Speed-up Techniques for Computation of Markov Chain Model to Find an Optimal Batting Order

Kiyoshi Osawa Kento Aida
{osawa,aida}@alab.ip.titech.ac.jp

Abstract

In this paper, we propose speed-up techniques for computation of the Markov chain model to find an optimal batting order in a baseball team. The proposed technique parallelizes computation of the Markov chain model for batting orders, where probabilities to obtain scores by the batting orders are computed using the D'Esopo and Lefkowitz model, on the Grid. In addition, the proposed technique improves the performance by sharing 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 for 3,278 seconds on the Grid testbed.

1 Introduction

Grid computing is new computing technology, which provides huge computational power with low costs by employing computing resources geographically distributed over the internet. The high cost-effectiveness drives researchers to solve large-scale problems, which have not been solved before due to lack of computational power. Particularly, researchers in the operations research community, which are not deeply related to high-performance computing, begin to take an interest in the Grid computing technology.

The problem of finding an optimal batting order is to solve a combinatorial optimization problem[4] to find the best batting order, which is expected to yield the maximum score in a baseball game. This problem essentially needs huge computation time. For instance, even for a small baseball team, which consists of nine players, we need to compute expected runs for 9!=362,880 batting orders. Thus, conventional schemes, such as a Malkov chain approach[1], are able to get the only near-optimal solution for small-scale problems of unrealistic problem size. The optimal solution for the large-scale problem has not been computed.

This paper proposes speed-up techniques for computation of the Markov chain model to find an optimal batting order. The proposed techniques reduce computation time of the problem by performing:

- reduction of computation by sharing common parameters to compute expected runs among multiple batting orders
- reduction of computation by solving the satisfiability problem about batter's capability for fielding positions.
- parallelization of computation on the Grid using GridRPC

The experimental results on the Grid testbed show that the proposed techniques significantly reduce the computation time. Also, the proposed techniques demonstrate that the optimal batting order of the baseball team, consisting of 12 players selected form Japanese professional baseball teams, is obtained for 3,278 seconds on the Grid.

2 The Method to Find an Optimal Batting Order

2.1 D'Esopo and Lefkowitz Model

In the D'Esopo and Lefkowitz model[1], 25 offensive states are defined in a half-inning. These states correspond to the combination of the number of outs (three possibilities: zero, one, or two outs) and the occupation of bases (eight possibilities: none, a runner on only first base, runners on first and second base, and so forth), and the end of the half-inning when the third out occurs[5]. The state transition occurs when a batter completes the plate appearance, that is, it advances runners on the bases or increases the number of outs.

For each batter, a state transition matrix P is defined from the batter's statistics computed by the past records. The probabilities of state transitions are computed by the batter's past records to occur events, a single hit, a double, a triple, a home run, a walk, and an out. In this paper, we

Table 1. D'Esopo and Lefkowitz runner advance model

Event	Probability	Rule of runner advance
Single	p_S	A batter moves to first base. A runner on first moves to second base. Other runners score.
Double	p_D	A batter moves to second base. A runner on first moves to third base. Other runners score.
Triple	p_T	A batter moves to third base. All runners score.
Home Run	p_H	A batter and all runners score.
Walk	p_W	A batter moves to first base. All runners advance one base if forced.
Out	p_O	All runners do not advance.

employ the D'Esopo and Lefkowitz runner advance model, which uses events described in Table 1.

In this model, a 25$\times$ 25 matrix P is defined for each batter as (1),

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

where A and B are 8$\times$ 8 submatrices represented as (2):

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

$$B = p_O I \tag{2}$$

In (1) and (2), I denotes an 8 $\times$ 8 identity matrix and F indicates an 8 $\times$ 1 vector as (3):

$$F = (p_O, \ldots, p_O)^T \tag{3}$$

The first 8 rows in the matrix P represent states when a batter goes to the plate with no outs. The next 8 rows represent those with one out and the next 8 rows represent those with two outs. The 25th row represents the absorbing state, which corresponds to three outs. Each column represents a state when a batter completes the plate appearance, respectively.

