Parallel Programming Final Project
Parallel Matchmaking

David May, Thomas Clarke, Kousuke Tachida
(Dated: April 2019)

I. ABSTRACT

Matchmaking is an aspect of gaming that can make or break a player's experience. In this paper, we analyze various matchmaking algorithms and simulate tens of thousands of players trying to participate in fair matches amongst themselves. Specifically, we implement the Elo and TrueSkill matchmaking rating systems along with head-to-head matchmaker. This simulation utilizes MPI (the Message Passing Interface) for parallelism to allow for quick testing on Blue Gene/Q of thousands of players over thousands of time steps.

Players are modeled with a hidden skill level and face off against each other in thousands of games over 10,000 time steps. Based on their hidden skill level, they win or lose these games. Utilizing the Elo or TrueSkill algorithms, their MMR is updated after each game, allowing us to do statistical analysis on the validity of these implementations.

A strong scaling analysis of this simulation shows that the runtime initially decreases with an increase in MPI ranks, then increases. We show that this is due to the overhead in matchmaking itself and that a ratio of 128 Players per Rank is optimal.

Based on the results of the statistical analysis showing that our implementation stays true to these MMR systems and a best runtime of 60 seconds for a 32 thousand player input, we conclude that a parallel approach for simulating matchmaking is viable. We compute a capability of 0.02 seconds per player with the optimal player to rank ratio of 128.

The quick runtime due to the parallel approach allows that implementations similar to this could be used for future research on matchmaking algorithms including team versus team, other MMR systems, or other attribute focused matchmakers. With such speed, quick iteration of algorithms and realistic player interaction is possible.

II. INTRODUCTION

Matchmaking is an important problem in multiple fields, including advertising, video gaming, dating, and employment. Matchmaking is applicable to all of these fields and more, so there are many different approaches to pairing things together. Gaming is a common interest among the members of our group, so we decided to place our focus on the gaming aspect of matchmaking. Many games have studied matchmaking to make the best possible experience.

These games take in a wide range of considerations in their algorithms, such as ping, skill, premade teams, variation, and settings. The best algorithm is one that maximizes enjoyment and competition for all players. Limitations exist in many algorithms, as ranking systems are usually uni-dimensional (give a ranking of a single number) in games that are very complex in what can make a player excel [7]. There are also common misconceptions about what can make a match "good." While unbalanced games are usually not enjoyable, this doesn't necessarily mean a balanced match would be. For example, in an FPS, there may be a match of evenly skilled teams. However, if both of these teams are filled with gamers who camp, this will make for an uneventful match [7].

It is not an easy task to make an accurate ranking system given all of the factors involved. An analysis of "Heroes of Newerth" showed that while player skill is captured generally well, there are still many ways the exploit the system where users can climb the ranking ladder above their actual skill level [3]. Another issue for the game's matchmaking algorithm is that it is generally slow in placing players in their rightful skill groups. This is a troubling sign because for the players, speed is a very important factor in their game-playing experience.

When players seek out matches, speed is a high priority. Taking a parallel approach in our analysis allows us to analyze each matchmaking algorithm quickly and effectively. In our project, we analyze two different algorithms for matchmaking to simulate on the Blue Gene/Q: Elo and TrueSkill. The Elo rating system was designed originally for chess, and is still used today in that regard. The TrueSkill system was created by Microsoft for games such as Halo. By analyzing both we will determine the efficiency of our parallel algorithm to enable further study by quickly simulating matchmaking.

III. MMR SYSTEMS

Our parallel algorithm is a discrete event simulation of many players undergoing matchmaking. Players have several attributes: Current MMR (matchmaking rating), latency (connection to game server), true MMR or actual skill (hidden), lenience, and in the case of TrueSkill MMR uncertainty. If players are not in a game during the current time step, the program will attempt to match them based on these attributes. The matched players will play and have their MMR updated based on the result of the game, which is determined by how well the players played. In our simulation, this will be represented by the players' true MMR. At the end of the simulation, we

will compare how close each player is to their true MMR and how well the players as a whole represent a normal distribution.

The two MMR systems we are utilizing in this simulation are the Elo system and TrueSkill. The Elo system was developed by Professor Arpad Elo for use in chess. Each player has a rating (MMR) that represents their skill level. A player with a higher rating has the edge in a given match. In fact, a player with 100 more points is expected to win 64% of the time and a player with 200 more points is expected to win 76% of the time. Using this property, a player that is favored to win does not get many rating points for winning, but in the case of an upset, they will lose much more points.

