Let's Depart Together: Efficient Play Request Dispatching in Cloud Gaming

Yusen Li
Multi-plAtform Game Innovation Centre
Nanyang Technological University
S080007@e.ntu.edu.sg

Xueyan Tang
School of Computer Engineering
Nanyang Technological University
ASXYTANG@ntu.edu.sg

Wentong Cai
School of Computer Engineering
Nanyang Technological University
ASWTCAI@ntu.edu.sg

***Abstract*—In this paper, we study the problem of how to dispatch the play requests to the cloud servers in a cloud gaming system. We show that the dispatching strategy of play requests may heavily affect the total service cost of the cloud gaming system. The play request dispatching problem can be considered as a variant of the dynamic bin packing problem. However, we show that the classical bin packing algorithms such as First Fit and Best Fit are not efficient in dispatching the play requests in cloud gaming due to the diurnal workload pattern of online games. To address this issue, we propose an efficient request dispatching algorithm that assigns the play requests according to the predicted ending times of the game sessions. Through extensive evaluations using real online gaming traces, we show that the proposed dispatching algorithm can significantly reduce the resource waste of the cloud servers and thus decrease the total service cost compared to the First Fit and Best Fit algorithms.**

I. Introduction

Recently, cloud gaming has attracted a great deal of interests from both academia and industry [20], [4], [5]. The basic idea of cloud gaming is to run games on cloud servers and players interact with games through thin clients. Specifically, the thin client sends player inputs to a cloud server; the cloud server runs the game instance, render 3D graphics, encodes the game scenes into a video and streams the video to the thin client; the thin client decodes and displays the video to the player. In this way, players can play games easily without installing them locally. Moreover, players can play GPU intensive games on their computers, tablets or even mobile phones without dedicated hardware equipped.

Running a game instance requires a certain amount of computational resources. Therefore, the number of game instances that can run concurrently on a cloud server is limited. In order to serve a large player population, the cloud gaming service provider needs to prepare sufficient number of cloud servers for running the game instances. However, the high workload variability of online gaming makes server provisioning a challenging issue. As shown by empirical data (see Figure 2 in Section III), the number of active players varies greatly over the day. Thus, setting up too many servers would lead to resource waste during slower gaming times while not maintaining enough servers would result in servers becoming overloaded during peak gaming times. The on-demand resource provisioning service in public clouds like Amazon EC2 provides a promising solution to the above problem. The users can rent the resources (i.e., virtual machines) on an as-needed basis in response to workload variation and pay for only the resources that they actually use. This approach frees the users from the complexities of purchasing, engineering and maintaining hardware infrastructures, and has been adopted by cloud gaming companies such as OnLive [4] and GaiKai [1].

Consider a cloud gaming system that rents virtual machines to serve as cloud servers. A main concern of the service provider is how to minimize the total renting cost of the virtual machines used. Generally, the total renting cost is proportional to the total running hours of all the virtual machines used. Given a workload, we shall show that the total running hours of all the virtual machines used is highly dependent on how the play requests of the players are assigned to the virtual machines. In our previous work [25], we modeled the problem of dispatching play requests to cloud servers for minimizing the total renting cost as a variant of the dynamic bin packing (DBP) problem, which is named the MinTotal DBP problem. We theoretically analyzed the worst-case performance of the classical bin packing algorithms such as First Fit and Best Fit for the MinTotal DBP problem. First Fit attempts to put a new item into the first bin (i.e., the earliest opened bin) that can accommodate the item, while Best Fit attempts to assign a new item to the bin with the smallest residual capacity that can accommodate it. First Fit and Best Fit are simple and make decisions based on the current system state only. However, we shall show that First Fit and Best Fit packing algorithms may lead to large resource waste of the virtual machines in cloud gaming due to its diurnal workload pattern.

In this paper, we propose an efficient play request dispatching algorithm which assigns play requests according to the predicted ending times of the game sessions. This approach is based on the assumption that the ending times of the game sessions can be predicted in some ways, e.g., according to the user habits, historical user behaviors or based on the expected game session length etc. With the predicted ending times, our request dispatching algorithm aims to "pack" the game sessions that have similar ending times to the same virtual machine, so that the virtual machine can keep high resource utilization during its running hours. We evaluate the proposed dispatching algorithm by using real game session traces and the results show that it can significantly reduce the resource waste of the virtual machines, thereby decreasing the total running hours compared to First Fit and Best Fit.