We assume that a batter performs constantly at any states with number of outs, and submatrices A and B in rows 9 through 16 of P present states with one out in common with those with no outs. Similarly, A and B in rows 17 through 24 of P present states with two outs.

The rows of matrices A and B correspond to occupation of bases as shown in Table 2. For example, when a batter goes to plate at the beginning of an inning, the state transition probability from a state for no runners on with no outs to one for a runner on first with no outs corresponds to the element on the first row and on the second column. The state transition occurs when a batter gets a single hit or base on balls; thus, the value of the probability is $p_S + p_W$ in the D'Esopo and Lefkowitz model.

Table 2. The correspondence of matrices rows to occupation of bases

Row	Runners on base
1	empty
2	first base
3	second base
4	third base
5	first and second bases
6	first and third bases
7	second and third bases
8	bases loaded

The non-zero elements in A represent probabilities of events which do not increase the number of outs and advance on the bases, and those in B represent probabilities of events which increase the number of outs and do not advance on the bases. The elements in F represent probabilities of events from two outs to three outs.

Suppose the set of N batters is represented as $S = \{b_1, b_2, \ldots, b_k, \ldots, b_N\}$($k$ represents a batter index). A state transition matrix P_k is defined by probabilities like p_S, p_D, p_T, p_H, p_W,and p_O, which are derived by statistics of each batter k. The multiplication of these matrices along an input batting order allows us to simulate a baseball game.

2.2 Calculation of Expected Runs Scored per Inning

The state transition matrix P_k is decomposed into $P_k^{(0)}$, $P_k^{(1)}$, $P_k^{(2)}$, $P_k^{(3)}$, and $P_k^{(4)}$. Matrices $P_k^{(r)}(r = 0, 1, 2, 3, 4)$ consist of probabilities to occur events that score r run(s) by the batter k, and the following equation is obtained:

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

Suppose the maximum value of runs scored in one inning is represented as R_{max}, and the matrix which expresses runs scored and probability distribution of the states is represented as U. The rows in the $(R_{max} + 1) \times 25$ matrix U correspond to runs scored in an inning, and columns of U correspond to 25 states defined by the number of out and occupation of bases. Suppose the initial value of U is represented as U_0. Since no runs scored and no runners on bases at the beginning of an inning, the element of U_0 on first row and on first column is 1, and others are 0. Suppose U when n batters complete the plate appearances since the beginning of an inning is represented as U_n, this is defined as (5):

$$U_n|_i = \sum_{r=0}^{4} U_{n-1}|_{(i-r)} P_k^{(r)} \quad (i = 1, 2, \ldots, R_{max} + 1) \tag{5}$$

Here, $U_n|_i$ expresses the i-th row of U_n, and k expresses a batter index. The index k is incremented with n and changes in a cyclic way between one to nine, which is the number of batters in a batting order. As the multiplication shown in (5) are repeated, the sum of probabilities in the 25th column of U_n converges on 1 when n reaches the infinity. Suppose the element on the i-th row and the j-th column in U is represented as $u_{i,j}$, the expected runs scored per an inning, ER, is defined as (6):

$$ER = \sum_{k=1}^{R_{max}+1} (k - 1) \times u_{k,25} \tag{6}$$

Composing batting orders from the set of batters S and defining the state transition matrix P_k for each batter k allow us to calculate ER.

