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

2005－HPC－103（10）
2005／8／3

グリッド環境における野球チームの最適打順決定手法の高速化

大澤　清[†]　合田憲人[†]

本稿では野球チームの最適打順を決定する計算をグリッド環境上で行う際の高速化手法について述べる．期待得点を最大にする最適打順を決定するには D'Esopo and Lefkowitz モデルに基づく確率計算を多数の打順それぞれについて行う必要があるが，この計算を Ninf-G を用いてグリッド環境上で並列に実行した．さらに計算パラメータの再利用により打順の組合せ数を 1/9 に削減して計算の効率化を行うとともに，各 PC に割り当てられる打順の組合せ数を実行時に決定して計算資源の有効利用を図った．その結果，2 拠点の PC クラスタを利用して 27,216,000 通りの打順の組合せについて期待得点を計算した結果，3,278 秒で最適な打順を決定することが可能となった．

A Speedup Technique for Finding an Optimal Batting Order on the Grid

Kiyoshi Osawa[†] and Kento Aida[†]

In this paper, we propose a speedup technique of computation for finding an optimal batting order in a baseball team on the Grid. The proposed technique parallelizes computation for performances of batting orders, where probabilities to obtain scores by the batting orders are computed using the D'Esopo and Lefkowitz model, using Ninf-G. In addition, the proposed technique improves the performance by reusing parameters about batting orders. On a Grid environment, load balancing is appropriately performed considering performances of computing resources. The experimental results show that the proposed technique finds the optimal batting order in 27,216,000 batting orders in 3,278 seconds on the Grid testbed.

1. はじめに

野球チームの最適打順を決定する手法としてマルコフ連鎖を用いた手法[1)] が知られている．この手法では，分析対象の球団から打者 9 人を選出し，構成し得るすべての打順について D'Esopo and Lefkowitz モデルを適用してその期待得点を算出する．しかし，解を得るためには 9!=362,880 通りの打順について期待得点を算出する必要があり，その中から最大値をとる打順を最適打順とするため，その計算量が単体の PC にとっては比較的大きいという問題がある．例えば文献 1) では，一部の攻撃力の低い打者を打順内で固定して打順の組合せ数を減らして計算時間を短縮し，その中で最大値をとる打順を最適打順とみなす手法が採られている．しかしチームの勝敗を予測する上で，その手法により得られる打順の期待得点と真の最適打順の期待得点との誤差による影響は，試合数が増えるにつれ大きくなる恐れがある．

本稿ではすべての打順の組合せについて期待得点を算出して真の最適打順を決定する手法の計算時間を短縮することを目的とし，その手法に対して以下の高速化を適用した結果について報告する．

- 打順の循環に着目した計算パラメータの再利用
- 守備位置の充足可能性判定
- 計算の並列化と実行時負荷分散

2 サイトに分散配置されたグリッド環境においてこれらの手法を適用して最適打順を探索した結果，27,216,000 通りの打順の組合せに対して 55 分程度の計算時間で最適打順とその期待得点を得られることが確認された．

2. 最適打順決定手法

2.1 D'Esopo and Lefkowitz モデル

D'Esopo and Lefkowitz モデルでは野球の攻撃をアウト数と走者状況毎で分類し，25 通りの状態を定義する (アウト数 3 種類 (無死，一死，二死)× 走者状況 8 種類 (無走者，一塁，二塁，三塁，一二塁，一三塁，二三塁，満塁)+ 三死における攻撃終了状態)．これより各行が “ある打者が打席に入った時点の状態” を表し，各列が “ある打者が打撃を行った結果の状態” を表す 25×25 状態遷移行列 P が定義される．さらに各打者の打撃による走者の進塁状況とその確率を以下

† 東京工業大学
Tokyo Institute of Technology

のように定義する．

- 凡打 (Out)：走者は進塁せず，アウト数を一つ増やす (確率：p_O)
- 四球 (Walk)：押し出される走者のみ一つ進塁 (p_W)
- 単打 (Single)：一塁走者は二塁へ，その他の塁の走者は生還 (p_S)
- 二塁打 (Double)：一塁走者は三塁へ，その他の塁の走者は生還 (p_D)
- 三塁打 (Triple)：すべての塁の走者が生還 (p_T)
- 本塁打 (Homerun)：すべての塁の走者と打者が生還 (p_H)

