P2P における分散ハッシュテーブルを用いた静的負荷分散方式

塩谷 康夫[†]　片山 薫[‡]　石川 博[‡]

†‡東京都立大学大学院工学研究科 〒192-0397 東京都八王子市南大沢 1-1

‡首都大学東京システムデザイン学部 〒192-0397 東京都八王子市南大沢 1-1

E-mail: †shioya@hikendbs.eei.metro-u.ac.jp, ‡{katayama,ishikawa}@eei.metro-u.ac.jp

あらまし 分散コンピューティングにおいて, 負荷を均一に分散させると, 各ノードの処理時間にばらつきが生じ, 全体の処理時間が最も負荷処理速度の遅いノードの処理時間に依存してしまう．それゆえ，各ノードの能力に応じて負荷を分散させれば，負荷処理時間の均一化により，全体としてより効率的に処理を実行できる．そのために必要なノードの処理能力を示す指標としてランクを転送遅延の影響を加味して定義し，隣接するノードのランクから決定する．本研究は，分散ハッシュテーブルを用いることで，各ノードに隣接ノードの処理能力の情報を持たせ，サーバを必要とせずにランクの計算を行い，負荷分散を行える方式を提案する．また，大規模ネットワークにおいて，多数の負荷があり，負荷分散により処理した場合にシステム全体の効率についてのシミュレーションを行い，負荷分散処理の有効性を確認した．

キーワード P2P ネットワーク，分散コンピューティング，負荷平衡

A static load distribution scheme using a distributed hash table in P2P

Yasuo SHIOYA[†]　Kaoru KATAYAMA[‡]　and　Hiroshi ISHIKAWA[‡]

†‡Graduate School of Engineering, Tokyo Metropolitan University, 1-1 Minami-Osawa, Hachioji-shi Tokyo,192-0397

‡Tokyo Metropolitan University, System Design, 1-1 Minami-Osawa, Hachioji-shi Tokyo,192-0397

E-mail: †shioya@hikendbs.eei.metro-u.ac.jp, ‡{katayama,ishikawa}@eei.metro-u.ac.jp

Abstract In distributed computing, when a load is distributed uniformly, the processing time varies with nodes. The whole processing time is dependent on the slowest node. If the load is distributed according to the capability of each node, we can execute more efficient total processing by equalizing the processing time on each node. Considering the influence of transmission delays, we define a “rank” as a metric which indicates the throughput of each node, and determine the “rank” of each node from “ranks” of adjacent nodes. This paper proposes a load distribution scheme that gives information on the processing performance of adjacent nodes to each node by using a distributed hash table and calculates “rank” without using the server. Moreover, when there are a lot of loads, and they are processed by load balancing in large scale networks, the efficiency of system is simulated, and the effectiveness of the load balancing processing is confirmed.

Keyword P2P Network, Distributed Computing, Load Balance

1. はじめに

ネットワークを介して複数のコンピュータを結ぶことで，大規模な計算を高速に行う Grid computing が注目されている．P2P（peer to peer）による分散コンピューティングもその例である．

P2P による分散コンピューティングにおいて，隣接するノードに処理を行わせる時，単純に負荷を均一に分散させるよりも，各ノードの能力に応じて分散させれば，全体としてより効率的に処理を実行できると期待できる．その際，ネットワークの各ノードの転送速度を考慮して，負荷を分散させれば，より効果的に処理が行えると期待できる．

本研究の負荷分散方式によって分散コンピューティングを行う目的は，ネットワークを構成する各ノードのユーザが，大規模な計算を安価な設備で，より効率的に，高速に処理が行えるようにすることである．この負荷分散方式の適用により，効率的な処理が行える例として，ネットワークの各ユーザが，データマイニングを行う場合や，大規模な計算（研究データの処理等）を処理する場合が挙げられる．

本研究は，転送する際に生じるネットワークの転送遅延の影響を加味し，各ノードの処理時間が均一にな

るように，負荷を分散させるための基準となるノードのランク付けの方式を提案する．また，このランク付けを用いるためのネットワークの構成方法も提案する．

文献[1]では，このランク計算を superpeer（peer よりも処理能力と安定性があるノード）を用いて行った．しかし，この方式だと，サーバ，クライアント型のネットワークになってしまい P2P ネットワーク上で行う利点が少なくなってしまう．そこで，本研究は分散ハッシュテーブルを用いることで，各ノードをオーバレイネットワーク上で結び，各ノードでランク計算を行うことによって superpeer を必要とせずに負荷分散を行えるシステムとした．

以降，第 2 章で関連研究について述べ，第 3 章では本研究で想定するネットワーク構造について述べる．第 4～6 章で,負荷分散の基準となるノードのランクを隣接するノードのランクから決定する方式について説明する．第 7 章で，大規模ネットワークで多数の負荷処理がある場合の負荷分散により処理した場合にシステム全体の効率についてのシミュレーションを行う．第 8 章でまとめを述べる．

