ICIC Express Letters
Part B: Applications
Volume **8**, Number **12**, December **2017** ICIC International ©2017 ISSN 2185-2766
pp. 1663–1670

RESEARCH ON DYNAMIC LOAD BALANCING STRATEGY BASED ON JIAOZHOU BAY WATER QUALITY FORECASTING SYSTEM

Weijian Huang, Zenglian Jiao, Yuanbin Han and Wei Du

School of Information and Electrical Engineering
Hebei University of Engineering
No. 199, Guangming Rd., Handan 056038, P. R. China
huangweijian0808@sina.com

Received June 2017; accepted August 2017

Abstract. *At present, the parallel calculation of Jiaozhou Bay water quality forecasting system is mainly realized by MPI (Message Passing Interface). Because the existing parallel methods for the Jiaozhou Bay water quality forecasting parallel system (JBWQFPS) mostly use the equipartition of the grid, the actual amount of calculation is determined by the wet and dry separation method on the grid. Therefore, the sharing of the grids does not guarantee the equipartition of the computational load in the actual calculation, leading to serious computational imbalance problems. In view of this problem, this paper proposes an automatic redistribution load balancing scheme (ARLBS). Compared with the situation before using load balancing optimization, parallel acceleration ratio has been significantly improved, and the parallel efficiency is up to 53.68%. The results show that the ARLBS is effective in the field of ocean numerical simulation.*
Keywords: Load balancing, Parallel computing, Water quality forecasting system, Jiaozhou Bay

1. Introduction.

At present, the water quality forecasting system is mostly compiled by serial programs. In practical applications, the calculation efficiency is often low, resulting in some limitations. With the application of various parallel technologies, the efficiency of the prediction system has been greatly improved. Since the JBWQFPS adopts the original automatic assignment of the grid nodes [1,2], which divides computing grid according to coordinate points, the size of the grid is divided into different parts, and there is often a load imbalance problem, which results in the waste of the idle nodes and the inefficiency of the operation. The key to MPI parallel computing is load distribution. If the load distribution is reasonable and the load between nodes is balanced, it can give full play to the performance of each node and the parallel efficiency is higher. There are three strategies for dynamic load balancing: distributed, centralized, and mixed/hierarchical. Each strategy has its own representative programs. For example, distributed strategy mainly has diffusion method, gradient method (GM) and other methods [3-6]. However, the implementation of each dynamic load balancing scheme needs to consider complex hardware performance, including: CPU, memory, etc. The requirements for the parallel programmers are higher, so it is difficult to implement. And there are load imbalance problems in the implementation of MPI parallelization mentioned in document [1,2]. In order to solve the problems mentioned above, this paper takes JBWQFPS as an example and implements a novel dynamic load balancing scheme – ARLBS to reduce the distribution imbalance of node computing tasks and increase the performance of parallel computing.

2. Related Work.

2.1. Introduction to MPI parallel programming.

MPI is one of the standards for messaging interfaces for developing parallel programs based on message passing [3]. The general execution flow of the MPI program is that the initialization function MPI_INIT must be executed before the process calls the MPI process. Then call the MPI_COMM_SI-ZE function to get the default group size. And then call MPI_COMM_RA NK to get the logical number of the current process in the default group, through this number to distinguish between the different processes in the group, each process can perform different tasks. Then, call MPI_SEND and MPI_RECV to exchange information with other nodes, so as to realize parallel and cooperation among processes. Finally, when the process has completed all operations and no longer needs MPI process, call MPI_FINALIZE to eliminate the MPI environment.

2.2. Parallel calculation method of fluid.

The model of Jiaozhou Bay water quality forecasting system is based on the C grid [1], which mainly adopts 3D coordinate system. The system divides the monitored sea area into a three-dimensional grid system composed of $i * j * k$ grid points. Combined with the horizontal resolution and the nested hydrodynamic calculation section of the predicted sea area, the actual grid number suitable for Jiaozhou Bay is calculated to be $159 * 185 * 5$.

