した負荷量を越えている場合に，そのノード上のプログラムが停止するようになっている．また，利用者の設定により，プログラムがそのまま停止しているのか，また負荷が低くなったころをみはからって復帰するのか，を選択できる．プロセスのマイグレーション機能は実装されていないため，ネットワーク上に存在する情報から再開することになる．それだけの情報では十分ではないアプリケーションが多く，また再開できるようにチェックポイントをとるにはデータ量が多すぎるものが多いため，再開することはあまり考慮されていない．ただ，計算が長時間にわたり，他のユーザとの衝突がおきやすいシステムを利用している場合は，再開可能なシステムを構築することが望ましい．

5. テストアプリケーション

本研究では提案システムを評価するためにいくつかのアプリケーションを実装した．まず，各ノードのトポロジがどのような構成かを示すモニタを実装した．次に，本システムを利用してクラスタの各ノードの状況がモニタできるようなシステムを構築した．実験に利用した PC クラスタを**表 1** に示す．

表 1 PC Clusters used in experiments

Name	CPU	Network
Forte	Pentium III 600MHz, 40 CPU	100BASE-TX
Cambria	Pentium III 800MHz, 256 CPU	100BASE-TX

```
forte
  forte04
    forte004-103
      forte004-101
    forte004-104
    forte004-105
      forte004-102
    forte004-106
    forte004-107
    forte004-108
    forte004-109
```

図 7 Text-based representation of current hierarchy

5.1 ネットワークトポロジのモニタ

ネットワークで構造がどのように変化していくのかを分析するために，本システムの上にネットワーク接続構造モニタを作成した．

マスタノードは各ホストからの全パケットを受信するように設定されている．各パケットの Seen-By 情報からはパケットが通った経路が記録されている．この情報から論理的なネットワーク構造を取得することができる．このデータを利用して，テキスト形式や，グラフィック画面などで構造を提示することが可能である．実際のトポロジ例は**図 7** にテキスト形式で示す通りである．この例では，ノード forte に downlink としてノード forte04 が接続している．forte04 には downlink として，forte004-103，forte004-104，forte004-105，forte004-106，forte004-107，forte004-108，forte004-109 が接続している．forte004-103 には downlink として forte004-101 が接続していて，forte004-105 には forte004-102 が接続している．forte004-102 で発生したメッセージは forte004-105 と forte04 を経由して forte まで到達する．

初期状態として，完全な C/S システムを与えた場合に，使用ノード数に対するツリー構造を構築するために必要な時間は，**図 8** に示す通りである．実験はクラスタシステム「forte」(表 1) で行った．図は 10 回試行の平均を示している．3 秒のインターバルで図 6 のアルゴリズムに示すオペレーションを行った．その結果ツリー構築に要する時間はノードに対してほぼ線形に増加している．

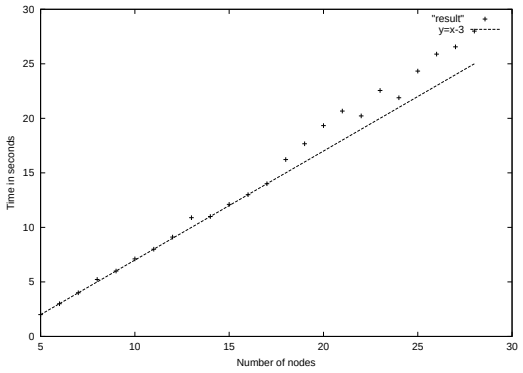

図 8 Time taken for restructuring of network structure

本システムには図 5 のアルゴリズムに基づいてノードの削除等に耐えることができるという耐障害性がある．また，図 6 のアルゴリズムに基づいて単一のノードに対する downlink 負荷が多すぎるとロードバランスをとるメカニズムを持つ．

5.2 クラスタのシステムモニタ

クラスタのシステムモニタをこのネットワークシステムを用いて実装した．各ホストにおいて情報収集プロセスが実行され，負荷平均や最も CPU を利用しているプロセス名とそのユーザ名などの情報を収集する．それは各ノードの CPU 負荷を示す「loadavg: 1.13」のようなタグとデータのペアとして中継されていく．

モニタアプリケーション自身もモニタするためにクラスタに負荷をかけている．その負荷の量が測定された．アプリケーションを Cambria クラスタシステム (表 1) 上で実行した．16 の servent が直接マスターノードに接続し，以下に 15 の servent が接続している．それぞれの 16 の servent からマスターノードは約 4kB のデータを一つのメッセージで受け取っている．これは，30 分間で合計 237kB になり，マスターノードにとっては合計 3MB となる．これは，論理的な 100BASE-TX の能力の限界の 1%になる．CPU の利用をみると，5 日間実行しておいたところ，CPU 時間を 22 秒利用していた．これは，システムモニタが全体の 0.006%を利用したということになる．

6. ま と め

本研究では PC クラスタ内における階層的構造をとり耐障害性の仕組みを備えたシステムを提案した．実際のアプリケーションの構築により本システムが有効であることが確認できた．クラスタのシステムモニタ等のアプリケーションが有効に動作している．

マスターシステムへの依存性を除くと，提案システムは非常に耐障害性が強い．マスターノードへの依存性はクラスタシステムの構造上必要なものであり，それを利用することにより効率の良いネットワークが構築できると考える．

今後は提案システムをハイパークラスタおよび Grid 環境に適用させ検討を行う．

謝辞 本研究は文部科学省学術フロンティア推進事業により支援されている．

参 考 文 献

1) Freenet Project: *Freenet*, http://freenetproject.org.
2) Wego Systems, Inc.: *Gnutella*, http://gnutella.wego.com (1999).
3) Napster: *Napster*, http://napster.com (2001).
4) Condor Team: *Condor Project*, http://www.cs.wisc.edu/condor/.
5) Microsoft Corporation: *Microsoft Network OpenNET File Sharing Protocol*, ftp://ftp.microsoft.com/ developer/drg/CIFS/corep.txt (1988).
6) Sharp, R.: Just what is SMB, http://anu.samba.org/ cifs/docs/what-is-smb.html (1999).
7) Oikarinen, J.: RFC1459: Internet Relay Chat Protocol, http://www.ietf.org/ rfc/rfc1459.txt (1993).
8) Stoica, I., Morris, R., Kaashoek, F. and Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, *SIGCOMM '01, August 27-31, 2001, San Diego, California, USA* (2001).
9) Rodriguez, P., Spanner, C. and Biersack, E. W.: Analysis of Web Caching Architectures: Hierarchical and Distributed Caching, *IEEE/ACM Transactions on Networking*, Vol.9, No. 4, pp. 404–418 (2001).

出典：第９回「ハイパフォーマンスコンピューティングとアーキテクチャの評価に関する北海道ワークショップ (HOKKE-2002)」
情報処理学会研究報告 2002-ARC-147 2002-HPC-89, Vol. 2002, No. 22, pp. 55–61 (2002)