これより行列 P は部分行列 A, B とベクトル F より

$$P = \begin{pmatrix} A & B & 0 & 0 \\ 0 & A & B & 0 \\ 0 & 0 & A & F \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1}$$

と定義される．打者が打席に入った時点の状態を表す P の各行は第 1 行から第 8 行が無死，第 9 行から第 16 行が一死，第 17 行から第 24 行が二死，第 25 行が三死の攻撃終了状態に対応する．打撃を行った結果を状態を表す各列も同様となる．このとき 8×8 行列 A, B と 8×1 ベクトル F は

$$A = \begin{pmatrix} p_H\, p_S + p_W\, p_D\, p_T & 0 & 0 & 0 & 0 \\ p_H & 0 & 0\ p_T\, p_S + p_W & 0 & p_D & 0 \\ p_H & p_S & p_D\, p_T & p_W & 0 & 0 & 0 \\ p_H & p_S & p_D\, p_T & 0 & p_W & 0 & 0 \\ p_H & 0 & 0\ p_T & p_S & 0 & p_D\, p_W \\ p_H & 0 & 0\ p_T & p_S & 0 & p_D\, p_W \\ p_H & p_S & p_D\, p_T & 0 & 0 & 0 & p_W \\ p_H & 0 & 0\ p_T & p_S & 0 & p_D\, p_W \end{pmatrix}$$

$$B = p_O I$$

$$F = (p_O, \ldots, p_O)^T$$

である．I は単位行列を，^{T} はベクトルの転置をそれぞれ表す．

A, B の各行，各列はそれぞれ上端，左端から “無走者，一塁，二塁，三塁，一二塁，一三塁，二三塁，満塁” の状態に対応する．A は打者の打撃結果が安打または四球である場合の遷移確率を表し，このときアウト数は増えることなく走者は生還または進塁する．例えばある打者が無死無走者の状態で打席に入った場合，打撃結果により無死一塁の状態に遷移する確率は第 1 行第 2 列の要素に対応し，D'Esopo and Lefkowitz モデルにおいてその状態遷移を実現する打撃結果は単打もしくは四球であるためその値は $p_S + p_W$ となる．式 (1) ではアウト数によらず打者の各打撃結果の確率は一定と仮定しているため，一死，二死における状態遷移確率も A により表される．B は無死，一死における打者の打撃結果が凡打の場合の遷移確率を表し，このときアウト数は一つ増え，走者は進塁しない．ベクトル F は凡打により二死から攻撃終了状態である三死に遷移する確率を表す．実際の野球における併殺打や凡打の間に走者が進塁する打撃については，今回は簡単のために考慮していない．

N 人の打者の集合を $S = \{b_1, b_2, \ldots, b_k, \ldots, b_N\}$($k$ は打者インデックス) とすると，打者毎に過去の成績から求められる各確率 $p_O, p_W, p_S, p_D, p_T, p_H$ を用いて状態遷移行列 P_k を定義し，それらを打順に従って掛け合わせることで攻撃の状態遷移がシミュレートできる．

2.2 1 イニング中に挙げる期待得点算出手法

1 イニング中に挙げる期待得点を状態遷移行列 P_k を用いて算出する方法を以下に示す．P_k を $P_k^{(0)}, P_k^{(1)}, P_k^{(2)}, P_k^{(3)}, P_k^{(4)}$ に分解する．$P_k^{(r)}(r = 0, 1, 2, 3, 4)$ は P_k の各確率のうち，打者 k の打撃により得点が r 点入る要素を抜き出したもので，

$$P_k = P_k^{(0)} + P_k^{(1)} + P_k^{(2)} + P_k^{(3)} + P_k^{(4)}$$

が成り立つ．1 イニング中に挙げる得点を行，アウト数および走者状況で定まる 25 通りの状態を列とする $(R_{max} + 1) \times 25$ 行列 U を得点と状態の確率分布を表す行列とする．ここで R_{max} は 1 イニングで挙げ得る最大得点とする．攻撃開始時すなわち 0 人の打撃終了時においては無得点，無走者であることから U の初期値 U_0 は第 1 行第 1 列のみが 1 であり，残りの成分はすべて 0 の行列となる．イニング開始から n 人の打撃終了時における U を U_n で表すと，これは以下の式で定められる．

$$U_n|_i = \sum_{r=0}^{4} U_{n-1}|_{(i-r)} P_k^{(r)} \quad (i = 1, 2, \ldots, R_{max}+1) \tag{2}$$