The Jiaozhou Bay water quality forecasting system includes the establishment of the three-dimensional water quality model with nutrient salt, chemical oxygen demand and dissolved oxygen as forecasting variables [2]. The system considers the biological processes such as sedimentation, sediment material flux, and exchange material with foreign matter. In 3D coordinates, 12 variables such as nitrogen, phosphorus, zooplankton, phytoplankton, nitrate, nitrite, ammonium salt, phosphate, silicate, various debris, and various chemical oxygen demand in water are under quantitative monitoring, including the first, the second, the third, the fourth chemical change separately, as well as submarine flux calculation, chemical changes and calculation of various open boundary conditions and so on. At the same time, the diffusion of various substances in water is divided into two directions, horizontal and vertical. The calculation of vertical diffusion does not require data transfer, while the horizontal diffusion requires the calculation data of adjacent grid points. So the flow field is divided into multiple calculation regions according to the vertical direction, and a new calculation grid is generated for each block. At the same time, the mesh on the common boundary of each region should be coincident, so that the continuity of material quantification calculation can be guaranteed at the mesh boundary. Multiple computing areas exchange information after a calculation, and the next round is calculated until the prediction task is completed.

3. Research on ARLBS.

3.1. Statistics and analysis of the time consumption and the task amount of the nodes.

The execution time of the program is recorded by the CPU_TIME method, and the time spent on each hour prediction of the JBWQFPS is recorded in the log file. And ultimately derive the time-varying data of each node on processing the 48-hour data of the parallel system. As shown in Figure 1, the time that the node p3 consumed is the most. The consumption time of the node p1 is close to that of p2, and the computing time of p4 is the least. That is when the node p4 completes the calculation task, it is necessary to wait for the completion of the task on others, while p4 is in idle state during which time. The idle time is the difference between the time consumed by the p3 and the p4 in performing their respective computational tasks: $T_{p3} - T_{p4}$. In parallel computing, time efficiency is measured by the longest time among nodes, that is, the time consumed by p3.

