3C3-IOS-2-2

Effective Integration Frameworks for Combining Computer Gaming programs with the Grid Computing System

Chih-Hung Chen and Shun-Shii Lin

Department of Computer Science and Information Engineering,
National Taiwan Normal University, Taipei, Taiwan, ROC

The high space complexity and the high time complexity are still severe limitations when the game tree is searched until a very deep depth in computer gaming programs. In order to reduce these limitations, we aim to parallelize computer gaming programs by using the grid computing system which was developed by Professor I-Chen Wu. In this system, it exploits spare resources such as computing powers and memory storages of desktops or some personal computers for the applications requiring huge amount of computation. However, when the gaming programs are combined with the grid computing system, it is necessary to resume the gaming programs for each move during a game. To overcome the above problem, in this paper, we introduce some approaches to avoid the resumption of the running programs. According to the initial property of the computer gaming programs, we design two frameworks to combine the computer gaming programs with the grid computing system. The experiments show that its performance is improved 8.27 times on a Connect6 program and its winning rate is enhanced 42.8% on a Go program. The results indicate that our approach combined with the grid computing system is quite powerful for the computer gaming programs. Furthermore, these frameworks are easy for integration between the computer gaming programs and the grid computing system.

1. Introduction

There are many applications (e.g., business intelligence [2], social network [12], and high complexity computer games [6]) that require massive computation resources. In the past, they usually were dealt with a single computer. However, we constantly need to increase the computing power, memory capacity, and storage space to overcome some more complex situations. This will cause a waste of resources because more advanced hardware devices are continuously installed to replace the old ones, and the renewed hardware devices are very expensive. Furthermore, there are some limitations for keeping strengthening the power of a hardware device on the manufacturing process under the current architecture. All of above are the reasons why those applications could not be resolved well in the past.

With the development of network technology, techniques of the grid computing, the volunteer computing, the cloud computing, etc. are proposed to connect lots of low-priced computation devices to construct a computing system which is more powerful than a single high-priced computer. Therefore, those problems that demand numerous computation resources might be solved by these kinds of network-connected computing systems.

However, the computation resources are usually not managed perfectly in daily life. In order to exploit exhaustively all resources we owned, we take advantage of the Desktop Grid computing system [7][13] which was developed by Professor I-Chen Wu at the National Chiao Tung University to integrate lots of computers in our laboratory. We design two frameworks for combining the Desktop Grid computing system with the computer gaming programs. The goal is to let the computer gaming programs achieve higher performance from the shared resources.

Contact: Shun-Shii Lin, Dept. of Computer Science and Information Engineering, National Taiwan Normal Univ., No. 88, Sec. 4, Ting-Chow Rd., Taipei, Taiwan, R.O.C., Tel: 886-2-77346671, Fax: 886-2-29322378, linss@csie.ntnu.edu.tw.

2. Desktop Grid computing system

The Desktop Grid computing system [7][13] which was developed by Professor I-Chen Wu at the National Chiao Tung University is a volunteer-computing-based grid computing system. It provides a suitable environment for the computer gaming programs.

The Desktop Grid computing system is constructed from three parts labeled APP, BROKER, and PC. The architecture diagram of the Desktop Grid is shown in Figure 1.

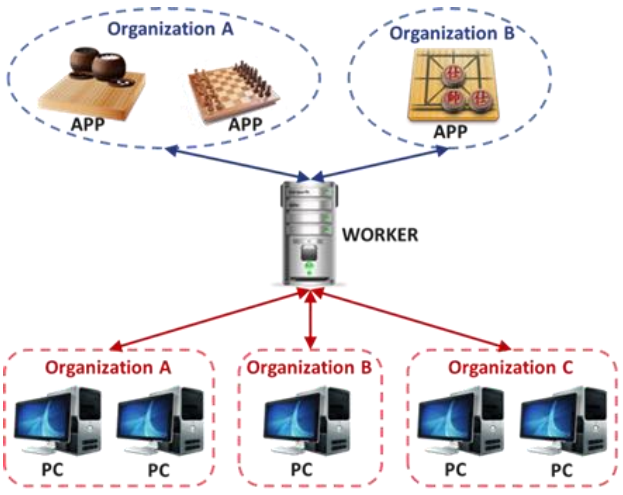

Figure 1 The architecture diagram of the Desktop Grid

The APP which is implemented by any user is a computer gaming program, the PC that is offered by an organization is a personal computer, and the BROKER is a coordinator which serves as a bridge between APPs and PCs.

