TABLE OF CONTENTS

Introduction 4

Problem Description 4

Optimization Targets 4

Basic Assumptions 5

Model 1: Prediction of Players' Growth 6

Preliminary Data Analysis 6

Preliminary Hypothesis 6

Preliminary Model Verification 7

Think Deeper 8

Time Series Analysis 8

ARMA model 9

Model 2: Elastic Server Cluster 10

Introduction 10

Further Analysis 11

Assumptions & Hypothesis 12

Modeling 12

Numerical Solution & Model Verification 13

Model 3: Intelligent Queuing 15

Introduction 15

Assumptions & Hypothesis 16

Multi-Queue Model Analysis 17

Monte Carlo Simulation 18

Improving Queuing Model 21

Assumptions for Simplified EDD Queuing Model 22

Simplified EDD Queuing Model Analysis 22

Numerical Solutions for Improved Model 24

Conclusion 25

References 25

Appendix 26

Shrewd Online Game Manager

Abstract With more and more computer games coming out these years, optimization problems about online computer games have drawn greater concern. Every player may face the longer queuing time due to the growth of the population in the online game server. Our objective is to balance the players' satisfaction and the profit of the game's operator. We utilized the Fourier transformation and time series analysis (ARMA) to produce the forecast to the changing statistical number of players per month, and we verified the relatively more accurate model with the provided data. We further discussed the game operator's response to the growth of players. Inspired by feedback model in automatic control theory, we proposed a naive but easy-to-deploy solution where the number of servers can be adjusted dynamically according to the predicted player growth model. Non-linear optimization and classical queuing theory are used in this model. We verify the model against the real aggregation data, proving its effectiveness in satisfying the growing demand and reducing the costs. With the residual fluctuation still causing users to queue in some cases, we further discussed how to reduce the total waiting time for players in queue. Motivated by the process-scheduling algorithm in computer operating system, we considered every server as many virtual threads, each of which can provide service to only one player at a time, and every player's occupy time as time slices. By assigning different size of time slices to different amount of virtual threads, we made different virtual queues. We explored a sorting rule that guarantees minimum total waiting time. We further proposed a refinement model, which selects the most valuable players and exclusively enhances their satisfaction level by offering them privilege when logging-in. We also tried to combine the game currency, tokens, with this model. The three models in this paper focus on different aspects and make hypothesis covering different situations, and together they form a comprehensive solution to the original problem.

Keywords Time series analysis, ARMA model, Non-linear programming, load balancing, priority queuing model

Introduction

Problem Description

A certain game company has several servers running at the same time, each of which can offer service to a fixed number of players. The company recently launched a new paid game, and promised that the game currency, i.e. tokens, will be favorable for the players who frequently participate in the game for more than a certain period of time. If a player has bought the game currency, he doesn't need to pay for anything in the game. Soon after the discount policy launched, many players join in the game. However, the game operator discovered that due to this policy, a lot of players are kept waiting for the server to be ready, and players complain a lot.

1. Establish a mathematical model, aiming at reducing players' complaint. Use numerical analysis methods to test the model.
2. A new plan is to be given to predict the increase in the number of game players. Help the company establish a reasonable new plan.

Optimization Targets

Based on the original problem, we drew a conclusion that there are two major concerns in this player-operator system. From the aspect of the player, the queuing time is expected to be as short as possible, which requires adequate number of servers to provide services. Form the aspect of the game operators, the new policy brought more players than its accommodating capacity. In the meantime, since the cost of daily operation should be as low as possible, the operator cannot simply make the number of servers way too large than actual demand. However, this value cannot be too low either, for the operator needs to consider the players' satisfaction which is associated with his profit. We summarized this problem as a contradiction between operator's profit and user's satisfaction, and the model is intended to strike a balance between them.

Basic Assumptions

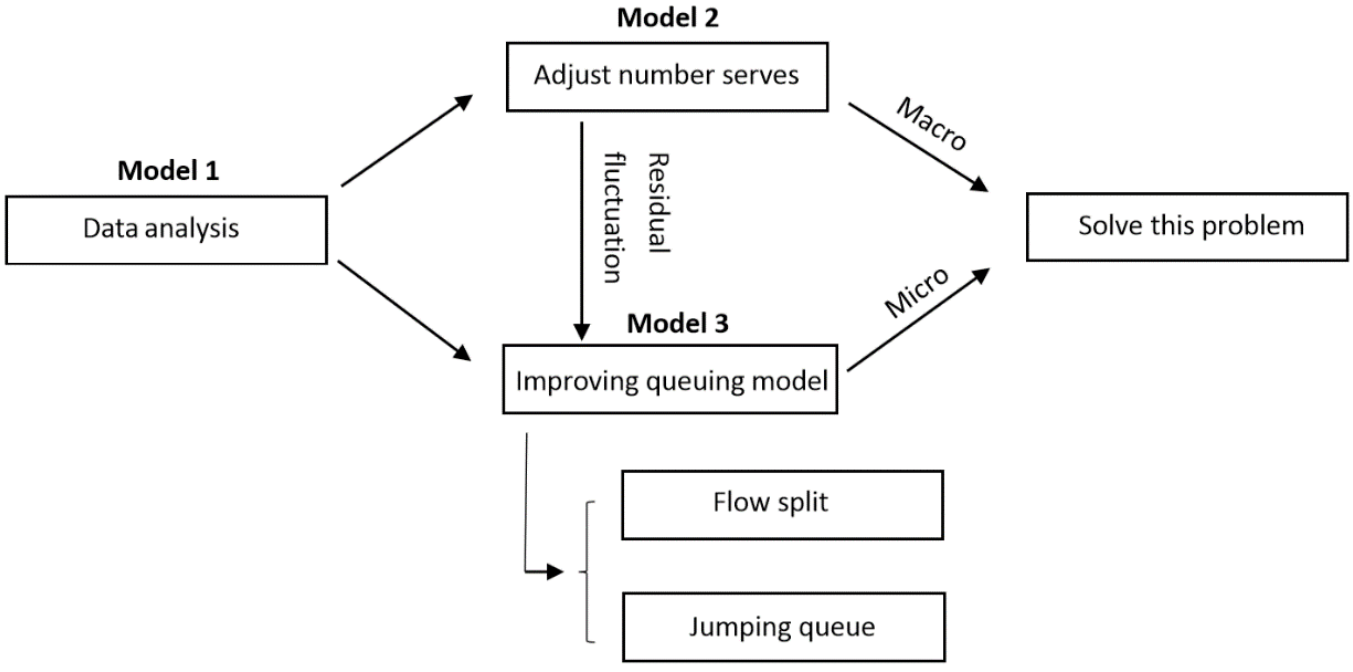
We proposed the following basic assumptions based on the analysis above.

1. The current discount policy works, and it has attracted more players to play this game.
2. The lack of severs is the major cause of long queue.
3. Players' complaint is caused by waiting in queue. The shorter queuing time leads to higher satisfaction level.
4. The satisfaction level is a factor affecting the growth of players.
5. The cost of the servers should be considered.
6. There are different types of players, and the time they spend in this game is not the same.

These assumptions are throughout the analysis in the flowing part of this paper.

In the following part of this paper, we proposed three different models, which focus on different aspects, and hypothesis covering different situations, and they together form a comprehensive solution to the original problem.

For convenience, we established a notation convention to express our model. In formulas of probability and statistics, we use upper-case letters to represent random variables and lower-case letters to represent observed variables.



Model 1: Prediction of Players' Growth

Preliminary Data Analysis

Driven by the idea that knowing the growth model may help the successive modeling process, we decided to predict the growth of players first.

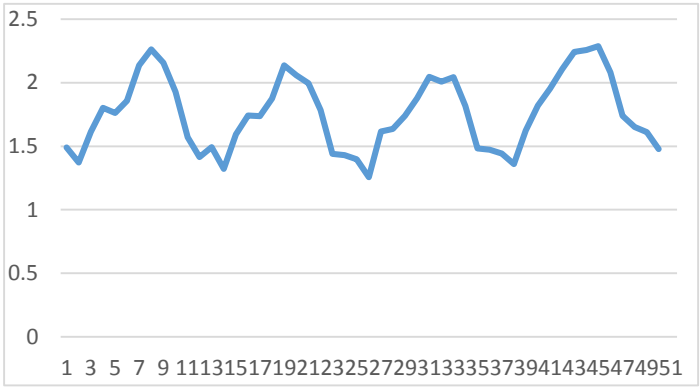


Figure 1Aggregation of players per month

The first glance of the aggregation of players per month (Figure 1) reminded us of the simple harmonic oscillation (SHO) in classical mechanics.

The data can be roughly regarded as oscillating between a certain number. The amplitude and frequency of oscillation are fixed. That leads to our preliminary hypothesis.

Preliminary Hypothesis

1. The number of players are oscillating between a certain number **H**, which is the holding power of the servers.
2. The increase of players causes longer queue. According to the basic assumption, this leads to decrease of players. The ratio of the number of queuing players **Q** to the decrease acceleration of players $\frac{\mathbf{d^2}}{\mathbf{dt^2}}\mathbf{Q(t)}$ can be expressed as constant k, so

$$\frac{\mathbf{d^2}}{\mathbf{dt^2}}\mathbf{Q(t)} = \mathbf{kQ(t)}$$

3. Likely, the decrease of players leads to shorter queue, which in turn draws more

players.