2. 関連研究

2.1. Grid computing

Grid computing は，ネットワークに接続された世界中のコンピュータ資源を活用して，高性能なコンピュータの代わりに，大規模な計算を高速に処理する技術である．具体的な活用の例として，SETI@home[2]がある．これは，インターネットに接続されたコンピュータを活用することで，電波望遠鏡によって受信した，宇宙からの電波の解析を行い，地球外生命体の発見を目的とするプロジェクトである．

また，医薬分野のたんぱく質の折り畳み予測[3]や数学分野の素数発見[4]等にも用いられている．

2.2. 分散ハッシュテーブル（distributed hash table）

分散ハッシュテーブルは主にP2P上においてサーバを必要としないでデータの検索を行うシステムを構築するのに用いられる．その仕組みは，アイテムとノード（peer）のそれぞれにハッシュ値を割り当て，アイテムのハッシュ値に最も近いハッシュ値を持つノードにアイテムを格納する．検索する際には，欲しいアイテムのハッシュ値を計算し，近いハッシュ値を持つノードに順々に問い合わせることで検索行うシステムである．分散ハッシュテーブルを実現する手法として Chord[5]，CAN[6]，Kademlia[7]，Pastry[8]等がある．

本研究は，分散ハッシュテーブルとして一般的な Chord を用いてネットワークを構成し，各ノードのハッシュ値とノードの処理能力と転送能力を対応させることで用いている．

2.3. SAN（Storage Area Network）による動的負荷分散機構

SAN を適用した PC クラスタによる並列データマイニングを行う合田らの研究[9]がある．これは，SAN という専用のネットワーク上で，動的負荷分散機構を設計し，HPA（Hash Partitioned Apriori）によってアソシエーションルールの抽出を行うものである．その利点は，構成ノードを動的に投入することにより，CPU パワーを増加させることで，CPU バウンドを抑制している点にある．また，データが存在しているストレージデバイスからデータを分割して，未使用のストレージデバイスにコピーすることにより，並列にアクセスを行うことを可能にし，ディスク I/O の帯域を拡張することで，I/O バウンド時の性能改善を図っている．

本研究は，SAN という専用のネットワークでは無く，LAN もしくは一般的な回線を想定しており，データ転送速度が遅いという点で異なる．また，構成ノードの CPU パワーを，負荷分散を開始するノードが任意に制御することができないという点で異なる．

3. ネットワークの構成

文献[1]では，superpeer を用いたサーバ，クライアント型のネットワークを想定していたが，この方式だと P2P ネットワーク上で行う利点が少なくなる．そこで，本研究におけるネットワーク構造は，図 1 に示すように，各ノード（peer）が Chord に基づきオーバレイネットワークを構成する．また，実際に負荷分散処理を行う場合には，ネットワークを構成している各ノードは，Chord によって自身が持つ指示先のノード（隣接ノード）の情報を利用して負荷処理のネットワークを構成した後，処理を行う．この負荷処理のネットワークにおいて負荷処理の実行単位を subpeer と表す．この時，subpeer は peer に内包された負荷処理の実行単位となる．

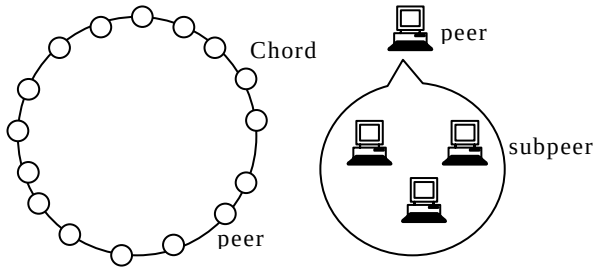


図 1　ネットワークの構成

3.1. Chord

分散ハッシュテーブルの利点は，各ノードに負荷を割り当てるので負荷分散の効果があること，大規模ネットワークに適用できること，ノードの頻繁な参加と

離脱に対応できることがあげられる．この特性から，P2P に用いるのに適している．

本方式では，分散ハッシュテーブルの 1 つである Chord を用いてネットワークを構成する．この際，Chord によって各ノードが持つ隣接ノードの情報に加えて，表 1 に示すような隣接ノードの subpeer の処理能力と転送能力の情報をハッシュ値に対応させて各ノードに持たせている．

3.2. peer

peer は Chord によってオーバレイネットワーク上で結ばれている．peer は，負荷分散処理の実行単位である subpeer を内包している．peer は,複数の subpeer を持ち，一対多の関係になっている．内包する subpeer の数は，peer に到達した負荷を処理する際，その負荷が処理待ちを生じないでできる様に随時変動している．この様に変動させるのは，peer において処理に使用されない余剰な能力を少なくするためである．

また，文献 [1]とは異なり，peer が隣接する peer の subpeer の処理能力と転送能力の情報を持つため, 負荷分散に必要なランクの計算を行う役割を持つ．