The PC must connect to the BROKER to share its resource to other users, thus it can also get powerful energy from other PCs offered from other organizations. All the APPs need to do is to create tasks that require to be run and to submit these tasks to the BROKER. Then the APPs wait for the response from the BROKER. Now, the BROKER will assign some PCs to do the tasks, and await the results. When a PC is assigned a task, it will begin doing the task. When a PC finishes its task, it will reply the results to the BROKER right away. The BROKER then returns the results to the original APP which creates the task. Finally the APP gets the results and does the next process. The steps are illustrated in Figure 2.

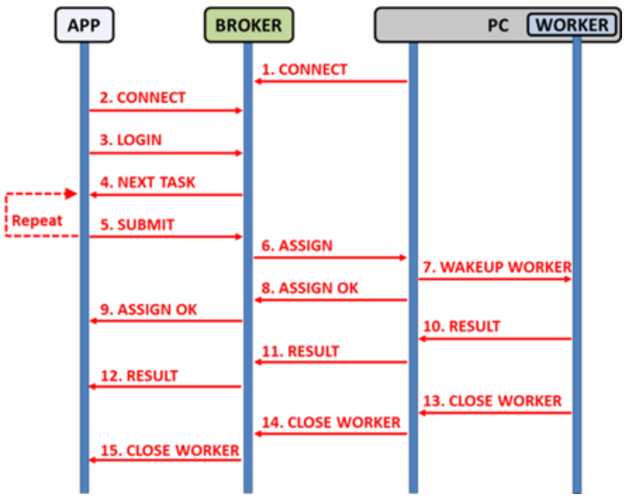

Figure 2 The flow chart of the Desktop Grid

3. Frameworks for Desktop Grid

The volume of research into computer games has increased steadily in recent years. When the gaming program is executed for searching the optimal move in the game tree, it consumes most resources. Within the time allotted, the more resources we have, the further evolution will be estimated well. To reinforce the power of the gaming programs, we try to take advantage of the Desktop Grid to share the resources of each computer in our laboratory. And we hope that our gaming programs can compete with human experts in the future.

For combining the gaming programs with the Desktop Grid, a computer gaming program should be divided into two parts; one is the controller (hereinafter referred to as “APP”), and the other is the engineer (hereinafter referred to as “WORKER”). The APP is the major portion of the gaming program, and it must support the communication protocol for connecting with the BROKER. When the game is playing, the APP may generate some instructions for the WORKERs and wrap them up into a task. Then, the APP submits the task to the BROKER and waits for the returned results. At this moment the APP can do something else until the results are returned. If the results of all tasks are received, the APP will integrate all the results and make a decision. Moreover, the WORKER program is usually a searching engine which expands a subtree of the whole game tree. The PC ought to keep the WORKER program running all the time. While the task is arrived, the PC wakes up the WORKER for performing the task created by the APP.

When the gaming programs are directly combined with the Desktop Grid, it is necessary to resume the WORKER for each move during a game. To overcome the above problem, in this paper, we introduce some approaches to avoid the resumption of the running programs. According to the initial property of the WORKERs, we design two different frameworks (named wakeup immediately [3] and wakeup non-immediately [4]) to integrate the computer gaming programs into the Desktop Grid. The advantages of these frameworks are that they require less running time and fewer codes to be modified. As following, we will describe these two frameworks in more detail.