$U_n|_i$ は U_n の第 i 行を表し，k は打者インデックスである．k は n とともにインクリメントされるが，周期 9(=打者数) でループする．このようにして U_n を定めると，$n \to \infty$ のとき U_n の第 25 列 (右端) の要素の総和は限りなく 1 に近付く．このとき $u_{i,j}$ を行列 U の第 i 行，第 j 列の成分とすると 1 イニング中に挙げる期待得点 ER は

$$ER = \sum_{k=1}^{R_{max}+1} (k-1) \times u_{k,25}$$

で表される．打者の集合 S より打順を構成し，各打者について状態遷移行列 P_k を定義して上記の計算を行うことで 1 イニング中に挙げる期待得点 ER が算出できる．

2.3 1 試合中に挙げる期待得点算出手法

あるイニングの先頭打者が i 番打者で，その次のイニングの先頭打者が j 番打者である確率を t_{ij}，その場合の期待得点を e_{ij} とする (図 1)． これらの値は式 (2) の乗算毎に U_n 最右端の列の要素から算出することができる．このとき m 回の攻撃が n 番打者で始まる確率 $p_{m,n}$ とその場合の初回からの合計得点 $a_{m,n}$ は

$$p_{m,n} = \begin{cases} 1 \quad (m=1,\, n=1) \\ 0 \quad (m=1,\, n=2,3,\ldots,9) \\ \displaystyle\sum_{k=1}^{9} p_{m-1,k} t_{kn} \\ (m=2,3,\ldots,10,\, n=1,2,\ldots,9) \end{cases} \tag{3}$$

$$a_{m,n} = \begin{cases} 0 \quad (m=1,\, n=1,2,\ldots,9) \\ \displaystyle\sum_{k=1}^{9} \frac{p_{m-1,k}}{p_{m,n}} t_{kn}(a_{m-1,k} + e_{kn}) \\ (m=2,3,\ldots,10,\, n=1,2,\ldots,9) \end{cases}$$

で表される．これらより 1 試合の期待得点は $\displaystyle\sum_{k=1}^{9} p_{10,k} a_{10,k}$ で得られる．

3. 期待得点算出手法の高速化

本稿では，期待得点算出手法について以下に示す高速化を提案する．

3.1 計算パラメータの再利用による高速化

野球では 9 人の打者が循環して打席に立つことを考慮すると，集合 S 内の打者を b_k で表した場合に例えば打順 $b_1, b_2, \ldots, b_9$ について求めた t_{ij} と e_{ij} は循環シフトした打順 $b_2, b_3, \ldots, b_9, b_1$ の期待得点