3.3. subpeer

subpeer は, 負荷処理のネットワークにおいて負荷処理の実行単位を表わしている．その役割は，Chord ネットワーク上の複数のノードが異なるタスク（負荷）を同時に負荷分散による処理を行う時，その個々の処理が相互に影響しないで，独立に処理を行えるようにするという役割である．例で示すと，図 2（a）のようなネットワークがあり，ノード A，B から異なるタスクを与えられ，負荷処理を行うとする．この時，図 2（b）のような負荷処理のネットワークを構成できる．その際，subpeer の役割は，ノード D に関し，2 つの処理が独立に処理できるようにする役割である．

このことから, subpeer は元のノードの処理能力を分割した処理能力を持ち，異なるタスクごとに個々に割り当てられる．

3.4. 負荷処理のネットワークの構成

Chord ネットワーク上の各ノードが負荷分散を行う時，必要な負荷処理のネットワークの構成の手順を以下に示す．その例を図 3 に示す．この時，ネットワークの隣接関係は変化させずに構成することを想定する．

① 負荷処理を開始するノード（ルート）は，負荷分散の範囲（転送回数 P）を決定する．

② 隣接ノードに負荷処理を行うか問い合わせる．

③ 問い合わせを受けたノードのユーザが，依頼を受けるかどうか判断し返答する．

④ 依頼を受けたノードは，同様にその隣接ノードに負荷処理を行うか問い合わせる．また，同様に③に基づき隣接ノードが返答する．

⑤ ④を転送回数が P になるか，隣接ノードが無くなるまで繰り返す．

⑥ 末端の親のノードは, 子のノード(peer)の subpeer の情報により，負荷処理のネットワークにおける負荷実行能力を表すランクの計算を実行する．

⑦ 得られたランク情報をそのノードの親のノードに転送する．

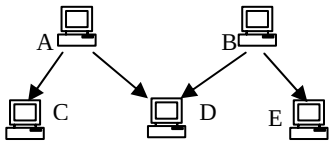
⑧ 中継のノードは転送されて来たランク情報と自身の subpeer の情報に基づき，ランクの計算を行い，親のノードにランク情報を転送する．

⑨ ⑧をルートに到達するまで繰り返す．

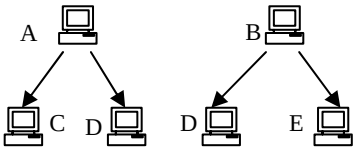
この時，隣接ノードに問い合わせる際，ルートのノ

表 1 本方式で付加したノードが持つ情報

ノード	ハッシュ値	subpeer の処理能力	転送能力
1	16543	1	3
2	16544	3	5
3	16546	4	9



（a） ネットワーク



（b） 負荷処理のネットワーク

図 2 subpeer の役割

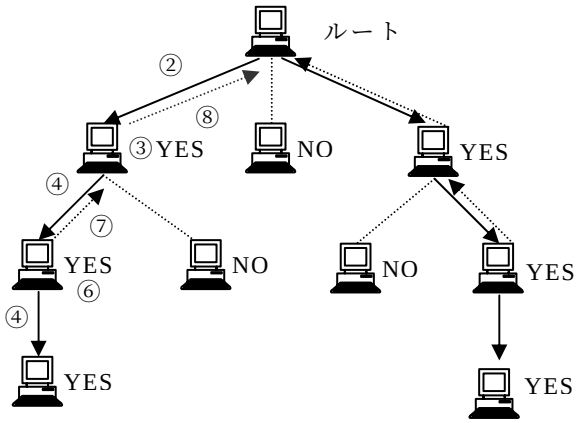


図 3 負荷処理のネットワークの構成

ードからそのノードに到達するのに必要な転送回数が増加するノードに対してのみ問い合わせを行う.

3.5. subpeer 情報の更新

文献[1]では, 各ノードはある一定期間(T)で subpeer の情報の更新を行う方式であったが, 本研究では, 各ノードはある一定タスク数（N）を処理するごとに, subpeer の情報に更新があった場合に自身を指しているノードにその情報を転送する方式を用いる. タスクに処理待ちが生じない様にし, subpeer の数を最小にするために行う. これにより, 各ノードの subpeer の数を必要最小限度にし, システム全体での負荷分散処理の効率を良くすることができるからである.

コンピュータの性能が変化した場合にも subpeer の処理能力が変化するので, その時点で更新を行う.

3.6. 処理の流れ

負荷処理のネットワークを構成した後の, 処理の流れをデータマイニングにおけるアソシエーションルールの抽出を例に使って以下で述べ, 図 4 に示す.

① ルートは, 隣接ノードのランク情報から負荷を分割し, 隣接ノードに転送する. 負荷転送と同時に, partition アルゴリズムを用いることにより, 割り当てられた負荷からローカルラージアイテムセット（L）の生成を行う. その際, 負荷は一定のデータサイズ（F）を基準に均等に分割して処理する.

② 中継のノードでも同様に, 隣接ノードのランク情報から負荷の分割をし, 隣接ノードに転送する. この時, ルートと同様な方式で割り当てられた負荷の処理を行う.