4. Hypothesis 2 and 3 combined to form a restoring force, so the system appears to be a SHO, which can be formulated as:

$$\mathbf{Q(t) = Asin(\omega t + \theta)}$$

Preliminary Model Verification

In order to get ω and A in the formula, Fourier transformation was used to analysis the input data, then we picked the maximum point from peaks in the frequency spectrum and drew the sin function curve according to the obtained parameters.

The MATLAB software was used to verify our hypothesis. We tried the first three years' data in the provided table. The fitting result has a fractional error of 5% in fitting the curve and a fractional error of 8.2% in predicting future data, which is not an ideal outcome.

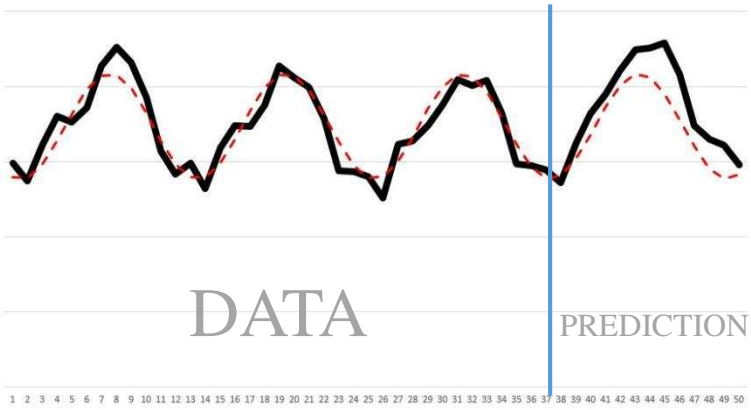
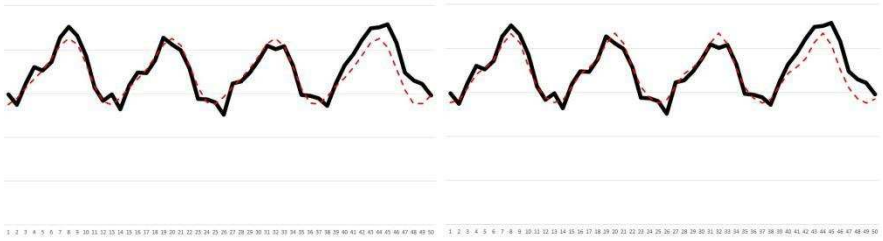


Figure 2 Sin Function Fitting

In Figure 2 we can directly find that there are many fitting errors and the prediction is generally smaller than the actual value.

We tried to increase the order **n**, which indicates the number of sin functions used to fit the data. However, the outcome is still bad.



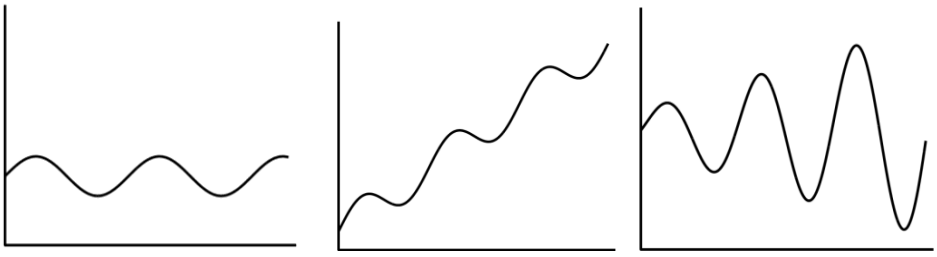
Think Deeper

During further discussion, we found the preliminary model has following drawbacks.

1. It totally ignores the increasing trend of the data. In the preliminary model, the steady linear differential system we built perfectly describes periodic, but has no component that describes the growth of average value per year.
2. This model cannot explain the oscillation well either. Hypothesis 2 and 3 predicts the oscillation caused by queuing, but the frequency should be much higher. What's more, the main oscillation we observed has a period of approximately one year, which is too long to be related to queuing oscillation.
3. We investigated that in some online games, like LOL, the population of gamers is larger than usual during some annual celebration event.

Time Series Analysis

When analyzing the drawback of the former model, we also found that the traditional models like population models and SHO model can only describe a part of factors of the growth. The actual growth has many components, such as steady oscillation, trend of increase and increase of amplitude, etc.



Other essential factors excluded in this system, like celebration events, may also affect the growth of players. Some of these factors are hard to quantify. In addition, it

is hard to adjust the model to include all of these factors. Thus, we drew a conclusion that we should find a modeling tool that can make full use of the input data without explicitly assigning specific variables to represent different kind of growth components.

After further discussion, we proposed that time series analysis is suitable for modeling this problem. Time series analysis can deal with the system that has many variables with trend, periodic, and other ones changing in an uncertain manner.

The basic idea about time series analysis is simple. For any crazy-looking sequence, we can use one-order differentiating to eliminate liner trend, and use k-order (in this case, 12-order for a year) to eliminate periodic component, then we get a differentiated sequence. If we can model this sequence, we can restore the original sequence and even predict the future of that sequence.

ARMA model

Autoregressive moving average model (ARMA model) is a method to describe differentiated sequence based on the assumption that differentiated sequence comes from a Gaussian ergodic process.

ARMA model can be represented by the relation:

$$Y_i + \sum_{j=1}^{l} \alpha_j Y_{i-j} = e_i + \sum_{j=1}^{m} \beta_j e_{i-j}$$

Where e_i is a sequence of independent and identically distributed Gaussian random variables, with zero mean and variance σ^2. In this formula, l and m are non-negative integers with at least one of them being non-zero. If l is zero, this model degrades into a MA model. If m is zero, it degrades into an AR model. α_j, β_j follow the constrain that roots of $Q_1(x) = x^l + \sum_{j=1}^{l} \alpha_j x^{i-j} = 0$ and $Q_2(x) = x^m + \sum_{j=1}^{m} \beta_j x^{m-j} = 0$ are all in the unit circle.

Since the chosen value of l, m will greatly affect the outcome when modeling with ARMA, we have to try different combinations of parameters and calculate coefficient

α_j, β_j separately, then evaluate the error rate.

Model Verification

We use the first three years of data to generate the fourth year's forecast. We choose l=3 and m=1 to obtain best result, and get α_1=0.921121, α_2=-0.140226, α_3=0.0541096, α_4=-0.400429, β_1=-0.14599, β_2=-0.264143.

As we can see in the Figure 3 , where the solid line represents the original data from year 2012 to 2014, and the dotted line represents the prediction data for the 2015, time series analysis can better predict the future of growth, with a fractional error of 4.4%

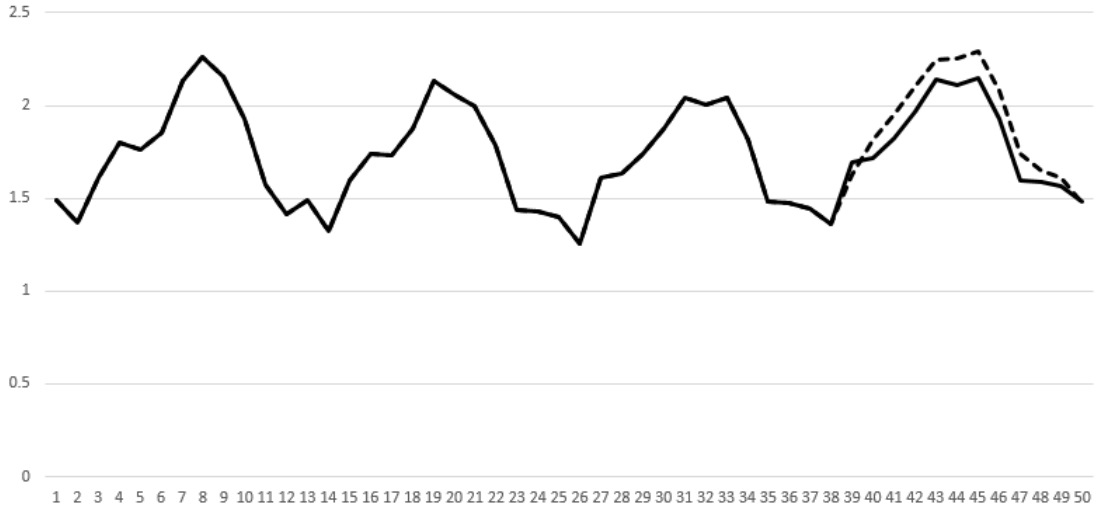


Figure 3 ARMA Prediction

Model 2: Elastic Server Cluster

Introduction

From the monthly data of the cumulative population in games, the number of players in the year can be 40% higher than average, with a max-min peak gap reaching 200 thousand people. Since there is no server can cope with such a huge load fluctuation, it is necessary to adjust the number of servers in the appropriate time to respond. Model 1 provide a macro forecast value of the number of players. Inspired by the feedback in control theory automatic, we designed a set of model that can not only suppress the fluctuation but also increase the number of players by adjusting the

number of servers.

The number of servers needs to be match with the number of players predicted in the model 1, otherwise it will be overload of idle. However, in the actual situation, the server can not exactly keep up with the changing populations. On the one hand, each server has its own holding power with only two states: open and closed. Once the server is open, the required cost is constant, and cannot guarantee a full load. On the other hand, since the server stores the current game files, opening or closing the server requires transferring these files to other servers and modifying the network topology. The cost of this adjusting cannot be ignored, so the server cannot be switched too frequently.