2.3 Calculation of Expected Runs Scored per Game

Suppose the probability that the i-th batter in a batting order leads off a certain inning and the j-th batter leads off the next inning is represented as t_{ij}, and the expected runs in that case is represented as e_{ij}(Figure 1). These values are obtained from the 25th column of U_n in each multiplication shown in (5). Suppose the probability that the n-th batter in a batting order leads off the m-th inning is represented as $p_{m,n}$, and the accumulated expected runs through the game in that case is represented as $a_{m,n}$. These parameters $p_{m,n}$ and $a_{m,n}$ are defined as (7) and (8):

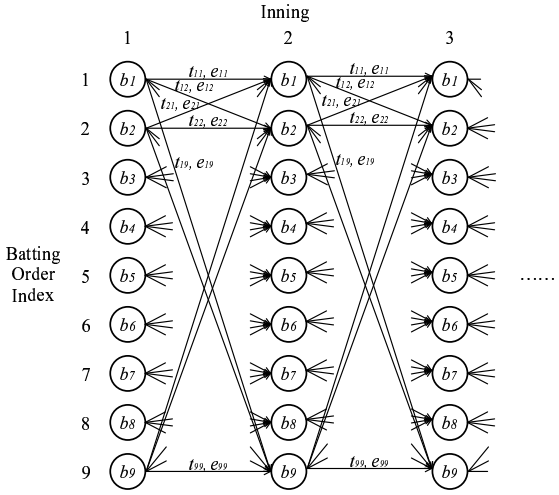

Figure 1. Definition of parameters t_{ij}, e_{ij}

$$p_{m,n} = \begin{cases} 1 \quad (m = 1,\ n = 1) \\ 0 \quad (m = 1,\ n = 2, 3, \ldots, 9) \\ \displaystyle\sum_{k=1}^{9} p_{m-1,k} t_{kn} \\ (m = 2, 3, \ldots, 10,\ n = 1, 2, \ldots, 9) \end{cases} \tag{7}$$

$$a_{m,n} = \begin{cases} 0 \quad (m = 1,\ n = 1, 2, \ldots, 9) \\ \displaystyle\sum_{k=1}^{9} \frac{p_{m-1,k}}{p_{m,n}} t_{kn} (a_{m-1,k} + e_{kn}) \\ (m = 2, 3, \ldots, 10,\ n = 1, 2, \ldots, 9) \end{cases} \tag{8}$$

From these values, the expected runs scored per a game is expressed by $\sum_{k=1}^{9} p_{10,k} a_{10,k}$. Calculating the expected runs scored per a game for each batting order organized from the set of batters S allows us to find the optimal batting order.

3 A Speed-up Technique to Calculate Expected Runs Scored

3.1 Parameters Sharing among Multiple Batting Orders

Since batters go to the plate in a cyclic way in a baseball game, calculation parameters, t_{ij} and e_{ij}, for the batting order $(b_1, b_2, \ldots, b_9)$ can be used for the batting order $(b_2, b_3, \ldots, b_9, b_1)$ by shifting the result for the order

