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

2006－ARC－167 (38)
2006－HPC－105 (38)
2006／3／1

大規模環境向け情報共有手法を用いた分散ジョブスケジューリングシステム

梅田典宏[†] 中田秀基[††,†] 松岡聡[†,†††]

ジョブスケジューリングシステムを利用することで分散した計算資源を統合し、利用することが可能になっている。しかし既存のシステムは、スケジューリングに必要な資源情報収集および実行ジョブと資源のマッチングを少数の計算機で行うことによる単一故障点の存在と資源・投入ジョブ数の増加に対するスケーラビリティの欠如という問題を抱えている。我々は、大規模環境向けの通信手法を用いて資源情報を共有し、耐故障性と資源数の増加に対しスケーラブルな分散ジョブスケジューリングシステムを提案し、シミュレータ上で比較評価を行った。その結果、大規模環境下に於いて既存のシステムより利用効率の低下を小さく抑えられることを確認した。

Decentralized Job Scheduling System based on Information Sharing framework for Large-Scale Computing Environment

NORIHIRO UMEDA,[†] HIDEMOTO NAKADA[††,†]
and SATOSHI MATSUOKA [†,†††]

Job scheduling system enables to unify and to use distributed computer resources However these systems has a single point of failure that just a few computers makes assignments job to resources, and lack of scalability to increase number of resources and jobs. We claim decentralized job scheduling system to share resources status using communication framework for large-scale computing environment and evaluated on simulations. The results showed that our proposal reduced a decline of efficency on large-scale computing environment.

1. はじめに

ネットワークで結ばれた広域に分散する多数の計算機を統合し、仮想的な一つの計算機として利用するグリッドが普及しつつある。グリッド環境では、利用可能な計算資源の状態 (アーキテクチャ、CPU 使用率、空きメモリ、ストレージ容量など) とユーザから投入されるジョブの実行に必要、ないしは実行により適した条件の情報を収集し、適切な資源割り当てを行うジョブスケジューリングシステムを用いることで計算資源の効率的な利用を可能にしている。代表的なジョブスケジューリングシステムとして Condor[1], XtremWeb[2] などが存在し、現実に最大 1000 台程度の計算機を集約して利用する環境が提供されている。

これらのシステムでは、計算資源の情報収集およびジョブの割り当てを一台ないしは少数の固定された計算機によって行っており、それらに障害が発生した際には管理下におかれた資源全体の利用が不可能になってしまう。また、情報収集にかかる通信量や資源とジョブのマッチングを求めるコストが少数の計算機に集中することから、今後予想される計算資源および投入されるジョブの増大に対するスケーラビリティに問題がある。

本研究では、コストの低い不完全な情報共有手法によって各計算資源の状態を複数のノードで共有し、スケジューリングを複数の計算機で分散して行うことにより、負荷集中と単一故障点の排除を目指すシステムを提案する。そして、本提案手法と既存のシステムが基盤としているモデルをシミュレーションで比較し、情報の不完全性と計算資源および投入ジョブ数が性能に与える影響を評価した。

2. 関連研究

2.1 Condor

Condor は、ウィスコンシン大学で開発されたジョブスケジューリングシステムであり、バックグラウンドでのジョブ実行によって生ずるレスポンスやパフォーマンスの低下といった計算機の主所有者に対する悪影響を抑えつつ、組織が所有する遊休計算機の稼働率を向上させることを目的としている。

† 東京工業大学
Tokyo Institute of Technology
†† 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology
††† 国立情報学研究所
National Institute of Informatics

```
MyType = "Machine"
TargetType = "Job"
Machine = "nostos.cs.wisc.edu"
Requirements = (LoadAvg ≤ 0.3) &&
               (KeyboardIdle ≤ 15*60)
Rank = target.Department == my.Department
Arch = "INTEL"
OpSys = "LINUX"
Disk = 3076076
Memory = 128
KeyboardIdle = 173
```

図 1 計算資源の状態を表す ClassAd の例

```
MyType = "Job"
TargetType = "Machine"
Requirements = (target.Arch == "INTEL" &&
target.OpSys == "LINUX")
&& target.Disk ≤ my.DiskUsage
Rank = (Memory * 10000) + KFlops
Cmd = "/home/tannenba/bin/sim-exe"
Owner = "tannenba"
DiskUsage = 6000
```

