社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－DPS－131 (13)
2007／6／7

管理情報の分散化と自律的協調による動的負荷分散システム

氏家武志† 菅谷至寛† 阿曽弘具†

高性能な計算機が広く普及しネットワークで接続されているが，パーソナルコンピュータの多くは遊休状態にあり，その計算資源は十分に利用されていないのが現状である．その一方で，膨大な計算タスクを抱えているが十分な計算資源を得られないという状況も存在し，遊休状態にある計算資源を集め，有効に活用したいという需要は大きい．そのため負荷分散システムの重要性がますます高まっているが，計算機の資源の管理に中央集権的な管理用サーバを用いると，その管理用サーバに障害が生じた場合にシステム全体が停止する恐れがある．本研究では，分散システムに参加する計算機同士が自律的に協調し，管理用の情報を分散共有して動的負荷分散を実現するシステムを提案する．シミュレーション実験の結果，本手法を用いることで管理用情報の分散共有が可能であることが示された．

A Dynamic Load Balancing System using Autonomous Cooperation of Hosts and Decentralization of Management of Hosts

TAKESHI UJIIE,† YOSHIHIRO SUGAYA† and HIROTOMO ASO†

High performance computers and broadband networking technologies are widely spread. However, many personal computers are idle most of the time, those computational resources have not been exploited. Therefore research on load balancing systems becomes more and more important to utilize these resources efficiently. In case of using centralized control server for resources management, there is a risk that a failure on the server halts the whole system. In this paper, we propose a dynamic load balancing system which utilize autonomous cooperation of hosts and decentralize management information of hosts. Computer simulation indicate that the proposed method can efficiently share decentralization information of management of hosts.

1. はじめに

近年の計算機の性能向上には目を見張るものがある．パーソナルコンピュータにも，高速なプロセッサ及び大容量のメモリを搭載したものが普及しており，さらには広帯域のネットワークによってそれらが相互に接続されている点を見逃してはならない．しかし，このような豊富な計算資源があるにもかかわらず，プロセッサの能力を限界まで使用している場合は少なく，むしろ遊休状態にある計算機が多数であると考えられる．このように，計算資源を決して有効に利用できてはいないのが現状であり，計算タスクを適切に分散し計算資源を有効利用するためのシステムが重要になっている．

グリッド・コンピューティングは注目を集めている分散処理の形態で，利用者が必要とする資源を世界中からその時々で集め，地理的な障壁を意識せずあたかも仮想的な巨大計算機のように使用するためのインフラストラクチャである[1]．資源情報の管理やタスクのスケジューリングは専用のサーバを用いて行うことが一般的である．これらのサーバに負荷が集中して障害が生じシステムがダウンする可能性がある．その対策として高性能なサーバの導入が考えられるが，コストが高い．

BOINC[2] はボランティア・コンピューティングのためのプラットフォームで，参加者は自分の計算機の遊休時間を提供し，管理用サーバから割り当てられたデータの分析を行う．BOINC を用いているプロジェクトには，SETI@home[3] などがある．このプラットフォームを用いてプロジェクトを立ち上げれば，他の計算機の資源を利用することが可能だが，参加者が互いの遊休資源を必要に応じて融通しあって利用することは困難である．

本研究では，計算資源の性能・負荷情報の管理に特定のサーバを用いず，参加する計算機同士が自律的に協調し動的負荷分散を実現するための，管理情報の分散共有手法を提案する．管理用に特定のサーバを設置

† 東北大学 大学院 工学研究科 電気・通信工学専攻
Department of Electrical and Communication Engineering, Graduate School of Engineering, Tohoku University

