E-03

利己的なクラスタで構成される異種計算グリッドでの負荷分散スケジューリング
Load Balancing In Selfish Heterogeneous Computational Grids

岡崎 祐也†　大下 福仁†　角川 裕次†　増澤 利光†
Yuya Okazaki　Fukuhito Ooshita　Hirotsugu Kakugawa　Toshimitsu Masuzawa

１．まえがき

近年，並列計算の実行環境として計算グリッドが注目を集めている[1]．生物学等の様々な分野において，大きなサイズの科学計算に計算グリッドが用いられている．複数の組織が計算資源としてクラスタを提供し，それらを接続すれば，一つの組織では構築できない規模の計算グリッドを構築することができる．本研究では，このように複数の組織が構成する計算グリッドに注目する．

計算グリッドでは，計算等のジョブを効率よく処理するために，負荷分散アルゴリズムが用いられる．これまでに多くの負荷分散アルゴリズムが提案されているが，その多くは計算グリッド内の全ての計算資源が負荷分散アルゴリズムに従うことを前提としている．しかし，複数の組織で計算グリッドを構築する場合，各組織は自身の利益を優先するために利己的に振る舞う可能性がある．例えば，組織はクラスタを提供することで自身のクラスタを別の組織に利用されることになり，計算グリッドに参加しない場合に比べて不利益を被る可能性がある．もし組織の不利益が利益より大きければ，その組織はクラスタの提供をやめてしまう可能性が高い．これは，計算グリッドの計算能力を低下させることになるため，各組織に十分な利益を与えるような負荷分散アルゴリズムを用いる必要がある．

組織が利己的に振る舞うことを考慮した負荷分散アルゴリズムは，これまでにいくつか研究されている．文献[2],[3],[4],[5] では，各組織が自身のジョブを割り当てる計算資源を利己的に選択するという仮定のもとで負荷分散アルゴリズムを考察し，利己的な選択により全ジョブの処理時間がどのくらい悪化するかを示している．また，文献[6],[7],[8] では，各組織のジョブの処理時間を犠牲にすることなく，全ジョブの処理時間を短くする負荷分散アルゴリズムが提案されている．

Rzadcaら[9] は，計算グリッドに属する各クラスタが利己的に負荷分散に協力するかどうかを決定するような環境で，負荷分散への協力を促す負荷分散アルゴリズムを提案している．各クラスタは，負荷分散に協力することで，他のクラスタへ自身のジョブを送って利益を得ることができるが，自身のクラスタで他のジョブを処理する必要もあり不利益を被る可能性がある．Rzadcaらの負荷分散アルゴリズムは，各クラスタが投入したジョブを公平に実行することで，各クラスタに一定の利益を与え，負荷分散への協力を促している．ただし，Rzadcaらは各クラスタの性能が全て同じという状況のみで評価を行っている．しかし，複数の組織が提供するクラスタは性能が異なることが想定されるため，クラスタの性能が異なる異種計算グリッドにおけるアルゴリズムの性能評価が必要である．

そこで，本報告では，Rzadcaらの負荷分散アルゴリズムを異種計算グリッドに対して適用し，シミュレーションによりその性能評価を行う．その結果，Rzadcaらの負荷分散アルゴリズムは，異種計算グリッドでも各クラスタを負荷分散へ協力させることができるものの，性能の高いクラスタほど協力の度合いが小さいことを示す．これは，負荷分散で得られる利益が性能の高いクラスタほど小さくなることに起因する．そこで，Rzadcaらの負荷分散アルゴリズムを改良し，性能の高いクラスタのジョブを優先的に実行することで，性能の高いクラスタに対してより大きな利益を与えるアルゴリズムを提案する．シミュレーション実験により，Rzadcaらのアルゴリズムと比較して，提案アルゴリズムが性能の高いクラスタの協力の度合いを大きくすることを示す．

2. 諸定義

本報告の諸定義をここに示す．

2.1 計算グリッド

計算グリッド(以下，グリッド) はn個のクラスタ$M_1, \ldots, M_n$によって構成される．本報告では，各クラスタは別々の組織によって所有されているものとする．

クラスタM_iは組織O_iに所有されており，組織O_iはクラスタM_iの動作を決めることができる．組織O_iでジョブ$J_{i,1}, J_{i,2}, \ldots$が順次発生し，クラスタM_iのキューに蓄積していく．クラスタM_iは自身のキューに並んでいるジョブをその順に実行する．ジョブ$J_{i,j}$はクラスタM_iのj番目のジョブを表し，その負荷の大きさを$p_{i,j}$と表記する．組織O_iで発生する全てのジョブを$J_i = \{J_{i,1}, J_{i,2}, \ldots\}$とする．また，ある時点に注目した時，その時点でクラスタM_iのキューに並んでいる全てのジョブの負荷の合計をL_iとする．クラスタM_iの性能はs_iと表わされ，M_iがジョブ$J_{i,j}$を処理するのにかかる時間は$p_{i,j}/s_i$で求められる．つまり，s_iはクラスタM_iが単位時間あたりに処理できる負荷の量を示している．