Idle phenomenon will lead to a waste of resources, which obviously increases the cost. Overload phenomenon will increase the waiting time for players or refuse some players to log into the game. Although this will not directly affect the profit, but it will give an indirect impact to the number of players. According to the conclusion of the model 1, it will have a positive feedback if the load capacity of the server can be slightly more than the number of players during the increasing population in games. In addition, when the number of players is decreasing, the capacity should not reduce too fast, otherwise, it will cause the negative feedback and the loss of players.

In order to quantify this phenomenon, we convert the loss of players, passed by the feedback of server overload, into a Penalty Cost. We assume that Penalty Cost is proportional to the average waiting time for the players, which is obtained by the MMS model in queuing theory.

Our goal is to use the non-liner programming to solve the total cost of the minimum value.

Further Analysis

In order to simplify this problem, we used single-queue model in the queuing theory. The queuing people of overloaded server divided by the number of the servers is regarded as average queue length.

The number of players is constant in one month. Although, the number of players and the length of queue will change periodically every day. However, considering the small fluctuations in these short time scales have little effect on the large scale in a month, and the situation is similar for every month, they can be approximately converted to a fixed value and then be added to the fixed monthly cost of the server.

Since the server cannot be switched too frequently, we assume that each server has a fixed switching cost, and each server will only be switched open or closed at the beginning of the month.

Assumptions & Hypothesis

1. Players per month count number of online players.
2. Every running sever has a constant cost, no matter it is over loaded or idle.
3. Penalty cost is proportional to the average waiting time for the players.
4. Switching the state of a server has a certain amount of cost.
5. Changing the state of a server can only happen once a month.
6. Cost per server per month is unit cost.

Modeling

We list some variables in this model, and their estimated value in the following table.

Symbol	Explanation	Estimation
N	Number of servers	Approximated 4 to 10.
n	Holding power of every server	About 1500 players
P	Players online	Total players × T Approximated 5000 to 9000.
C_o	Switching cost	About $40 \times \zeta$
C_r	Cost per server per month	Unit cost $1 \times \zeta$
C_f	Penalty cost	Explained in the formula

t	Average playing time	3 hours ~ 0.125
x	People in queue	$x = p - nN$, negative for free space

According to queuing theory, queue length can be represented as:

$$x_i = \frac{x}{N}$$

Thus average waiting time is:

$$x_i * \frac{t}{n}$$

Total waiting time is:

$$w = \frac{(P - n * N)^2}{n * N}$$

According to the assumption, the Penalty Cost is:

$$C_f = \zeta \times w$$

So total cost per month is:

$$C_r * N + C_f$$

Considering one year, we have:

$$\sum_{i=1}^{d} (C_r * N(i) + C_f(i) + |N(i-1) - N(i)|)$$

Simplify this expression and we get the optimization target:

$$\min \sum_{i=1}^{d} (C_r * N(i) + k_1 \frac{(P - n * N)^2}{n * N} + k_2 |N(i-1) - N(i)|)$$

Subject to the constrain that every N(x) is positive integer.

Numerical Solution & Model Verification

The given data of 5 years is used to conduct the model verification. We estimate the parameter as $k_1 = 0.125,\ k_2 = 1, k_3 = 750$, by the typical value from some online cloud computing providers. (other solutions using different parameters can be found in the appendix.)

We use the Differential Evolution algorithm in Mathematica to solve for a minimum cost. The optimized solution for numbers of servers is expressed by bar chart together with the number of players in Figure 4. The number of servers does not strictly follow

the changes in the number of players, and it is tend to be smooth, for what we wish.

Parameter: k1=0.125 k2=1 n=750;
Solution: {Minimum total cost: 539.54, When: {X[1] -> 8, X[2] -> 8, X[3] -> 9, X[4] -> 10, X[5] -> 10, X[6] -> 10, X[7] -> 12, X[8] -> 12, X[9] -> 12, X[10] -> 11, X[11] -> 9, X[12] -> 8, X[13] -> 8, X[14] -> 8, X[15] -> 9, X[16] -> 10, X[17] -> 10, X[18] -> 10, X[19] -> 12, X[20] -> 11, X[21] -> 11, X[22] -> 10, X[23] -> 8, X[24] -> 8, X[25] -> 8, X[26] -> 8, X[27] -> 9, X[28] -> 9, X[29] -> 10, X[30] -> 10, X[31] -> 11, X[32] -> 11, X[33] -> 11, X[34] -> 10, X[35] -> 9, X[36] -> 9, X[37] -> 9, X[38] -> 9, X[39] -> 9, X[40] -> 10, X[41] -> 11, X[42] -> 12, X[43] -> 12, X[44] -> 12, X[45] -> 12, X[46] -> 12, X[47] -> 10, X[48] -> 9, X[49] -> 9, X[50] -> 9}}

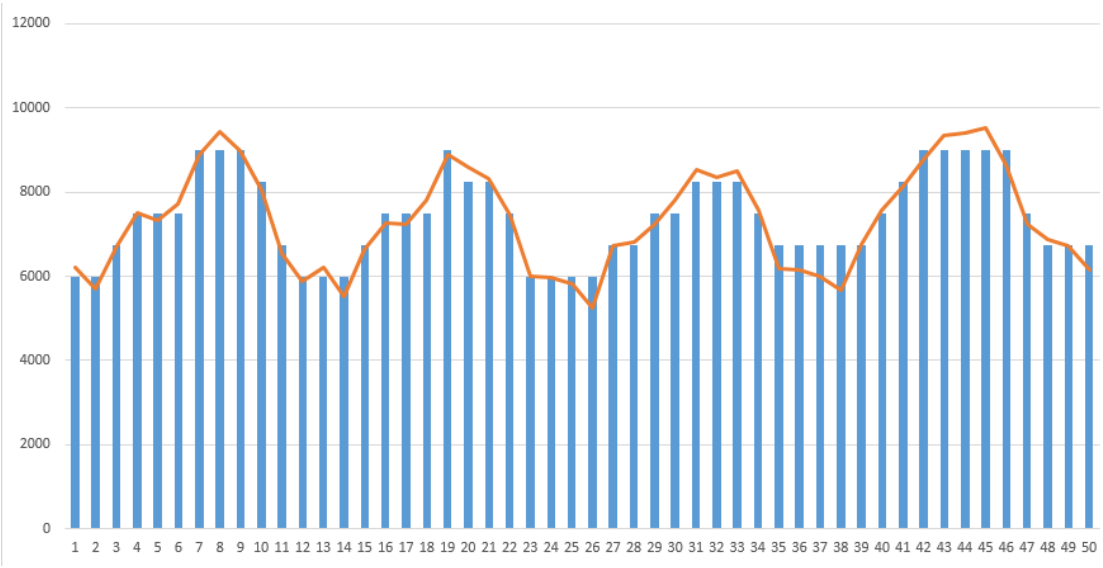


Figure 4 Optimized Solution

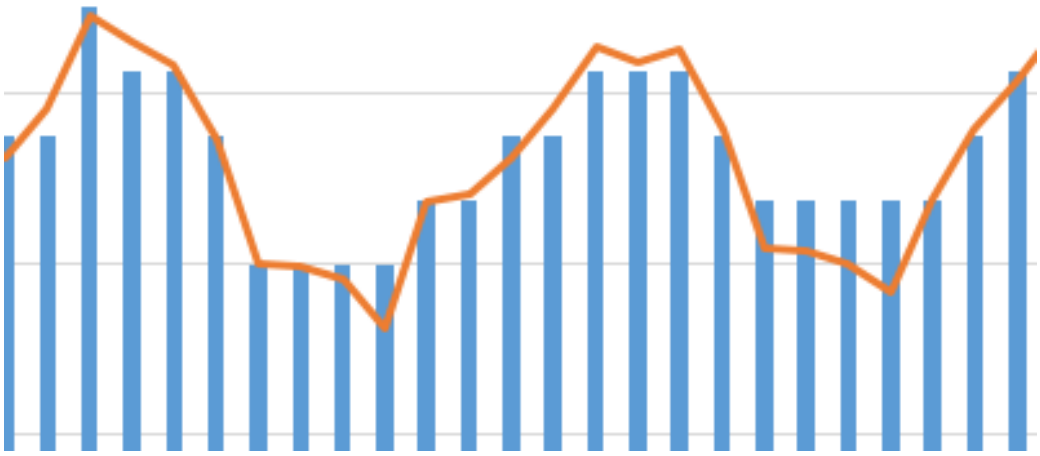


Figure 5 A slice of Fig.4

Model 3: Intelligent Queuing

Introduction

Based on the results of the model 2, we draw a line graph of the number of the holding power of ideal number of servers minus the number of online players. As can be seen from the Figure 6, even with the model 2 adopted, the number of people is still affected by residual fluctuation. The overload status means that players need to wait for a long time, or even unable to play the game. For further optimization, we introduced a third model.