The rest of this paper is structured as follows. The related

978-1-4799-6882-4/14/$31.00 (c)2014 IEEE

works are summarized in Section 2. In Section 3, the play request dispatching problem in cloud gaming is introduced and the performance of the classical bin packing algorithms is evaluated. Our proposed play request dispatching algorithm is presented in Section 4. In Section 5, the experimental evaluations and results are discussed. Finally, conclusions are made and future work is discussed in the last section.

II. Related Works

Cloud gaming systems have been implemented for both commercial use and research studies [20], [4], [5]. GamingAnyWhere [20] is the first open source cloud gaming system, which provides a complete cloud gaming testbed for researchers and developers. OnLive [4] is the first company offering a commercial cloud gaming platform. As mentioned in the introduction, the basic idea of cloud gaming is to render games on cloud servers and stream videos to thin clients. A lot of research work has been conducted towards measuring and optimizing the video-based cloud gaming solution. The measurement works have mainly concentrated on measuring the latency and network traffic of the existing commercial cloud gaming platforms [10], [24], [26], [12]. The optimization works have mainly focused on video encoding techniques and graphic rendering techniques for bit rate reduction [17], [6], [7], [11], [27].

There is also some research work studying the resource management issues in cloud gaming systems. Wu *et al.* [28] studied the request dispatching and server provisioning issues in a multi-region multi-datacenter cloud gaming system. The objective of their work is to reduce the latency. Hong *et al.* [19] considered how to consolidate the game instances on the physical servers to strike a balance between the Quality of Experience (QoE) perceived by players and the net profit of the service provider. They found that both the QoE and the provider's profit are highly dependent on how the VMs are placed on the physical servers. Some heuristic algorithms were proposed to maximize the provider's profit while guaranteeing the QoE. However, it was assumed that the operational cost of the cloud gaming service provider is proportional to the CPU and GPU utilizations of its servers. This is valid when the service provider maintains its own server infrastructure. But it does not really match the pricing model of the on-demand resources offered by public clouds. If the service provider rents server resources from public clouds, the servers rent would be charged according to their running hours regardless of their utilizations. Thus, to save the total renting cost, it is important to reduce the total running hours of the servers.

The play request dispatching problem studied in this paper is related to the bin packing and interval scheduling problems. The classical bin packing problem aims to pack the items into the least number of bins. The problem and its variations have been studied extensively in both the offline and online versions [13], [15]. It is well known that the offline version of the classical bin packing problem is NP-hard already [16]. A variant of the classical bin packing problem is dynamic bin packing [14]. This generalization assumes that items may arrive and depart at arbitrary times. The objective is to minimize the maximum number of bins ever used over time. The play request dispatching problem considered in this paper is a case of dynamic bin packing. However, we study the problem from different view points in that we aim to minimize the total renting cost. In our previous work [25], we have formulated the play request dispatching problem for minimizing the total service cost of a cloud gaming system that rents servers from public clouds. We modeled the problem as a MinTotal DBP problem and theoretically analyzed the worst-case performance of the classical bin packing algorithms such as Best Fit and First Fit for the MinTotal DBP problem. However, no experimental evaluation was conducted using real gaming workloads. The interval scheduling problem is also related to the play request dispatching problem [21]. The classical interval scheduling problem considers a set of jobs, each associated with an interval in which the job should be processed. Each machine can process only a single job at any time. Given a fixed number of machines, the objective is to schedule a maximum feasible subset of jobs [8]. However, the online version of the problem has seldom been studied.

III. Classical Bin Packing Algorithms for Play Request Dispatching

Suppose the cloud gaming service provider rents virtual machines as the cloud servers. When a play request is received, it should be assigned to an active virtual machine that has enough computational resources to run the game instance of this request. Several game instances may share the same virtual machine as long as the computational resources of the virtual machine are not saturated. If no active virtual machine is able to accommodate the play request, a new virtual machine should be started. Generally, once a game instance starts, it will reside in the same virtual machine during the entire game session. Thus, each play request corresponds to one game session and these two terms can be used interchangeably. The migration of game instances from one virtual machine to another is not preferable due to large migration overheads and interruption to game play [19]. A virtual machine can be shutdown and released for saving cost if no game instance is running on it.