For players A and B with ratings R_A and R_B respectively, define the following values:

$$Q_A = 10^{R_A/400} \qquad Q_B = 10^{R_B/400}$$
$$E_A = \frac{Q_A}{Q_A + Q_B} \qquad E_B = \frac{Q_B}{Q_A + Q_B}$$

Then, based on whether A won or B won, S_A and S_B are ± 1, where a 1 indicates a win. The Elo system uses all of these to determine each players' updated rating, R'_A and R'_B.[10]

$$R'_A = R_A + 32(S_A - E_A) \qquad R'_B = R_B + 32(S_B - E_B)$$

This is repeated for every match the players play, hoping to closely approximate their true skill. Several issues arise with the Elo system, including the fact that it is static over time. Past games have the same relevance as current games, implying that every game has the same relevance on a players' current skill, which may not be true [8]. Another issue with the Elo system is that players can play whoever they want to, meaning they could target higher rated players with low skill and artificially boost their own Elo [6]. These problems have been improved on by several successors, including TrueSkill [13], Glicko, or some Bayesian [6] systems.

The TrueSkill system, developed at Microsoft Research, is a ranking system primarily used for Xbox Live. What sets the TrueSkill system apart from other ranking systems is that it skill ratings are based on two numbers as opposed to just a single number rating. One of the numbers, called μ, is the player's average skill rating. The other number is the degree of uncertainty in the players skill, called σ [13]. The function of keeping track of uncertainty is that if a player has a high uncertainty, the rating system can make big changes to the player's rating until a string of consistent games are played and the system becomes more confident in that player's skill rating. If two players are matched and play a game, where player A wins and has μ_A and σ_A and player B loses and has μ_B and σ_B, the way the system would update there rating is [9] [13]:

$$c^2 = 2\beta^2 + \sigma_A^2 + \sigma_B^2 \quad (1)$$
$$\mu'_A = \mu_A + \frac{\sigma_A^2}{c} * \Phi(\frac{\mu_A - \mu_B}{c}) \quad (2)$$
$$\mu'_B = \mu_B - \frac{\sigma_B^2}{c} * \Phi(\frac{\mu_A - \mu_B}{c}) \quad (3)$$
$$\sigma'^2_A = \sigma_A[1 - \frac{\sigma_A^2}{c^2} * \Psi(\frac{\mu_A - \mu_B}{c})] \quad (4)$$
$$\sigma'^2_B = \sigma_B[1 - \frac{\sigma_B^2}{c^2} * \Psi(\frac{\mu_A - \mu_B}{c})] \quad (5)$$

Here, there are several unknowns: $\Phi(x), \Psi(x), \beta$. β is a constant related to uncertainty. $\Phi(x)$ and $\Psi(x)$ are functions that allow for the following: The player with a higher uncertainty gets a bigger uncertainty delta and a bigger skill delta [13]. These functions normalize the difference in μ to account for likelihood of winning similar to Elo. We estimate these functions as

$$\Phi(x) = MAX(1.0 - x, 0.1)$$

and

$$\Psi(x) = 1 - \frac{1}{e^{-2x+2} - 1}$$

for ease of computation [14].

With this system, eventually players will reach their "true" skill, because their uncertainty will approach 0. As you can see from equations 4 & 5, uncertainty is always decreasing. This forgoes the issue of non-permanence from Elo, but TrueSkill is still not perfect. Critics argue that TrueSkill does not take into account the match length, players' contributions during a match, and eventually uncertainty goes down to zero, meaning those games accomplish nothing. They argue that uncertainty should have a minimum value above 0 [12].

Comparatively, the Elo and TrueSkill systems are not perfect, but are heavily studied and work in a bubble [12]. Therefore, we can use them to model our player interactions and changes in this simulation and know approximately how they will behave.

IV. THE ALGORITHM

The algorithm we implement to simulate matchmaking is a player-based model. Players are modeled with several attributes: MMR, true MMR, latency, uncertainty, wait time, and lenience. These players are stored in a player array. To run in parallel, our algorithm utilizes MPI. Each MPI rank has its own player array, containing a portion of the total players. The simulation runs for a certain number of time steps.

Initially, all players begin with an average MMR, based on the average ratings in the Elo and TrueSkill systems. All players are randomly assigned a true MMR and a latency. Wait time is initially -1 and lenience is initially 1.

The importance of these values arises later in the algorithm. We decided on including latency in this algorithm because of its real-world importance in matchmaking. A study of mobile gamers determined that the most important factor in a match is low latency [11], and another study determined that once a player is connected, their connection will most likely remain constant during the course of their connection [1].