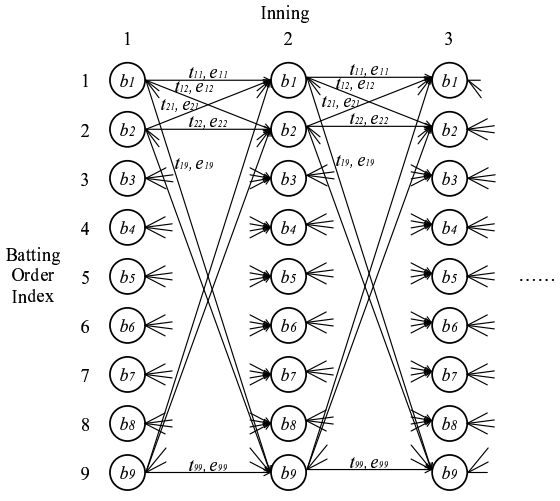

図 1 パラメータ t, e の定義

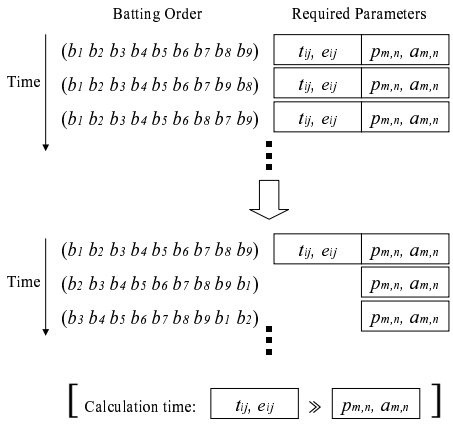

図 2 パラメータの再利用による高速化

を求める際に再利用可能である．具体的には式 (3) で $p_{1,1} = 1, p_{1,n} = 0\,(n = 2,3,\ldots,9)$ として求めていた $p_{m,n}\,(m = 2,3,\ldots,10,\, n = 1,2,\ldots,9)$ を $p_{1,1} = 0, p_{1,2} = 1, p_{1,n} = 0\,(n = 3,4,\ldots,9)$ に変更して求めることで t_{ij} と e_{ij} の再計算が不要になる．同様に打順 $b_3, b_4, \ldots, b_9, b_1, b_2$ 等の期待得点を求める際にも t_{ij} と e_{ij} を再利用可能である．$p_{m,n}$ と $a_{m,n}$ の計算コストは t_{ij} と e_{ij} のそれに比べて小さく，t_{ij} と e_{ij} を共通して使用できる打順は 9 通りであるため，これらの打順について連続して期待得点の算出を行うことで，すべてのパラメータを求める場合と比較して 9 倍程度の速度向上が期待される (図 2)．

3.2 守備位置の充足可能性判定

前節までに述べた期待得点の算出方法を単純に実装した場合，打者の集合 S の要素数すなわち打者数 N が打順を構成し得る最小の値 (=9) の場合においても 9!=362,880 通りのすべての打順の組合せについて期待得点を算出する必要があった．3.1 節で示したパラメータの再利用による高速化手法を用いるとその計算量はおよそ 1/9 に削減されるが，打者数を増やしてより大規模な問題を扱う場合，$N = 10$ 人の場合は ${}_{10}P_9 = 3,628,800$ 通り，11 人の場合は 19,958,400 通り，12 人の場合は 79,833,600 通りの打順について計算することになり，計算量が大きくなることは避けられない．しかし実際の野球において打者は指名打者を除き守備を行う必要があり，各打者について技術的に守れる守備位置と守れない守備位置が存在するため，S から 9 人を選出した際，守備が破綻しないようにする必要がある．S から選出されたある打者の組合せですべての守備位置が充足可能かどうかを判定するため，各打者について守れる守備位置を示すパラメータを定める．期待得点を算出する前にその組合せに含ま

Batting Order	Position								
	P	C	1B	2B	3B	SS	LF	CF	RF
b_3	0	1	1	0	0	0	0	0	1
b_5	0	0	0	0	1	0	0	0	0
b_9	0	0	0	0	0	0	1	0	1
b_1	1	0	0	0	0	0	0	0	0
b_4	0	0	0	1	1	1	0	0	0
b_{10}	0	0	0	0	0	0	1	0	0
b_2	0	1	0	0	0	0	0	0	0
b_8	0	0	0	0	0	0	1	1	1
b_6	0	0	0	0	0	1	0	0	0

図 3　ある打順における打者が守備可能なポジション

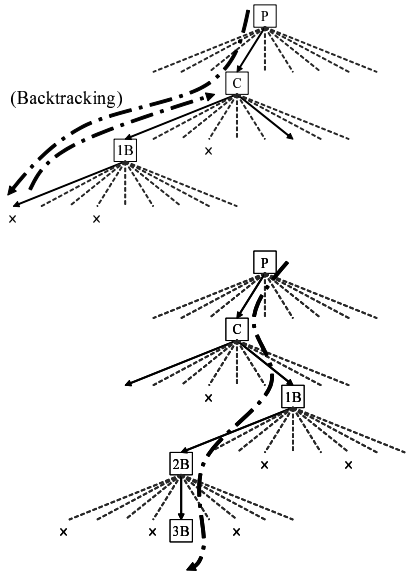

図 4　守備位置の充足可能性判定

れる打者ですべての守備位置が充足可能かどうかを判定し，可能な場合のみ期待得点を算出することで，計算量の増加を抑えることが可能になる．以下にその判定を行うための基本的な考えについて述べる．

3.2.1　守備可能なポジションを示すパラメータ

各打者について各守備位置に就くことが可能かどうかを表すパラメータ列を用意し，守備可能の場合は 1，そうでない場合は 0 を与える．守備力 (守備範囲，肩の強さ，捕球の確実性) については考慮しない．ある打順に沿ってパラメータ列を並べると図 3 のような表が得られる．