In the online scenario, each play request must be dispatched as it arrives without any knowledge of the future play request arrivals. For a given set of play requests, the dispatching strategy directly affects the total running hours of the virtual machines used to serve all the play requests. Consider the following example. Suppose for simplicity that all the play requests have the same computational resource requirements. Assume each virtual machine is able to host at most 2 game instances concurrently. Let each play request (or game session) be represented by a pair: (*starting time*, *ending time*). Assume there are three play requests to be served, which are represented by $r_1 = (0, 10)$, $r_2 = (0, 1)$, $r_3 = (0, 10)$. It is easy to see that the three play requests arrive at the same time and two virtual machines are needed to host them. Suppose r_1 and r_2 are packed into the same virtual machine, and r_3 is assigned to the other virtual machine (see Figure 1(a)). In this case, the running hours of the virtual machine serving r_1 and r_2 would be $\max\{1, 10\} = 10$, and the running hours of the virtual machine serving r_3 would be 10. Therefore, the total running hours of the two virtual machines is $10 + 10 = 20$. On the other hand, if r_1 and r_3 are packed into the same virtual machine, then r_2 would be assigned to the other virtual machine (see Figure 1(b)). In this case, the running hours of the virtual machine serving r_1 and r_3 would be 10, and the running hours of the virtual machine serving r_2 would be 1.

Thus, the total running hours of the two virtual machines is $10 + 1 = 11$. It can be seen from the above example that the total running hours of the virtual machines is highly dependent on how the play requests are dispatched.

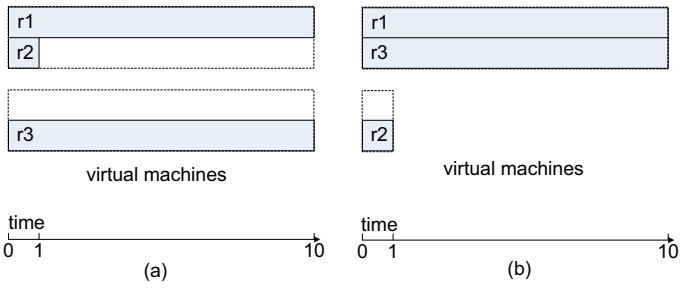

Fig. 1. Two dispatching strategies

A. Classical Packing Algorithms

Since the play request dispatching is a variant of the bin packing problem, we first examine how the classical bin packing algorithms perform for this problem. We conducted some simulations to evaluate the two most commonly used bin packing algorithms: First Fit and Best Fit. We consider the following implementations of First Fit and Best Fit for play request dispatching in cloud gaming:

- **First Fit Dispatching**: Each time a play request arrives, First Fit tries to assign it to the earliest started active virtual machine that has enough resources to run the game instance for the request. If no such virtual machine is found, a new virtual machine is started and the request is assigned to the new virtual machine.
- **Best Fit Dispatching**: Each time a play request arrives, Best Fit tries to assign it to the "best" active virtual machine, which is the one with the smallest amount of spare resources that can accommodate the game instance for the request. If no such virtual machine is found, a new virtual machine is started and the request is assigned to the new virtual machine.

We implemented an event-driven simulator to simulate the play request dispatching process by using First Fit and Best Fit. In order to make the simulations more realistic, we use the data from the World of Warcraft Avatar History (WoWAH) dataset [23]. The World of Warcraft (WoW) is the most popular MMORPG in the world which has over 12 million subscribers [2]. Hence, the data of WoW is often studied by researchers from various areas [9], [18], [22]. The WoWAH dataset records continual observations of the status of all players in a WoW realm in Taiwan over a 3-year period from Jan. 2006 to Jan. 2009. During the monitored period, 91065 players and 667032 game sessions associated with the players were observed.