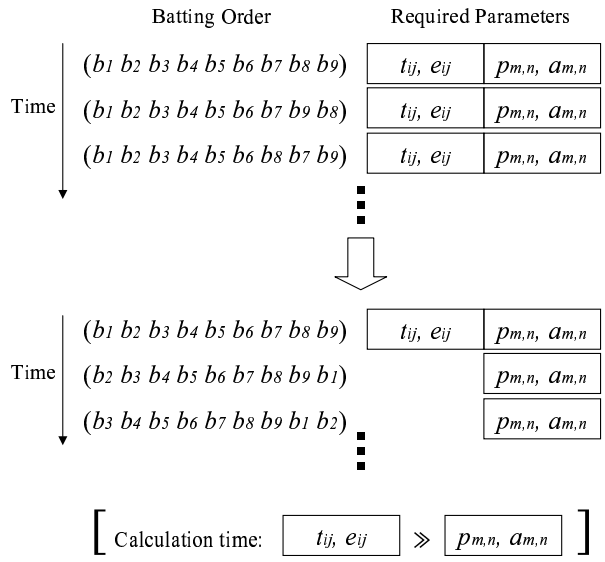

Figure 2. Speed-up by sharing common parameters

$(b_1, b_2, \ldots, b_9)$. In particular, $p_{m,n}(2 \leq m \leq 10)$ for the batting order $(b_1, b_2, \ldots, b_9)$ is calculated under the condition that $p_{1,1} = 1, p_{1,n} = 0(2 \leq n \leq 9)$ as shown in (7). Also, $p_{m,n}$ for the order $(b_2, b_3, \ldots, b_9, b_1)$ can be calculated under the condition that $p_{1,2} = 1, p_{1,n} = 0(n = 1, 3 \leq n \leq 9)$ using t_{ij} and e_{ij}, which have been computed for the order $(b_1, b_2, \ldots, b_9)$. Similarly, $p_{m,n}$ for the batting orders obtained by shifting the results. For example, $p_{m,n}$ for the order $(b_3, b_4, \ldots, b_9, b_1, b_2)$ can be calculated by only changing the initial value of $p_{1,n}$ without calculating t_{ij} and e_{ij} each time.

Since the computational complexity of $p_{m,n}$ and $a_{m,n}$ are very small in comparison with that of t_{ij} and e_{ij}, and nine batting orders can use parameters $p_{m,n}$ and $a_{m,n}$ in common, we can expect about nine times speed-up in comparison with the case that $p_{m,n}$ and $a_{m,n}$ is calculated each time (Figure 2).

3.2 Investigation of Batters' Capability for Fielding Position

Suppose the number of elements in the set of batters S is nine, the expected runs scored need to be calculated for 9! (or 362,880) batting orders to determine the optimal batting order. If the number of elements in S increases, the number of batting orders increases according to $O(N^9)$. For example, ${}_{10}P_9(= 3,628,800)$ expected runs scored need to be calculated for 10 batters, ${}_{11}P_9(= 19,958,400)$ expected runs scored for 11 batters, and ${}_{12}P_9(= 79,833,600)$ for 12 batters.

Position

Batting Order	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

Figure 3. Fielding Position Table

The computation to find an optimal batting order is further reduced by solving the satisfiability problem about batter's capability for fielding positions. In this problem, batter's capability for fielding positions is given by the fielding position table. By checking the feasibility of batting orders, or checking if batters in the order fill nine fielding positions, the computation for unfeasible batting orders can be omitted.

3.2.1 Fielding Position Table

Figure 3 shows an example of the fielding position table. Elements in the table indicate a batter's capability of fielding position. An element equals one if the batter is capable to take the position indicated by the column, and it equals to zero if he/she is not capable. For example in Figure 3, the batter b_1 can play as the pitcher(P) but cannot play in other fielding positions. Defensive skill, such as range of defense, strength of arm, and catch reliability, are not taken into account in the table.

3.2.2 Investigation of Feasible Batting Orders

The feasibility of batting orders to fill nine fielding positions is investigated by generating the tree and finding the feasible path on the tree. The tree has hierarchical structure as an example in Figure 4, the feasibility is investigated as follows:

First, the leftmost column in the fielding position table, pitcher position(P), is assigned to the root node on the tree. The table indicates that the fourth batter, b_1, can fill the pitcher position, then the fourth path (from the leftmost path) originating from the root node is chosen. Next, the next position, the catcher(C), in the table is assigned to the node under the fourth path. Here, because the fourth batter has been already occupied by the pitcher, the fourth path