Our algorithm models latency as a Gaussian distribution centered around 50ms. This is due to a study on League of Legends players that found that the average ping is 50ms and follows a single peak Gaussian distribution shape [16].

Our matchmaking algorithm attempts to match players with similar MMR and a low total ping. To do this we use the lenience model of matchmaking (or queue model). Every time step, players that are not in a game (players with negative wait time) try to match with other players that are not in a game. Our determination for this is:

$$\begin{aligned}(lenience_A + lenience_B)/2 \\ > M * |MMR_A - MMR_B| \\ + P * (latency_A + latency_B)\end{aligned}$$

This equation combines lenience, MMR, and latency with two coefficients, M and P. The first term on the right hand side lowers as the MMR differential lowers and the second term lowers as the latency total lowers. With good definition of M and P, we can fine tune the effect of MMR difference and total ping. Lenience, then, is basically the acceptable total difference in MMR and acceptable total ping. To ensure players get a match, lenience will increase when players remain unmatched.

To get a proper definition of M and P, we want to consider what MMR differences and what latency totals are acceptable. With an average lenience of 1, the players are matched immediately. For equally ranked players, this is when $P * (latency_A + latency_B) < 1$. Based on our research, an unacceptable latency is ~ 200 [1], so P should be $\frac{1}{200}$.

For players with average ping, 50ms and a 1 lenience, we can determine an acceptable M value based on the equation $1 > M * |MMR_A - MMR_B| + \frac{1}{2}$. Players should be matched immediately if their MMRs differ by less than 25 for the Elo system or 50, so this gives an M of $\frac{1}{50}$ for Elo and $\frac{1}{100}$ for TrueSkill. These values could vary based on the strictness of a particular matchmaking algorithm and could be improved in later studies.

This tells us if a player will be matched with another player, and if they are not matched, their lenience increases. In our algorithm, we simply try to match a player with every other "nearby" player by going over the list of players and trying to match them with unmatched players.

To give us a parallel approach to this algorithm, we use the concepts of rankings, which are in many games such as League of Legends, Overwatch, and the aforementioned Heroes of Newerth [16] [3]. Each MPI rank contains a portion of the total players, split up by their current MMR. The first rank will have the lowest grouping of players with the worst MMRs and similarly, the last rank will have the highest grouping of players.

To ensure that players can leave their rankings, during each time step we do the following: Sort, Exchange, and then Match. We sort all the players in the rank based on their current MMR (using quicksort), then exchange the lowest in the rank with the previous rank and the highest in the rank with the next rank. This allows players to eventually reach their intended rank and keeps the runtime fast due to MPI's scalability.

Compared to having no ranks, each MPI rank will only need to look for matches within itself, decreasing the search space from 1 million potential matches to a fraction of that size. With an $O(n^2)$ or $O(n^3)$ algorithm this is a drastic decrease in runtime.

V. STRONG SCALING ANALYSIS

For testing, we ran all of our strong scaling tests on a pool of 32768 players. See Figure 1 for the strong scaling of the simulation. From the graph, we can conclude that the ELO and TrueSkill systems have very similar runtimes, differing by seconds. This is promising, because it implies that any implementation of any matchmaking rating system will have a similar runtime.

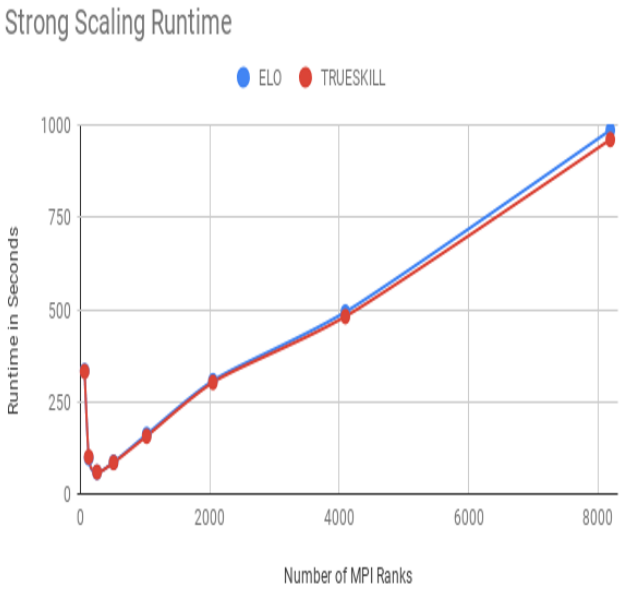

FIG. 1. MPI Ranks vs. Time

The next clear indication from Figure 1 is that runtime decreases and then increases when increasing node count for this problem size. This implies two things: first, that there is an optimal ratio of problem size to

MPI ranks; second, that there is significant overhead for a large amount of ranks.