In the simulations, each game session in the WoWAH dataset represents a play request in cloud gaming. We assume that all the game sessions have the same computational resource requirements and each virtual machine is able to run up to 4 game instances simultaneously (a normal GPU generally can support $3 \sim 5$ game instances concurrently [29]). Figure 2 shows the number of active players over a sample period of 3 days and Figure 3 shows the instantaneous number of virtual machines used by different dispatching algorithms over the sample period. The "Optimal" curve represents the minimum number of virtual machines required for accommodating all the active game sessions, which is calculated by dividing the total number of active game sessions by the virtual machine capacity (i.e., 4 game sessions on each virtual machine).

According to how the number of active game sessions changes (see Figure 2), we divide 24 hours of the day into two periods: a climbing period (from 7:00 to 23:00) and a declining period (from 23:00 to 7:00 of the next day). The number of active players generally increases during the climbing period and decreases during the declining period. As can be seen from Figure 3, the numbers of virtual machines used by First Fit and Best Fit dispatching are very close to Optimal in the climbing period. This is because there are more play request arrivals than departures in the climbing period. In this case, no matter how the play requests are assigned, all the active virtual machines are almost fully occupied by the continuously arriving play requests. However, in the declining period, First Fit and Best Fit occupy many more virtual machines than Optimal. It implies that the active virtual machines under First Fit and Best Fit dispatching are not fully utilized in the declining period. This is because there are more player departures than arrivals in the declining period. If the players on the same virtual machine do not leave the system at the same time, the "leftover" game sessions would cause many virtual machines to run at low levels of resource utilization so that these virtual machines cannot be shut down timely. As indicated in Figure 3, the area between the "Optimal" curve and the First Fit (or Best Fit) curve represents the "wasted" running hours of virtual machines under First Fit (or Best Fit) dispatching. It is easy to see that there exist a large number of wasted running hours for both First Fit and Best Fit dispatching.

IV. Ending Time based Play Request Dispatching

In this section, we propose an efficient play request dispatching algorithm that can greatly reduce the wasted running hours compared to First Fit and Best Fit. The basic idea is to assign play requests according to the predicted ending times of the game sessions, and pack those game sessions that have "close" ending times into the same virtual machine. In this way, all the game sessions assigned to the same virtual machine are expected to complete at around the same time. Therefore, each virtual machine can keep a high level of resource utilization during its running hours and can be shut down timely after all its game sessions end. The details of the dispatching algorithm are described in Algorithm 1.

Denote the play request to be dispatched by r. We first predict the ending time of r's game session, which is denoted by $e(r)$ (line 1). For each active virtual machine v, let $T(v)$ denote the *expected* shutdown time of v, which is defined as the latest predicted ending time of all the game sessions currently running on v. If there is no active virtual machine that has enough computational resources to serve r, a new virtual machine needs to be started and the play request r is assigned to the new virtual machine (lines 3-4). Otherwise, r would be assigned to one of the active virtual machines that have sufficient spare computational resources. Among all