図 2 ジョブの実行条件を表す ClassAd の例

Condor では、利用対象となる各計算資源およびそれらを管理する計算機の集合を Condor プールと呼ぶ。Condor プールには、ユーザからのジョブの投入を受け付け、計算資源に適切に割り当てる Central Manager が一台のみ存在する。

2.1.1 資源情報の収集とジョブの割り当て

Condor におけるジョブと計算資源の割り当ては Matchmaking[3] と呼ばれる手法を用いている。以下にその詳細を述べる。

計算資源を提供するすべてのマシンは、アーキテクチャや OS、CPU 使用率、メモリ・ストレージ容量といった自らの状態を示す情報を定期的に収集している。また、各資源の所有者は Condor によって外部から投入されるジョブが満たすべき利用ポリシーを定義しており、それらをあらかじめ固定された Central Manager に ClassAd と呼ばれるフォーマットで定期的に送信する。図 1 にその例を示す。

Central Manager はプール内に一台のみ存在し、計算資源から送信された ClassAd をすべて回収する。一方ユーザは、ジョブの実行に必要ないしは望ましい条件を図 2 のようなフォーマットでジョブそのものと一緒に Central Manager に投入する。Central Manager は与えられたジョブと資源の情報を参照し、できるだけ実行に望ましい計算資源にジョブの実行を割り当てる。計算ノードでジョブ実行が終了するとその結果は Central Manager によって回収される。

2.2 XtremWeb

XtremWeb は、個人の PC やワークステーションのみならず PDA や携帯電話までを含めるネットワーク上に広く分散した計算機の遊休時間を利用してマスタワーカ型分散ソフトウェアを大規模に実行させるためのミドルウェアである。XtremWeb は計算資源を提供する Worker とそれらに計算ジョブを割り当て管理する Server から構成される。

Worker はアーキテクチャ、OS といった自らの状態およびキーボードやマウスの使用状況などから得られる資源所有者の利用状況と計算に利用するプログラムやライブラリ、データなどの所有情報を Server 側に定期的に送信する。

Server は、ユーザからのジョブ投入を受け付け、Worker から送られてきた情報を参照して適切にジョブを割り当てる役割を担っている。

マスタワーカ型計算モデルでは、一台ないしはごく少数のマスタに対し多数のワーカがぶら下がる形になる。XtremWeb では、マスタに相当する Server を必要に応じて増やすことで、ワーカの増加に対応している。具体的には、Linux Virtual Server[4] や DNS ラウンドロビンといった IP レベルでの冗長化手法によって、Worker からのアクセスを複数台の Server に分散させることが可能である。

2.3 大規模環境向け情報共有手法

この章では、多数の計算機間で情報を共有する手法の一つである Gossip Protocol について述べる。

2.3.1 Gossip Protocol

Gossip Protocol[5] は、多数のノード間において高速かつ効率的な情報共有を実現するための通信手法である。共有する情報の生成元は、次に示す過程を反復して実行され、最終的にすべてのノードで停止する収束状態をもって伝達が終了する。

(1) メッセージの発信元は接続可能なノードをランダムに選択し、メッセージを転送する。
(2) メッセージを受信したノードは、メッセージが既知か否かで次の動作に分かれる:
  (a) 既知であったときは、それを送信元に伝える。
  (b) 新しいメッセージを受信したときには、自身もそのメッセージの発信元となり、過程 (1) を開始する。
(3) 送信元は、一定回数以上既知の応答を受けるまで繰り返す。回数を超えたら停止する。

(1) に於いて毎回通信経路が動的に選択されるため、あるノード間の通信路ないしはノード自身に障害が発生した際、それを迂回する耐故障性を備えている。

Gossip Protocol では、全ノードへメッセージを伝達させる保証はできないが、高い伝達率を実現している。また、周期ごとに伝達ノードが指数的に増加するため、全体のノード数増加に対する収束回数の増加は