組織O_iは，自分の組織内で発生した全てのジョブの処理時間の合計F_iを短くしようとしている．F_iはジョブ$J_{i,j}$の発生時刻を$r_{i,j}$，完了時刻を$C_{i,j}$として，$F_i = \sum_j (C_{i,j} - r_{i,j})$で求められる．定義の式から明らかなように，処理時間にはジョブがキューに到着してから処理が開始されるまでの待ち時間も含まれている．クラスタは一度に一つのジョブの処理し，複数のジョブを並行して行わない．

ジョブ$J_{i,j}$は自由に分割することができ，並行してその断片を複数のクラスタで処理することができるとする．本報告では，どのクラスタも相互に接続されており，負荷を任意のクラスタへ送ることができるとする．組織O_iはジョブの発生時間をあらかじめ知ることはできない．そして，ジョブの負荷の大きさはキューに到着してすぐに判明する．

† 大阪大学大学院情報科学研究科，Graduate School of Information Science and Technology, Osaka University

2.2 負荷分散

グリッド中に含まれる二つ以上のクラスタによって，ジョブの負荷分散が行われる．負荷分散は一定時間tごとに全クラスタを集中制御型で管理するシステム(以下，負荷分散システム) が行う．一定時間Tごとにグリッド中のクラスタは一定時間Tの間に負荷分散に参加するかどうかを決定する．協調活動に参加するクラスタの集合$\{M_1, \dots, M_m\}$を連立と呼ぶ．連立に参加したクラスタは一定時間Tの間，負荷分散システムに従って負荷分散を何度も行う．一度連立に参加すれば，そのクラスタは次に連立を作る時刻まで連立から抜けられない．一方，参加しなかったクラスタも次に連立を作る時刻まで連立に参加できない．負荷分散システムは，負荷分散を行う際の連立中の各クラスタのキューを参照し，キューに並んでいるジョブを並び替える．また，負荷分散システムは一度キューへ並んだジョブの完了時刻を保証する．負荷分散システムは完了時刻を保証する必要があるため，一度キューに並んだジョブの実行を遅らせることができない．また，負荷分散に参加するクラスタはシステムに割り当てられたジョブを順番通りに実行する．クラスタが負荷分散に参加しなかった場合，自分のキューに並んでいるジョブを順に処理していくこととなる．

3. 既存研究[9] のアルゴリズム

既存研究[9] は，全てのクラスタの性能が一定$(s_1 = s_2 = \cdots = s_n = 1)$である場合に対して，Bounded, iterative load balancingアルゴリズム(以下，BILB)を提案している．BILBはシステム中のクラスタに負荷分散への参加を促す．負荷分散を促すため，単純に負荷を平均化する負荷分散よりも公平に負荷分散を行う．BILBは制限法(Bounded)，反復法(Iterative) の２つの手法を使っている．それぞれについて個別に説明する．

3.1 制限法

各クラスタは負荷分散により，外部から負荷を受け取るが，その負荷が大きいほど次に発生する自分のジョブの処理の開始が遅れる可能性が増える．また，外部へ負荷を送れば自分のジョブの完了時刻は早くなる．

制限法では，各クラスタにキューの長さの限界値を設定させることで，負荷分散に参加することによって生じる利益や損失を制御させる．限界値を高く設定すれば，負荷分散によって発生する損失が大きくなる可能性が増えるが，同時に大きな利益を得られる可能性も増える．逆に低い限界値を設定すれば，発生する損失が小さくなる傾向になるが，得られる利益も同様に小さくなる可能性が高い．

まず，各クラスタM_iは負荷分散に参加するかどうかの選択をする代わりに，キューの長さの限界値として参加レベルl_iを宣言する．参加レベルl_iを宣言すると，$L_i \geq l_i$のとき，そのクラスタは外部から負荷を受け取ることがない．また$L_i < l_i$のとき，外部から負荷を受け取るが，キューの長さは$L_i > l_i$にならない．また，低い参加レベルを宣言することを防ぐため，各クラスタが外部へ送ることができる負荷の大きさも参加レベルによって制限する．すなわち，M_iが外部へ送る負荷の総量をΨ_iとすると，$\Psi_i \leq l_i$ が成立する．

3.2 反復法

単純に平均より負荷の大きいクラスタから小さいクラスタへ負荷を送るなら，負荷分散に参加したクラスタは得をできるものと損をするものに分かれる．反復法では，負荷の少ないクラスタから順に，そのクラスタより負荷の小さなクラスタへ負荷を送ることで，より多くのクラスタが得をできることを目指す．

まず，クラスタを負荷の小さい順に並べ，小さいものから順に$M_1, \dots M_k, \dots, M_m$とする．$m$は負荷分散に参加しているクラスタの数である．そして，クラスタ$M_k(k = 2,3, \dots, m)$はM_1からM_{k-1}に負荷を送り，M_1からM_kまでの負荷が一定になるようにする．M_kは負荷を送る時，M_1からM_kまでの負荷の大きさの平均$\overline{L_k}$を求める．$\overline{L_k}$は次の式で求められる．

$$\overline{L_k} = \frac{1}{k}\sum_{i=1}^{k} L_i \tag{1}$$

M_kは$L_k - \overline{L_k}$だけの負荷をM_1からM_{k-1}へ送ろうとする．しかし，制限法により，送る量が制限されるので，実際に送ることができる負荷の総量Ψ_k^*は次の式で求められる．