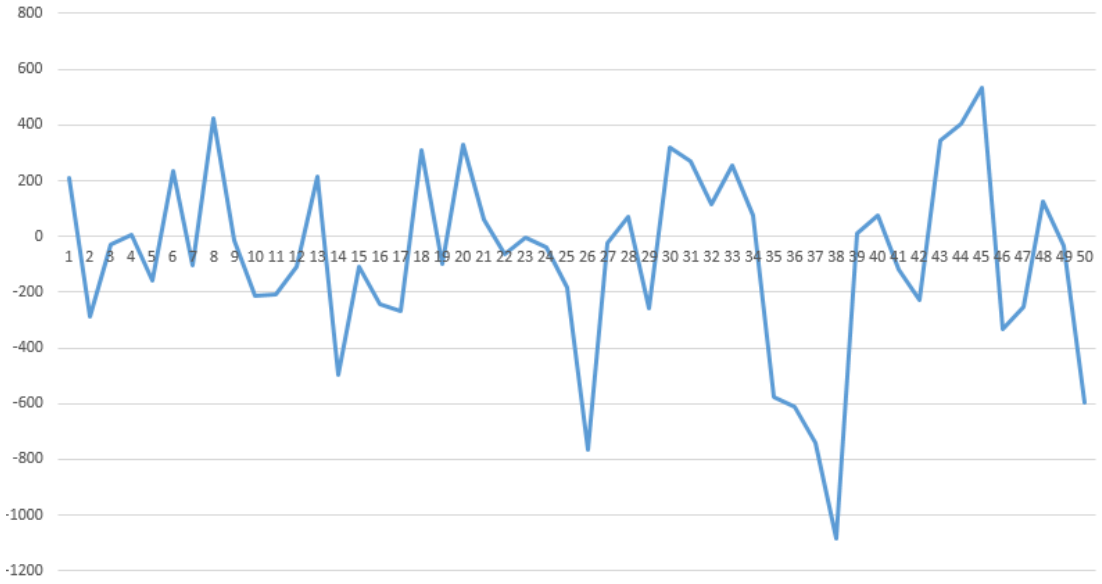


Figure 6 Residual Fluctuation

In this model, we take into consideration the relationship between the average waiting time and the demand time of the players in the queue. We assume that each player's demand time is known, so each player's waiting time can be pre-calculated. Because of the first in first out rule, the waiting time of every single player is equal to the sum of demand time of all the players ahead.

The following example in Figure 7 illustrates the two extreme cases. The arrival order arrayed according to the player's demand time from small to large indicates the best situation where the total waiting time of the queue is the minimum, while the contrary

is the worst case.

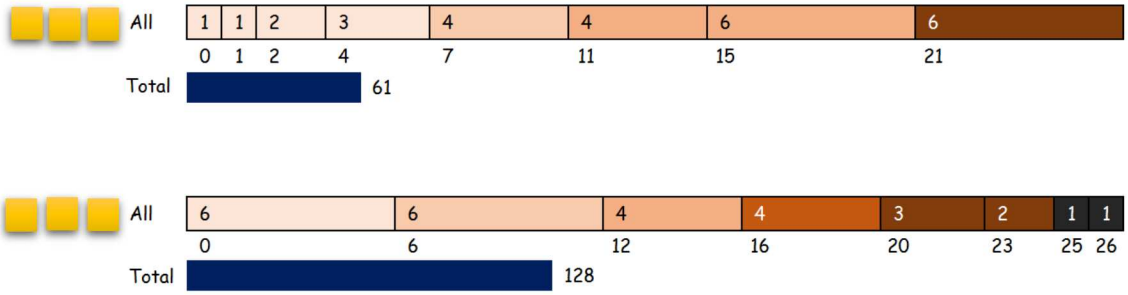


Figure 7 Best Case & Worst Case

Motivated by the process-scheduling algorithm in computer operating system, we abstracted every server into many identical virtual threads, each of which can provide service to only one player at a time. By assigning different size of time slices to different amount of virtual threads, we make different virtual queues. This model allows multiple players to enter virtual queues with many parallel threads to eliminate the effect of arrival order. As shown in Figure 8, we sort every player into different virtual queues according to his demand time. This method guarantees minimum total waiting time without adding more servers.

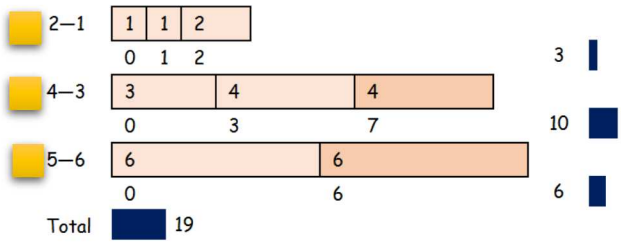


Figure 8 Multi-Queue Model

Assumptions

This model is based on the following assumptions.

1. Every player's demand playing time can be measured or estimated.
2. Once a player enters the queue, all parameters about him is constant. Player in queue will not change his demand time.
3. Maximum waiting time is considered when players are not patient enough.

Multi-Queue Model Analysis

The key point of this model is to adjust the appropriate sizes of virtual queues.

We assume t as the average gaming time, λ_t is the percentage of players' whose demanded playing time is t, and k_t is the size of virtual queue for players' whose demanded playing time is t, which is represented by percentage of total computing power of all servers.

According to classical queuing theory, we further define that for every virtual queue for players' whose demanded playing time is t, L_t is the average queue length, μ_t is the average player's demanded time, and T_t is the average waiting time.

Assume L_t is proportional to λ_t, then the average queue length is:

$$L_t = t_t \lambda_t$$

Average serving time is:

$$\mu_t = t_t \frac{\lambda_t}{k_t}$$

Average waiting time is:

$$T_t = \mu_t L_t = C_1 \frac{t_t \lambda_t^2}{k_t} k_t$$

So the total playing time for all players is:

$$\sum_{i=1}^{n} C_1 \frac{t_i \lambda_i^2}{k_i} \lambda_i = \sum_{i=1}^{n} C_1 \frac{t_i \lambda_i^3}{k_i}$$

Substitute $C_1 \frac{t_i \lambda_i^3}{k_i}$ as $\frac{a_i}{k_i}$, then we have:

$$\sum_{i=1}^{n} \frac{a_i}{k_i}$$

Considering

$$\sum_{i=1}^{n} k_i = 1$$

Thus we have

$$\sum_{i=1}^{n} \frac{a_i}{k_i} = \sum_{i=1}^{n} \left(\frac{a_i}{k_i} \sum_{j=1}^{n} k_j \right) = \sum_{i=1}^{n} \left(a_i \sum_{j=1}^{n} \frac{k_j}{k_i} \right)$$

Differentiate by k_i we derive:

$$\frac{\partial \sum_{i=1}^{n}\left(\mathrm{a_i}\sum_{j=1}^{n}\frac{k_j}{k_i}\right)}{\partial \mathrm{x}} = \sum_{j=1,j\neq i}^{n}\left(-\mathrm{a_i}\frac{\mathrm{k_j}}{\mathrm{k_i^2}}+\frac{\mathrm{a_j}}{\mathrm{k_j}}\right)$$

Assuming all components of derivative for k_i are zero, then we have equation:

$$\mathrm{a_i}\frac{\mathrm{k_j}}{\mathrm{k_i^2}} = \frac{\mathrm{a_j}}{\mathrm{k_j}}$$

which is the minimum solution. Then we have:

$$\forall\, i = 1\ to\ n\ , j = 1\ to\ n \quad \frac{\mathrm{k_j^2}}{\mathrm{k_i^2}} = \frac{\mathrm{a_j}}{\mathrm{a_i}}$$

Substitute it back and we have:

$$\forall\, i = 1\ to\ n\ , j = 1\ to\ n \quad \frac{\mathrm{k_j}}{\mathrm{k_i}} = \frac{\lambda_j^{\frac{3}{2}}\sqrt{t_j}}{\lambda_i^{\frac{3}{2}}\sqrt{t_i}}$$

So the total waiting time is smallest when:

$$\mathrm{k_i} = \mathrm{C}\,\mathrm{t_i^{\frac{1}{2}}}\lambda_\mathrm{i}^{\frac{3}{2}}$$

Then we can assign the size of virtual queues according to this conclusion.

Monte Carlo Simulation

There are two groups, an experimental group, which utilizes this multi-queue model, and a control group, which uses single queue model. They have the same total capacity.

We use the same Gaussian distributed sample to conduction Monte Carlo simulation. Simulating with the ratio of k1:k2:k3=1:1.41:1.73, which is set according to the derivation above, we obtained the following data.

Comparing the experimental group with the control group, the average waiting time is reduced by 40%. This model proves to be effective to increase satisfaction.

In Figure 9, X-axis is time; Y-axis is the queue length at this moment. The black curve is the data of control group and the other three curves are three parts of experimental server.