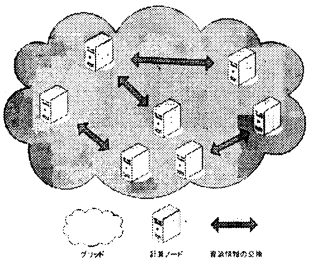

図 3　分散ジョブスケジューリングシステムの概要

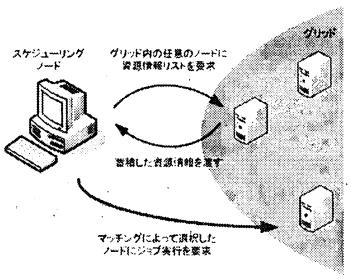

図 4　スケジューリングに必要な資源情報の取得

小さい。

3. グリッド環境の大規模化に伴う問題点

今後グリッドを構成する計算資源の量およびそこに投入されるジョブの数が共に飛躍的に増大することが予想される。

Condor では、資源情報の収集およびジョブと資源のマッチングを行う Central Manager を一台の固定された計算機で実行している。よって、資源数とジョブ数の増加に伴う負荷が無視できないものとなる。また、Central Manager に障害が発生すると計算資源全体の利用が不可能になることから、Central Manager には高い信頼性が必要となる。

XtremWeb では、Server を複数台で構成することが出来るため、資源とジョブの増加による負荷への対応は可能である。しかし、Server のアドレスはあらかじめ固定されたものになるため、それらが存在するネットワークに障害が起きた際にはシステム全体に影響が及んでしまう。

4. 分散ジョブスケジューリングシステムの提案

グリッド環境の大規模化に伴う負荷と障害数の増加への対応は、既存のシステムでは不十分なものとなっている。そこで我々は、スケーラブルでかつ単一故障点の排除を目標とした分散ジョブスケジューリングシステムを提案する。

4.1 概　　要

スケジューリングに必要な計算資源の情報を、大規模環境に適したスケーラブルな情報拡散手法を用いて複数のノードで共有する。システムにジョブの投入を行うマシンは、グリッドを構成する任意のノードから共有する資源情報を取得し、その情報を基にジョブと資源のマッチングを形成し実行を行う。

本提案手法においては、次の二種類の要素から構成される。

計算ノード

計算資源を提供する計算機である。また、計算ノード同士で自らの資源状態を表す情報を互いに交換・共有する。

スケジューリングノード

計算ノードからグリッドを構成する計算資源の状態を表す情報を取得し、ユーザから投入されたジョブを適切に割り当てる。

なお、単一のマシンが両方の機能を担うこともありうる。

4.2 資源情報の共有

各計算資源は、定期的に自らの CPU 使用率・空きメモリ・ディスク容量などに代表される資源情報を収集し、資源の所有者によってあらかじめ定義された資源利用に関するポリシーとともに他のマシンに伝達する。伝達された情報は、Gossip などの大規模環境に適した手法を用いてグリッドを構成する多数のノードに広報され、互いの資源状態・利用ポリシーについての情報を共有する形になる。

ユーザがジョブを投入するには、それらグリッドを構成する任意のノードを選択し、そこからグリッド内の資源情報を取得する。そこで得られた情報を元に、資源とジョブのマッチングを形成してスケジューリングを行い、ジョブの実行を割り当てる。

4.3 ジョブと計算資源のマッチメイキング

スケジューリングノードはグリッドに参加している任意のノードを選択し、それらが蓄積しているスケジューリングに必要な資源情報を取得する。ユーザはそれらスケジューリングノードに対し、ジョブと計算資源のマッチメイキングを依頼する。利用する資源が決定したら、まずその計算ノードに対し割り当て要求を出し、実行可能という返答があったら正式にジョブの実行を依頼する。すでに別のジョブが投入されている、あるいはグリッドから離脱するなどジョブを実行できないときはその旨を元のスケジューリングノードに返す。その場合はマッチングをやり直し別のノードに再度要求を出して実行されるまで繰り返す。

4.4 大規模環境に適した情報共有手法

単一故障点および負荷の集中を防ぐためには、できるだけ多数のノードで情報を共有する必要がある。しかし、ブロードキャストなどが利用できない環境下に

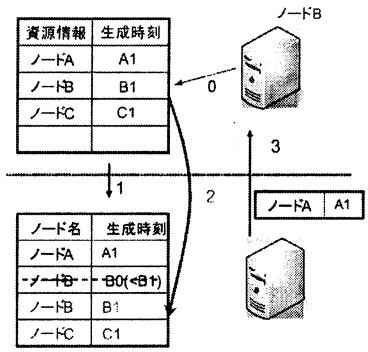

図 5　Gossip Protocol による資源情報の交換

おいて多数のノード間で同じ情報を完全に共有することは非常にコストが高い。

また、計算資源の動的な追加・離脱やノードや通信路での障害発生への対応も必要であることから、あらかじめ静的に決定された経路による情報伝達を行うことは本提案手法において現実的ではなく、状態の動的な変化に対応した通信手法を用いる必要がある。