$$\Psi_k^* = \min(L_k - \overline{L_k}, l_k) \tag{2}$$

M_kが$M_i(1 \leq i \leq k-1)$に送ろうとする負荷の大きさを$\varphi_{i,k}$とすると，$\varphi_{i,k} = \Psi_k^*/(k-1)$となる．しかし，制限法により，送る量が制限されるので，実際にM_kがM_iに送ることができる負荷$\varphi_{i,k}^*$は次の式で求まる．

$$\varphi_{i,k}^* = \min(\frac{\Psi_k^*}{k-1}, l_i - L_i) \tag{3}$$

このアルゴリズムを用いると，M_1以外のクラスタは自分より負荷の小さいクラスタが存在するので，外部クラスタへ負荷を送ることができ，ジョブの処理時間を短縮できる．M_k自身の負荷の保証完了時刻は，最大でも$\overline{L_k}$となる．

4. BILBの異種計算グリッドへの適用

本節では，3. 節のアルゴリズムBILBを異種計算グリッドへ適用する．適用するため，制限法，反復法を変更する．本報告の負荷分散による利益をジョブの処理時間の短縮とし，BILBを異種グリッドにおいて使用できるように拡張する．

4.1 制限法

BILBでは，参加レベルを宣言し，外部とやり取りする負荷を制限していた．詳しく言えば，参加レベルでキューの長さを制限することで，次に発生する自分のジョブの待ち時間を制限していた．異種計算グリッドでは同じ長さのキューでもクラスタごとに処理時間が変わるので，参加レベルを時間に置き換える．つまり，宣言した時間以内に処理が完了するようにジョブが外部から渡される．

つまり，ある参加レベルl_iを宣言した時，負荷分散後の負荷を処理するための時間がl_i以下に抑えられるようにする．クラスタM_iが参加レベルl_iを宣言すると，$L_i/s_i \geq l_i$のとき，クラスタM_iは外部から負荷を渡されない．また$L_i/s_i < l_i$のとき，$L_i/s_i > l_i$にならない程度の負荷を受け取る．

クラスタM_iが外部へ送ることができる負荷の合計は，クラスタM_iが時間l_iで処理できるだけの負荷$(=l_i s_i)$と制限する．

4.2 反復法

本報告における負荷分散の目的はジョブの処理時間の短縮としている．処理時間を短縮するには，自分の負荷を全て処理するための時間が長いクラスタから，短いクラスタへ負荷を送れば良い．つまり，負荷分散はL_iの大きさではなく，L_i/s_iの大きさを比較して行う．

つまり，L_i/s_iの小さい順に$M_1, \dots M_k, \dots, M_m$とクラスタを並べる．そして，$M_k(k=2,3,\dots,m)$は$M_1$から$M_{k-1}$に負荷をおくる．クラスタ$M_k$の処理時間を短縮するように，かつ，分散後の$M_1$から$M_k$の負荷の処理時間が等しくなるようにする．そのために，まずはM_kの負荷を分散した後の$M_i(1 \le i \le k-1)$の負荷$L'_{k,i}$を求める．

$$L'_{k,i} = \frac{s_i \sum_{j=1}^{k} L_j}{\sum_{j=1}^{k} s_j} \tag{4}$$

クラスタM_kは，M_iにそれぞれ$L'_{k,i} - L_i$だけの負荷を送っていく．この値は 3.2 節と同様に制限される．これを$k=2,3,\dots,m$の順に繰り返す．

4.3 シミュレーション

4 節で示したように，BILBを異種計算グリッドにおいて実行し，クラスタの性能の違いによって生じる挙動の違いを評価する．評価の対象は各クラスタがどれだけ負荷分散に協力するかと，負荷分散においてどれだけの利益を得られたかである．

4.3.1 シミュレーションの設定

異種計算グリッド

負荷分散に参加するクラスタ数は$m=5,10,15$とする．各クラスタの性能は$s_i = 1 + 1/(m-1)$で求められる．つまり各クラスタの性能は1と2の間で等間隔な値としている．

各クラスタで発生するジョブ$J_{i,j}$はポアソン過程に従い発生し，その大きさ$p_{i,j}$はガンマ分布に従うとする．異種計算グリッドの場合，ジョブのサイズや発生間隔は性能に応じて別の分布に従うことが予想される．従って，クラスタで発生するジョブのサイズ，ジョブの発生間隔について，次の二つの場合に分けてシミュレーションを行った．

I. クラスタの性能に応じてジョブのサイズが大きくなり，発生間隔が同じ
II. クラスタの性能に応じて単位時間あたりのジョブの発生個数が多くなり，サイズの期待値が同じ

(I)のとき，クラスタM_iのジョブのサイズは$\alpha_i = 2$，$\beta_i = 2s_i$のガンマ分布に従い，発生間隔は$\lambda_i = 0.2$の指数分布に従う．(II)のとき，クラスタM_iのジョブのサイズは$\alpha_i = 2$，$\beta_i = 2$のガンマ分布に従い，発生間隔は$\lambda_i = 0.2s_i$の指数分布に従う．どのクラスタも利用率$\rho = \lambda_i(\alpha_i \beta_i / s_i)$が同じである．