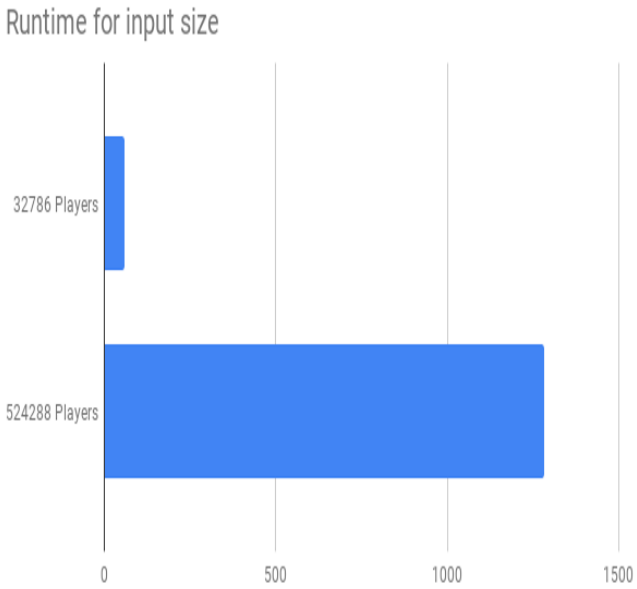

FIG. 2. Runtime vs. Input Size

The first is interesting because this means for a given input size we can calculate the optimal number of ranks. Here, we had 32768 players and the number of ranks was 256, meaning an optimal number of ranks per player is $\frac{32768}{256} = 128$. We can test this by running a larger input size of 524288 players with 128 nodes (8192 ranks $\frac{524288}{8192} = 128$) and confirming that it is fast up to a factor of input size. See Figure 2 and Figure 3 for this result, where Figure 3 shows that runtimes per player are quite close.

The second implication of a significant overhead for more ranks can be justified by the fact that every player makes an $O(n)$ search every tick when they are not in a game. Here, n is the number of players per rank. This is initially large with a low amount of ranks because n is large, but becomes small as matches are found. When a player finds a match, they are removed from the search pool for the duration of the match, 10 ticks, so finding matches improves the runtime of the algorithm. Conversely, when n is small with 4 players per rank, it is hard for these players to find an adequate match and thus runtime increases.

Over all, we can conclude that this simulation is not embarrassingly parallel, but still benefits from parallelism.

VI. CORRECTNESS

To analyze that our simulation was a success, we must not only look at the parallel runtime, but the correctness

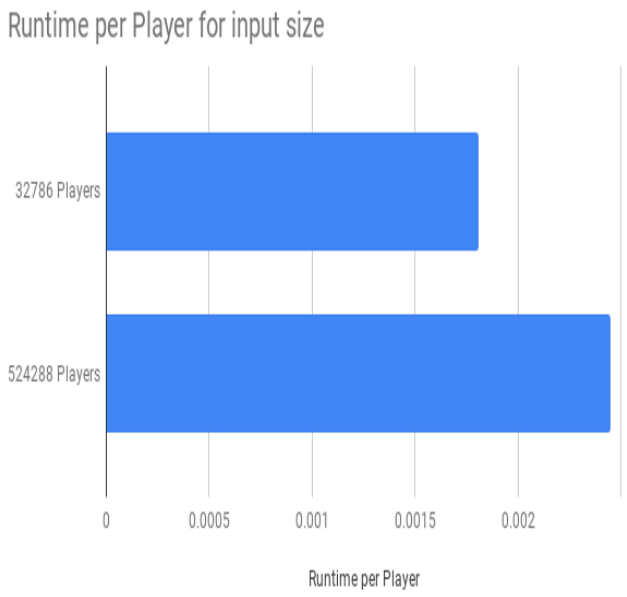

FIG. 3. Runtime per Player vs. Input Size

of the algorithms.

We analyze the distribution of players over time, the average wait time for matches over time, the average win-rate over time, and average true MMR over time.

For the Elo systems, we can see that in the first few time steps, the players distribute themselves into a Gaussian in Figure 4.