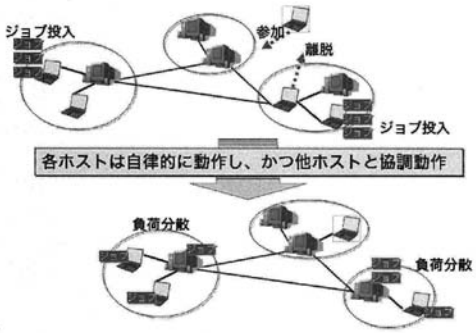

図 1 ホストの自律的協調による分散システムの概念図

しないことで，サーバの障害によってシステム全体が停止することを避けるとともに，高性能なサーバの導入のためのコストも抑えることが可能である．

本稿の構成は以下の通りである．2 章ではホストの自律的協調による動的負荷分散システムについて述べる．3 章で DHT を用いた管理情報の分散共有手法を提案する．4 章では実験とその結果を述べ，最後に 5 章でまとめと今後の課題を述べる．

2. 負荷分散システム

我々はこれまで，分散システムに参加する計算機 (ホスト) 同士の自律的協調によって動的負荷分散を実現するシステム[4] を提案している．各ホストの情報を一元的に管理する中央管理サーバ (ブローカ) を設置しないブローカレス型のシステムであるという特徴を持つ．ネットワーク的に近い位置に存在するホスト同士でクラスタを構成し，各ホストの情報をクラスタごとに管理することで，ホストの頻繁な参加・脱退にも対応可能である．まず，分散システムに参加するホストによるオーバーレイネットワークの構築法について述べ，そのネットワーク上で動作する負荷分散アルゴリズムについて述べる．

2.1 オーバーレイネットワークの構築

分散システムの参加ホストによるオーバーレイネットワークを構築する．オーバーレイネットワークとは，物理的なネットワークとは異なり，各ホストで動作する分散システムのアプリケーション同士が構築する論理的なネットワークである．したがって，何の規則もなくコネクションを張ってしまうとオーバーレイネットワークには物理ネットワークの構成は考慮されないこととなり，通信遅延の増大やネットワークのトラフィック増加などを招く恐れがある．そこで，参加ホストを物理ネットワーク上での距離に従ってクラスタリング (グループ化) し，物理ネットワーク上での距離が近いホスト同士が同一クラスタのメンバーとなるようにする．距離の指標には通信遅延を用いる．各クラスタでは，クラスタのメンバー中から 1 台をマスタホストに選定し，クラスタ内のホストの負荷情報の管理を行う．このマスタホストの役割は常に特定の 1 台が担うのではなく，ホストの新規参加・離脱，所属クラスタの変更に伴ってマスタの役割の委譲を可能としている．そのため，すべてのホストがマスタの役割を担う可能性がある．各クラスタのマスタホストは，他クラスタのマスタホストとの間でコネクションを張る．このとき接続数に制限を設け，限度を越えた場合は最も距離の遠いクラスタのマスタホストとの接続から切断していくことにより，距離が近いクラスタのマスタホスト同士が隣接するようにする．他クラスタに属するホストの負荷情報の取得は，それぞれのクラスタのマスタホストを介して行われる．

	ホスト1	ホスト2	ホスト3	和S_d
ホスト1	0	1	10	11
ホスト2	1	0	10	11
ホスト3	10	10	0	20

クラス間分散最大で分割

11, 11, 20

図 2 クラスタ再構築における S_d の計算例

2.1.1 参加手順

分散システムへの新規参加手順は以下のようになる．ただし，新規参加を希望するホストは，何らかの手段を用いて，既に分散システムに参加しているホストを 1 つ知っていることが前提となる．

(1) 事前に知っているホストに対して，そのホストの所属クラスタのマスタホストを問い合わせる．
(2) 得られたマスタホストとの距離の計測を行う．
(3) さらに，隣接するクラスタのマスタホストを問い合わせる．
(4) 得られた別のクラスタのマスタホストとの距離の計測を行い，距離が小さいクラスタを記憶する．
(5) 距離が小さいマスタホストが見つからなくなるまで，3,4 の作業を繰り返す．
(6) 最終的に距離が最小のマスタホストの所属クラスタを初期所属クラスタと定める．
(7) 初期所属クラスタ決定後，そのクラスタに属する全ホストとの距離を計測し，その和 S_d を全遅延指標として保持する．既にそのクラスタに属していたホストも，自身の全遅延指標に新規参加ホストとの距離を加える．