負荷分散の実行される間隔を$t=1$，参加レベルの変更が行われる間隔を$T=1000$としてシミュレーションを行う．また，参加レベルが500回変更されるまでシミュレーションを行う．

参加レベル

各クラスタ参加レベルをTごとに変更する．同じ参加レベルで作業をする一定時間Tを1ラウンドとする．最後に参加レベルを変化させた時刻をt_{prev}とする．時刻t_{prev}から1ラウンドの間，tごとにBILBによる負荷分散が実行される．クラスタは各ラウンドで得られた利益，または損失をもとにして次ラウンドの参加レベルを決めるとする．本シミュレーションでは，クラスタM_iは次の式を用いて参加レベルを変化させる．

$$l_i = l_i + \sigma l_i \delta (R_i - 1) \tag{5}$$

σは$\sigma \in (0,1]$を満たす係数で，本節では$\sigma = 0.02$としている．δ，R_iを求める手順を示す．まず，時刻$t_{prev} + 0.2T$から$t_{prev} + 0.8T$の間に，M_iで発生したジョブの処理時間の合計F_iを計算する．次に，t_{prev}から1ラウンドの間に負荷分散を行わなかったと仮定し，時刻$t_{prev} + 0.2T$から$t_{prev} + 0.8T$の間に，M_iで発生したジョブの処理時間の合計の予測$\widetilde{F_i}$を計算する．F_iと$\widetilde{F_i}$との比よりδ，R_iが求められる．詳しい算出方法は次の通りとなる．

$$\begin{cases} \delta = -1 & (F_i \geqq \widetilde{F_i} \text{のとき}) \\ \delta = +1 & (\widetilde{F_i} > F_i \text{のとき}) \end{cases} \tag{6}$$

$$\begin{cases} R_i = F_i / \widetilde{F_i} & (F_i \geqq \widetilde{F_i} \text{のとき}) \\ R_i = \widetilde{F_i} / F_i & (\widetilde{F_i} > F_i \text{のとき}) \end{cases} \tag{7}$$

評価指標

本シミュレーションでは，各クラスタがどれだけ負荷分散に協力するか，負荷分散によりどれだけの利益を得られるかを評価する．前者については，参加レベルが高いほどクラスタが協力的であると考えられるため，参加レベルを調べることで評価する．クラスタの利益の指標になるものとしてスコアを定義する．スコアとは負荷分散によってどれだけジョブの処理速度が上がったかを示す値である．BILBの効果を評価する際は，ある参加レベルにおけるクラスタごとで得られる利益を求める．スコアの求め方であるが，まず，1ラウンドの間，同じ参加レベルでtごとにBILBを行う．ラウンドの開始時刻はt_0とする．M_iで時刻$t_0 + 0.2T$から$t_0 + 0.8T$の間に発生したジョブの処理時間の合計F_iを計算する．次に，時刻t_0から1ラウンドの間，負荷分散を行わなかったと仮定し，M_iで時刻$t_0 + 0.2T$から$t_0 + 0.8T$の間に発生したジョブの処理時間の合計$\widetilde{F_i}$を計算する．そして，クラスタM_iの得られるスコアS_iはF_i，$\widetilde{F_i}$を用いた次の式で求められる．

$$\begin{cases} S_i = 1 - F_i / \widetilde{F_i} & \left(F_i \geqq \widetilde{F_i} \text{のとき}\right) \\ S_i = 1 - \widetilde{F_i} / F_i & \left(\widetilde{F_i} > F_i \text{のとき}\right) \end{cases} \tag{8}$$

$S_i = 1$はM_iのジョブの処理される速度が100%上がったことを示す．つまり，処理時間が負荷分散を行わない場合に比べて半分になっている．実際に評価する対象となるのは，同じレベルで100回シミュレーションを行った時の平均である．本報告では，参加レベルを1,10,100,1000に固定して，スコアを計測した．