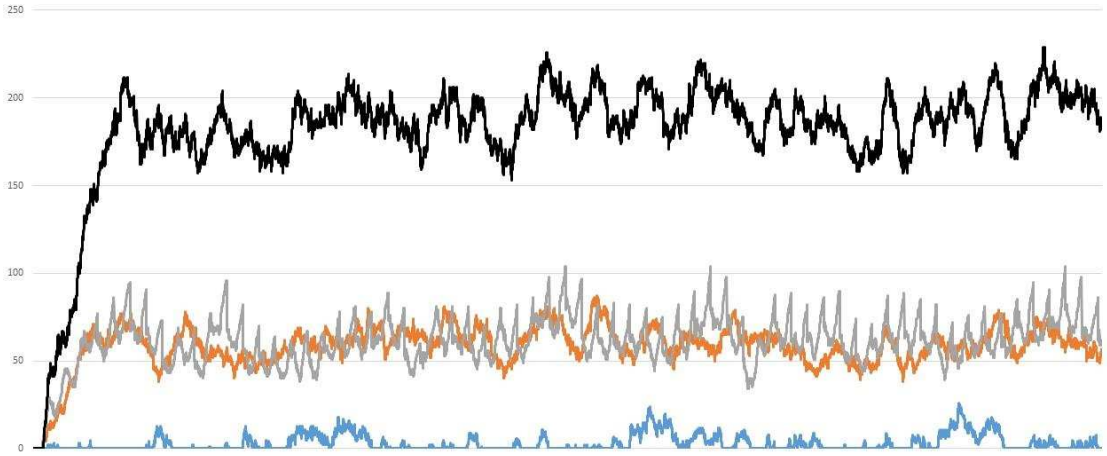


Figure 9 Queuing length

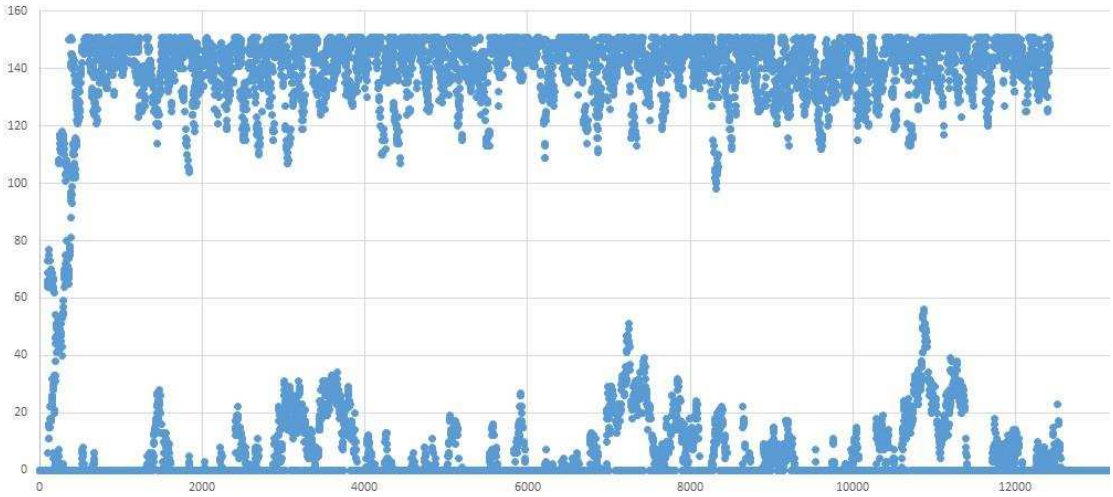


Figure 10 Waiting time & number of players

Figure 10 is the waiting time of control group and Figure 11 is about experimental group. X-axis is the number of players; Y-axis is the waiting time.

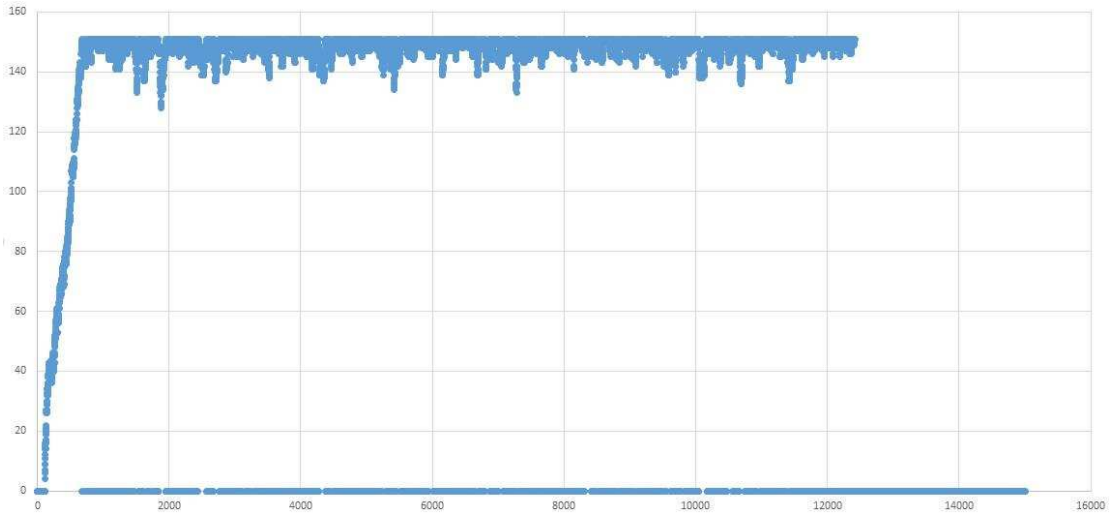


Figure 11 Waiting time & number of players

Concluded from Figure 9, Figure 10 and Figure 11, it obvious that the average waiting time of players, whose playing time is shorter and profits is higher, is reduced. Moreover, the average waiting time of players who play for long time, is also slightly lower. The purpose of reducing the total waiting time is achieved.

We further repeated the experiment and drew another two figures:

Figure 12: the total length is the waiting time of control group and the blue half is that of experimental group. Figure 13: X-axis is the total waiting time of control group; Y-axis shows the ratio of reducing.

Concluded from picture Figure 12 and Figure 13, this method is stable overall.

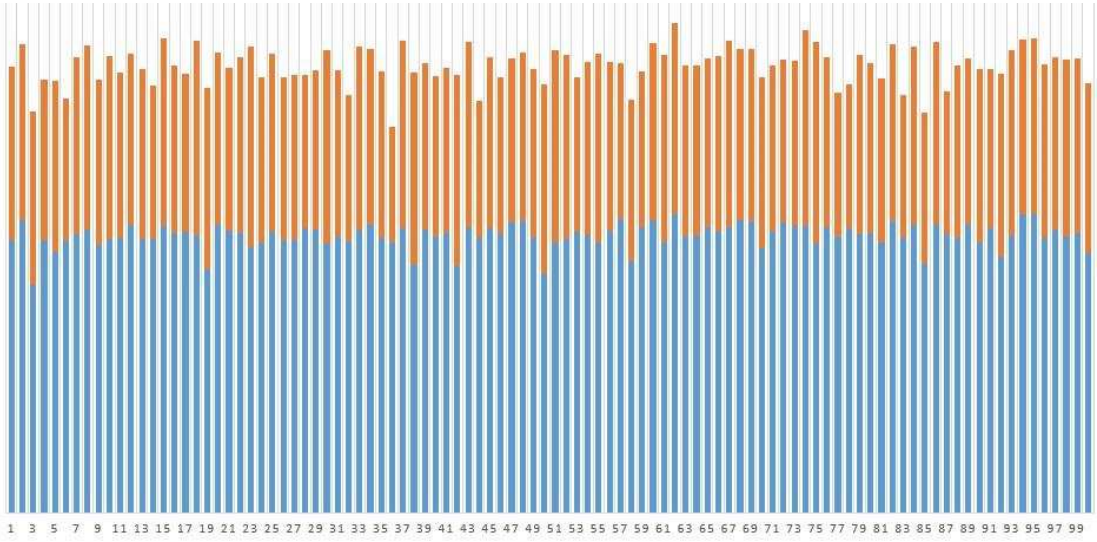


Figure 12

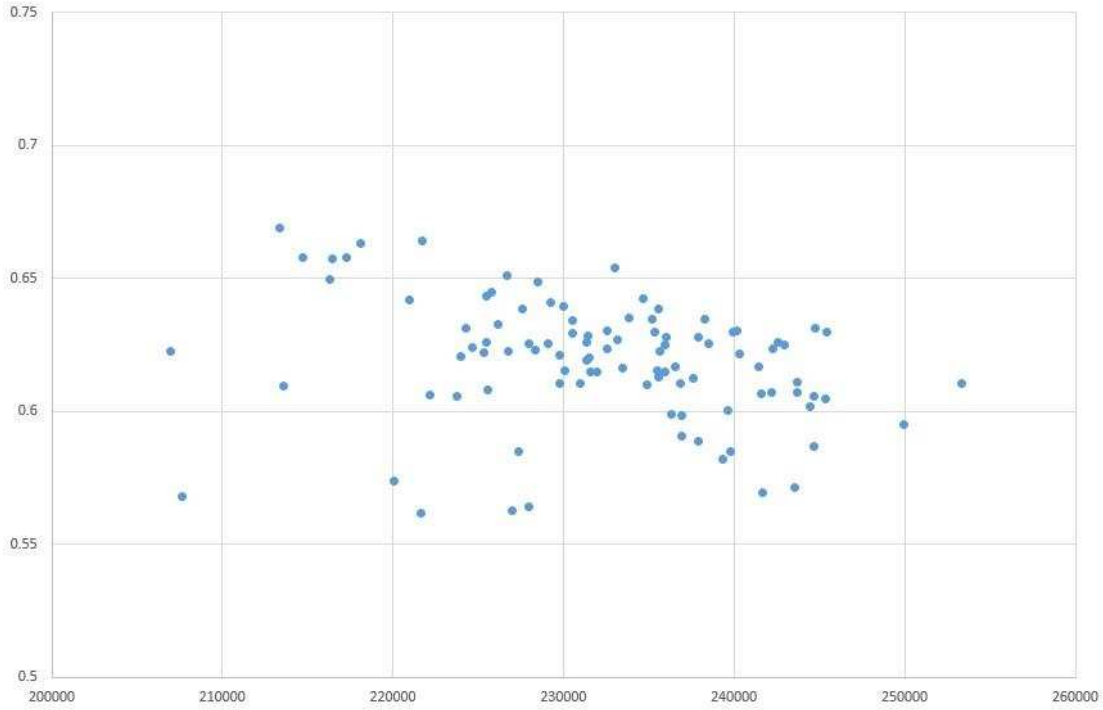


Figure 13

Improving Queuing Model

The model above can optimize user queuing satisfaction holistically. Based on Pareto principle, that is, 20% of users create a benefit of 80%, and the fact that residual fluctuation in model 2 is mainly caused by the low-quality players, who come and go unpredictably, we proposed a modified queuing model to refine high-end players.