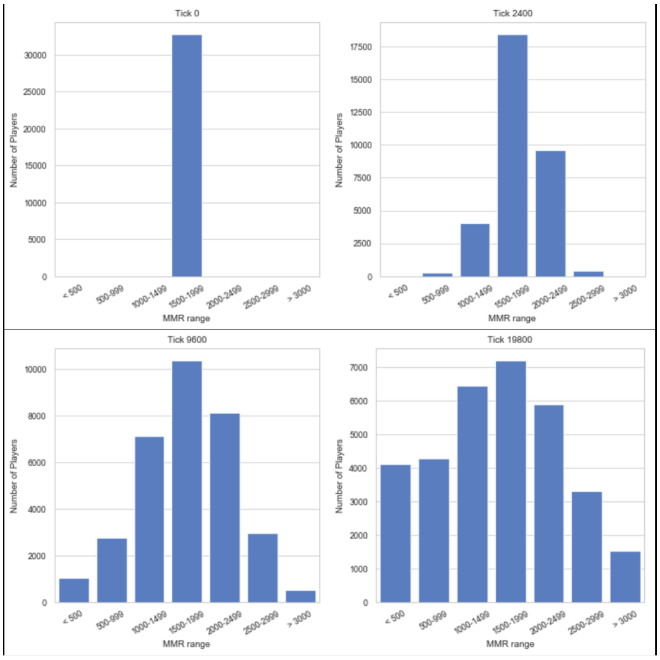

FIG. 4. Average Player MMR distribution in the first few time steps for the ELO system

Looking at the long-time statistics, we can see that the number of players in each grouping of Elo ends up in a grouping proportional to the Gaussian. Initially, the majority of people are in the singular MMR range, then they spread out. See Figure 5 for this.

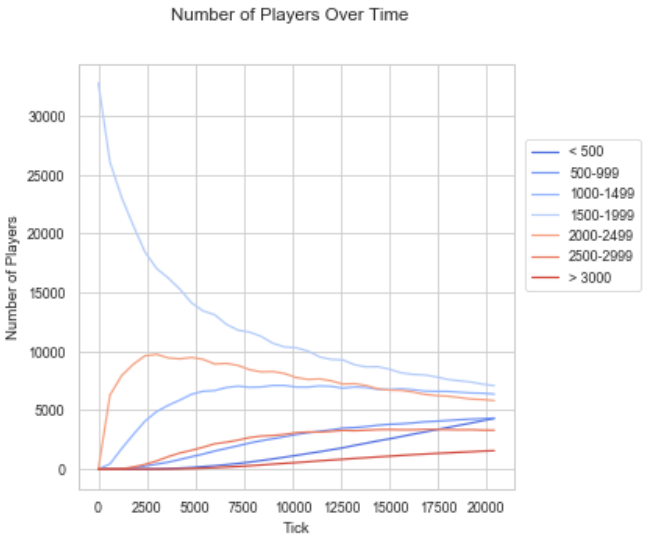

FIG. 5. Tick Number vs. Number of players for each Elo grouping

We also see that as time goes on, people wait less and less time for their matches. See Figure 6 for this.

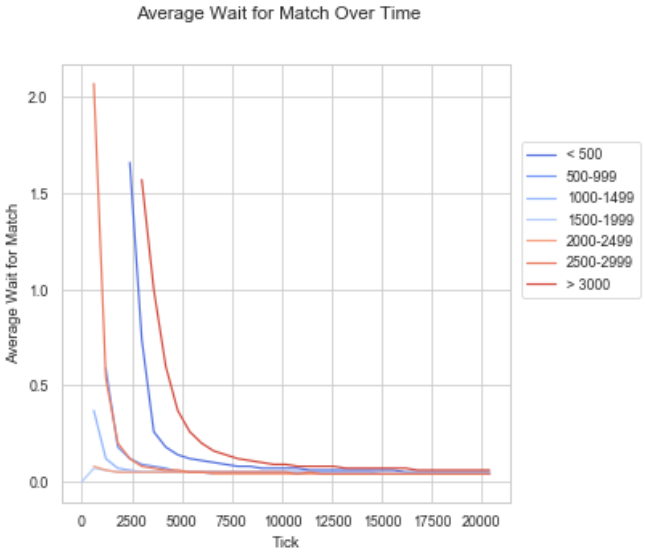

FIG. 6. Tick Number vs. Average wait time for each Elo grouping

We see that win-rate goes towards 50% in Figure 7, implying that matches become balanced.

We see that players' MMR comes close to their true MMR in Figure 8.