③ 末端のノードに到達するまで, ②を繰り返す. あるいは, 負荷のデータサイズが F＋ΔF 以下に収まった時点で終了する.

④ 末端のノードで, ルートと同様な方式で割り当てられた負荷の処理を行う.

⑤ L を負荷の転送経路を逆順にルートに転送する.

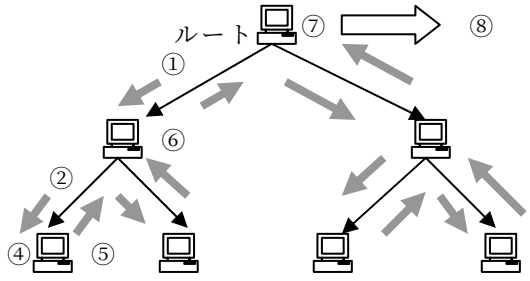


図 4　負荷分散の処理の流れ

⑥ 中継ノードで L を統合し, 転送経路を逆順にルートに転送する.

⑦ ルートは, L を統合する.

⑧ 統合結果より, アソシエーションルールを得る.

4. 処理能力の定量化

4.1. peer の処理能力の定量化

ある負荷処理を実行する時, 処理時間は, そのコンピュータの CPU の種類, 速度, メモリの容量, 動作クロック数, マザーボードの種類, HDD の速度等の複数の要因が互いに影響して決まる. そのため, コンピュータの処理能力を, コンピュータの構成要素から定めるのは困難である. そこで, 本研究においてコンピュータの処理能力を定量化する方法を説明する.

① ネットワークを構成する全てのノードに, 処理能力を 1 と定めるベンチマーク用データ（サイズ d）の処理時間 t_0 を設定し, 基準となる処理速度 v_d[kB/s]を（1）式と定める.

$$v_d = \frac{d}{t_0} \quad \text{[kB/s]} \quad (1)$$

② ネットワークを構成するノードは, ベンチマーク用のデータを処理し, 処理時間 t を測定する.

③ ノードのコンピュータの処理能力 a は, t_0 と t の比として（2）式で与える.

$$a = \frac{t_0}{t} \quad (2)$$

4.2. subpeer の処理能力

subpeer の処理能力を定めるには, 各ノードで負荷分散処理を効率的に行うために必要な subpeer の数を定めておく必要がある. それゆえ, subpeer の数はネットワークの状況に応じて変動させる必要がある. そこで, ネットワークの各ノードは, ある一定数（N）の異なる負荷分散処理のタスクを処理した時に subpeer の数の更新を行うとする. n 回目の更新による subpeer の数を sc(n)とおく. 以下で sc(n)の定め方を説明する.

① タスクに処理待ちが生じた場合

N のタスクを処理した間に, 同時に処理待ちの状態になった最大のタスク数を m とする. sc(n)は, (3) 式で与えられる.

$$sc(n) = sc(n-1) + m \quad (3)$$

② タスクに処理待ちが生じなかった場合

N のタスクを処理した間に, 実際に同時に使用した subpeer の数を u とする. sc(n)は, (4)式で与えられる.

$$sc(n) = u \quad (4)$$

ただし, ネットワークに初めてノードが参加した時

は，subpeer の数はある基準値 q の値を持つとする（sc(0)=q）.

subpeer の処理能力 sp は，コンピュータの処理能力 a を subpeer の数 sc(n)で分割したものと定義し，定義式を（5）式で与える．これは，あるノードが子のノードが持つコンピュータの最大限使用できる処理能力を定めることを意図している.

$$sp = \frac{1}{sc(n)} a \quad (5)$$

4.3. 転送速度の定量化

データの転送速度は，データのサイズをその転送時間で割ることによって得ることができる.したがって，あるノードにおいて親のノードからのデータ転送速度 v_f[kB/s]から，（1）式で定めた v_d を用いて，そのノード間の転送速度の定量化が行える．ノード間のデータ転送能力 ta は，（6）式で与える.

$$ta = \frac{v_f}{v_d} \quad (6)$$

5. 負荷実行能力を表すランクの定義式

隣接ノードに負荷を分散させるにあたって，隣接ノードの子のノードの影響と，転送遅延の影響を考慮した負荷分散を行う必要がある．そのため，この様な影響を考慮したノードの負荷実行能力を表すランクを定義する.