以上を踏まえると、大規模なグリッド環境で完全な情報をすべてのノードで矛盾なく共有するのはコストが高くまた困難が多い。よって、妥当な精度でより負荷の軽い手法として本研究では Gossip Protocol を採用した。

4.5　資源情報広報への Gossip Protocol の適用

Gossip Protocol を以下のように用いることで大規模なグリッド環境下での高速で安定した資源情報の広報を行う。図 5 に概要を示す。

0　定期的に自らの資源情報を更新
　計算ノードは、定期的に自らの状態を収集し、資源情報を更新する。資源情報には、それを生成した時刻と生存期限が付加されている。

1　ランダムに選択したノードと情報を交換
　各計算資源は、グリッドを構成する計算ノードをランダムに選択して接続し、互いに自らの持つ計算ノード群の各資源情報を交換する。

2　受信ノードは自らの保持する情報を更新
　受信ノードは、送られてきた情報のうち保持していないものを自らのテーブルに追加する。同じ情報源から生成された情報を持っている場合は、タイムスタンプを参照してより鮮度の高い情報を採用する。

3　既知であった情報を送信元に通知相手から送られてきた情報のうち、既知であったものを送信元に通知する。

計算資源が新たにグリッドに参加するときには、スケジューリングノードと同様にグリッドに参加している近隣のノードに対し資源情報を要求することで Gossip に必要な他ノードの位置を取得する。また、計算資源がグリッドを離脱する際には、離脱メッセージを Gossip で広報することによって、そのメッセージを受信した各ノードはその資源が利用不可能であるとしてその資源情報を破棄する。また、各自ノードは保持する情報の生存期限をチェックし、期限を過ぎた情報は破棄を行うことによって、計算機やネットワークの障害によってアクセスできなくなった資源は自動的に利用対象から除外される。

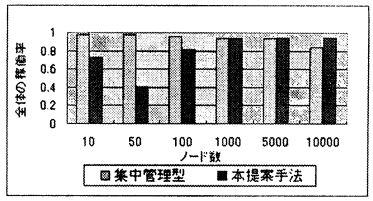

図 6　メッセージ伝達率

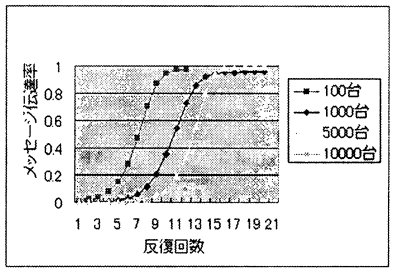

図 7　反復回数によるメッセージ伝達率の変化

5. 性能評価

5.1　Gossip Protocol の性能

5.1.1　Gossip Protocol の基礎評価

一周期に選択する伝達ノード数を 1、既知応答を受ける回数を 1 としたときの収束時における全ノードに対する資源情報の伝達率を図 6 に、過程の反復回数による伝達率の変化を図 7 に示す。

5.1.2　通信の非対称性が存在する環境

現在 NAT やファイアウォールなどによって、外部から内部への接続が制限される環境に属する計算機同士が多数存在する。Gossip Protocol では各ノードが直に接続することで情報交換を行うため、そのようなサイト間をまたいで直接資源情報を交換することはできない。

しかし、TCP のように内部から接続を張ることで双方向通信が可能な通信手法であれば情報の交換を行うことができるため、双方向通信を行えるホストを経由して資源情報の流通を行うことが可能である。ノード 1000 台のうち、外部から通信を張ることができないノードの割合を変えたときの資源情報の流通速度を測定した。結果を図 8 に示す。

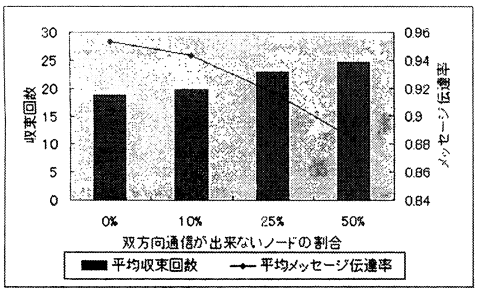

図 8 メッセージ伝達率

双方向通信が不可能なノード数が増えるにつれ収束に要する回数は増加し最終的な情報の伝達率も低下するが、ノードの割合が大きくてもその低下は比較的小さく、情報の完全性が極端に悪化することがないことを確認した。