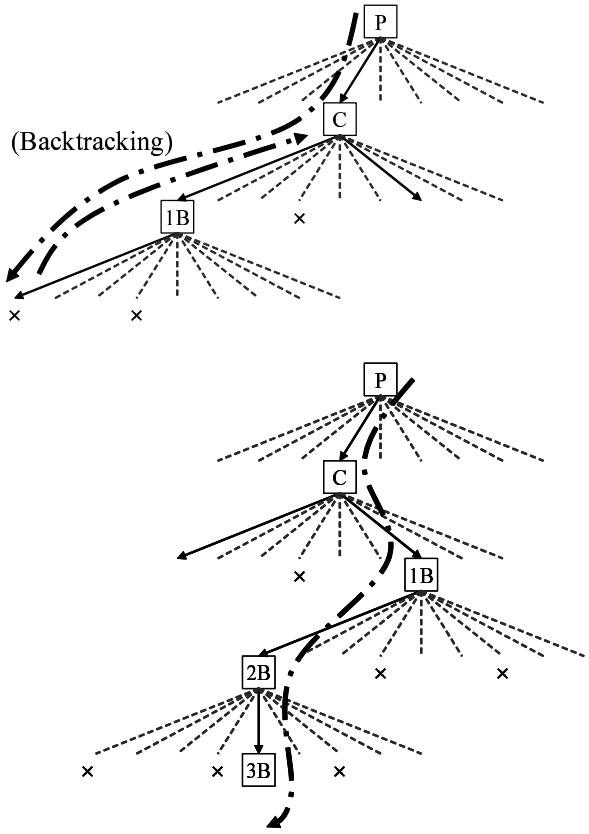

Figure 4. The method to determine whether batters in lineup fill all positions

under the node C is marked with "×". The table in Figure 3 indicates that the first batter, b_3, can play as the catcher; thus, the first path originating from the node C is chosen and the 1B is assigned to the node under the first path. For paths originating from the node 1B, the first and fourth paths have already occupied, and there is no feasible path. In this case, we backtrack to the ancestor node, C, and choose other feasible path.

The batting order is feasible if a trace on the tree reaches the node RF, the position in the rightmost column in Figure 3. If there are no feasible paths to reach the node RF, the batting order is not feasible and the computation for the order can be omitted.

In the previous work[3], it is necessary to extract batters in view of the fielding ability by hand. We propose a method to automate the process to determine whether batters in lineup fill all positions. And also, it reduces the computational complexity in comparison with that for all possible batting orders.

3.3 Parallelizing Computation and Load Balancing in Execution

The proposed technique parallelizes the computation on the Grid. Load balancing is performed by dispatching computation of batting orders proportionally to performances of computing resources. Since the calculation for each batting order is independent, the calculation can be distributed over multiple PCs to reduce the time to obtain an optimal batting order. As the scale of the problem increases, the Grid environment is needed as a computing platform to address it. Generally, the Grid is composed of computing resources which have heterogeneous computational powers, and efficient load balancing is needed in order that computing resources are utilized effectively.

Load balancing in the proposed technique is performed in two phases. In the first phase, in order to measure computational power for each computation node, each computation node calculates expected runs scored for a small subset of batting orders. In the second phase, the size of computation, or the number of batting orders, assigned to each computation node is calculated by the measured computational power.

Suppose N means the number of batters in a team and $s({}_N P_9)$ denotes the number of feasible batting order, which is defined in Section 3.2.2. The number of batting orders to compute their expected runs scored is reduced to $s({}_N P_9/9)$ by sharing common parameters among batting orders as described in Section 3.1. In the first phase, $f \times s({}_N P_9/9)/M$ batting orders is assigned to each computing nodes to measure the performance, where M denotes the number of computing nodes and f means the constant$(0 < f < 1)$ to define the ratio of batting orders used for the measurement. Then, the number of computing nodes assigned to the computing node k in the second phase is defined by $c(k)$.

$$\begin{aligned} S_t &= \sum_k \frac{1}{T(k)} \\ c(k) &= s({}_N P_9/9) \times \frac{1/T(k)}{S_t} \end{aligned} \tag{9}$$

Here, $T(k)$ means the elapsed time to compute for $f \times s({}_N P_9/9)/M$ batting orders in the first phase.

4 Performance Evaluation

In this section we present performance evaluation of the proposed techniques on the Grid testbed. The Grid testbed used in the experiment consists of two PC clusters distributed over two sites in Japan. These are the Blade cluster[1], which consists of 36 computation node and set up in

[1]Each node has PentiumIII 1.4GHz, 512MB memory. The OS is Red Hat Linux 7.1(Kernel 2.4.10), the compiler is gcc 2.96, and the compile option specified for optimization is -O3.