負荷処理のネットワーク上のノード i を構成する subpeer の処理能力を sp_i とし，協調する隣接ノードの負荷実行能力も加味したノード i の処理能力を na_i とする.親のノード h とノード i 間のデータ転送能力を ta_{hi} とし，親のノード h からみた転送遅延を考慮したノード i の負荷実行能力を表すランクを r_{hi} とする．ノード i が指している隣接ノードの集合を S_i とおき，ノード i が負荷処理のネットワークにおいて，隣接ノードから指されているノード数を n_i とおく（だだし，ルートのノードについては，n_i=1）．ノードの処理能力 na_i の定義式を（7）式で与える．また，親のノード h から見た転送遅延を考慮したノード i の負荷実行能力を表すランク r_{hi} の定義式を（8）式で与える.

$$na_i = \frac{1}{n_i}(sp_i + \sum_{j \in S_i} r_{ij}) \quad (7)$$

$$r_{hi} = \frac{ta_{hi} \cdot na_i}{ta_{hi} + na_i} \quad (8)$$

（8）式から，ノード間の転送能力が十分に処理能力より大きければ，r_{hi}= na_i となり，ノード間の転送能力にノードの負荷実行能力が依存しない．また，逆にノード間の転送能力が十分に処理能力より小さければ，r_{hi} = 0 となり，負荷実行能力は 0 となる．ゆえに，実際の状況に即した定義を表している.

6. 負荷分割の方式

負荷を単純に処理すると，負荷のデータ量の増加に対し処理時間は一般に線形に増加しない．すなわち，負荷のデータ量とその処理時間は比例関係にはなっていない．そこで，負荷の処理方式を工夫することで，負荷のデータ量と処理時間の間に比例関係を持たせる.

本研究では，元の負荷をある一定のサイズ（F+ΔF）単位に負荷を均等に分割し，分割した負荷を順々に処理する方式を使うことで，負荷のデータ量の増加に対し処理時間を線形に増加させることを可能にした．F は任意のデータサイズに定めることができるが，分割しないで処理した方が効率的に処理を行える負荷の最大データサイズに定める．また，F+ΔF は分割しないで処理した時，処理時間が F の処理時間の 2 倍となるデータサイズとする．これからΔF を定められる．この様に定めることで，各ノードにおける負荷を分割し処理する時の基準値による処理時間の誤差を 2 倍以内に収めることを意図している.

この処理方法を，例を使って説明する．1F の負荷の処理時間が a とする.図 5 のような 5F の負荷を処理する場合を考える．まず，1F 単位に 5 個に負荷を分割する.次に，分割した負荷を順々に処理する.その結果，処理時間は 5a となり，1F の負荷の処理時間の 5 倍となる．よって，負荷のデータ量と処理時間の間に比例関係を持たせられる.

したがって，各ノードの負荷の処理時間の均一化は，各ノードのランクに応じて負荷を比例配分で割り当てることで可能となる.

分割する負荷のサイズを s とおく．s が F+ΔF よりも大きければ負荷を分割し，子ノードへ転送する．この時，ノード i とその子ノード k に割り当てる負荷は，（9）式で与える.

ノード i：$\frac{sp_i}{sp_i + \sum_{j \in S_i} r_{ij}} s$　子ノード k：$\frac{r_{ik}}{sp_i + \sum_{j \in S_i} r_{ij}} s$ (9)

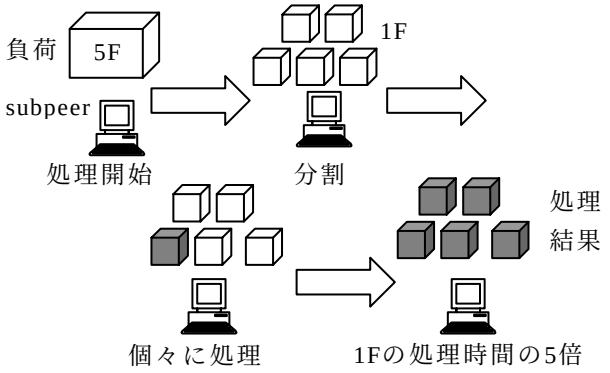
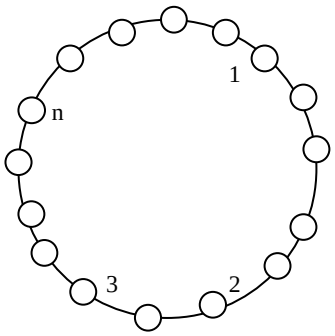


図 5　負荷の処理方式



①負荷がランダムに発生（n 個発生）
②負荷処理のネットワークを形成して処理
③最後の負荷を処理するまでの処理時間を測定

図 6 シミュレーション方法

表 2　シミュレーションの条件

ネットワークのノード数	2^{14}=16384
コンピュータの処理能力	1～10
転送能力	1～10
subpeer の数（初期状態）	1
subpeer の更新タスク数	5
ノードがあるタスクの負荷分散処理の依頼を受ける確率	0.9
負荷 w	14400I
発生する負荷の大きさ	w～10w
負荷の発生間隔	1～100[s]

7. シミュレーション

文献[1]で，1 個の負荷（タスク）の場合について，本方式に基づいて負荷を分散処理させると，確かに各ノードの負荷の処理時間が均一化されることで，処理の効率化がなされることが確かめられた．しかし，一般的な状況（大規模ネットワークで多数の負荷がある場合）において，本方式を用いることで，負荷の処理効率が良くなることを確かめていない．そこで，一般的な状況での本提案方式の有効性を確かめるためのシミュレーションを行った．