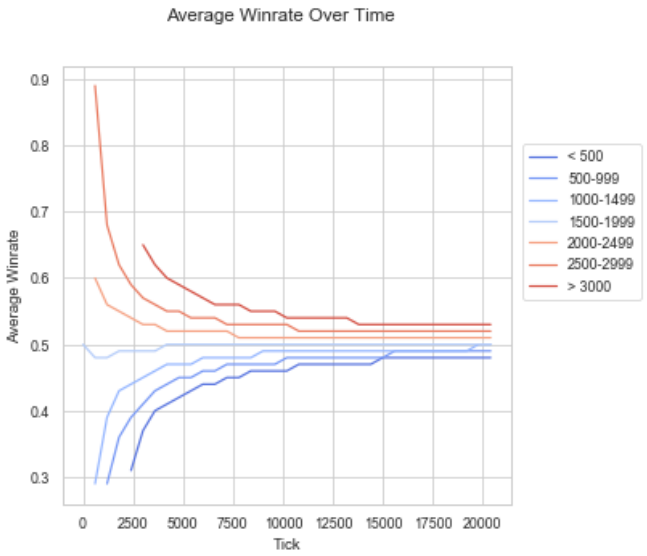

FIG. 7. Tick Number vs. Average win-rate for each Elo grouping

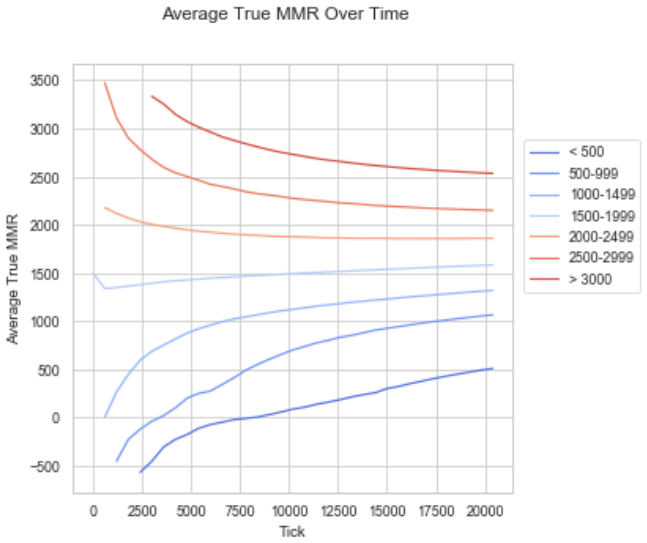

FIG. 8. Tick Number vs. Average True MMR of each Elo grouping

In each of the above Figures, we see that as time goes on, each statistic converges towards what it should be: the number of players approach a Gaussian distrubtion, wait time approaches 0, winrate approaches 50%, and average true MMR approaches the MMR grouping.

We can see similar trends for the TrueSkill system. Initially, the average player MMR distribution in the first time steps also expands to a Gaussian in Figure 9, we can explain the bi-modal distribution in the first two frames by the uncertainty factor in TrueSkill.

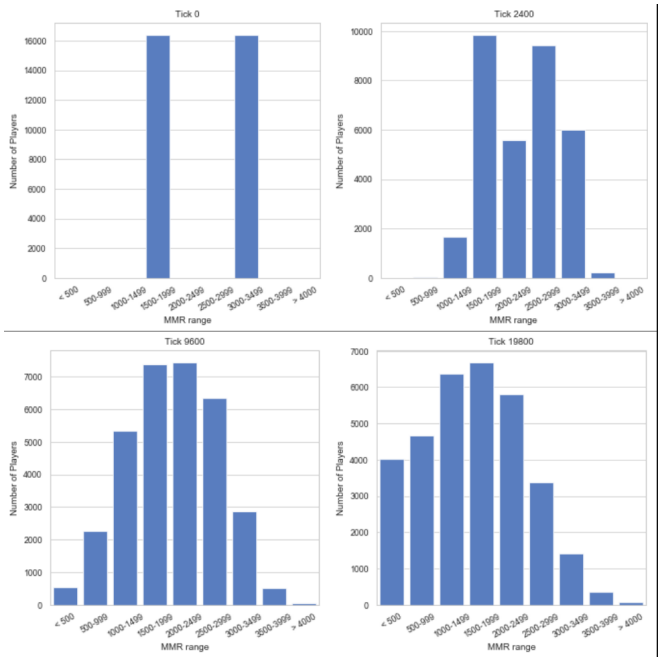

FIG. 9. Average Player MMR distribution in the first few time steps for the TrueSkill System

Similarly to Elo, we can see that the number of players over time converges cleanly to a Gaussian. This implies that TrueSkill is a little more stable than the Elo system. See Figure 10 for this.