2.1.2 再構築手順

ホストの参加・離脱が頻繁に発生することを想定した場合，定期的にクラスタの再構築を行う必要がある．

(1) 各ホストは，自身の全遅延指標の値 S_d をマスタホストに送信する．
(2) 自クラスタの各所属ホストから S_d を受け取ったマスタホストは，それらを適当な値で 2 分割し，判別分析により適切な分割点を求める．クラスタ内に距離が遠いホストが存在するとクラ

ス間分散が大きな値をとることになる．

(3) 様々な分割点でクラス間分散を求め，その最大値がある閾値をこえた場合，S_d が最大のホスト H_c を所属クラスタ変更候補とし，そのことを H_c に通知する．あわせて，隣接するクラスタのマスタホストの情報も H_c に通知する．

(4) ホスト H_c は，隣接する各クラスタの所属ホストの情報を得て距離を計測し，その情報を用いた判別分析法の適用をその隣接する各クラスタのマスタホストに依頼する．

(5) 隣接する各クラスタに H_c を移動したと仮定して計算したクラス間分散がある閾値をこえないものがある場合，クラス間分散が最小となる隣接クラスタに移動する．すべての隣接クラスタで閾値をこえた場合，新たなクラスタを生成し，自身がマスタホストとなる．

2.2 負荷分散アルゴリズム

構築されたオーバーレイネットワークを利用して自律的負荷分散を行う．タスクを適切に配置し負荷を分散するためには，分散システムに参加しているホストの中から処理するタスクに見合った性能を持ち，負荷の低いホストを検索して利用しなければならない．この検索のためにオーバーレイネットワークを利用する．クラスタを跨いだ検索要求については，マスタホスト同士の協調により処理される．また，距離が近いホストのクラスタができているため，タスク移動先決定の際にその情報を利用することで，余分な通信コストがかからないようにすることができる．

タスクの移動にはタスクマイグレーションを用いる．タスクマイグレーションとは，実行中のタスクを一時停止して実行状態をデータ化し，他のホストに転送した後に復元してタスクの実行を再開する仕組みである．本研究では，タスクマイグレーションを実現する環境としてモバイルエージェント環境 Aglets[5] を用いる．

2.2.1 タスク移動要求の発生

本研究で対象とするものは非均質環境であり，プロセッサの性能がホストごとに異なる．そのため，各ホストの負荷情報として，分散システムへの参加時に共通の小さなプロセスを実行しその実行時間を求め，それと Load Average を併せて用いる．

各ホストでは監視エージェントを動作させ，動的にオーバーレイネットワークの構築を行う．すなわち，定期的に自ホストの負荷取得及び実行中タスクの通信遅延を監視し，これに基づいて自ホストのタスクを移動すべきか判断する．タスク移動要求が生じる契機には次の 2 つがある．

- 取得された負荷値がある閾値をこえた場合
- 実行中タスクが他ホストと行っている通信の遅延が閾値をこえた場合

通信遅延が閾値をこえた場合のタスク移動により，通信遅延が大きくなってタスク実行時間に影響が及ぶことを防ぐ．

2.2.2 移動するタスクの決定

移動要求が発生したホストでは，自ホストに複数存在するタスクの中から移動すべきタスクを決定しなければならない．この移動タスク決定法には Credit-Based Load Balancing[6] アルゴリズムを用いる．それぞれのタスクに対してホストとの親和力を決定する評価値を与えることで，親和力の最も小さいタスクを移動するべきタスクとして決定する．この評価値は，自ホスト内のタスクとの通信が多いと大きくなり，ネットワーク上での距離が遠いホストとの通信が多いと小さくなる値である．(詳細は[4] 参照)