3.1 Wakeup immediately

In this type of framework, the WORKERs take a little time from initial state to ready state when the WORKERs are executed. It is so fast that we can ignore this little time. When the PCs are assigned some tasks, they will wake up the WORKERs to deal with the tasks. At the beginning, the APP submits tasks to the BROKER. Then, the BROKER assigns the tasks to some PCs. Now, the PCs wake up the WORKERs and assign the tasks. The WORKERs then start to do the tasks. When a WORKER completes its task, it will output the results and terminate itself. At last, the results are passed from the BROKER to the APP. And the task is finished. These steps are illustrated in Figure 3.

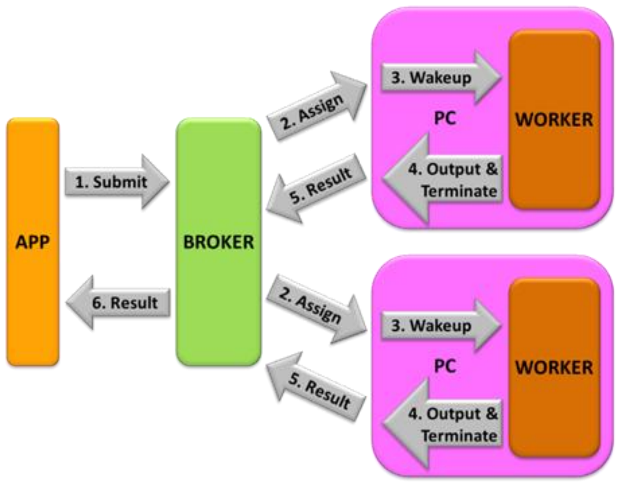

Figure 3 The diagram of wakeup immediately

3.2 Wakeup non-immediately

In this type of framework, the WORKERs take a longer time to get ready when they are performed. When a PC is assigned a task, it will wake up the WORKER to do the task. The WORKER takes more time to finish the task due to its delay. If we can eliminate the delay caused by the WORKERs, then the WORKERs will terminate earlier. In other words, we may make good use of the time to let the WORKERs do more work.

To aim at the above purpose, we run the WORKERs firstly and allow it to reside on the PCs. When the Desktop Grid computing system is working, the WORKERs always stand by for doing tasks. To adjust the framework to fit the operations of the Desktop Grid, we implement a program called ADAPTER to be waked up by the PCs. Thus, the ADAPTERs replace the role that is originally played by the WORKERs. Now, an ADAPTER acts as a mouthpiece between a PC and a WORKER. The PCs communicate with the WORKERs through the ADAPTERs, and vice versa.

The steps are refined and illustrated in Figure 4. To start with, the WORKERs reside at the PCs in advance. Next, the APP submits tasks to the BROKER. The BROKER then assigns these tasks to the PCs. The PCs wake up the ADAPTERs and assign the tasks to them. Now, the ADAPTERs pass the tasks to the WORKERs that are ready for work. When a WORKER achieves its task, it will pass the results to the ADAPTER and wait for the next task. Then, the ADAPTER outputs the results and terminates itself. Finally, the results will be returned to the APP step by step. At last, the processes of the task are over.