In our model, tokens and RMB's exchange rate maintain constant. The policy is simple: Players can buy a fixed period of time. During this time, players can enter the server freely without paying any more. More time, more discount. So different player may have different consumption in a period (like one month or longer).

All players are ranked by the money they spend in a period. Considering every virtual queue, which can be abstracted into a room for all players playing at the same time, there are two categories of players. One category of players is the high priority players, who rank in the top 20 and the other category of players is low priority players (ordinary players). In the premise of ensuring the user experience of the majority of players. We focus on improving the satisfaction of the first category of

players, which will improve the company's revenue.

As we know, there are two widely used queuing policy, one is FCFS (first come first service), and the other is HOL (head of line). The first model fully appreciates the fair, but cannot increase the high-end customers' experience. The second model takes into account the high-end customers. However, when the queue is long enough, it will cause serious injury to ordinary players' experience. The ordinary player at the end may never be able to enter the game.

For this reason, the introduction of a simplified earliest due date rule (EDD) queuing model which take the both problems of former models into account. In this model, players have two different levels of priorities. Our main job is to calculate the appropriate maximum jump length for privilege players.

Assumptions for Simplified EDD Queuing Model

1. Players will not leave the queue before entering the server
2. Queue length is limited.
3. Time utility function is a straight line when there is no player jumping the queue
4. There are two types of users, low-priority and high-priority.

Simplified EDD Queuing Model Analysis

We define the following variables for further analysis.

Symbol	Explanation
L	the length of queue
μ	the number of bits a high-priority player jumps forward
$\mathbf{L_{max}}$	the maximum length of a queue
ι	player's position in the queue
α	gain sensitivity coefficient
$\boldsymbol{\beta}$	loss sensitivity coefficient
$\mathbf{U}(\iota)$	the time utility when player's position in the queue is ι

In the EDD model, the players are sensitive about the length of queue, not time. Suppose the length of queue is L, a high-priority player jumps ahead μ when entering the queue. Utility curve is to measure satisfaction levels of queuing. If you suffered a sudden loss, the decrease of utility is higher than under normal circumstances, and vice versa. Time utility curve can be established using common power function curve. Assume $L_{max} \neq 0$ is the maximum length, $\mathrm{U}(\iota)$ represents the time utility when player's position in the queue is ι, and $\mathrm{U}(L_{max}) = 0$. When no one jumps the queue, the players enter server in order, utility value increases linearly:

$$\mathrm{U}(\iota) = \frac{L_{max} - \iota}{L_{max}}$$

When one jumps the queue, it is expected to shorten the waiting time for players who jumps. The function has a convex curve. Assume gain sensitivity coefficient $\alpha \in (0,1)$ we get:

$$U_1(\iota) = \left(\frac{L_{max} - \iota}{L_{max}}\right)^{\alpha}$$

When one jumps the queue, players behind him are expected to wait more time, a sudden loss. The function has a concave curve. Assume loss sensitivity coefficient $\beta \in (0,1)$,then we get:

$$U_2(\iota) = \left(\frac{L_{max} - \iota}{L_{max}}\right)^{\beta}$$

For determining a reasonable maximum number of jumping the queue, the EED model requires jumping without reducing overall utility, or in other words, $\Delta\mathrm{U} = \mathrm{U}^{+} - \mathrm{U}^{-} \geq 0$. U^{+} is the increasing of the amount of total utility in the entire system. U^{-} is the decrease in the entire system. Suppose a player reaches the queue and queue length is L. After the jump, the player's queue position is $\iota \in (0, \mathrm{L}]$, have

$$\Delta\mathrm{U} = \left(\frac{L_{max} - \iota}{L_{max}}\right)^{\alpha} - \sum_{i=\iota}^{L}\left(\frac{L_{max} - i}{L_{max}}\right) + \sum_{i=\iota+1}^{L}\left(\frac{L_{max} - i}{L_{max}}\right)^{\beta} \geq 0$$

Through questionnaires and curve fitting, α and β can be calculated. In this case, the formula can give the maximum number of positions to jump forward.

Numerical Solutions for Improved Model

Due to the lack of relevant data, we use the data from the reference to test the rationality of the model. According to reference, through questionnaires and curve fitting, $\alpha = 0.25108$, $\beta = 1.257538$, and suppose L_{max}=18. ΔU can be plotted using MATLAB. In Figure 14, X-axis is the player's position in the queue, Y-axis is ΔU.

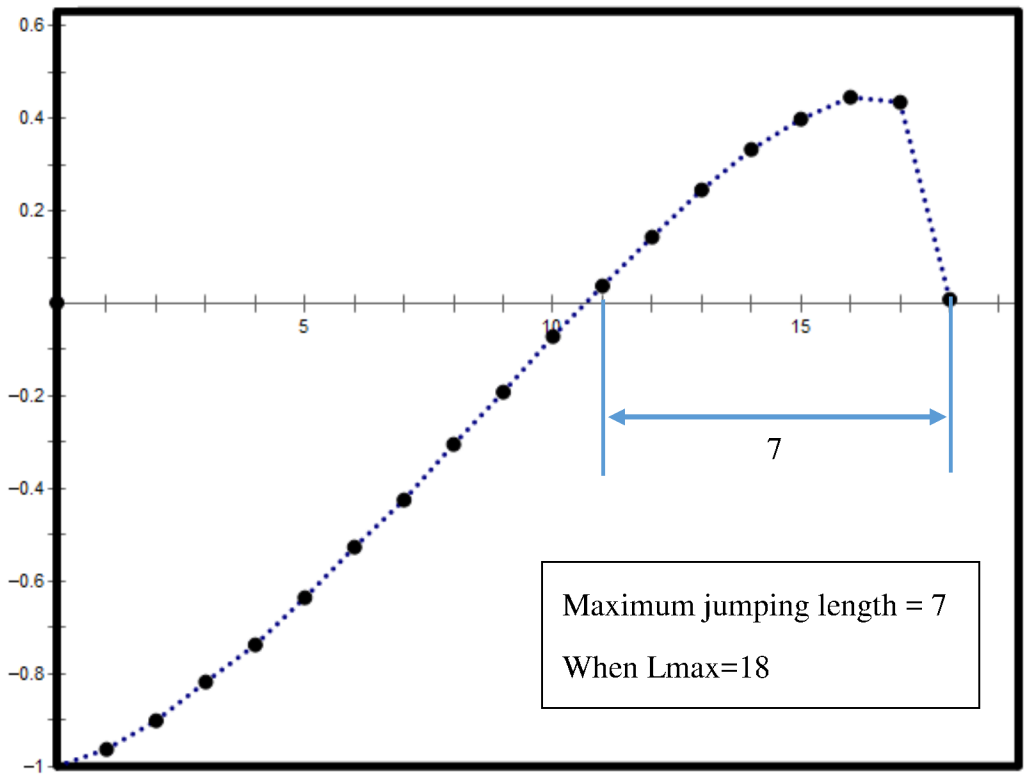


Figure 14

We conclude that for different maximum length of queue, the maximum jumping length is also different, so the operator should choose the appropriate parameter according to its own investigations and researches.

Maximum Length of Queue	18	17	16	15~13	12~9	8~5	4~2	1
Maximum Jumping Length	7	6	5	4	3	2	1	0

Conclusion

With the three models combined, we provide a comprehensive solution to the original problem. We build a model to predict players' growth accurately, which guided the second model to eliminate large gaps between sever load and player's demand, with a manner of dynamically changing the number of servers, With the third model, we reduce the average waiting time and especially that for high-end players. These three models together give a practical solution to improve the satisfaction of consumers while obtaining the best revenue for the game operator.

References

[1].厉譞.苏强.朱岩.LI Xuan.SU Qiang.ZHU Yan 改进 EDD 策略在门诊排队管理中的应用[期刊论文]-清华大学学报（自然科学版） 2007(11)

[2].张世斌.数学建模的思想和方法.上海交通大学出版社.2015(1)

[3]. Mobile Game Data Analysis White Paper THINKDATA Inc.