2.2.3 タスク移動先ホストの決定

基本的にタスクの移動先ホストは，所属するクラスタ内から優先して選択する．これは，タスク移動時の通信コストを抑えるため，また他クラスタの低負荷ホストの検索にかかるコストを抑えるためである．移動先選択のため，自クラスタのマスタホストに負荷の低いホストの検索を要求する．要求を受けたマスタホストは，自クラスタに所属するホストから負荷の低いホストを選択し，要求元ホストに返答する．

所属クラスタ内に低負荷のホストが存在しない場合は，マスタホストを介して隣接するクラスタに低負荷ホストの検索を依頼する．また移動するタスクが他クラスタに所属するホストと通信を頻繁に行っている場合は，その通信相手が所属するクラスタに低負荷ホストのマスタホストに検索を依頼する．

3. 管理用情報の分散共有

2 章で紹介したシステムにおいては，ネットワーク的に近い位置にあるホスト同士をクラスタリングし，各クラスタのマスタホストがクラスタ内のホストの負荷情報を管理してタスクの配置に利用する．異なるクラスタに所属するホストの情報の取得は，それぞれのクラスタのマスタホスト同士の協調により実現されているが，マスタホスト間のネットワークは非構造化オーバーレイを用いており，検索効率などがよくなかった．

本論文では，分散システムに参加しているホストの情報をクラスタを跨いで効率よく分散共有するために，また，ユーザの要件にあう性能の計算機を検索したいという要求に応えるため，マスタホスト同士間に構造化オーバーレイを構築する (図 3)．

ここで，構造化オーバーレイとはトポロジーに何らかの制約があるものであり，非構造化オーバーレイとはトポロジーに制約のないネットワークである．

3.1 分散ハッシュテーブル (DHT)

分散ハッシュテーブル (Distributed Hash Table) 法とは，構造化オーバーレイの一種を実現する手法であり，与えられたキーとその内容を保持するホストとの対応付けを基にしたアルゴリズムである．検索の対象

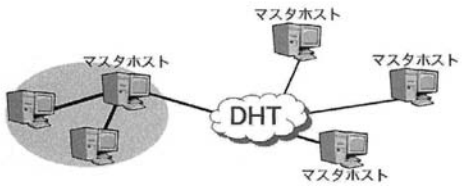

図 3 マスタホスト間で DHT を利用するイメージ

となるものにキーをつけ，そのキーのハッシュ値から配置するホストを決める．著名な DHT のアルゴリズムには，Chord[7]，Pastry[8]，Tapestry[9] などがある．

一般的に，DHT は検索効率は非構造化オーバーレイより優れているが，柔軟な検索は苦手であると考えられている．それは，キーが僅かでも異なればハッシュ値は全く異なるものになるためで，曖昧なキーワードを用いた検索を行うことが困難である．

3.2 DHT を用いた管理用情報の分散共有

提案システムでは，各クラスタのマスタホストによる構造化オーバーレイを構築し，DHT を用いた管理情報の分散共有を行う．これによりマスタホスト間での管理情報検索の効率化を図り，ユーザが所望する性能の計算機の情報を高速に検索し，タスクを移動する際の移動先候補選定に用いることが可能となる．

構造化オーバーレイを構築し DHT を利用するホストはマスタホストに限ることにした．これは DHT では地理情報を反映したトポロジを構築することが困難であると考えられるためである．ネットワーク的に近くにあるホストを効率よく利用する必要があるため，各クラスタのマスタホストのみで構造化オーバーレイを構築することとし，マスタ以外の一般ホストは自クラスタのマスタに情報の登録及び検索を依頼する形をとる．このような形態をとることで，管理用情報をマスタホスト間で効率よく分散共有して検索効率を高めるとともに，ネットワーク的に近いホストもクラスタリングされているため利用しやすい．