7.1. シミュレーション方法

図 6 のように，Chord に基づいたネットワークにおいて，多数の負荷がランダムに順次発生し処理した時の，システム全体の処理効率についてのシミュレーションを行った．この時，ネットワーク上で発生する負荷の個数を 100000 個まで変化させ,その時のシステム全体の処理効率を測定した．その際，上述のように，個々の負荷それぞれについて，ツリー構造の負荷分散のネットワークを構成して処理を行うものとした．

この時の処理効率は，負荷分散を行わずに個々のノードで処理した場合の処理時間を基準に，負荷分散による処理との比較により定める．処理効率の式を（10）式で与える．

$$x = 1 - \frac{y - t}{z - t} \quad （10）式$$

ここで，x がシステム全体の処理効率，y が負荷分散による処理時間，z が負荷分散を行わずに個々のノードで処理した場合の処理時間，t が最後の負荷の発生時間を表す．（10）式の様に定義することで，最後の負荷が発生してから全ての負荷の処理が終わるまでの処理時間を比較することで正味のシステム全体の処理効率を表している．

また，このシミュレーションにおいて用いた負荷分散方式は，本方式と均等に負荷を分散した場合の 2 方式で行った．各方式について，subpeer を使用した場合と subpeer を使用しなかった（ノードの処理能力を分割しない）場合の 2 種類について行った．

シミュレーションを行う時の，ネットワークの基本条件を示したのが表 2 である．ここで，I は単位負荷でランク 1 のノードが処理に 1[s]かかる負荷とする．また,シミュレーションの基準として用いる負荷 w は，ランク 1 のノードが処理に 4 時間かかる負荷とした．本シミュレーションでは，負荷を極小に分割しても効率的に処理できるものとし, F, ΔF は極小な値とした．

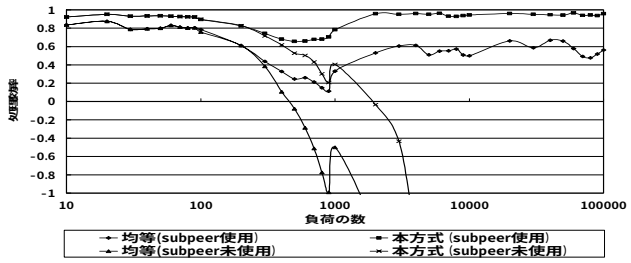
7.2. シミュレーション

（1）　転送範囲の違いの影響

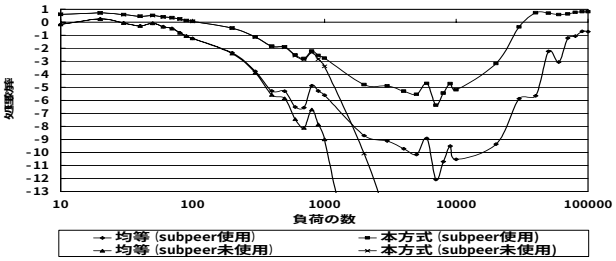
負荷分散を行う時の最大転送範囲の違いの影響についてのシミュレーションを行った．転送範囲として転送回数（ポップ数）が 4 以下，転送回数 14 以下の 2 つの場合に行った．シミュレーション結果を示したのが図 7 である．

図 7(a)の負荷の転送範囲がネットワークの規模に対して狭い範囲で分散させた場合に，負荷の数が少ない範囲では，subpeer を使用しなくても，均等，本方式のどちらの分散方式でも負荷分散を行わない場合よりも効率的に処理が行えるが，負荷の数が多くなると 2 つの方式とも負荷分散を行わない場合よりも効率的に処理が行えない．一方，subpeer を用いて負荷分散処理させた場合には，均等，本方式のどちらの分散方式でも負荷分散を行わない場合よりも効率的に処理が行える．ゆえに, subpeer を用いて負荷分散処理を行う有効性が示せた. また, 本方式によって分散させた場合の方が，均等に負荷を分散処理させた場合よりも，転送回数が 4 以下の場合には平衡状態で約 0.3，転送回数が 14 以下の場合には平衡状態で約 2，処理効率が良くなっている．よって，各ノードの能力に基づき負荷を分散させる有効性を示せた．

図 7（b）から転送範囲を広くすると，subpeer を使用した本方式による負荷分散の場合でも，処理効率が最小値-6.4 で 0 以下になり，負荷分散を行わずに処理した方が効率的に処理を行える範囲（負荷の数で 100～30000 の範囲）がある．この要因は，負荷分散処理のネットワークとしてツリー構造のネットワークを形成しているので，負荷を転送する中継ノードで転送待