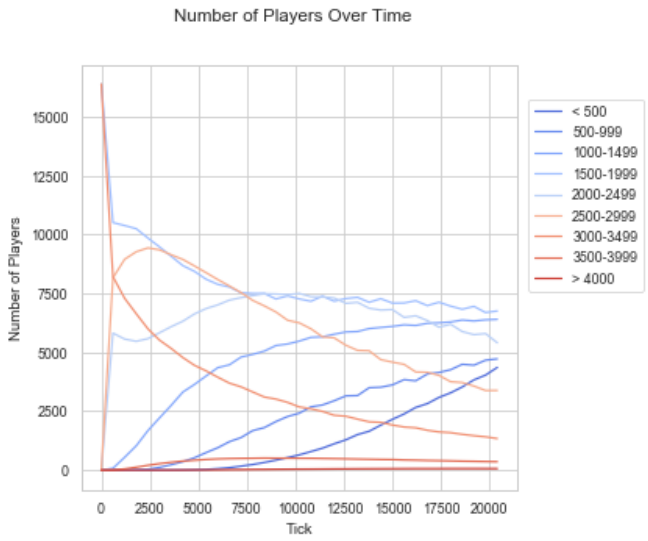

FIG. 10. Tick Number vs. Number of players for each TrueSkill grouping

We see that wait time for TrueSkill approaches 0 for the average groupings in Figure 11. The only exception to this is the > 4000 grouping, which has very few people and as a result they will find less matches.

We see that average win-rate approaches 50% as expected for TrueSkill groupings as well in Figure 12.

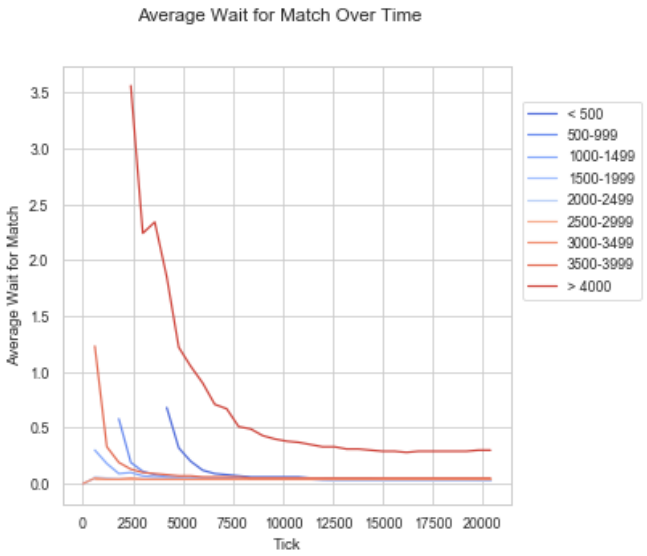

FIG. 11. Tick Number vs. Average wait time for each TrueSkill grouping

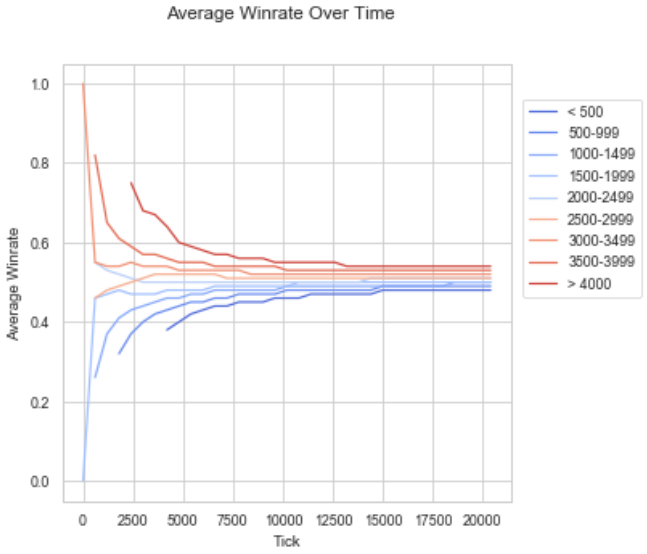

FIG. 12. Tick Number vs. Average win-rate for each TrueSkill grouping

For TrueSkill groupings, we also see that the MMR groupings have convergent true MMRs as well, implying that the algorithm is correct. This can be seen in Figure 13.