3.2.1 ホスト情報の登録

要件に合う性能のホストを検索するためには，まず，各ホストの性能を何らかの指標をもとに事前に評価しておく必要がある．一つには，分散システムのアプリケーションを起動する際に性能評価用の小さなプロセスを実行しているので，その数値を用いる方法が考えられる．また，プロセッサ名によって分類する方法も考えられる．

いずれにせよ，DHT の検索は柔軟性がなく完全一致検索的であることから，これを解消するために以下のような手順でホストの性能情報を登録する．なお，今回行ったシミュレーションではプロセッサの種類で分類する方法を用いたので，以下の説明でもプロセッサの種類で分類する方法で説明する．例として，性能情報を登録するホストのホスト名が Host1，所属クラスタのマスタホストが HostX，搭載されているプロセッサが Core2Duo 1.86GHz であるとする．DHT を用いて情報を共有するため，(キー, 値) の組を 2 段階で登録する (表 1)．まず 1 段階目の (キー, 値) の組として，(Core2Duo, 1.86GHz) を登録する．続いて 2 段階目の (キー, 値) の組として，(Core2Duo_1.86GHz, Host1) を登録する．

表 1 ホストの性能情報の登録例

段階	キー	値
1	Core2Duo	1.86GHz
2	Core2Duo_1.86GHz	Host1

表 2 ホストの所属クラスタ情報の登録例

段階	キー	値
1	Host1	HostX

このように 2 段階で情報を登録しておくことにより，まずはおおまかな性能 (プロセッサ名) で検索してどのようなクロックのものがあるか知った上で，次に詳細な性能 (プロセッサ名と動作クロック) により検索することができ，DHT を用いて柔軟な検索を行うことが可能となる．

性能評価用のプロセスを用いてホストの性能を数値で表した場合でも，数値を離散化してある範囲は Low，ある範囲は High などとランク付けして分類することで同様な手順で情報の登録が行える．

ホストの所属クラスタの情報については，所属クラスタのマスタホスト名を登録する (表 2)．この情報は，ホストの負荷情報を得る場合などに用いられる．(キー, 値) の組としては，例えば，(Host1, HostX) のようになる．

これらの登録する (キー, 値) の組には有効期限を設定し，期限が切れた (キー, 値) の組は DHT 上から自動的に削除することとする．性能情報に有効期限を設けない場合，あるホストが分散システムから離脱しようとする際に該当する情報を削除する必要が生じる．しかし，システムからの離脱の際に特定の処理が必要となるように設計すると，故障などによって突然離脱してしまった場合に問題を生じることとなるので，このような設計は避けなければならない．よって，性能情報に有効期限を設定し，この期間が経過した後には自動的に削除することとした．これに伴い，性能情報は定期的に再登録を行うこととした．

また，所属クラスタ情報にも同様に有効期限を設定する．所属クラスタは，クラスタ再構築により変更される可能性があるため，情報の再登録が必要になる．有効期限を設定することにより，再登録の際に古い情報の削除を行う必要がなくなる．

3.2.2 ホスト情報の検索

登録されているホストの性能情報・所属クラスタ情

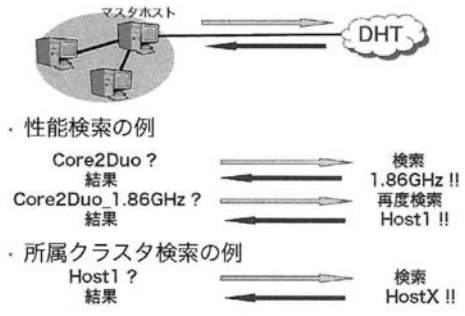

図 4 ホスト情報の検索例

報の検索について例を示す．登録されている情報は，先ほど例を示した 3 組であるとする．

まず，ユーザが必要とするプロセッサ名で検索する．このとき，検索要求が発生したホストが一般ホストであれば，検索はマスタに依頼される．ユーザが Core2Duo というキーで検索すると，1.86GHz という検索結果が得られる．他のクロックのものが登録されていれば，このときに複数のクロックが検索結果として得られることになる．