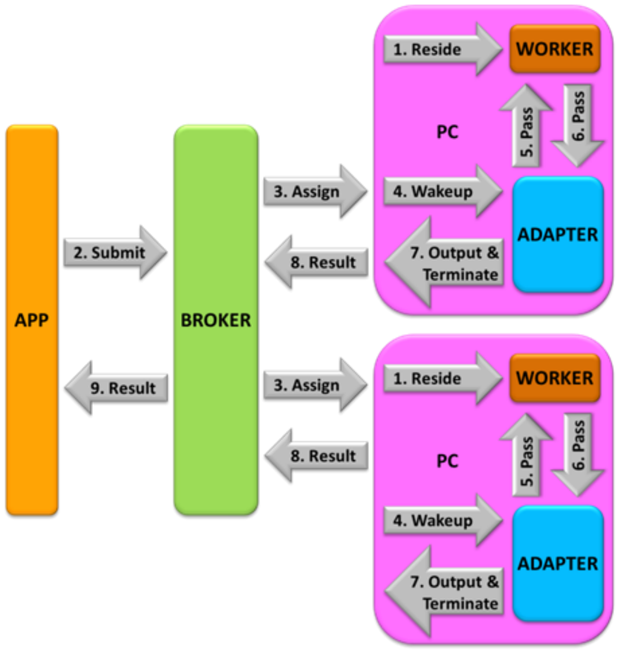

Figure 4 The diagram of wakeup non-immediately

3.3 Summary

In the Desktop Grid computing system, the messages that are passed by the BROKER are communicated through the network. It takes more time to communicate with the network than within a computer. To share the penalty for the networking communication, we suggest to let each task do more instructions to reduce the times of the networking communication. Therefore, we should balance between the number of the instructions and the times of the networking communication within a fixed time limit.

If we already had some computer gaming programs, and we want these programs to be integrated into the Desktop Grid computing system, we can choose one of these two frameworks introduced above to combine with the Desktop Grid according to the property of the gaming programs. To integrate with these two frameworks, fewer codes are needed to be modified in the gaming programs. By the way, combining with the framework of wakeup non-immediately, the time saved can be used for doing other important things when the messages are communicated on the network. Thus, we can utilize our resources more efficiently.

4. Experiments

In this section, we select one gaming program for each framework to experiment. For the sake of fairness, we pick some computers which have same specification to do the experiments. Each computer uses a single core processor to handle the tasks in the experiments. The results of the experiments are showed in the following subsections.

4.1 Wakeup immediately

We choose a Connect6 [5][8] program named Ant [3] to perform the experiments for the framework of wakeup immediately. Ant uses threat-space search [1] for exploring the game tree. When the program is searching, it will check if there exists a threat within the current branch of the game tree. If there is a threat on the branch, then the children nodes of this branch will be expanded. Now, the program repeats to check for the presence of successive threats. When the program finds a path that has a sequence of threats to let its opponent lose the game, it will win the game with the sequence of the threat moves.

The WORKERs that get ready quickly are classified as the framework of wakeup immediately. The searching engine of Ant just belongs to this type. Therefore, we let Ant complete 16880 different searches on a single computer, and record the searching time for each search. Then, we let Ant do these 16880 searches with multi-computers by the Desktop Grid computing system. Finally we compare the searching time on a single computer with multi-computers. The results are shown in Table 1.

Table 1 The experimental results for Ant

Number of computers	**Total time for 16880 searches**	**Average time for each search**	**Speedup**
1	155572.83 (s)	184.33 (s)	1.00
4	62001.08 (s)	73.46 (s)	2.51
8	33021.50 (s)	39.13 (s)	4.71
12	24463.34 (s)	28.99 (s)	6.36
16	21724.56 (s)	25.74 (s)	7.16
20	18801.79 (s)	22.28 (s)	8.27

The experiments indicate that it takes 184.33 seconds for a search with a single computer in average. But it just needs 22.28 seconds with 20 computers working together. So the performance of 20 computers working together is 8.27 times faster than a single computer. Furthermore, our results show that the more resources we have the higher performance we will get.

4.2 Wakeup non-immediately

We select a Go program called ERICA [10][11] to do the experiments for the framework of wakeup non-immediately. ERICA uses Monte-Carlo Tree search which is a best-first search algorithm based on random playouts to estimate the values of the most promising moves accurately [9].

The WORKERs which get ready slowly are classified as the framework of wakeup non-immediately. The searching engine of ERICA just belongs to this type. However, when the game tree is grown incrementally in a best first manner guided by the simulations, it has the shared memory problem in the Monte-Carlo Tree search. To reduce the space complexity of the experiments, we force the WORKERs to simulate locally on its computer. At last, the outputs from each computer will vote for the final results. In addition, the number of the random playouts is set to 10000. We then let ERICA run on a single computer to compete with another ERICA running on multi-computers. The results are shown in Table 2.