5.2 本提案手法の性能評価

本研究で提案した分散ジョブスケジューリングシステムと、Condor をモデルとした既存の集中管理型システムをそれぞれシミュレータ上に実装し、同じ計算資源、ネットワークに対し同じジョブを投入した際のスループットを測定した。

両システムで共通とした資源とジョブのマッチングに必要な処理時間は、Condor の MatchMaking システムをモデルにしたものを Java で実装し、そこで実際に処理にかかった時間をコストとして用いた。

すべての実験において、Gossip による資源情報の交換を行う頻度は 1 分に 1 回とした。ノード間を結ぶネットワークのトポロジおよび遅延は GridG[6] によって生成されたものを利用し、ノード間遅延が最大 200ms 程度に分散する環境を想定した。

5.2.1 本提案手法と集中管理型システムとの性能比較

本提案手法と既存の集中管理型システムの比較を行うため、スケジューリングノード数を集中管理型と同じ 1 台に固定し、グリッドを構成する計算ノード数の規模による性能変化の評価を行った。常にスケジューリングノードの実行待ちキューにジョブが蓄えられており、利用可能な計算ノードが存在すればそこに常に新しいジョブが割り当てられる状態とした。投入されるジョブの実行に必要な時間は 30 分から 1 時間の間になるように設定した。十分長い時間が経過しグリッド全体が定常状態と見なせるときの平均稼働率を表 1 に示す。

ノード台数が 10 台から 100 台までの中小規模な環境においては、本提案手法が集中型システムに比べ大きく性能が劣っている。これは、集中管理型システムが各計算ノードから 1 ホップで直接資源情報を回収しているのに対し、Gossip Protocol の場合他ノードを複数回経由して伝達される遅れに起因する情報の不完全性がスケジューリングに大きく影響しているためである。

表 1 集中管理型と本提案手法のシステム平均稼働率の比較

計算ノード数	集中管理型	本提案手法	性能比
10	0.981	0.735	0.749
50	0.977	0.410	0.420
100	0.958	0.817	0.853
1000	0.936	0.939	1.003
5000	0.935	0.951	1.017
10000	0.844	0.948	1.121

ノード数が増えるにつれ、集中管理型システムでは情報収集の集中によるオーバーヘッドが無視できなくなり、相対的に Gossip による通信時間よりも大きくなるため、現在大規模とされる 1000 台以上の環境では、既存のシステムとほぼ同程度ないしは最大 10%程度上回る稼働率を示している。

5.2.2 システム全体の稼動率による性能の変化

計算資源が処理能力の総和に対する計算量が 10 %,50 %,100 %,200 %となるような投入間隔にしたがうジョブの集合を、一台のスケジューリングノードからそれぞれ 100 台,1000 台,5000 台のマシンに投入するシミュレーションを行った。集中管理型システムを 1 とした際の本提案手法の性能比を表 2 に示す。

資源数が少ないときは、ジョブ数が増加しシステムの利用率が高くなるにつれて性能が既存システムに比べ低下する。これは、利用率が高くなると利用可能な計算資源が少なくなり、5.2.1 で述べた Gossip の情報の不完全性がよりスケジューリングに悪影響を与えるためである。ノード数が増加すると同様に集中型のオーバーヘッドが Gossip のそれを上回るため本提案手法がより高い性能を示している。

表 2 計算資源数の変化による稼働率の性能比

	資源全体の能力に対するジョブの計算量			
	0.1	0.5	1.0	2.0
100 台	0.998	0.995	0.853	0.823
1000 台	1.000	1.002	1.003	1.012
5000 台	1.007	1.016	1.017	1.060
10000 台	1.010	1.019	1.121	1.092

5.2.3 複数マシンによるスケジューリングの性能

本提案手法の上で計算資源とジョブのマッチングを行うノードの台数による性能の変化を測定した。計算資源 1000 台のシステムに対し、処理能力の総和に対する計算量が 100 %となるような投入間隔に従うジョブ列を投入し、それぞれ 1 台、2 台、4 台、8 台でマッチングした際のシステム全体の稼働率を表 9 に示す。台数が増えることでより効率的な割り当てが行われている結果が得られた。