3.2.2　守備位置の充足可能性判定手法

図 3 の表から守備位置の充足可能性判定を行う手法を以下に示す．表において投手の守備位置を表す左端の列を上から走査し，守備可能を表す「1」が i 行目に現れた場合は図 4 において「P」で示される投手のノードの左から i 番目の枝 (図 3 の表の上から i 番目の打者に対応する) が利用可能であることを示す．利用可能な枝は実線の矢印で，そうでない枝は点線で示されている．図 4 では投手のノードの左から 4 番目の枝から捕手のノードに移ったため，捕手のノードから 4 番目の枝を再び利用しないように「×」のラベルを付与する．枝を辿った先の「C」で示される捕手のノードにおいて同様の処理を行い，これを繰り返す．図 4 では「1B」で示される一塁手のノードにおいて利用可能な枝が存在しない (利用可能な枝に「×」のラベルが付与されている) ため，N-Queens 問題等で用いるバックトラッキングの手法を用いて捕手のノードまで戻り，再び処理を繰り返す.「RF」で示される右翼手のノードにおいて打者が割り当て可能の場合，その打順は守備位置を充足可能であると判定される．逆にバックトラッキングを繰り返し，最上位にある投手のノードにおいて利用可能な枝が存在しなくなった場合は，充足不可能であると判定される．

守備位置を考慮しつつ最適打順を決定する従来手法[4] では，あらかじめ人間が守備位置の充足性を判定した上で期待得点の計算を行っていたため，多くの組合せについて最適打順を決定するのは困難であった．本研究では以上の手法を用いることで守備位置の充足性をプログラムにより判定し，かつ人数の増加による組合せ数の増加を抑えることが可能となった．

3.3　最適打順決定計算の並列化と実行時負荷分散

個々の打順についての計算は独立に実行できるため，複数の計算機に割り当てることにより計算時間の短縮が期待できる．集合 S から構成し得る打順がすべて守備位置を充足し，かつ計算プラットフォームが単一拠点内の均一な計算ノードから構成される PC クラスタ上である場合，打順毎の期待得点計算時間に大きな差は無いため，組合せ数を計算ノード数で均等に分割して割り当てれば負荷は均衡する．しかし実際はすべての打順について守備位置を充足するとは限らず，また規模の大きな問題を扱うために複数拠点内の PC クラスタを用いて計算を行う場合は計算ノードの単体性能が拠点間で異なるため，負荷の均衡を図る必要がある．

本稿では，次に示す実行時負荷分散方式を実装した．初めに少量の計算を各計算ノード上で実行することにより計算ノードの性能を評価し，その結果をもとにして全計算の各計算ノードへの割り当て量を決定する．具体的には以下の手順で負荷分散を行った．

最初に N 人の打者の集合 S から構成し得る ${}_NP_9$ 通りの打順のうち，守備位置を充足する打順の数 $s({}_NP_9)$ を求める．ただし，3.1 節で示したパラメータの再利用により，集合 S から構成し得る打順のうち循環シフ

トして一致する打順について守備位置の充足判定を行う必要は無く，実際には ${}_NP_9/9$ 通りの打順について判定を行い $s({}_NP_9/9)$ を求める．

実際の期待得点算出の前に，各計算ノードにおいて $f \times s({}_NP_9/9)/$(全計算ノード数) 通り ($f < 1$) の打順について期待得点を算出し，その計算に要した時間 $T(k,l)$(拠点 k の計算ノード l における計算時間) を 1 台のノードに集約する．

各ノードの計算時間を集約したノードにおいて，式 (4) により各ノードに割り当てる打順の数 $c(k,l)$ を決定する．

$$S_t = \sum_{k,l} \frac{1}{T(k,l)}$$

$$c(k,l) = s({}_NP_9/9) \times \frac{1/T(k,l)}{S_t} \quad (4)$$

ここで，各計算ノードの計算性能をより正確に取得するために，$T(k,l)$ には全処理時間のうち認証処理や守備位置充足判定処理等を除いた期待得点計算処理に要する時間を用いる必要がある．これは，ある割合 f についてのみ期待得点を算出しているため，その全処理時間のうち期待得点処理に要する時間以外の時間の比率が比較的高く，それらを含めた時間を $T(k,l)$ として 1 台のノードに集約し $c(k,l)$ を定めると結果的に負荷の均衡が図れない場合が生じるためである．再び各計算ノードに $c(k,l)$ 通りの打順を割り当て，各計算ノード間で負荷の均衡が取れた状態で期待得点を算出することで最適打順決定処理の高速化が期待できる．現在の実装においては各計算ノードにおいて一部の (割合 f の) 打順について期待得点の計算を行い，割り当てられる打順の数 $c(k,l)$ を決定した後に再びすべての計算すべき打順の組合せ $s({}_NP_9/9)$ 通りについて期待得点の計算を行っている．既に計算した一部の打順についてはそれまでの期待得点を最大にするような打順を保存しておくなどして，再び期待得点の計算を行わないようにする高速化手法も考えられる．