Tokyo Institute of Technology, and the F32 cluster[2], which consists of 64 computation node and set up in Advanced Industrial Science and Technology. We developed the application program to find the best batting order based on the D'Esopo and Lefkowitz model and implemented it on the Grid testbed using GridRPC middleware, Ninf-G 2.3.0[6].

Ninf-G is reference implementation of GridRPC API. The client program is able to invoke server programs, or executables, on remote computing resources using the Ninf-G client API. Ninf-G is implemented on the Globus Toolkit[2]. When the client program starts its execution, it accesses MDS to get interface information to invoke the remote executable. Next, the client program requests GRAM to invoke the remote executable. In this phase, authentication is performed using GSI. After the invocation, the remote executable connects back to the client to establish connection. Finally, the client program dynamically encodes its arguments according to the interface information, and transfers them using Globus I/O and GASS.

To execute the product of matrix-vector, which is the kernel of computation, we utilize ATLAS[7], which is automatically tuned numerical calculation library.

4.1 Effects of Sharing Common Parameters among Batting Orders

To verify the effect of sharing common parameters, t_{ij} and e_{ij}, among batting orders, we compare the elapsed times to calculate expected runs scored in cases where t_{ij} and e_{ij} are shared and not shared. Suppose the number of elements in S is 10, and all batting orders composed from S fill all positions.

The elapsed times in each case with 40 nodes on F32 cluster is shown in Table 3. As mentioned in Section 3.1, the computational complexity of $p_{m,n}$ and $a_{m,n}$ are very small in comparison with that of t_{ij} and e_{ij}. Because batting orders can share parameters t_{ij} and e_{ij}, 8.81 (close to 9) times speed-up is obtained.

Table 3. The effect of sharing common parameters

calculate t_{ij}, e_{ij} for each order	use t_{ij}, e_{ij} in common	speed-up ratio
3,258(sec)	370(sec)	8.81

[2]Each node has Xeon 3.06GHz, 4GB memory. The OS is Red Hat Linux 8.0(Kernel 2.4.24), the compiler is gcc 3.3.3, and the compile option specified for optimization is -O3.

Table 4. Statistics of 12 batters

Batter	AVG	HR	SLG	OBP
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

AVG indicates an abbreviation of Batting Average,
HR indicates Home Runs,
SLG indicates Slugging Percentage,
and OBP indicates On-base Percentage.

Table 5. Optimal batting order at June 23, 2005

	Batter	AVG	HR
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
Expected runs scored: 6.88			

4.2 Results on the Grid Testbed

To verify the effect of load balancing on the Grid, we conduct the experiment on the Grid testbed consisting of 30 nodes on the Blade cluster and 40 nodes on the F32 cluster. We choose 12 batters who showed good performance from April to June 23, 2005 in the Central League of NPB(Nippon Professional Baseball). The batters' batting statistics and fielding parameters are shown in Table 4 and Figure 5, respectively. The computed optimal batting order and expected runs scored by the order are shown in Table 5.

The number of all possible batting orders composed of 12 batters is ${}_{12}P_9 = 79,833,600$. By sharing common t_{ij} and e_{ij}, the number of batting orders to be computed is ${}_{12}P_9/9 = 8,870,400$. Among these batting orders,

Batters	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

Figure 5. Batters' capabilities for positions

the number of orders which fill all positions is 3,024,000 ($= s({}_{12}P_9/9)$).

To verify the effect of load balancing, we compare the elapsed times to calculate expected runs scored in cases where the load balancing is performed and computation is evenly distributed over computing nodes. The elapsed times and the breakdown of them in each case are shown in Table 6. Before actual calculation, computation to measure computational power of each node is conducted in the first phase. The ratio of the calculation in the first phase, or f is set to 0.05. When we do not apply load balancing, the calculation in the first phase is not conducted, and simply $s({}_{12}P_9/9)/M$ batting orders ($s({}_{12}P_9/9)$ is the number of target batting orders, M is the number of all computation node) are evenly distributed over all nodes and expected runs scored are calculated at each node.