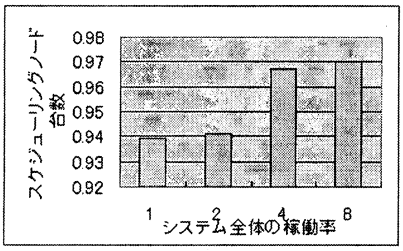

図 9 スケジューリングノード数によるシステム全体の稼働率

6. 考　　察

本提案手法に於いて性能が低下する主な要因は次の二つである。

- 情報の不完全性
- Gossip の伝達時間のオーバヘッド

これらについての考察を述べる。

6.1 情報の不完全性

前述の通り、Gossip Protocol では情報の不完全性が避けられないが、5.1.1 および 5.1.2 によれば、ノード台数の増加に対し広報に必要な時間・伝達度ともに高い結果が得られており、広報時間に対しジョブの実行時間が十分長いものであれば実用的であると思われる。

5.2.3 の結果では、マッチングを行うノードが多いほど稼働率が向上し、効率の良いスケジューリングが行われている。これは、マッチングの負荷が分散されたためと、情報の取得を複数地点から行ったため、Gossip の欠点である情報の不完全性が小さくなったためと考えられる。

6.2 Gossip の伝達時間のオーバヘッド

本提案手法に於いては、資源情報が複数のノードを経由して伝達されるため、前者に比べ伝達時間が長くなってしまう。よって、他ノードを経由中に状態が変化し Gossip で伝播されている情報がすでに現状を反映しておらず、その古い情報を基にしたスケジューリングによって性能の低下が起きる可能性がある。

5.2.2 の結果では、資源の数によって投入ジョブ列の多少による性能変化の形が大きく異なる。これは、Gossip の伝播速度に原因があると考えられる。グリッド全体の稼働率が高く、実行待ちジョブが多く存在する状態では、計算資源情報の伝達速度がスケジューリングの効率に大きい影響を与える。図 6 によると、ノードが少ないときの収束回数は多いときのそれと比べ相対的に大きい。つまり、計算ノードおよび投入ジョブの絶対数が少ないときはより単純な収集機構である集中管理型のほうが高速となる。

しかし、集中管理型では情報収集のコストが計算資源数に対し線形に増加するため、資源数が増えると Gossip の伝達速度より通信の集中の影響が相対的に大きくなる。よって、本稿で提案した手法のほうがより効率的なスケジューリングが可能になると考えられる。

7. おわりに

7.1 ま と め

不完全な情報共有手法で資源情報の収集を行う分散ジョブスケジューリングシステムを提案し、シミュレータ上で集中型との性能比較を行った。現在小中規模とされる台数下では既存システムに対する通信コストが大きいことにより性能が大きく低下するが、それを超える大規模環境においては同等ないしは 10%程度上回る利用効率を確認した。また、スケジューリングマシンの複数化によってグリッド全体の利用効率をより向上することを確認した。

7.2 今後の課題

今後の課題としては以下が挙げられる。

- Gossip 以外の情報共有手法の検討
- 広報される情報の検証機構
- 実環境での検証

参 考 文 献

1) Litzkow, M.J., Livny, M. and Mutka, M.W.: Condor - A Hunter of Idle Workstations, *Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS)*, Washington, DC, IEEE Computer Society, pp. 104–111 (1988).

2) GillesFedak, Ce'cileGermain, V.N. and Cappello, F.: XtremWeb : A Generic Global Computing System, *CCGRID2001, workshop on Global Computing on Personal Devices* (2001).

3) Raman, R., Livny, M. and Solomon, M.: Matchmaking: Distributed Resource Management for High Throughput Computing, *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, Chicago, IL (1998).

4) : The Linux Virtual Server Project - Linux Server Cluster for Load Balancing, http://www.linuxvirtualserver.org/.

5) Jenkins, K., Hopkinson, K. and Birman, K.: A Gossip Protocol for Subgroup Multicast, *International Workshop on Applied Reliable Group Communication (WARGC 2001)* (2001).

6) Lu, D. and Dinda, P.: Synthesizing Realistic Computational Grids, *"Proceedings of ACM/IEEE Supercomputing 2003 (SC 2003)"* (2003).