4.3.2 シミュレーション結果

図:1～2 は(I)，(II)におけるクラスタ数5のときの参加レベルの変化を示している．性能の高い方のクラスタは低い

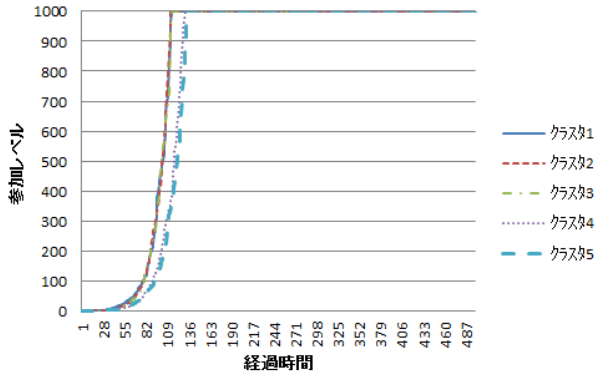

図: 1 (I)における参加レベルの変化

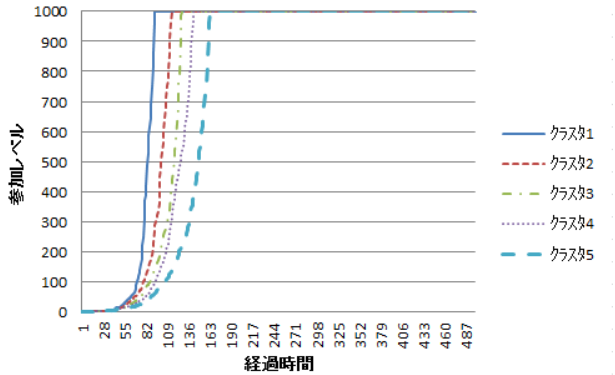

図: 2 (II)における参加レベルの変化

方に比べて参加レベルの上がり方が緩やかである．また，(I)では，性能の高いクラスタの参加レベルの上がり方がIIに比べて速くなっている．逆に，性能の低いクラスタの参加レベルの上がり方は，(I)では(II)に比べて少し遅くなっている．これらの結果より，各クラスタが得られるスコアは，(II)のとき，性能が高いほど低くなっていると考えられる．また(I)のとき，各クラスタが得られるスコアは性能ごとの差が小さいとも考えられる．実際に，各参加レベルにおいて得られるスコアの表でそれが確認できる．表:1〜6で(I)，(II)におけるクラスタ数5,10,15のときのスコアの変化を示す．

先ほど述べたように，(II)の結果では，クラスタの性能によって得られるスコアが違っている．(I)では，(II)に比べて性能の高いクラスタと低いクラスタの結果の差が小さくなっている．また，(I)では，性能の低いクラスタの得られるスコアが減っており，これは(I)では性能の低いクラスタの参加レベルの上昇が遅くなっていることに影響している．

4.2.3 考察

三つの場合に分けてシミュレーションを行ったが，(I)，(II)の間で結果に差が生じた．これはクラスタM_iで発生するジョブについて，到着時刻から処理が開始されるまでの平均時間W_iを計算することで説明がつく．W_iは，クラスタM_iが単位時間あたりに処理できるジョブの平均個数をμ_iとして次の式で求めることができる．

$$W_i = \frac{\lambda_i}{\mu_i} \frac{1}{(\mu_i - \lambda_i)} \tag{9}$$

W_iはある瞬間におけるキューに並んだジョブの処理時間と考えることができる．異種計算グリッドにおけるBILBはキューに並んだジョブの処理時間を基準に，ジョブを外部へ送る順番を決めるので，クラスタM_iがジョブを外部へ送ることができない可能性はW_iが大きいほど高くなる．

(I)，(II)の間で結果に差が生じた理由を述べる．

(I)のW_i

ジョブの単位時間あたりの平均発生個数λ_iは一定であるので$\lambda_i = \Lambda$とする．一方，発生するジョブの平均サイズはクラスタの性能に応じて大きくなっており，$\alpha_i \beta_i = C s_i$(Cは定数)で表わされ，単位時間あたりに処理できるジョブの平均個数μ_iは$\mu_i = 1/C$で求められる．(9) 式にこれを代入すると，

$$W_i = \frac{C^2 \Lambda}{1 - C\Lambda} \tag{10}$$

が得られる．s_iの値が変わってもW_iの値は変わらず，キューに並んでいるジョブの平均の処理時間もクラスタの処理能力にかかわらず一定である．つまり，s_iが増えても，外部からジョブを受け取る可能性は変わらない．

(II)のW_i

ジョブの単位時間あたりの平均発生個数λ_iはクラスタの性能に応じて多くなっており，$\lambda_i = \Lambda s_i$で表される．発生するジョブの平均サイズは同じであり$\alpha_i \beta_i = C$(Cは定数)とすると，単位時間あたりに処理できるジョブの平均個数μ_iは$\mu_i = s_i / C$で求められる．(9) 式にこれを代入すると，

$$W_i = \frac{C^2 \Lambda}{s_i (1 - C\Lambda)} \tag{11}$$

が得られる．s_iが増えるとW_iが小さくなることは明らかであり，キューに並んでいるジョブの平均の処理時間も短くなる．つまり，s_iが増えるほど，外部からジョブを受け取る可能性が増えていく． (10)，(11)式を比べると，各場合でs_iと受信者になる確率の関係が違うことが分かる．これが(I)，(II)の間で結果に差が生じた理由となる．

シミュレーションの結果，クラスタの性能によって参加レベルの上昇速度，スコアが異なることが示された．(I)において，その差はあまり大きくないが，(II)ではその差が顕著であった．次の節では，性能による参加レベルの上昇速度，スコアの差が小さくなるアルゴリズムを提案している．

l_i	S_1	S_2	S_3	S_4	S_5
1	2.455	2.384	2.366	2.305	2.386
10	3.623	3.553	3.468	3.403	3.348
100	3.872	3.729	3.492	3.435	3.199
1000	3.738	3.475	3.644	3.443	3.233

表: 1 (I)で，$m = 5$のときのスコア

l_i	S_1	S_2	S_3	S_4	S_5
1	2.434	2.390	2.550	2.554	2.395
10	5.019	4.239	3.564	3.121	2.813
100	5.044	4.234	3.857	3.106	2.783
1000	5.144	4.073	3.325	3.094	2.879