ユーザは，得られたクロックから所望のものを選び，2 度目の検索を行う．1.86GHz が所望のクロックであるとすると，Core2Duo_1.86GHz を検索キーとする．その検索結果として，この性能のプロセッサを搭載したホスト名が得られる．この例では，Host1 が検索結果として得られる．同一性能のホストが複数登録されていれば，複数の検索結果が得られることになる．

さらに，得られたホスト名をキーとして検索することにより，そのホストの所属クラスタのマスタホストが得られる．この例では，Host1 をキーとして検索すると，HostX という結果を得ることができる．ここで得られたマスタホスト HostX に問い合わせることで，Host1 の負荷情報などを得ることができる．

4. 実　　験

3 章で述べた管理情報の分散共有手法について，エミュレータ上に実装し，実際に動作可能であるのか検証を行った．まず，DHT の 3 種類のアルゴリズムによるメッセージ数の比較を行い，適切なアルゴリズムを見極める．また，各ホストが保持するキー数を確認し，キーの分布に偏りが生じるかどうか検証する．

なお，DHT を実装するために，首藤らが開発しているオーバーレイ構築ツールキット Overlay Weaver[10] を使用し，このツールキットに同梱されている分散環境エミュレータを用いて実験を行った．実験を実施した計算機の環境は次の通りである．

- CPU : PentiumIII 1GHz Dual
- メモリ : 512MB
- OS : Debian/GNU Linux 3.1 Sarge (Kernel 2.4.21)
- Java : JDK 5.0 update10
- オーバーレイ構築ツールキット : Overlay Weaver 0.6

4.1 実験詳細

実施したシミュレーションの詳細は次のようなものである．

まず，エミュレータ上に 100 ホスト (すべてマスタホストである) を起動する．それぞれのホスト名は emu0 ～99 である．各ホストにはユニークな ID が割り振られる．今回は，ホスト名を SHA-1(160bit) でハッシュしたものを ID として用いる．

各ホストを起動後，ランダムな順序でオーバーレイに参加し，ホスト情報の登録を行う．ホスト情報の登録先は，キーのハッシュ値により決まり，そのハッシュ値と同じ値を ID として持つホスト (存在しなければ，ID 空間で次のホスト) である．ホストの情報の有効期限は，性能情報が 10 分，所属クラスタ情報が 5 分とした．したがって，この間隔で再登録の操作を行うこととなる．

全ホストが参加した後，ランダムに検索を行う．

なお，ルーティング様式は Iterative ルーティングであり，比較する DHT アルゴリズムは Chord，Pastry，Tapestry の 3 つである．

4.2 実験結果

DHT アルゴリズムによるメッセージ数の比較を図 5 に示す．初めにメッセージ数が多いのは，各ホストをオーバーレイに参加させる手続きを行っているためである．また，約 600 秒毎に大きなピークが，その間に小さなピークが生じているが，これは性能及び所属クラスタ情報の再登録が行われているためである．

図 5 より，Tapestry はネットワーク上でやりとりされるメッセージ数が少なく，ネットワークに対して負荷をかけないことが明らかである．Chord は，オーバーレイの維持に多くのメッセージを必要とすると考えられる．

Tapestry を用いたときの各ホストが保持するキー数を図 6 に示す．横軸はキー数でソートした結果のホスト ID である．21 番目以降は省略したが，キー数は 1 である．Chord，Pastry でも同様の結果であった．これより，どのアルゴリズムを用いた場合でも，ホスト毎に保持するキー数に偏りが生じてしまっていることが分かる．これは，ある特定の性能のプロセッサが分散システムに参加するホストに大量に存在している場合に生じてしまう問題である．分散システム上にあるプロセッサの種類が様々であればよいが，実際にはある特定の種類のプロセッサが多量に用いられている可能性が高く，今後考慮しなければならない問題である．