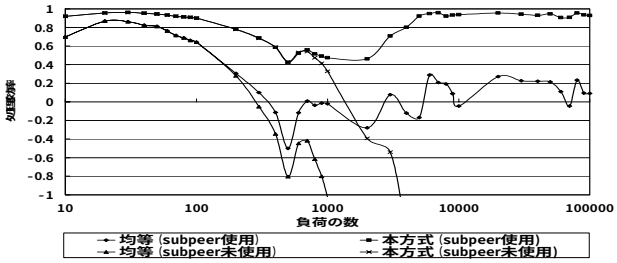


（a）　転送回数 4 以下

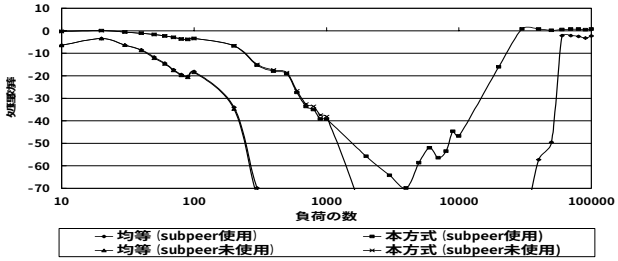


（b）　転送回数 14 以下

図 7　転送範囲の違いの影響



（a）　転送能力が処理能力の約 10 倍の場合



（b）　処理能力が転送能力の約 10 倍の場合

図 8　ノードの能力の影響

ちが生じると以降の転送先のノードで処理時間が大きくなるため，全体として処理時間が大きくなり処理効率が悪くなるからである．これは，各ノードに伝達される subpeer の更新情報と，実際にノードで subpeer の更新に時間差があることに起因している.すなわち，subpeer の処理能力を動的に変動させずに,静的に変動させているため，前の subpeer の状態による負荷処理が少なくとも１つ終わらないと，新しい subpeer の状態に更新することができないからである．また，負荷の数が大きくなると処理効率が良くなっているのは，subpeer の数が最適化されるので,処理待ちがほぼ生じなくなるため，処理効率が良くなるのである．

（2）　ノードの能力の影響

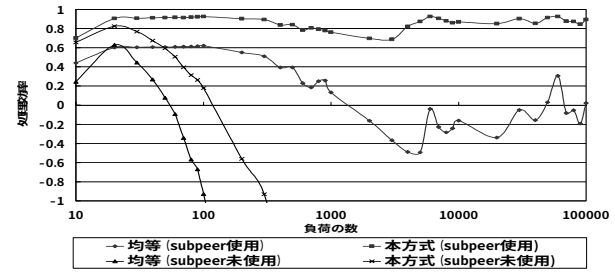
各ノードの転送能力を 10～100 に設定した場合（転送能力がコンピュータの処理能力の約 10 倍の場合)と各ノードのコンピュータの処理能力を 10～100 に設定した場合（コンピュータの処理能力が転送能力の約 10 倍の場合）についてシミュレーションを行った．この時の転送範囲は 14 以下で,他の条件は表 2 と同じ条件に設定した．シミュレーション結果を図 8 に示す．

図 8（a）から，本方式の場合（subpeer を用いた場合）では，ノードの転送能力がコンピュータの処理能力よりも相対的に大きい程，負荷を転送した時の転送遅延の影響が少なくなり効率的に処理が行える．その結果，subpeer の数が最適化された状態では，最も効率の良い値（0.92）になる．また，この条件下では，処理効率が最小値 0.42 で 0 以下にならずに処理が行える.逆に，図 8（b）から，ノードの転送能力がコンピュータの処理能力よりも相対的に小さい程，負荷を転送した時の転送遅延の影響が大きくなり処理効率が悪くなる．また，この時 subpeer の数の最適化にも，時間が多くかかり，最適化されるまでの負荷の数も増大するため，処理効率が 0 以下の範囲が表 2 の条件下の場合よりも広くなり（負荷の数で 10～40000 の範囲），処理効率の最小値も-70.0 と最も悪くなる．

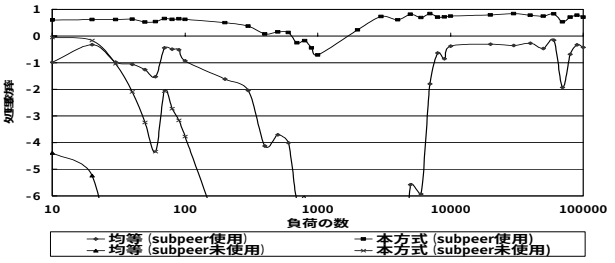
以上から，コンピュータの処理速度よりも転送速度が速い程，本方式は効率的に処理が行える負荷分散方式だと言える．すなわち，実際のネットワークがより速い回線で結ばれると効率的に行える方式と言える．