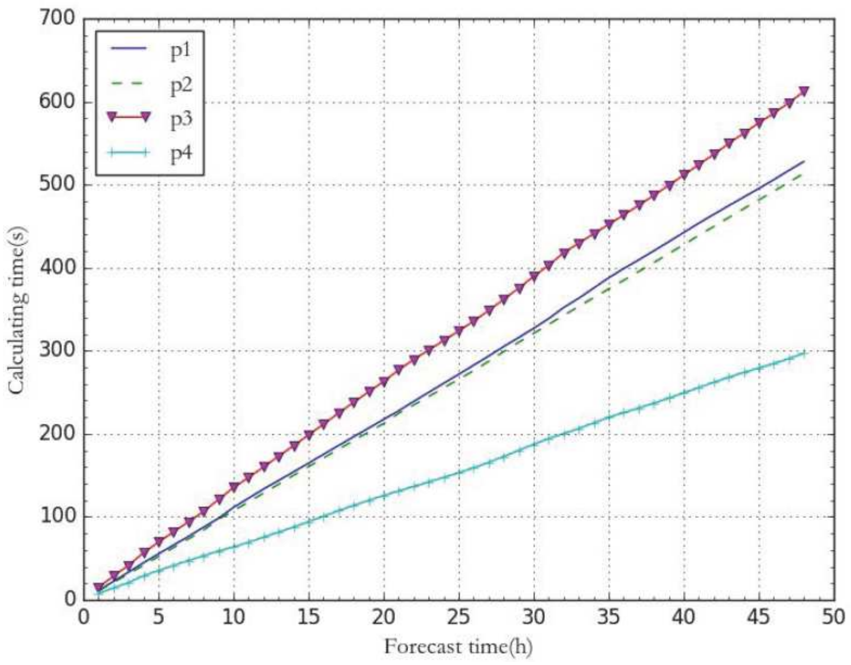

Figure 1. Forecast time versus time consumption

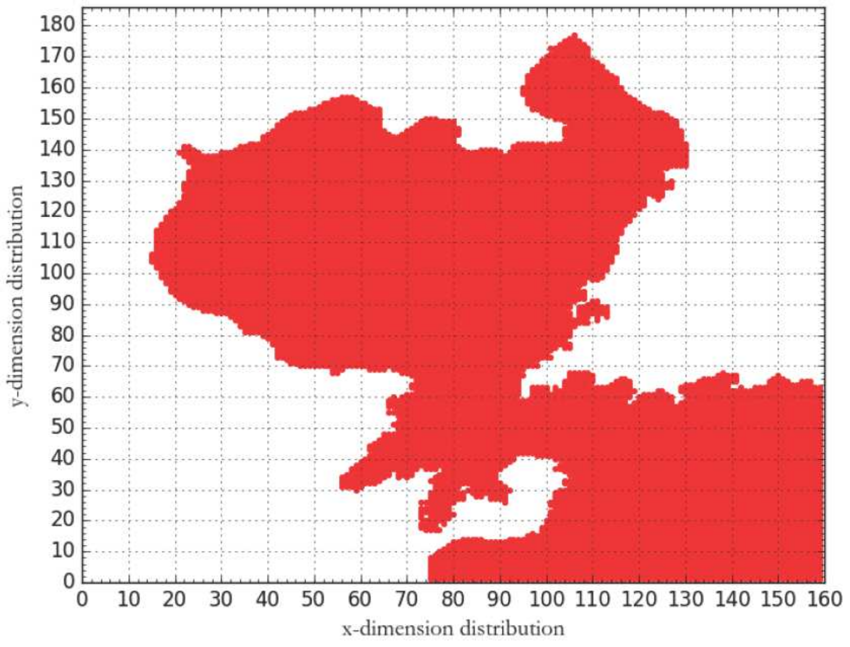

Figure 2. Diagram of the distribution of calculate points in the grid

By studying the process control points in the program, it is determined that the main control variable in the program is wet_mask. Then the distribution information of computing points in two-dimensional coordinates is derived. According to the coordinate information of the calculation points, the sketch map of the distribution of the points involved in the calculation in the grid is drawn, as shown in Figure 2.

It can be seen in Figure 2 that the distribution of the computational tasks in the whole region is an irregular shape. With the increase of y value, the integral area of the whole calculation area is non-linear.

Through the analysis and study, the distribution information of the main calculation points in the system is found and the tasks that need to deal with by each node to simulate the hourly calculation are plotted in Figure 3.