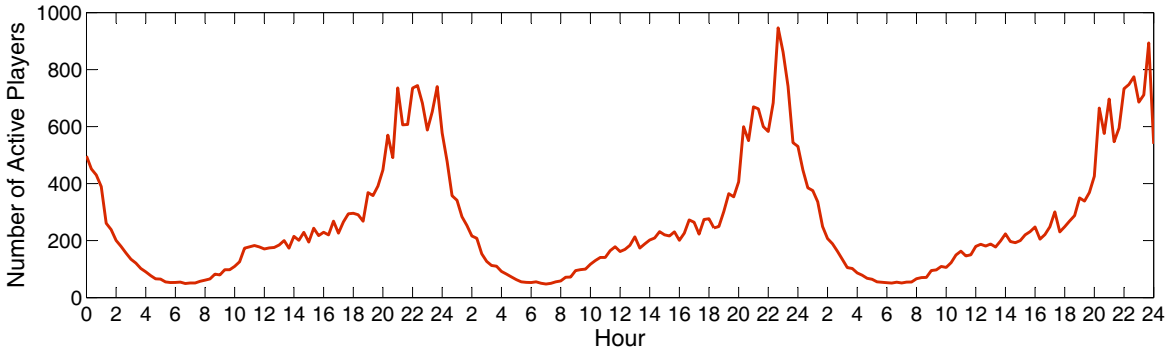

Fig. 2. Number of active players over a sample period of 3 days

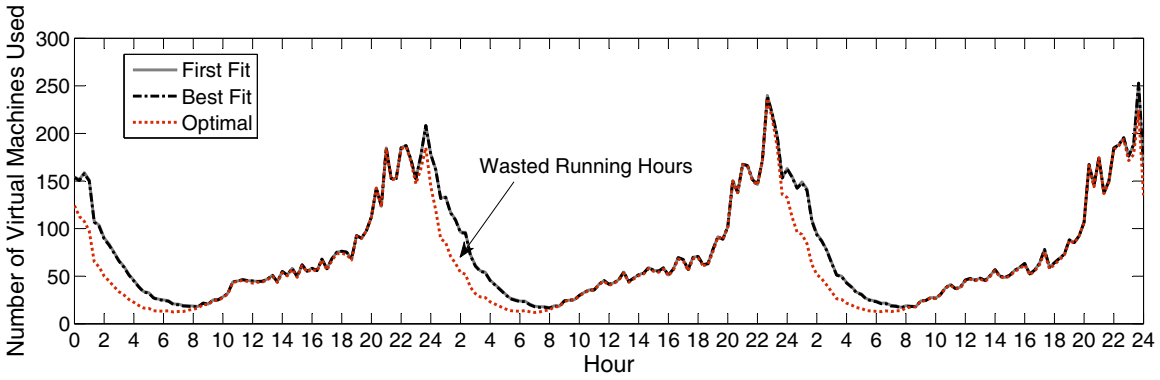

Fig. 3. Performance of First Fit and Best Fit dispatching for a sample period of 3 days

the active virtual machines that can accommodate r, we first consider the virtual machines whose expected shutdown times are later than $e(r)$ (lines 6-8). The rationale behind is that if r is assigned to a virtual machine whose expected shutdown time is later than $e(r)$, the total running hours of the virtual machines will not be increased after the assignment. This is because the virtual machine serving r needs to keep active at least to its expected shutdown time no matter whether r is assigned to it or not. If all the active virtual machines that can accommodate r have their expected shutdown times earlier than $e(r)$, r is then assigned to the virtual machine that has the latest expected shutdown time. The motivation here is to minimize the increase of the total running hours after the assignment (line 10).

V. Experiments

We conducted extensive experiments using the WoWAH dataset to evaluate the benefit of the proposed play request dispatching algorithm. In each experiment, we used different algorithms to dispatch the play requests (i.e., the game sessions in the WoWAH dataset) and collected the statistics of active virtual machines used at each time point. We also evaluated the impacts of the virtual machine capacity and the prediction errors of the game sessions' ending times on the performance of our ETPRD algorithm.

First, we assume the ending times of the game sessions are perfectly predicted in the ETPRD algorithm. Moreover, we assume each virtual machine is able to host up to 4 game instances concurrently. Figure 4 shows the number of active virtual machines used over the sample period as shown in

Algorithm 1 Ending Time based Play Request Dispatching (ETPRD)

1: Predict $e(r)$ for the play request r
2: Denote the set $V = \{v_1, v_2, ...\}$ as the set of active virtual machines that have enough computational resources to run the game instance for r
3: **if** $V = \emptyset$ **then**
4: Start a new virtual machine and assign r to the new started virtual machine
5: **else**
6: Denote by $V' \subseteq V$ the set of all the virtual machines v such that $T(v) > e(r)$
7: **if** $V' \neq \emptyset$ **then**
8: Assign r to any of the virtual machines in V'
9: **else**
10: Assign r to the virtual machine v' such that $T(v')$ is the latest (largest) among all the virtual machines in V
11: **end if**
12: **end if**

Figure 2. It can be seen that the active number of virtual machines used by the ETPRD algorithm in the declining period has been greatly reduced compared to the First Fit and Best Fit algorithms. This is because in the climbing period, the ETPRD algorithm packs the game sessions that have similar ending times into the same virtual machine. In the declining period, the game sessions assigned to the same virtual machine end at