（3）　subpeer の数の初期値と下限値の影響

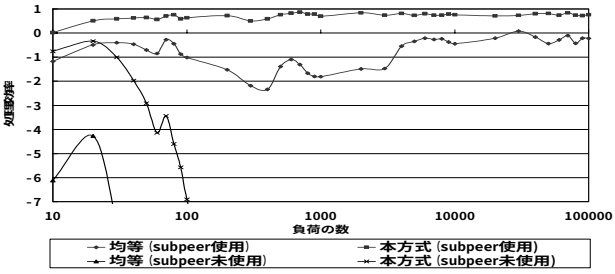
転送回数を制限しないと負荷分散を行うよりも，行わない時の方が処理効率に良い状態があるのでは，一般的な状況で用いることができるシステムといえない.そこで，負荷分散を行わない時よりも処理効率が悪くなる事を防ぐ必要がある．そのためには，初期状態からある程度 subpeer を持たせ，subpeer の数がある一定値（下限値）以下にしないことで可能であると考えられる．そこで，転送範囲が 14 以下の時に，表 2 の条件について subpeer の数の初期値を 5 とし，subpeer の数の下限値を 5 に設定して,シミュレーションを行った．また，各ノードのコンピュータの処理能力を転送能力の約 10 倍に設定した場合についても同様な条件で行った．この時のシミュレーション結果を図 9 に示す．各ノードのコンピュータの処理能力を転送能力の約 10 倍に設定した場合（図 9（b））では，処理効率が 0 以下（最小値-0.71）で負荷分散を行わない時よりも悪くなる場合がある.そこで,条件を再度変更し,subpeer の数の初期値を 7，下限値を 7 に設定してシミュレーションを行った．シミュレーション結果を図 9（c）に示す．図 9（a),（c）では，システム全体の処理効率



（a）　表 2 の条件の場合



（b）　処理能力が転送能力の約 10 倍の場合



（c）　処理能力が転送能力の約 10 倍で，subpeer の数の初期値を 7，下限値を 7 に設定した場合

図 9　subpeer の数の初期値と下限値の違いの影響

は 0 以下（処理効率の最小値は，図 9（a）で 0.69，図 9（c）で 0.02）にならず，負荷分散を行わない時よりも処理効率が悪くなることを防げている．よって，図 7（a），図 9（a），（c）から，各ノードのコンピュータの処理能力，転送能力，転送範囲に依らず，subpeer の数の初期値と下限値を適切に設定すれば，本方式の負荷分散により，負荷を分散しないで処理するよりも効率的に行える．また，転送能力がコンピュータの処理能力の相対的に小さい程, subpeer の数の初期値と下限値は大きく取る必要がある．

しかしながら，subpeer の数を最初から多く取りすぎると，ネットワーク上で発生する負荷の数が少ない場合には，各ノードで使用されていない余剰な処理能力が増大し，結果として負荷分散を使用しないで処理した方が効率的に処理を行える場合もある．

8. まとめ

本研究では，分散ハッシュテーブルを用いることで，サーバを必要とせずに，負荷分散処理を行うネットワークシステムの方式を示した．また，協調する隣接ノードの負荷実行能力と負荷の転送遅延を考慮して，各ノードの処理能力に応じて負荷を分散させるための，負荷実行能力のランク付けの方式を提案した．

シミュレーション結果より，ネットワークに多数の負荷がある場合に本システムを用いることで効率的に処理が行えることを示せた．この時，負荷の転送範囲，各ノードの処理能力や転送能力に依存せずに，負荷分散による処理を行わない場合よりも効率的に行えることを示した．

本研究の今後の課題を以下に挙げる．

- シミュレーション上では，本提案方式の有効性を確認したが，実際の大規模ネットワークでも，本提案方式が有効であることを確認する．
- システム全体の処理効率に影響する subpeer の数の初期値と下限値の定め方を検討する．
- ネットワークの状態や各ノードの状態が動的に変化した場合における負荷分散方式（動的な負荷分散方式）を検討する．

謝辞

この研究の一部は科学研究費補助金（特定領域研究，研究課題番号：16016273）による．

文　献

[1] 塩谷康夫，太田学，片山薫，石川博:P2P による転送遅延を考慮した静的負荷分散方式，電子情報通信学会，第 16 回データ工学ワークショップ DEWS2005, 2005. 3

[2] SETI@home,http://setiathome.ssl.berkeley.edu/

[3] Rosetta@home, http://boinc.bakerlab.org/rosetta/

[4] GIMPS, http://www.mersenne.org/prime.htm

[5] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, ACM SIGCOMM 2001, San Deigo, CA, August 2001, pp. 149-160.

[6] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker, A Scalable Content-Addressable Network, In Proceedings of SIGCOMM'01, pp.161-172, August 2001.

[7] Petar Maymounkov, and David Mazieres, Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In Proceedings of IPTPS02, Cambridge, USA, March 2002.

[8] Antony Rowstron, and Peter Druschel, Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Proceedings of the 18th IFIP/ACM Int'l Conf. on Distributed Systems Platforms

[9] 合田和生，田村孝之，小口正人，喜連川優: SAN 結合 PC クラスタ上での動的資源割り当てを用いた並列データマイニング処理，電子情報通信学会第 13 回データ工学ワークショップ DEWS2002, 2002. 3