表: 2 (II)で，$m = 5$のときのスコア

l_i	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
1	2.811	2.967	2.865	3.113	2.835	2.791	2.850	2.843	2.732	2.862
10	6.352	6.556	6.790	6.454	6.559	5.970	6.239	5.988	5.759	5.966
100	6.539	6.549	6.646	6.525	5.639	6.402	6.189	5.847	6.015	5.617
1000	6.932	6.602	6.756	6.412	6.151	6.209	5.811	6.294	5.904	5.834

表: 3 (I)で，$m = 10$のときのスコア

l_i	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
1	2.802	2.847	3.064	2.871	2.937	2.922	2.942	3.013	3.082	3.146
10	8.148	7.353	6.414	6.254	5.790	5.764	5.648	5.005	5.025	4.637
100	8.017	7.060	6.932	6.097	5.977	5.602	5.046	5.162	4.950	4.494
1000	7.631	7.824	6.582	6.171	6.131	5.354	5.458	4.964	5.035	4.717

表: 4 (II)で，$m = 10$のときのスコア

l_i	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}
1	2.893	3.211	3.061	2.922	3.058	2.985	3.008	2.867	2.922	2.960	3.032	2.871	3.072	2.963	2.929
10	7.190	8.572	7.826	8.358	8.107	8.068	7.904	8.154	7.688	8.652	7.892	7.681	8.300	7.208	7.742
100	8.702	8.508	8.301	8.352	8.235	8.527	8.335	8.487	8.333	8.016	7.722	7.892	7.927	7.953	8.072
1000	8.925	9.120	9.004	8.412	7.717	7.619	8.297	7.805	8.353	7.995	8.017	7.946	7.720	7.676	7.437

表: 5 (I)で，$m = 15$のときのスコア

l_i	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}
1	3.060	3.039	3.032	3.045	3.086	3.036	3.089	3.031	3.163	3.102	3.250	3.181	3.304	3.174	3.326
10	10.45	9.244	9.036	8.534	8.436	7.683	7.647	7.035	6.798	7.340	6.372	6.360	6.045	5.835	5.711
100	10.22	8.805	8.518	8.502	7.920	7.532	7.566	7.479	6.726	6.575	6.520	6.497	6.585	6.105	5.622
1000	9.533	9.001	9.054	7.978	8.462	7.888	7.357	7.251	7.104	7.123	6.701	6.142	6.161	6.050	5.438

表: 6 (II)で，$m = 15$のときのスコア

5. BILB の拡張

5.1 提案アルゴリズム

4 節の結果で性能の高いクラスタは低い方に比べてスコアが低くなり，それに応じて参加レベルも上がり辛くなっていることが確認できた．(II)では，(I)に比べてそれが顕著に現れた．参加レベルやスコアが小さくなる理由として，性能が高いと負荷を外部へ送ることができる可能性が低いことが挙げられる．つまり，性能が高いクラスタはシステム内で最も負荷が小さい可能性が高い．よってシステム内で最も負荷が小さくてもそのクラスタのジョブの処理時間を短縮させる方法があれば，性能の高いクラスタでも参加レベルやスコアが高くなることが見込まれる．そこで性能の高いクラスタのスコアを高くする手法として，BILBによって一度割り当てられたジョブを，そのジョブの保証された完了時刻(以下，保証完了時刻)を基にスケジューリングを組み直す手法を提案する．

5.1.1 提案アルゴリズムの説明

まず，クラスタの参加レベルを基にBILBを実行する．これにより，キューに並んでいる各ジョブに対してそのクラスタの参加レベルに応じた保証完了時刻を決定することができる．提案アルゴリズムでは，各クラスタに対してBILBより大きな損失を与えることなくスケジュールを組み直す．すなわち，BILB実行後のスケジュールをS，組み直した後のスケジュールをS'とするとき，以下の条件を満たすようにする．

1) 各ジョブのS'における完了時刻がSにおける完了時刻より遅くならない．
2) 各クラスタのキューの長さがSとS'で等しい．

条件1と条件2より，負荷分散の実行時に存在するジョブと負荷分散の実行後に発生するジョブについて，各クラスタの損失がBILBより大きくならないことを保証している．提案アルゴリズムは，上記の条件を満たしながら，性能の高いクラスタのジョブの完了時刻をより早めるようにスケジュールを組み直す．

以降，提案アルゴリズムの詳細を説明する．BILBに参加したクラスタを$M_1,\dots,M_m$，BILBの実行後にクラスタM_iが持つキューの長さをL_i，M_iの性能をs_iとする．ジョブ$J_{i,j}$のサイズを$p_{i,j}$，BILB実行後の完了時刻（保証完了時刻）を$C'_{i,j}$とする．クラスタM_iが自分のキューに並んでいる全てのジョブを処理するのにかかる時間を$f_i = L_i/s_i$とする．説明の簡単化のため，$f_1 \geq,\dots,\geq f_m$であるとする．また，$f_{m+1} = 0$と定義する．