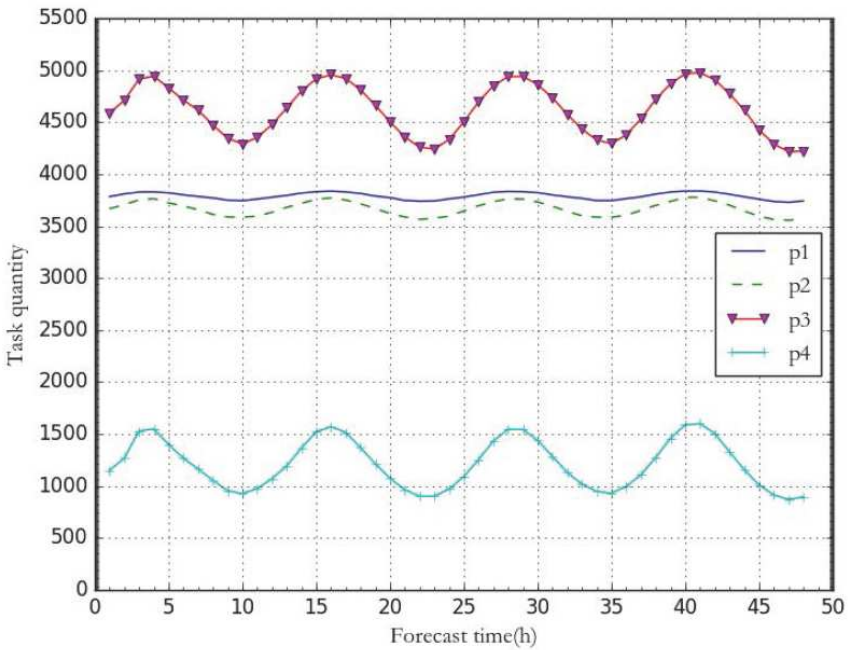

FIGURE 3. Comparison of the number of tasks per hour on each node

In combination with Figures 1, 2 and 3, it is found that the main factors causing the unbalanced calculation of the nodes are that the number of effective grid points is significantly different among the nodes. The number of grid points on p3 is obviously higher than that on others, while that on p4 is far less than that on other nodes, which leads to a serious imbalance between nodes. And the node with small load completes the calculation task ahead of time and into the idle state, while the overloaded node is still in the busy state, which causes the waste of computing resources of the idle node and unnecessary time overhead. The execution time of each process is directly proportional to the calculation scale of nodes. At the same time, the task quantity alternates at different time points due to the impact of the tide. However, the main factor affecting the computing time is the number of grid points involved in the calculation, that is, the area between the calculated point distribution and the x, y axis.

The entity waters of Jiaozhou Bay are complex and irregular in shape; thus the whole area is divided into several blocks. When divided into four block regions, the size of each grid block is shown in Table 1.

TABLE 1. Distribution table of the number of calculated units

Block-number	Number of units	Ratio (%)
1	3647	27.96
2	3406	26.13
3	4603	35.30
4	1384	10.61

For the original load distribution scheme, it can be seen in Table 1 that when using 4 processors, the load of two processors is relatively balanced, in addition to two processors, one for 10.61%, the other 35.30%, which causes the entire load to be unbalanced, resulting in a large delay in the calculation completion time and the total calculation time is longer.

3.2. **Design of ARLBS.** Load distribution is the key to multi-block parallel computing [7]. When the number of processors is more than that of grid blocks, each processor allocates one grid task, which is a relatively simple distribution scheme. However, it cannot play the performance of multi-node cluster system. And it is a relatively inefficient scheme [8]. At the same time, the number of computing units in each grid is not the same, and some are even very different; thus this scheme can cause serious load imbalance and affect the efficiency of parallel computing [9]. To solve this problem, an ARLBS is proposed in this paper.

Traditionally, the system automatically allocates grid blocks to the idle processor. Since the large difference in the size of each grid block, the load is often uneven. So the ARLBS is to re-divide the grid area to achieve the purpose of load balancing.

The process of the ARLBS is: To run the program according to the traditional scheme and analyze the relevant data collected. Then automatically write the best task allocation program to the source program to determine the size of the load on each processor according to the number of processors and the grid data. In computational domain, the integration is performed in the Y direction and the computational grids are evenly distributed to each computational node, through which the entire grid area is re-segmented into new mesh blocks, and then they are automatically distributed. In this way, the load can be better allocated to each processor. In practical applications, the water quality forecasting system can run continuously. In order to make timely automatic adjustment, it will automatically collect the data to produce the latest load balancing scheme.

4. Experiment and Analysis.

4.1. Computing environment.

The simulation environment used in this paper is: one workstation server equipped with the AMD Opteron(tm) Processor 6433 with sixteen cores, and 16GB of memory, system for CentOS Linux release 7.2.1511 (Core) operating system, MPI parallel environment, the compiler environment for the MPIF90 ver.11.1; gcc ver.4.8.5, and the use of FORTRAN language programming.

4.2. Application of ARLBS.

In view of the above situation, in order to further balance the computational tasks of each computing node, this paper adopts the ARLBS to reallocate the task.