Table 2 The results of multi-computers vs. a single computer

Number of computers	**Play black Win / Lose**	**Play white Win / Lose**	**Winning rate**
3	11 / 10	14 / 15	50 %
5	17 / 12	14 / 7	62 %
7	16 / 8	21 / 5	74 %
9	16 / 9	22 / 3	76 %
11	18 / 6	18 / 8	72 %

The experiments indicate that when the number of computers is greater than or equal to 7, the winning rates reach a fixed percent. To confirm the results, we test 500 games for comparing ERICA running on a single computer with another ERICA running on 7 computers together. In order to save the time for experiments, we set the number of the random playouts to 500. The results are shown in Table 3. It reveals that the winning rates of 7 computers working together is 71.4 % against a single computer. In general, when a program competes with itself using two machines with the same specification, the result should be fifty-fifty. So the winning rate is improved 42.8% by using 7 computers together.

Table 3 The results of 7-computers vs. 1-computer

Number of computers	**Play black Win / Lose**	**Play white Win / Lose**	**Winning rate**
7	175 / 66	182 / 77	71.4 %

5. Conclusion

When we want to find a better move for a complicated game, the gaming program will consume a lot of time and require massive computation resources to explore the game tree. We make good use of the Desktop Grid computing system to connect many computers in our laboratory to provide a powerful computing resource. According to the initial property of the computer gaming programs, we design two frameworks to integrate the computer programs into the Desktop Grid computing system. The experiments show that its performance is enhanced 8.27 times on a Connect6 program and its winning rate is improved 42.8% on a Go program. The results indicate that our frameworks integrated with the Desktop Grid are really powerful for computer games. In general, if we want to combine some computer gaming programs with the Desktop Grid by these frameworks, then we will need fewer codes to be modified in the gaming programs and less executing time to do the tasks. So these two frameworks are efficient approaches for integrating computer gaming programs into the Desktop Grid computing system.

References

[1] Allis, L.V., Herik, H. Jaap van den, and Huntjens M.P.H., “Go-Moku and Threat-Space Search,” Report CS 1993-02, Department of Computer Science, Faculty of General Sciences, University of Netherlands.

[2] Business intelligence, http://en.wikipedia.org/wiki/Business_intelligence.

[3] Chih-Hung Chen, Shun-Shii Lin, and Min-Huei Huang, “Volunteer Computing System Applied to Computer Games,” TCGA 2012 conference, Hualien, Taiwan, 2012.

[4] Chih-Hung Chen, Terry Lao, and Shun-Shii Lin, “Design a Monte-Carlo Go to Combine with the Volunteer Computing System,” TAAI 2012 conference, Tainan, Taiwan, 2012.

[5] Connect6, http://www.connect6.org/web/index.php?lang=en

[6] H. Jaap van den Herik, Jos W. H. M. Uiterwijk, Jack van Rijswijck, “Games solved: Now and in the future, ” Artificial Intelligence, 134(1-2): 277-311, 2002.

[7] I-Chen Wu, Chingping Chen, Ping-Hung Lin, Guo-Zhan Huang, Lung-Ping Chen, Der-Johng Sun, Yi-Chih Chan, and Hsin-Yun Tsou, “A Volunteer-Computing-Based Grid Environment for Connect6 Applications,” The 12th IEEE International Conference on Computational Science and Engineering (CSE-09), Vancouver, Canada, 2009.

[8] I-Chen Wu, Dei-Yen Huang and Hsiu-Chen Chang, "Connect6", ICGA Journal (SCI), Vol. 28, No. 4, pp. 235-242, December 2005.

[9] Monte-Carlo Tree search, http://chessprogramming.wikispaces.com/Monte-Carlo+Tree+Search.

[10] Shih-Chieh Huang, Remi Coulom, and Shun-Shii Lin, “Monte-Carlo Simulation Balancing Applied to 9×9 Go,” ICGA Journal, Vol. 33, No. 4, pp. 191-201, 2010.

[11] Shih-Chieh Huang, Remi Coulom, and Shun-Shii Lin, “Time Management for Monte-Carlo Tree Search Applied to the Game of Go,” International Workshop on Computer Games, Hsinchu, Taiwan, November 18-20, 2010.

[12] Social network, http://en.wikipedia.org/wiki/Social_network.

[13] V-Taiwan, http://aigames.nctu.edu.tw/vtaiwan/.