Appendix

时间序列预测数据：

1.4905	1.4905
1.3706	1.3706
1.6129	1.6129
1.802	1.802
1.7615	1.7615
1.8558	1.8558
2.1356	2.1356
2.2615	2.2615
2.1572	2.1572
1.9283	1.9283
1.5705	1.5705
1.414	1.414
1.4917	1.4917
1.3207	1.3207
1.5942	1.5942
1.7416	1.7416
1.7356	1.7356
1.8747	1.8747
2.1364	2.1364
2.0586	2.0586
1.9945	1.9945
1.7847	1.7847
1.4396	1.4396
1.4304	1.4304
1.3962	1.3962
1.256	1.256
1.6144	1.6144
1.6364	1.6364
1.7379	1.7379
1.8764	1.8764
2.045	2.045
2.0074	2.0074
2.0417	2.0417
1.8177	1.8177
1.4822	1.4822
1.4733	1.4733
1.4428	1.4428
1.3594	1.3594
1.6228	1.6945

//以下为预测值

1.8185	1.7206
1.9519	1.8262
2.1055	1.9687
2.2425	2.1415
2.2569	2.1079
2.2878	2.1463
2.0798	1.9264
1.7389	1.595
1.6506	1.5902
1.6115	1.5638
1.4765	1.4845

服务器数量计算结果

k1=0.125 k2=0.01 n=500

737.085, {X[1] -> 12, X[2] -> 11, X[3] -> 13, X[4] -> 15, X[5] -> 15,
 X[6] -> 15, X[7] -> 18, X[8] -> 19, X[9] -> 18, X[10] -> 16,
 X[11] -> 13, X[12] -> 12, X[13] -> 12, X[14] -> 11, X[15] -> 13,
 X[16] -> 14, X[17] -> 14, X[18] -> 15, X[19] -> 18, X[20] -> 17,
 X[21] -> 16, X[22] -> 15, X[23] -> 12, X[24] -> 12, X[25] -> 12,
 X[26] -> 10, X[27] -> 13, X[28] -> 13, X[29] -> 14, X[30] -> 15,
 X[31] -> 17, X[32] -> 17, X[33] -> 17, X[34] -> 15, X[35] -> 12,
 X[36] -> 12, X[37] -> 12, X[38] -> 11, X[39] -> 13, X[40] -> 15,
 X[41] -> 16, X[42] -> 17, X[43] -> 18, X[44] -> 19, X[45] -> 19,
 X[46] -> 17, X[47] -> 14, X[48] -> 14, X[49] -> 13, X[50] -> 12}

k1=0.125 k2=0.05 n=750

{500.018, {X[1] -> 8, X[2] -> 8, X[3] -> 9, X[4] -> 10, X[5] -> 10,
  X[6] -> 10, X[7] -> 12, X[8] -> 13, X[9] -> 12, X[10] -> 11,
  X[11] -> 9, X[12] -> 8, X[13] -> 8, X[14] -> 7, X[15] -> 9,
  X[16] -> 10, X[17] -> 10, X[18] -> 10, X[19] -> 12, X[20] -> 11,
  X[21] -> 11, X[22] -> 10, X[23] -> 8, X[24] -> 8, X[25] -> 8,
  X[26] -> 7, X[27] -> 9, X[28] -> 9, X[29] -> 10, X[30] -> 10,
  X[31] -> 11, X[32] -> 11, X[33] -> 11, X[34] -> 10, X[35] -> 8,
  X[36] -> 8, X[37] -> 8, X[38] -> 8, X[39] -> 9, X[40] -> 10,
  X[41] -> 11, X[42] -> 12, X[43] -> 12, X[44] -> 13, X[45] -> 13,
  X[46] -> 12, X[47] -> 10, X[48] -> 9, X[49] -> 9, X[50] -> 8}}

k1=0.125 k2=0.2 n=750

{506.6, {X[1] -> 8, X[2] -> 8, X[3] -> 9, X[4] -> 10, X[5] -> 10,
  X[6] -> 10, X[7] -> 12, X[8] -> 12, X[9] -> 12, X[10] -> 11,
  X[11] -> 9, X[12] -> 8, X[13] -> 8, X[14] -> 8, X[15] -> 9,
  X[16] -> 10, X[17] -> 10, X[18] -> 10, X[19] -> 12, X[20] -> 11,
  X[21] -> 11, X[22] -> 10, X[23] -> 8, X[24] -> 8, X[25] -> 8,
  X[26] -> 7, X[27] -> 9, X[28] -> 9, X[29] -> 10, X[30] -> 10,
  X[31] -> 11, X[32] -> 11, X[33] -> 11, X[34] -> 10, X[35] -> 8,
  X[36] -> 8, X[37] -> 8, X[38] -> 8, X[39] -> 9, X[40] -> 10,
  X[41] -> 11, X[42] -> 12, X[43] -> 12, X[44] -> 13, X[45] -> 13,
  X[46] -> 12, X[47] -> 10, X[48] -> 9, X[49] -> 9, X[50] -

```
> 8}}
k1=0.125 k2=0.4 n=750
{515., {X[1] -> 8, X[2] -> 8, X[3] -> 9, X[4] -> 10, X[5] ->
10,
   X[6] -> 10, X[7] -> 12, X[8] -> 12, X[9] -> 12, X[10] ->
11,
   X[11] -> 9, X[12] -> 8, X[13] -> 8, X[14] -> 8, X[15] ->
9,
   X[16] -> 10, X[17] -> 10, X[18] -> 10, X[19] -> 12, X[20]
-> 11,
   X[21] -> 11, X[22] -> 10, X[23] -> 8, X[24] -> 8, X[25] -
> 8,
   X[26] -> 7, X[27] -> 9, X[28] -> 9, X[29] -> 10, X[30] ->
10,
   X[31] -> 11, X[32] -> 11, X[33] -> 11, X[34] -> 10, X[35]
-> 8,
   X[36] -> 8, X[37] -> 8, X[38] -> 8, X[39] -> 9, X[40] ->
10,
   X[41] -> 11, X[42] -> 12, X[43] -> 12, X[44] -> 13, X[45]
-> 13,
   X[46] -> 12, X[47] -> 10, X[48] -> 9, X[49] -> 9, X[50] -
> 8}}
```

```
EDDmodel
syms x;%插队数
a=0.251083;
b=1.257538;
Lmax=18;
syms U(x) U1(x) U2(X);
U(x)=(Lmax-x)/Lmax;
U1(x)=((Lmax-x)/Lmax)^a;
U2(x)=((Lmax-x)/Lmax)^b;
hold on;
for u=1:1:Lmax%插队后位置
    sumU=0;
    sumU2=0;
    for i=u:1:Lmax
        sumU=sumU+U(i);
    end
    for i=(u+1):1:Lmax
        sumU2=sumU2+U2(i);
    end
    dU=U1(u)-sumU+sumU2;
    plot(u,dU,'*','MarkerEdgeColor','b');
    title('U 随排队位置的变化');
    xlabel('排队位置');
    ylabel('U');
end
```