The average number of batting orders assigned to computation nodes by applying load balancing, the average elapsed times for each PC cluster and the maximum elapsed times are shown in Table 7. Figure 6 and Table 7 presents that the average elapsed times between PC clusters are similar and that the whole elapsed time can be reduced by load balancing.

5 Conclusion

In this paper, we report speed-up techniques for computation of the Markov chain model to find an optimal batting order in a baseball team.

The proposed techniques reduce computation time by: sharing common parameters among multiple batting orders, omitting computation for unfeasible batting orders, parallelizing computation on the Grid. The experimental results

Table 6. The effect of load balancing

	Load balancing (sec)	
	Off	On
Trial	0	618
Actual	4,441	2,503
Other	91	157
Total	4,532	3,278
Speed-up	1.00	1.38

(Trial indicates the elapsed time to measure computational power, Actual indicates the elapsed time to compute expected runs scored, Other indicates the elapsed time to initialize, terminate, synchronize, etc.)

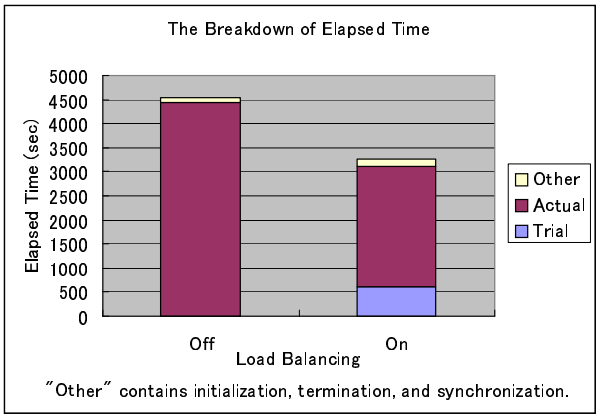

Figure 6. The effect of load balancing

show the effectiveness of the proposed techniques. First, by sharing common parameters, we attain 8.81 times speed-up. Next, the investigation of batters' capability for fielding positions significantly omits computation for unfeasible batting orders. Finally, parallelization of the computation on the Grid allows us to compute large-scale problems and the run-time load balancing gives 1.38 speed-up compared with computation with static load balancing. The experimental results also show that the proposed technique finds the optimal batting order in 27,216,000 batting orders for 3,278 seconds with 70 computation nodes on the Grid testbed. The results mean that the large-scale problem, which have not been solved before due to lack of computational power, in the operations research field can be solved by applying the proposed techniques, and the exact optimal batting order is obtained by calculating expected runs scored for all possible batting orders in the reasonable time.

acknowledgment We would like to sincerely thank the Grid Technology Research Center at the National Institute

Table 7. Result of load balancing

	Blade	F32
Avg. Assigned Order Count	23,597	57,903
Avg. Elapsed Time (sec)	2,196	2,013
Max. Elapsed Time (sec)	2,503	2,213

of Advanced Industrial Science and Technology for allowing us to use their computing resources for our experiments.

References

[1] B. Bukiet, E. Harold, and J. Palacios. A Markov Chain Approach to Baseball. *Operations Research*, 45(1):14–23, Jan. 1997.

[2] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputing Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[3] N. Hirotsu and C. Miyaji. A Mathematical Method to Find an Optimal Lineup in Baseball Game - A Case Study for All Japan Team -. *Journal of the Operations Research Society of Japan*, 49(6):380–389, June 2004.

[4] R. Horst, P. M. Pardalos, and N. V. Thoai. *Introduction to Global Optimization*. Kluwer Academic Pub., 2000.

[5] Major League Baseball. Official Info: Official Rules. http://mlb.mlb.com/NASApp/mlb/mlb/official_info/official_rules/definition_terms_2.jsp.

[6] Y. Tanaka, H. Nakada, S. Sekiguchi, and S. Matsuoka. Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing. *Journal of Grid Computing*, 1(1):41–51, June 2003.

[7] R. C. Whaley. Automatically Tuned Linear Algebra Software(ATLAS). http://math-atlas.sourceforge.net/.