As shown in Figure 4, the division ranges are set by x and y axes and each scale of which for one unit. By moving the position of the y, the area of the effective calculation

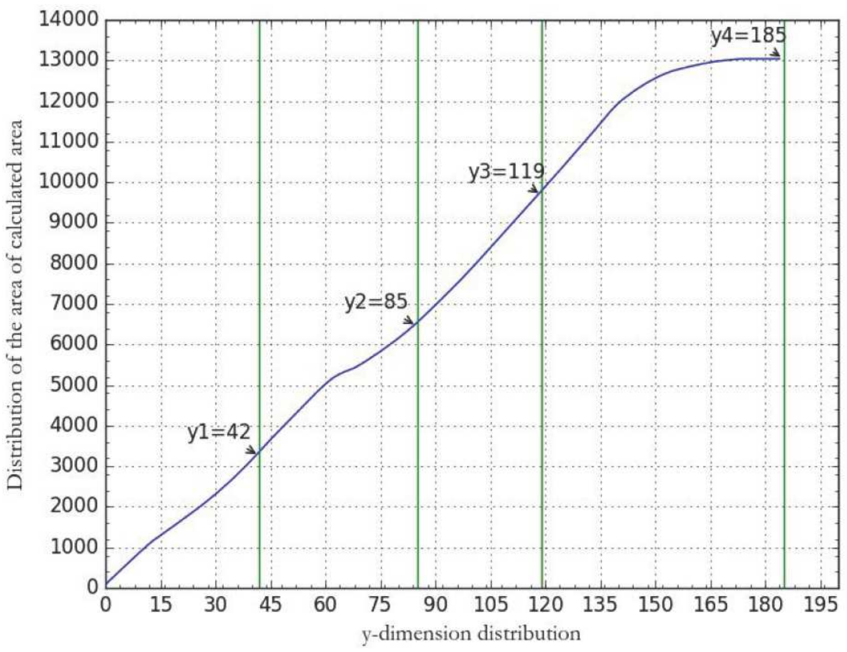

FIGURE 4. Schematic diagram of automatic re-partitioning

area is calculated by the integrating in the y direction. And when the area is less than 1/4 of the total area and it will be larger than 1/4 if y moves by one scale, it is denoted as $y1$; then start from $y1$, in turn find the value of $y2$ and $y3$ by repeating the above procedure; for the sake of computational integrity, $y4$ takes the maximum value of y.

4.3. **Experimental results and analysis.** Run time is one of the important indexes to measure the performance of parallel programs. A parallel arithmetic program is usually executed by multiple processes. Each process runs on one computing node, and the execution time of the parallel program is determined by the longest time. In order to better reflect the efficiency of parallel computing, speedup and parallel efficiency are used as the two evaluation indexes. They are defined as:

$$S_p = \frac{t_{p1}}{t_{p2}}, \quad E_p = \frac{S_p}{p} \tag{1}$$

S_p is the speedup of the parallel calculation, E_p is the parallel efficiency, p is the number of processors used in the parallel calculation, t_{p1} is the time consumed to perform the solution using p processors before using the load optimization, and t_{p2} is the time after optimization.

At the end of the experiment, the experimental data are processed. As shown in Figure 5, the calculation time of the four nodes is relatively close, and it saves nearly 100s relative to that before the load balancing.