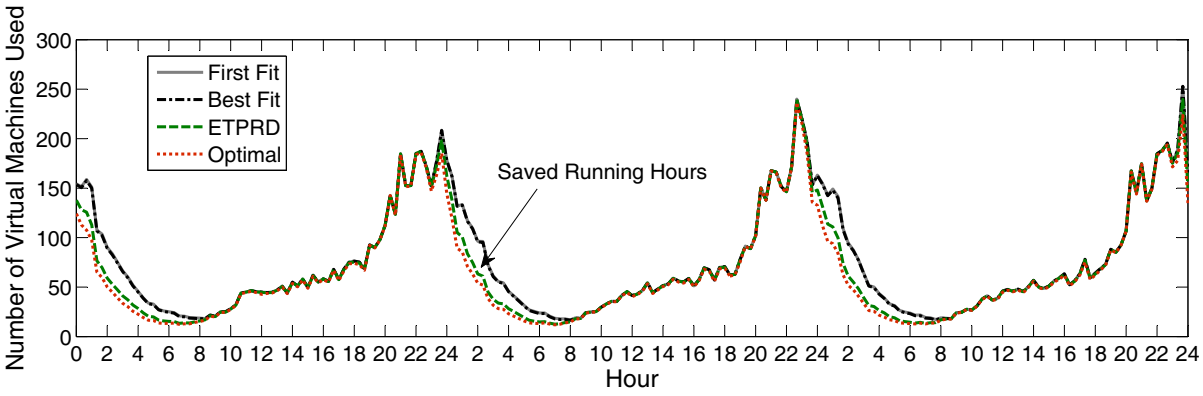

Fig. 4. Performance of different dispatching algorithms for a sample period of 3 days

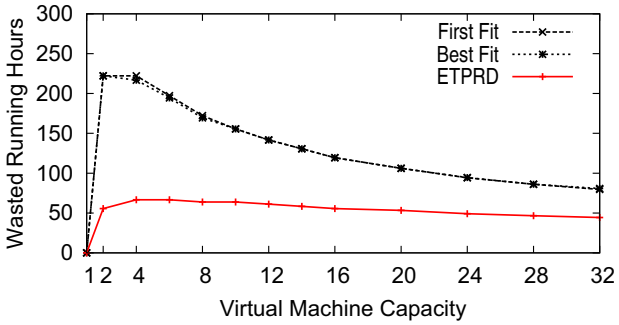

Fig. 5. Performance for different virtual machine capacities

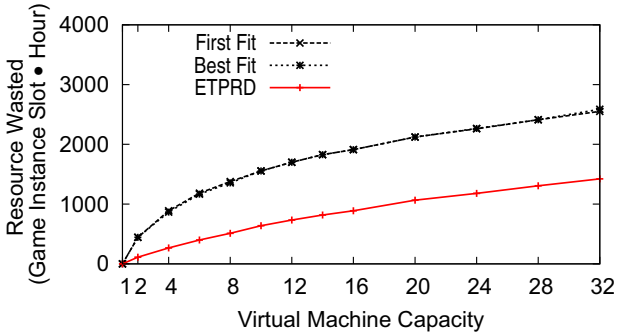

Fig. 6. Performance for different virtual machine capacities

around the same time. Therefore, each virtual machine is able to maintain high resource utilization throughout its running hours and can be shut down timely when the players depart. The area between the First Fit curve and the ETPRD curve is the total running hours that are saved by ETPRD compared to First Fit. It is easy to see that ETPRD can save almost 70% wasted running hours compared to First Fit and Best Fit dispatching.

Next, we evaluate how the virtual machine capacity (in terms of how many game instances can run on a virtual machine concurrently) would affect the performance of the ETPRD algorithm. We still assume that the ending times of the game sessions are perfectly predicted in this experiment. The virtual machine capacity is varied in the range from 1 to 32 (the recent GPU technology is able to support up to 32 game instances on a single board [3]). Figure 5 shows the average daily wasted running hours (i.e., the additional running hours compared to the optimal running hours) produced by different dispatching algorithms over the entire WoWAH dataset (i.e., the 3-year period from Jan. 2006 to Jan. 2009). It can be seen that for most of the virtual machine capacities, ETPRD can save more than 50% wasted running hours compared to First Fit and Best Fit dispatching. In the extreme case where the virtual machine capacity is only 1 (i.e., each virtual machine can run one game instance only), it is intuitive that all the dispatching algorithms (including Optimal) would occupy exactly the same number of virtual machines at any time and there is no wasted running hour at all. On the other hand, when a virtual machine can accommodate multiple game instances, the percentage of reduction in wasted running hours by our ETPRD algorithm gradually decreases with increasing virtual machine capacity. This is because given a set of play requests, when the virtual machine capacity is larger, it becomes more difficult to find sufficient number of play requests that share similar ending times to fill up a virtual machine to its capacity. It should be noted that a wasted running hour of a high-capacity virtual machine actually wastes more resources than a wasted running hour of a low-capacity virtual machine. Figure 6 shows the absolute amount of resource waste in terms of the game instance slots (which is given by the wasted running hours times the virtual machine capacity). It can be seen that as the virtual machine capacity increases, the resource waste cut by our ETPRD algorithm compared to First Fit and Best Fit dispatching also increases.