4. 性能評価

性能評価に用いたグリッド環境は東京工業大学の PC クラスタ (Blade クラスタ☆，計算ノード 36 台) と産業技術総合研究所の PC クラスタ (F32 クラスタ☆☆，計算ノード 64 台) で構成される．同環境上に D'Esopo and Lefkowitz モデルを実装し，GridRPC ミドルウェアである Ninf-G2.3.0[2] を用いて並列化を行った．計算のカーネル部分となる行列ベクトル積計算には性能の自動チューニングが施された数値計算ライブラリの ATLAS[3] を用いた．

☆ 各計算ノードの CPU は Pentium III 1.4GHz，メモリは 512MB，OS は Red Hat Linux 7.1(Kernel 2.4.10)，コンパイラは gcc 2.96 を使用し最適化オプションとして-03 を指定した．

☆☆ 各計算ノードの CPU は Xeon 3.06GHz，メモリは 4GB，OS は Red Hat Linux 8.0(Kernel 2.4.24)，コンパイラは gcc 3.3.3 を使用し最適化オプションとして-03 を指定した．

表 1 計算パラメータの再利用による高速化の効果

再利用なし	再利用あり	速度向上比
3,258(sec)	370(sec)	8.81

4.1 計算パラメータの再利用による高速化の効果

計算パラメータの再利用による高速化の効果を示すために，F32 クラスタにおいて再利用を行った場合と行わなかった場合の実行時間を比較した．集合 S に含まれる打者数は 10 人とし，再利用による効果のみを測定するためにすべての打順の組合せにおいて守備位置を充足するような打者を用いた．40 台の計算ノードを使用して測定した結果を表 1 に示す．

3.1 節で述べた通り，$p_{m,n}$ と $a_{m,n}$ の計算コストは t_{ij} と e_{ij} のそれに比べて非常に小さく，循環シフトにより一致する 9 通りの打順について t_{ij} と e_{ij} を共通して使用できるため，9 倍に近い速度向上が得られた．

4.2 守備位置充足性判定と実行時負荷分散による高速化の効果

実行時負荷分散による高速化の効果を示すために，Blade クラスタの計算ノード 30 台と F32 クラスタの計算ノード 40 台を用いて検証を行った．集合 S に含まれる打者として，2005 年 6 月 23 日の時点で好成績を収めているセントラル・リーグの打者の中から守備位置を勘案して 12 人を選出した．選出された打者の打撃成績，守備可能位置を表すパラメータと期待得点を計算した結果得られた最適打順を図 5，図 6 と表 2 に示す．

12 人の打者により構成可能な打順の組合せ数は ${}_{12}P_9 = 79,833,600$ 通りであり，循環シフトにより一致する打順を同一とみなすと ${}_{12}P_9/9 = 8,870,400$ 通りである．このうち図 6 のパラメータより守備位置を充足する打順の組合せ数は 3,024,000 通り ($=s({}_{12}P_9/9)$) となる．ここで，期待得点を算出する打順の組合せ数は最終的に $s({}_{12}P_9) = 27,216,000$ 通りになることに注意されたい．これは各計算ノードに割り当てられる打順の組合せ数は $s({}_{12}P_9/9)$ であるが，各計算ノードにおいて 1 通りの打順から循環シフトした 9 通りの打順の期待得点が得られるためである．

実行時負荷分散を行わずに計算を行った場合と 3.3 節で述べた手法を用いて負荷分散を行った場合の実行

打者	打率	本塁打	長打率	出塁率
b_1	.284	8	.436	.351
b_2	.305	7	.480	.349
b_3	.222	16	.472	.336
b_4	.296	5	.389	.326
b_5	.298	13	.496	.363
b_6	.276	19	.569	.374
b_7	.280	3	.366	.360
b_8	.324	1	.421	.402
b_9	.333	17	.632	.435
b_{10}	.344	21	.688	.418
b_{11}	.340	15	.582	.401
b_{12}	.304	8	.480	.406