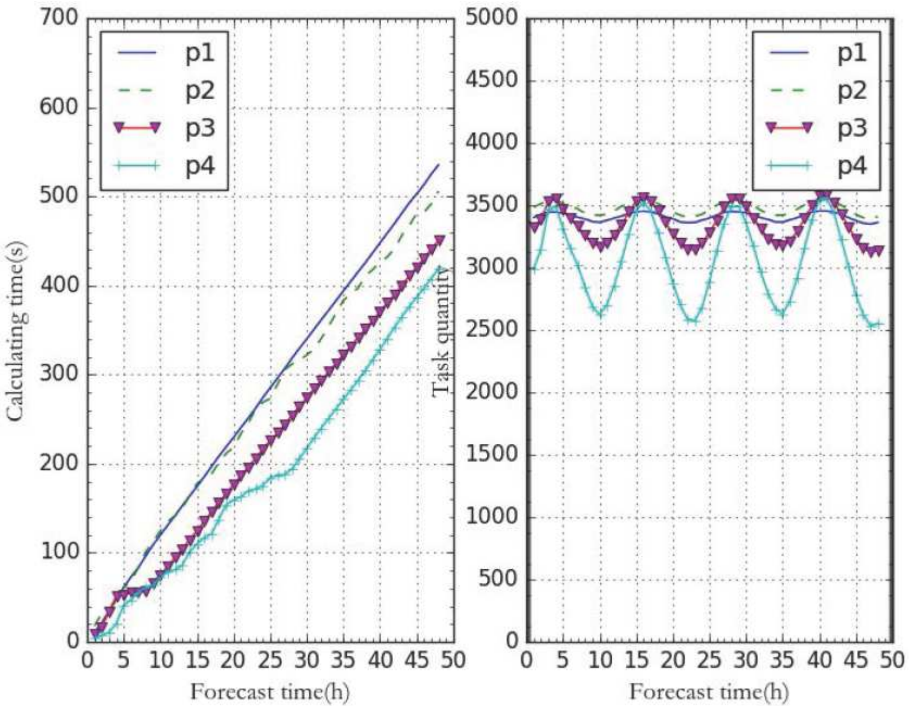

FIGURE 5. Forecast time versus time consumption and task quantity of each process

After applying the ARLBS, the number of tasks for each of the 4 processes and the percentage of each process's task in total computation are counted, as shown in Table 2.

As it can be seen in Table 2, the workload between processes is basically balanced, so the load balancing scheme is feasible and effective.

According to the above scheme, performance benchmarks are performed on system before and after using ARLBS at the case of different nodes. Respectively, to run four times in each case of using 2, 4, 8 computing nodes and to calculate the average time of four times as the final result. Through the statistics of performance changes in the

Table 2. Distribution table of the calculated units after load balancing

Block-number	Number of units	Ratio (%)
1	3258	24.98
2	3196	24.52
3	3229	24.76
4	3357	25.74

Table 3. Speedup and parallel efficiency of JBWQFPS

	2CPU	4CPU	8CPU
Calculation time before load optimization (s)	990.89	620.38	431.68
Calculation time after load optimization (s)	922.98	539.13	304.47
Speedup ratio	1.074	1.151	1.418
Parallel efficiency	53.68%	28.77%	17.72%