提案アルゴリズムでは，完了時刻の遅いジョブをキューの後ろから順に割り当てていく．割り当てたジョブの総量をΦとする．複数のクラスタへ割り当てられる場合，ジョブの開始時刻が同じになるように，全クラスタへ均等に割り当てていく．割り当て可能なクラスタを$M_1,\dots,M_l(1 \leq l \leq m)$とする．なお，$l$は$f_l > t' \geq f_{l+1}$となる値である．負荷分散の開始時点を時刻$t_r$としたとき，時刻$t_r + t'(0 \leq t' \leq f_1)$から時刻$t_r + f_1$の間に処理できるジョブの総量を$G(t')$とする．このとき，$G(t')$は以下のように定義される．

$$G(t') = \sum_{i=1}^{l}(f_i - t')s_i \qquad (12)$$

また，$G(t')$の逆関数を$G^{-1}(x)$で表す．
このとき，アルゴリズムは以下の通り動作する．

1. $\Phi = 0, t' = f_1, J' =${負荷分散の対象となる全てのジョブ}とする．
2. J'が空になるまで以下を繰り返す．

2-1. J'に含まれるジョブのうち，$C'_{i,j} \geq t'$で，発生させたクラスタのs_iが最も小さい$J_{i,j}$を選択する．

2-2. $J_{i,j}$の完了時刻をt'，開始時刻を$G^{-1}(\Phi + p_{i,j})$として，ジョブを$M_1,\dots,M_l$へ割り当てる．

2-3. $t' = G^{-1}(\Phi + p_{i,j}), \Phi = \Phi + p_{i,j}, J' = J'/\{J_{i,j}\}$とする．

2-1 では，$C'_{i,j} \geq t'$となるジョブのみを割り当ての対象として選択している．これにより，S'において，各ジョブがBILBの完了時刻より遅くならないことを保証している．また，そのようなジョブが複数存在する場合，s_iが最も小さいクラスタ，すなわち，最も性能の低いクラスタのジョブを選択している．このアルゴリズムでは先に選択されたジョブの完了時刻が遅くなるため，これにより性能の高いクラスタのジョブを優先的に実行することができる．

5.2 提案アルゴリズムのシミュレーション

5.1 節で説明した提案手法を 4 節のBILBの中に組み込み，4 節と同じ設定でシミュレーションを行った．

5.2.1 シミュレーション結果

クラスタ数5で，提案手法を実行した場合の参加レベルの変化を図:3～4 に示す．(I)，(II)いずれの場合においても，参加レベルの上昇速度が上がっていることが確認できる．(I)の方は上がり具合がわずかである．(II)においては，これが顕著であり，クラスタの性能による参加レベルの上昇速度の差が小さくなっていることもはっきりと確認できる．(II)では各クラスタで発生するジョブの数が多く，システム全体のジョブの数が多い．従って，同時に処理されているジョブの数も多い．

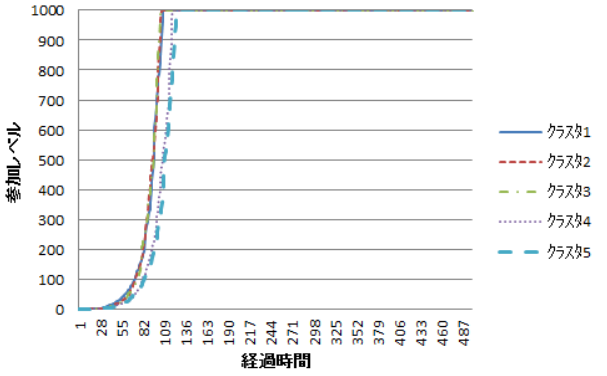

図: 3 (I)における参加レベルの変化

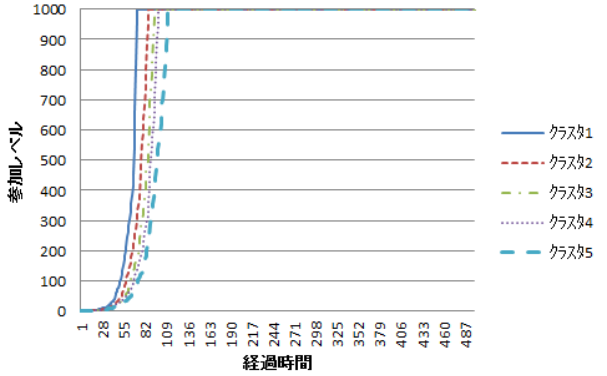

図: 4 (II)における参加レベルの変化

l_i	S_1	S_2	S_3	S_4	S_5
1	1.025	1.025	1.035	1.045	1.033
10	1.054	1.062	1.065	1.064	1.064
100	1.035	1.038	1.043	1.046	1.046
1000	1.036	1.039	1.044	1.044	1.047

表: 7 (I)で，$m = 5$のときのスコア

l_i	S_1	S_2	S_3	S_4	S_5
1	1.280	1.124	1.101	1.075	1.107
10	1.271	1.254	1.254	1.281	1.299
100	1.262	1.249	1.248	1.269	1.289
1000	1.291	1.252	1.283	1.284	1.298

表: 8 (II)で，$m = 5$のときのスコア

l_i	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
1	1.048	1.019	1.064	1.020	1.041	1.039	1.023	1.039	1.049	1.027
10	1.048	1.046	1.050	1.050	1.053	1.053	1.057	1.057	1.057	1.057
100	1.037	1.039	1.040	1.043	1.045	1.047	1.049	1.050	1.051	1.047
1000	1.038	1.037	1.038	1.043	1.045	1.047	1.045	1.051	1.050	1.047