なお，20 分間に 300 回の検索要求をランダムな時間

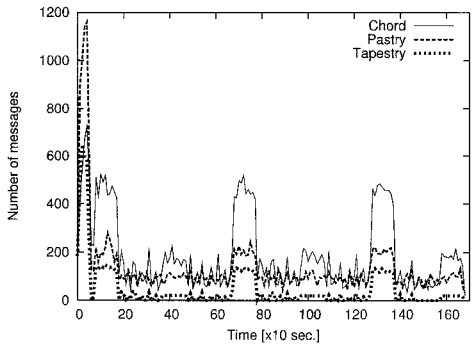

図 5 DHT のアルゴリズムによるメッセージ数の比較

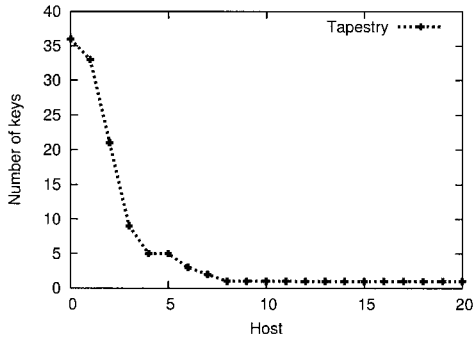

図 6 Tapestry を用いたときの各ホストが保持しているキー数

間隔とランダムなキーで発生させたが，失敗はなかった．検索時間は非常に小さいものであった．

5. ま と め

本論文では，計算資源の性能・負荷情報の管理に特定のサーバを用いず，分散システムに参加するホスト同士が自律的に協調し，管理情報を分散共有して動的負荷分散に用いる手法を提案した．管理用に特定のサーバを設置しないことで，サーバの障害によってシステム全体が停止することを避けることができる．また，情報の分散共有のために DHT を用い，効率的な検索が可能とした．

実験の結果，検索要求は失敗しなかったが，各ホストが保持するキー数に偏りが生じてしまうことが明らかとなった．

今後は，各ホストが保持するキー数の偏りを少なくする方法について検討し，より大規模なシミュレーション実験を行っていく．また，実際の分散システムに実装し，実環境での実験を行い，本手法の有効性を検証していく．

参 考 文 献

1) 日本アイ・ビー・エム システムズ・エンジニアリング株式会社："グリッド・コンピューティングとは何か"，ソフトバンクパブリッシング，Apr. 2004

2) BOINC(Berkeley Open Infrastructure for Network Computing): http://boinc.berkeley.edu/

3) SETI@home: http://setiathome.berkeley.edu/

4) 氏家，菅谷，阿曽："ホストの自律的クラスタリングを用いた動的負荷分散アルゴリズム"，平成 18 年度電気関係学会東北支部連合大会，1F-3, p.202, Aug. 2006

5) D. B. Lange and M. Oshima : "Programming and Deploying Java Mobile Agents with Aglets", ADDISON-WESLEY, 1998

6) K. P. Chow and Y. K. Kwok : "On Load Balancing for Distributed Multiagent Computing", IEEE Trans. on Parallel and Distributed Systems, Vol.13, No.8, pp.787–801, Aug. 2002

7) I. Stoica, R. Morris, D. Karger, M. F. Kaashoek and H. Balakrishnan : "Chord: A Scalable Peer–to–peer Lookup Service for Internet Applications", Proc. of ACM SIGCOMM 2001, pp.149–160, Aug. 2001

8) A. Rowstron and P. Druschel : "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems", Proc. of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), pp.329–350, Nov. 2001

9) B. Y. Zhao, J. Kubiatowicz and A. D. Joseph : "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing", Technical Report UCB//CSD-01-1141, University of California, Berkeley, Apr. 2001

10) 首藤，田中，関口："オーバーレイ構築ツールキット Overlay Weaver"，先進的計算基盤システムシンポジウム SACSIS2006, pp.183–191, May 2006