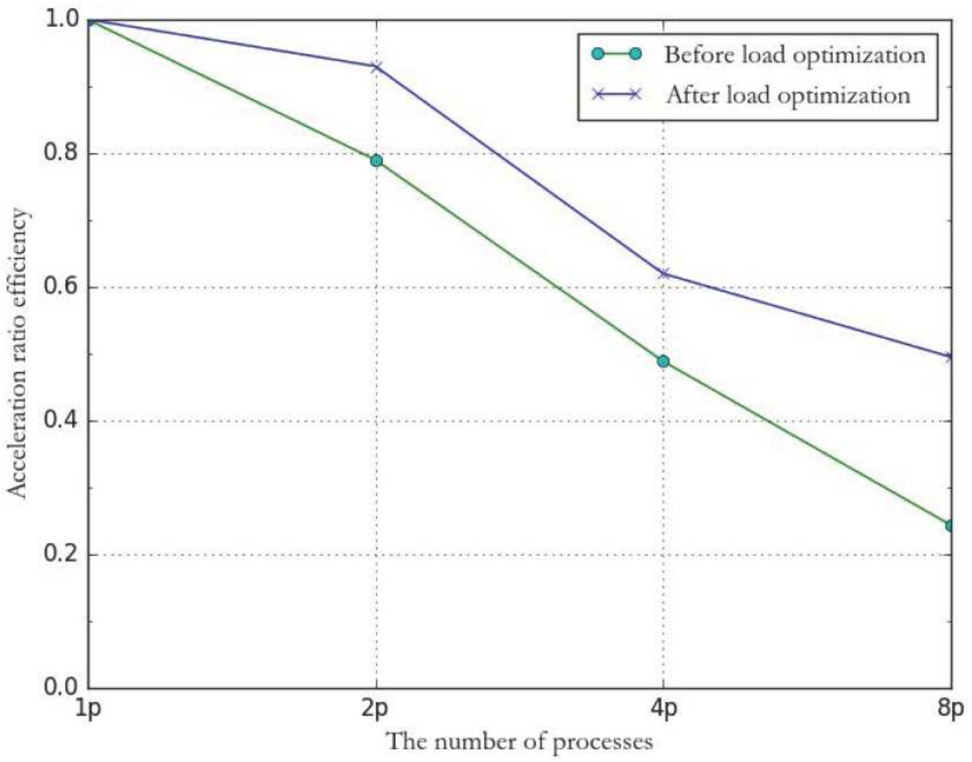

Figure 6. Comparison graph of acceleration ratio trends

case of using different number of nodes, the speedup ratio and parallel efficiency data are obtained before and after load balancing, as shown in Table 3.

In order to analyze the acceleration effect better, the acceleration ratio trend chart before and after using the optimization is plotted, as shown in Figure 6.

The results of the performance benchmark in Figure 6 show that the time consumption of the system before load balancing is relatively large and the load imbalance problem is more serious when the number of processes exceeds four. Although the parallel procedures still remain accelerating with the increase of the process number, the acceleration effect tends to be stable. The reason for this phenomenon is that the number of transmission data is increased and the load imbalance between nodes is more prominent. After the load balancing optimization, the acceleration efficiency eventually tends to be smooth with the increase of the node, which is caused by the message transmission between nodes, but the occurrence of this state is delayed.

Through the comparison of the operation data of the system before and after load balancing, it is found that the load of each node is more balanced after using ARLBS and the calculation time is significantly reduced. And also the speed of numerical simulation and the efficiency of the system are improved.

5. Conclusions.

Parallel computers are becoming more and more important in the field of scientific research, and the requirements for their performance optimization will be strengthened with the high requirements of business oriented numerical simulation. In order to solve the problem of load imbalance in the task allocation of JBWQFPS, the paper analyzed the load situation by using the time data generated by the first run of the system. Then, the spatial distribution of the computational grid is reallocated based on the actual computational tasks, and each group of tasks is automatically assigned to multiple processes. The feasibility of the ARLBS is illustrated by the experimental data. The purpose of load balancing has been achieved by using the approach of this paper; it has a certain degree of impact to the performance optimization in using parallel computer and shortens the time of numerical simulation.

Acknowledgment. This work is supported by the National Natural Science Foundation of China (No. 51509066), the National Marine Public Welfare Industry Research Special of China (No. 201205018)

REFERENCES

[1] C. Li, *Application of Mixed Parallel Computing in Marine Water Quality Forecasting System*, Master Thesis, Hebei University of Engineering, 2014.

[2] W. Du, P. Niu and W. Huang, Application of MPI technology in Jiaozhou Bay water quality forecasting system, *Computer Engineering and Design*, vol.34, no.6, pp.2257-2261, 2013.

[3] G. Utrera, J. Corbalán and J. Labarta, Dynamic load balancing in MPI jobs, *High-Performance Computing*, pp.117-129, 2008.

[4] S. Nian and L. Guangmin, Dynamic load balancing algorithm for MPI parallel computing, *International Conference on New Trends in Information and Service Science*, pp.95-99, 2009.

[5] V. Kale and W. Gropp, Load balancing for regular meshes on SMPs with MPI, *EuroMPI*, vol.10, pp.229-238, 2010.

[6] H. Liu, Multi-block structured grid dynamic load balancing method, *Proc. of the 22nd Annual Conference of Beijing Institute of Mechanics*, 2016.

[7] M. Rietmann, D. Peter, O. Schenk et al., Load-balanced local time stepping for large-scale wave propagation, *International Parallel and Distributed Processing Symposium (IPDPS)*, pp.925-935, 2015.

[8] G. Martin, M. C. Marinescu, D. E. Singh et al., FLEX-MPI: An MPI extension for supporting dynamic load balancing on heterogeneous non-dedicated systems, *European Conference on Parallel Processing*, pp.138-149, 2013.

[9] M. Bhandarkar, L. V. Kalé, E. de Sturler et al., Adaptive load balancing for MPI programs, *International Conference on Computational Science*, pp.108-117, 2001.