図 5　12 人の打者の打撃成績

Batting Order \ Position	P	C	1B	2B	3B	SS	LF	CF	RF
b_1	1	1	0	0	0	0	0	0	0
b_2	1	0	1	0	0	1	0	0	0
b_3	1	0	1	0	0	0	0	0	0
b_4	1	0	0	1	0	0	0	0	0
b_5	1	0	0	0	1	0	0	0	0
b_6	1	0	0	0	1	0	0	0	0
b_7	1	0	0	0	0	1	0	0	0
b_8	1	0	0	0	0	0	0	1	0
b_9	1	0	0	0	0	0	1	0	0
b_{10}	1	0	0	0	0	0	0	1	1
b_{11}	1	0	0	0	0	0	1	0	1
b_{12}	1	0	0	0	0	0	0	1	1

図 6　12 人の打者が守備可能なポジション

表 2　2005 年 6 月 23 日時点の最適打順

打順	打者	打率	本塁打
1	b_{12}	.304	8
2	b_9	.333	17
3	b_{11}	.340	15
4	b_{10}	.344	21
5	b_6	.276	19
6	b_1	.284	8
7	b_3	.222	16
8	b_2	.305	7
9	b_4	.296	5
期待得点: 6.88			

表 3　実行時間の内訳 (単位：秒)

実行時 負荷分散	試行	実際	その他	合計	速度 向上比
なし	0	4,441	91	4,532	-
あり	618	2,503	157	3,278	1.38

時間とその内訳を表 3 に示す．なお，実際の計算の前に拠点毎のクラスタの計算性能を取得するため，実際の計算の 5%($f = 0.05$) の組合せ数について計算を行っている．実行時負荷分散を行わない場合は，実際の計算に先立って行う計算自体を行わず，単純に打順の組合せ数 $s({}_{12}P_9/9)$ を全計算ノード数 70 で割った組合せ数の打順を各計算ノードに割り当てている．

実行時負荷分散により各計算ノードに割り当てられた打順の組合せ数をクラスタ毎に平均した値と平均実行時間，クラスタ内での最大実行時間を表 4 に示す．

表 4 より実行時負荷分散を行うことで 2 クラスタ間の平均実行時間が近い値となり，全体の実行時間を短縮することが確認できる．

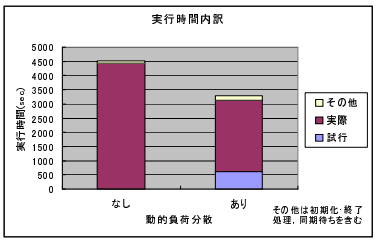

図 7　実行時負荷分散による高速化の効果

表 4　負荷分散の結果

	Blade	F32
平均割り当て数	23,597	57,903
平均実行時間 (sec)	2,196	2,013
最大実行時間 (sec)	2,503	2,213

5. ま と め

本稿では，グリッド環境において野球チームの最適打順を決定する計算の高速化手法とその性能評価について報告した．本手法を適用した結果，計算パラメータの再利用により 8.81 倍の高速化が図られ，さらに実行時に負荷分散を行うことで各拠点毎の実行時間が平準化され，拠点の計算性能を取得するオーバヘッドを含めても手法を適用しない場合に比較して 1.38 倍の高速化が図られた．以上の高速化を適用し，2 拠点の PC クラスタを用いて 12 人の打者の集合からなる 27,216,000 通りの打順について期待得点を計算した結果，3,278 秒で最適打順を決定することが可能となった．

謝辞 性能評価に御協力頂いた産業技術総合研究所グリッド研究センターに感謝致します．

参 考 文 献

1) Bukiet, B., Harold, E. and Palacios, J.: A Markov Chain Approach to Baseball, *Operations Research*, Vol. 45, No. 1, pp. 14–23 (1997).
2) Tanaka, Y., Nakada, H., Sekiguchi, S. and Matsuoka, S.: Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing, *Journal of Grid Computing*, Vol. 1, No. 1, pp. 41–51 (2003).
3) Whaley, R. C.: Automatically Tuned Linear Algebra Software(ATLAS), http://math-atlas.sourceforge.net/.
4) 廣津信義, 宮地力: 野球チームのラインナップ選定のための数理的一手法—日本代表チームの選定を例として—, オペレーションズ・リサーチ, Vol. 49, No. 6, pp. 380–389 (2004).