表: 9 (I)で，$m = 10$のときのスコア

l_i	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
1	1.195	1.164	1.117	1.124	1.103	1.120	1.095	1.072	1.093	1.070
10	1.156	1.150	1.162	1.161	1.162	1.170	1.168	1.175	1.181	1.189
100	1.166	1.162	1.160	1.160	1.171	1.168	1.175	1.182	1.176	1.183
1000	1.178	1.174	1.176	1.177	1.172	1.185	1.182	1.192	1.189	1.189

表:10 (II)で，$m = 10$のときのスコア

l_i	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}
1	1.042	1.018	1.038	1.024	1.017	1.025	1.029	1.058	1.060	1.034	1.017	1.052	1.058	1.031	1.040
10	1.046	1.056	1.054	1.055	1.047	1.040	1.052	1.065	1.060	1.052	1.049	1.058	1.043	1.062	1.067
100	1.035	1.037	1.037	1.039	1.041	1.040	1.042	1.043	1.043	1.046	1.046	1.046	1.046	1.047	1.044
1000	1.035	0.989	1.038	1.038	1.039	1.042	1.042	1.044	1.045	1.044	1.023	1.048	1.046	1.048	1.044

表:11 (I)で，$m = 15$のときのスコア

l_i	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}
1	1.201	1.168	1.159	1.127	1.114	1.132	1.079	1.100	1.083	1.093	1.097	1.096	1.057	1.098	1.048
10	1.118	1.123	1.120	1.118	1.122	1.119	1.124	1.124	1.126	1.131	1.129	1.133	1.145	1.142	1.138
100	1.134	1.124	1.130	1.129	1.131	1.131	1.132	1.130	1.133	1.140	1.141	1.150	1.145	1.146	1.150
1000	1.117	1.120	1.115	1.117	1.116	1.116	1.120	1.123	1.123	1.125	1.128	1.127	1.131	1.132	1.135

表:12 (II)で，$m = 15$のときのスコア

複数のクラスタが複数のジョブを並行して処理しているスケジュールを考える．このスケジュールを，複数のクラスタでジョブを一つずつ同時に処理するように変えると，どのジョブの完了時刻も遅くすることなく，いくつかのジョブの完了時刻を早くすることができる．(II)においては，複数のジョブを並行して処理するのを，一つずつ同時に処理していくスケジューリングに変えやすいので，結果が(I)よりも良くなったと考えられる．

表:7〜12 で提案手法を行ったときに得られたスコアが，BILBに比べてどれだけ大きくなったかを示す．(I)において，3~7%ほどのスコアの上昇が確認できる．(II)においては，少々スコアの上昇が大きく，10~30%の上昇が確認できる．しかし，クラスタの性能が上がれば，上昇率が上がるといったことははっきりと確認できない．

6. おわりに

本報告では，RzadcaらによるBILBアルゴリズムを異種計算グリッドへ適用し，各クラスタの協力の度合いや，得られるスコアを評価した．そして，性能の高いクラスタは性能の低いクラスタ程ではないものの，負荷分散システムに協力し，利益も得られることが確認できた．また，これを改良しさらに協力の度合いを増し，高いスコアの得られる手法の提案を行った．具体的には，BILBによって保証終了時刻を基に，スケジュールを組み直すというものであった．

本報告で提案した手法でシミュレーションを行った結果，各クラスタの得られる利益の上昇やそれに伴った協力度の上昇という結果が得られた．今回は，BILBを実行した際に決まった保証完了時刻を基にスケジュールを組み直したが，BILBよりも良いアルゴリズムが存在する可能性がある．例えば，この手法と組み合わせることで，クラスタごとのスコアや協力度を完全に等しくするようなアルゴリズムが考えられる．そのようなアルゴリズムと今回の手法を組み合わせることを今後の課題としたい．

参考文献

[1] I. Foster, C. Kesselman (Eds.), The Grid 2. Blueprint for a New Computing Infrastructure,Elsevier, 2004.

[2] G. Christodoulou, E. Koutsoupias, and A. Nanavati. Coordination mechanisms. In Proceedings of the 31st International Colloquium on Automata, Languages and Programming, pp. 345-357, 2004.

[3] E. Koutsoupias and C. Papadimitrious. Worst-case equilibria. In Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science, pp. 404-413, 1999.

[4] A. S. Schulz and N. Stier. On the performance of user equilibria in traffic networks. In Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 86-87, 2003.

[5] E. Anshelevich, A. Dasgupta, J. Kleinberg, ´E. Tardos, T.Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. In Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 295-304, 2004.

[6] F. Pascual, K. Rzadca, and D. Trystram. Cooperation in multi-organization scheduling. Concurrency and Computation: Practice and Experience, 21(7), pp. 905-921, 2009.

[7] F. Ooshita, T. Izumi, and T. Izumi. A generalized multi-organization scheduling on unrelated parallel machines. In Proceedings of the International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 26-33, 2009.

[8] J. Cohen, D. Cordeiro, D. Trystram, and F. Wagner. Analysis of multi-organization scheduling algorithms. In Proceedings of the International Euro-Par Conference, pp. 367-379, 2010.

[9] K. Rzadca and D. Trystram. Promoting cooperation in selfish computational grids. European Journal of Operational Research, 199, pp. 647-657, 2009.