蒙特卡罗代码及注释

```
N=5000;%总人数
N1=100;%服务器承载人数
for j=1:1
    for i=1:N
        for j=1:4
            player(i,j)=0;
        end
    end
    time=exprnd(80/N1,1,N);
    playin(1)=time(1);
    for i=2:N
        playin(i)=playin(i-1)+time(i);
```

```
k1=0.125 k2=1 n=750;
{539.54, {X[1] -> 8, X[2] -> 8, X[3] -> 9, X[4] -> 10, X[5] -
> 10,
   X[6] -> 10, X[7] -> 12, X[8] -> 12, X[9] -> 12, X[10] ->
11,
   X[11] -> 9, X[12] -> 8, X[13] -> 8, X[14] -> 8, X[15] ->
9,
   X[16] -> 10, X[17] -> 10, X[18] -> 10, X[19] -> 12, X[20]
-> 11,
   X[21] -> 11, X[22] -> 10, X[23] -> 8, X[24] -> 8, X[25] -
> 8,
   X[26] -> 8, X[27] -> 9, X[28] -> 9, X[29] -> 10, X[30] ->
10,
   X[31] -> 11, X[32] -> 11, X[33] -> 11, X[34] -> 10, X[35]
-> 9,
   X[36] -> 9, X[37] -> 9, X[38] -> 9, X[39] -> 9, X[40] ->
10,
   X[41] -> 11, X[42] -> 12, X[43] -> 12, X[44] -> 12, X[45]
-> 12,
   X[46] -> 12, X[47] -> 10, X[48] -> 9, X[49] -> 9, X[50] -
> 9}}
```

```
    end
    playin=floor(playin);
    %       playlong=poissrnd(4,1,N);
    %       playlong=playlong*25;
    playlong=rand(1,N);
    playlong=floor(playlong*3+1)*50;
    for i=1:N
        player(i,1)=playin(i);%玩家进入时间
        player(i,2)=playlong(i);%玩家计划停留时间,
    end
    clear playin time playlong
    for i=1:N1
        for j=1:3
            n(i,j)=0;
        end
    end
    for i=1:N
        for j=1:5
            q1(i,j)=0;
        end
    end
    for i=1:N
        for j=1:5
            q2(i,j)=0;
        end
    end
    for i=1:N
        for j=1:5
            q3(i,j)=0;
        end
    end
    for i=1:N1
        for j=1:3
            n_t(i,j)=0;
        end
    end
    for i=1:N
        for j=1:5
            q_t(i,j)=0;
        end
    end
```

```
    player_t=player;%两组采相同样本,
    for i=1:N
        for j=1:4
            long(i,j)=0;
        end
    end
end
tmax=150;%最长等待时间
k1=1;
k2=1.4142;
k3=1.7323;%服务器分区比例
k11=k1/(k1+k2+k3);
k21=k2/(k1+k2+k3);
k31=k3/(k1+k2+k3);
k1=floor(k11*N1);
k2=floor(k21*N1);
k3=floor(k31*N1);
k=k1+k2+k3;

m0=1;
m11=1;
m12=1;
m21=1;
m22=1;
m31=1;
m32=1;
l1s=1;

m1_t=1;
m2_t=1;
m0_t=1;

out=0;
out_t=0;


for t=0:2000
    for i=1:N1
        if n(i,2)<t && n(i,1)~=0
            n(i,1)=0;
            n(i,2)=0;
            n(i,3)=0;
        end
    end%将到时的玩家去除

    while player(m0,1)<=t
        if player(m0,1)==t
            if player(m0,2)==50
                q1(m11,1)=m0;
                q1(m11,2)=t;
                player(m0,3)=1;
                m11=m11+1;
            end
            if player(m0,2)==100
                q2(m21,1)=m0;
                q2(m21,2)=t;
                player(m0,3)=2;
                m21=m21+1;
            end
            if    player(m0,2)==150
                q3(m31,1)=m0;
                q3(m31,2)=t;
                player(m0,3)=3;
                m31=m31+1;
            end
        end
        m0=m0+1;
    end
    %根据玩家时间放到不同的服务器分区

    for i=m12:m11-1
        for j=1:k1
            if n(j,1)==0&&q1(i,5)==0
                n(j,1)=q1(i,1);
                n(j,2)=t+player(q1(i,1),2);
                n(j,3)=t;
                q1(i,3)=t;
                q1(i,4)=q1(i,3)-q1(i,2);
                player(q1(i,1),4)=q1(i,4);
                q1(i,5)=q1(i,5)+1;
                m12=i;
                break;
            end
        end
    end

    for i=m22:m21-1
        for j=k1+1:k1+k2
            if n(j,1)==0&&q2(i,5)==0
                n(j,1)=q2(i,1);
                n(j,2)=t+player(q2(i,1),2);
                n(j,3)=t;
                q2(i,3)=t;
                q2(i,4)=q2(i,3)-q2(i,2);
                player(q2(i,1),4)=q2(i,4);
                q2(i,5)=q2(i,5)+1;
                m22=i;
                break;
            end
        end
    end

    for i=m32:m31-1
        for j=k1+k2+1:k1+k2+k3
            if n(j,1)==0&&q3(i,5)==0
                n(j,1)=q3(i,1);
                n(j,2)=t+player(q3(i,1),2);
                n(j,3)=t;
                q3(i,3)=t;
                q3(i,4)=q3(i,3)-q3(i,2);
                player(q3(i,1),4)=q3(i,4);
                q3(i,5)=q3(i,5)+1;
                m32=i;
                break;
            end
        end
    end
    %将队列中最前排的玩家放入服务器
    %以上为实验组
    for i=1:N1
        if n_t(i,2)<t && n_t(i,1)~=0
            n_t(i,1)=0;
            n_t(i,2)=0;
            n_t(i,3)=0;
        end
    end
```

```
while player_t(m0_t,1)<=t
    if player_t(m0_t,1)==t
        q_t(m1_t,1)=m0_t;
        q_t(m1_t,2)=t;
        player_t(m0_t,3)=1;
        m1_t=m1_t+1;
    end
    m0_t=m0_t+1;
end

for i=m2_t:m1_t-1
    for j=1:N1
        if n_t(j,1)==0&&q_t(i,5)==0
            n_t(j,1)=q_t(i,1);
            n_t(j,2)=t+player_t(q_t(i,1),2);
            n_t(j,3)=t;
            q_t(i,3)=t;
            q_t(i,4)=q_t(i,3)-q_t(i,2);
            player_t(q_t(i,1),4)=q_t(i,4);
            q_t(i,5)=q_t(i,5)+1;
            m2_t=i;
            break;
        end
    end
end
%至此为对照组，结构与实验组类似

for i=m2_t:m1_t-1
    if (t-(q_t(i,2)))>tmax&&q_t(i,5)==0
        q_t(i,5)=2;
        player_t(q_t(i,1),4)=1/0;
        for j=1:N1
            if n_t(j,1)==i
                n_t(j,1)=0;
                n_t(j,2)=0;
                n_t(j,3)=0;
            end
        end
        out_t=out_t+1;
    end
end
for i=m12:m11-1
    if (t-(q1(i,2)))>tmax&&q1(i,5)==0
        q1(i,5)=2;
        player(q1(i,1),4)=1/0;
        for j=1:N1
            if n(j,1)==i
                n(j,1)=0;
                n(j,2)=0;
                n(j,3)=0;
            end
        end
        out=out+1;
    end
end
for i=m22:m21-1
    if (t-(q2(i,2)))>tmax&&q2(i,5)==0
        q2(i,5)=2;
        player(q2(i,1),4)=1/0;
        for j=1:N1
            if n(j,1)==i
                n(j,1)=0;
                n(j,2)=0;
                n(j,3)=0;
            end
        end
        out=out+1;
    end
end
for i=m32:m31-1
    if (t-(q3(i,2)))>tmax&&q3(i,5)==0
        q3(i,5)=2;
        player(q3(i,1),4)=1/0;
        for j=1:N1
            if n(j,1)==i
                n(j,1)=0;
                n(j,2)=0;
                n(j,3)=0;
            end
        end
        out=out+1;
    end
end
%将等待时间超长的玩家去除
long(t+1,1)=m11-m12-1;
long(t+1,2)=m21-m22-1;
long(t+1,3)=m31-m32-1;
long(t+1,4)=m1_t-m2_t-1;
%队列长度
end
%   plot(q1(1:2000,4));
%   hold all;
%   plot(q2(1:2000,4));
%   hold all;
%   plot(q3(1:2000,4));
%   hold all;
%   plot(q_t(1:2000,4))
ans(1,1)=sum(q1(1:N,4));
ans(1,2)=sum(q2(1:N,4));
ans(1,3)=sum(q3(1:N,4));
ans(2,1)=ans(1,1)+ans(1,2)+ans(1,3);%实验组总时长
ans(2,2)=sum(q_t(1:N,4));%对照组总时长
ans(2,3)=ans(2,1)/ans(2,2);
ans(3,1)=(m12+m22);
ans(3,2)=m2_t;
ans(3,3)=(m12+m22)/m2_t;

ans(4,1)=out;
ans(4,2)=out_t;
ans(4,3)=out/out_t;
```

多次实验两组数据对比

141011	230979	0.610492729
151774	242554	0.625732827
117950	207655	0.568009439
141572	224256	0.631296376
135471	223738	0.605489456
141232	214713	0.657771071
144436	235602	0.613050823
147019	242192	0.607034914
139034	223988	0.620720753
141595	236332	0.599135961
142615	227976	0.625570235
149366	237878	0.627910105
141599	230049	0.615516694

141909	220989	0.642154134
148359	245381	0.604606714
144844	231369	0.626030281
145321	227564	0.638593978
143586	244660	0.586879751
126263	220088	0.57369325
149199	238497	0.625580196
146166	230502	0.634120311
145109	235919	0.615079752
137599	241647	0.569421512
140161	225316	0.622064123
145505	237615	0.612356122
141146	225497	0.625932939
141161	226741	0.622564953
147606	226700	0.65110719
146965	229244	0.641085481
139286	239319	0.582009786
143300	229106	0.625474671
140527	216301	0.649682618
146532	241554	0.606622122
149335	240295	0.621465282
142275	228341	0.623081269
139903	199852	0.700033024
147186	244481	0.602034514
128642	227960	0.564318301
146591	232529	0.630420292
143051	226142	0.63257157
145104	230551	0.629379183
127747	226999	0.562764594
148004	243710	0.607295556
142742	213325	0.669129263
147415	235939	0.624801326
145114	225476	0.643589562
150396	235534	0.638532017
151283	238295	0.634855956
142723	229765	0.621169456
124499	221664	0.561656381
140305	239810	0.585067345
141811	236949	0.598487438
145571	225750	0.64483278
143879	233421	0.616392698
140020	237916	0.588527043
146166	233133	0.626964008
152408	233011	0.654080709
130217	213613	0.609593049
148209	228501	0.648614229
151865	242974	0.625025723
139940	236937	0.590621136
154611	253320	0.610338702
143289	231370	0.619306738
143636	231534	0.620366771
148182	235318	0.629709584
145855	236526	0.616655251
148249	244691	0.605862087
151384	240173	0.630312317
151063	239913	0.629657417
137140	225526	0.608089533
145445	231422	0.628483895
150738	234634	0.642438862
148453	233812	0.63492464
148748	249955	0.595099118
139180	243531	0.57150835
148179	236042	0.627765398
142937	217264	0.657895464
147294	221723	0.664315385
144591	236822	0.610547162
145064	232571	0.623740707
140229	224660	0.62418321
151099	242291	0.623626136
142222	216409	0.657190782
148935	241472	0.616779585
128877	206962	0.622708517
148920	243658	0.611184529
144616	218137	0.662959516
142373	231543	0.614887947
149205	235154	0.634499094
140307	229746	0.610704865
147111	229992	0.639635292
132948	227374	0.584710653
143836	239595	0.600329723
154511	244773	0.631242008
154549	245423	0.629725005
142591	231949	0.614751519
146741	235635	0.622747045
143320	234882	0.610178728
144906	235455	0.6154297
134644	222143	0.60611408