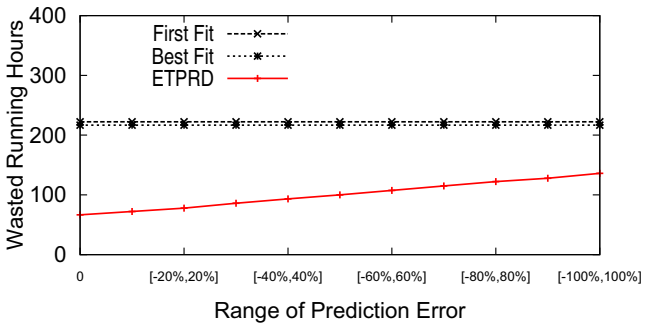

Fig. 7. Performance for different prediction errors

Finally, we evaluate how prediction errors of the ending times would affect the performance of the ETPRD algorithm. We again assume that the capacity of each virtual machine is 4. Since it is not the main focus of this paper to investigate specific prediction algorithms, we study the sensitivity of the ETPRD algorithm by assuming given prediction errors. Consider a play request r. Let $s(r)$ and $e(r)$ denote the starting time and ending time of its game session respectively. Denote by $p(r)$ the predicted ending time of the game session. The prediction error for r is defined as the ratio of $\frac{p(r)-e(r)}{e(r)-s(r)}$, i.e., the difference between the predicted and real ending times normalized by the actual length of the game session.

In the simulations, we tested different ranges of prediction errors. Given a range, we randomly generated a prediction error in the range for each play request. The daily averaged wasted running hours of ETPRD for different ranges of prediction errors are shown in Figure 7. As can be seen, the wasted running hours of ETPRD increases almost linearly as the prediction error increases. It can be observed that ETPRD has noticeable performance gains (38% reduction in wasted running hours compared to First Fit and Best Fit dispatching) even when the prediction errors are as high as [-100%, +100%].

VI. Conclusion and Future Work

In this paper, we have proposed a play request dispatching algorithm for cloud gaming systems to optimize their service costs. The proposed algorithm assigns the play requests according to the predicted ending times of game sessions. The experimental results show that the proposed algorithm can significantly reduce the total service cost compared to the classical First Fit and Best Fit bin packing algorithms. It is also seen from the experimental results that the benefit of the proposed algorithm is dependent on how accurately the ending times are predicted. Therefore, an important direction of future work is to investigate effective algorithms for predicting the ending times of game sessions.

VII. Acknowledgment

This research is supported by the Singapore National Research Foundation under its IDM Futures Funding Initiative and administered by the Interactive & Digital Media Programme Office, Media Development Authority.

References

[1] Gaikai web page. http://www.gaikai.com/.

[2] Mmogdata.net. http://users.telenet.be/mmodata/Charts/Subs-1.png.

[3] Nvidiagrid: http://www.nvidia.com/object/grid-technology.html.

[4] Onlive web page. http://www.onlive.com/.

[5] Streammygame web page. http://www.streammygame.com/smg/index.php.

[6] H. Ahmadi, S. Khoshnood, M. R. Hashemi, and S. Shirmohammadi. Efficient bitrate reduction using a game attention model in cloud gaming. In *IEEE International Symposium on Haptic Audio Visual Environments and Games (HAVE), 2013*, pages 103–108. IEEE, 2013.

[7] H. Ahmadi, S. Z. Tootaghaj, M. R. Hashemi, and S. Shirmohammadi. A game attention model for efficient bit rate allocation in cloud gaming. *Multimedia Systems*, pages 1–17, 2014.

[8] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM (JACM)*, 48(5):1069–1090, 2001.

[9] V. H.-h. Chen and H. B.-L. Duh. Understanding social interaction in world of warcraft. In *Proceedings of the international conference on Advances in computer entertainment technology*, pages 21–24. ACM, 2007.

[10] S. Choy, B. Wong, G. Simon, and C. Rosenberg. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *Proceedings of the 11th annual workshop on network and systems support for games*, page 2. IEEE Press, 2012.

[11] S.-P. Chuah and N.-M. Cheung. Layered coding for mobile cloud gaming. In *Proceedings of International Workshop on Massively Multiuser Virtual Environments*, pages 1–6. ACM, 2014.

[12] M. Claypool, D. Finkel, A. Grant, and M. Solano. On the performance of onlive thin client games. *Multimedia Systems*, pages 1–14, 2014.

[13] E. G. Coffman, J. Csirik, G. Galambos, S. Martello, and D. Vigo. Bin packing approximation algorithms: Survey and classification. *Handbook of Combinatorial Optimization (second ed.), Springer*, 2013.

[14] E. G. Coffman, Jr, M. R. Garey, and D. S. Johnson. Dynamic bin packing. *SIAM Journal on Computing*, 12(2):227–258, 1983.

[15] G. Galambos and G. J. Woeginger. On-line bin packingła restricted survey. *Zeitschrift für Operations Research*, 42(1):25–45, 1995.

[16] M. R. Gary and D. S. Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.

[17] M. Hemmati, A. Javadtalab, A. A. Nazari Shirehjini, S. Shirmohammadi, and T. Arici. Game as video: Bit rate reduction through adaptive object encoding. In *Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 7–12. ACM, 2013.

[18] T. Henderson and S. Bhatti. Modelling user behaviour in networked games. In *Proceedings of the ninth ACM international conference on Multimedia*, pages 212–220. ACM, 2001.

[19] H.-J. Hong, D.-Y. Chen, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu. Placing virtual machines to optimize cloud gaming experience. *IEEE Transactions on Cloud Computing*, 2014.

[20] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen. Gaminganywhere: an open cloud gaming system. In *Proceedings of the 4th ACM Multimedia Systems Conference*, pages 36–47. ACM, 2013.

[21] E. L. Lawler, J. K. Lenstra, A. H. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. *Handbooks in operations research and management science*, 4:445–522, 1993.

[22] Y.-T. Lee and K.-T. Chen. Is server consolidation beneficial to mmorpg? a case study of world of warcraft. In *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pages 435–442. IEEE, 2010.

[23] Y.-T. Lee, K.-T. Chen, Y.-M. Cheng, and C.-L. Lei. World of warcraft avatar history dataset. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 123–128. ACM, 2011.

[24] Y.-T. Lee, K.-T. Chen, H.-I. Su, and C.-L. Lei. Are all games equally cloud-gaming-friendly? an electromyographic approach. In *11th Annual Workshop on Network and Systems Support for Games (NetGames), 2012*, pages 1–6. IEEE, 2012.

[25] Y. Li, X. Tang, and W. Cai. On dynamic bin packing for resource allocation in the cloud. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '14, pages 2–11. ACM, 2014.

[26] M. Manzano, J. A. Hernández, M. Uruenña, and E. Calle. An empirical study of cloud gaming. In *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*, page 17. IEEE Press, 2012.

[27] S. Shi, C.-H. Hsu, K. Nahrstedt, and R. Campbell. Using graphics rendering contexts to enhance the real-time video coding for mobile cloud gaming. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 103–112. ACM, 2011.

[28] D. Wu, Z. Xue, and J. He. icloudaccess: Cost-effective streaming of video games from the cloud with low latency. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(8), 2014.

[29] M. Yu, C. Zhang, Z. Qi, J. Yao, Y. Wang, and H. Guan. Vgris: Virtualized gpu resource isolation and scheduling in cloud gaming. In *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '13, pages 203–214, New York, NY, USA, 2013. ACM.