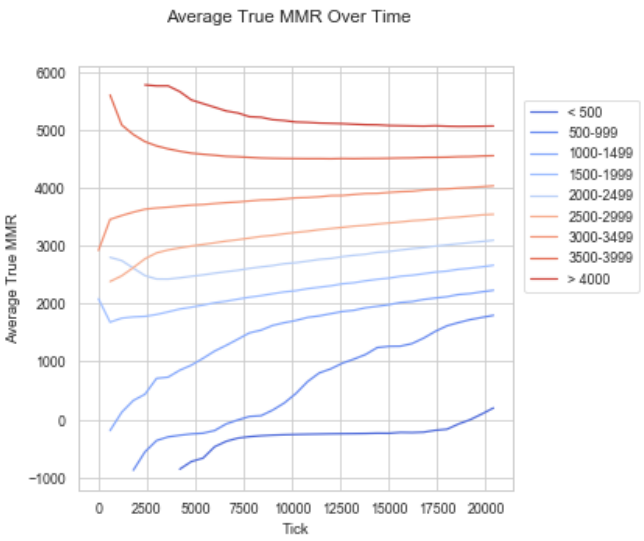

FIG. 13. Tick Number vs. Average True MMR of each TrueSkill grouping

VII. CONCLUSION AND FURTHER IDEAS

This simulation, based on the results expressed, is accurate for the Elo and TrueSkill systems with the given algorithm. The primary goal of this simulation, though, is to simply replicate a many-player matchmaker quickly in parallel on the Blue Gene/Q system and potentiall other similar supercomputers. Therefore, if we were to further explore this idea we could use the same general structure but change a variety of variables. This includes implementing a variety of different ranking systems, including TrueSkill 2 [12], EOMM [5], and Glicko.

Because our run-time was approximately one minute running at 256 ranks (the fastest run-time computed), implementing and iterating these various models should run at similar times. Development of these new models will be very efficient, allowing for easy analysis to compare different algorithms and a better future for matchmaking models. While implementing and testing different models is an effective way to extend our code further, there also exists opportunity to expand the simulation in terms of team size.

Due to the limits of the project, we were only able to incorporate head-to-head match-ups among single players. If this were to be expanded, the simulation could incorporate team matches. One could implement different constraints on teams, meaning team members must be within a certain distance in their skill rankings or allow for any combination of rankings. Incorporating teams will make room for more team-matchmaking specific algorithms to be implemented and tested in our simulation. These algorithms usually calculate a weighted team skill level based on some set of criteria to match up against different teams [15]. The weighted skill rating calculated could be an average of a teams skill rating, the best players skill rating, the average of the best and worst players rating, etc.

Another potential improvement to this simulation would be to use ROSS, or Rensselaer's Optimistic Simulation System. This would allow our simulation to run on an optimized framework for discrete simulations [4] [2], which is exactly what we implemented here. The constraints of our time in this course did not allow us to fully explore ROSS and that is why we did not use it here.

Our analysis shows that our implementation accurately models these players, and previous research doing similar matchmaking tells us that a simulation of matchmaking is accurate in later real-world analysis [17]. We hope that this project can lay the groundwork for future simulations of varying team-sizes and algorithms due to its parallel structure and quick run-time. This will aid in the development and optimization of matchmaking systems for gaming.

[1] Agarwal, S. and Lorch, J. R., in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09 (ACM, New York, NY, USA, 2009) pp. 315–326.

[2] Barnes, Jr., P. D., Carothers, C. D., Jefferson, D. R., and LaPre, J. M., in *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM PADS '13 (ACM, New York, NY, USA, 2013) pp. 327–336.

[3] Caplar, N., Suznjevic, M., and Matijasevic, M., "Analysis of player's in-game performance vs rating: Case study of heroes of newerth," (2013), arXiv:1305.5189.

[4] Carothers, C. D., Bauer, D., and Pearce, S., in *Proceedings Fourteenth Workshop on Parallel and Distributed Simulation* (2000) pp. 53–60.

[5] Chen, Z., Xue, S., Kolen, J., Aghdaie, N., Zaman, K. A., Sun, Y., and El-Nasr, M. S., "Eomm: An engagement optimized matchmaking framework," (2017), arXiv:1702.06820.

[6] Coulom, R., in *M.H.M. Computer and Games* (Beijing, China, 2008) pp. 113–124.

[7] Delalleau, O., Contal, E., Thibodeau-Laufer, E., Ferrari, R. C., Bengio, Y., and Zhang, F., IEEE Transactions on Computational Intelligence and AI in Games **4**, 167 (2012).

[8] Glickman, M. and Jones, A., "Rating the chess rating system,".

[9] Graepel, T. and Herbrich, R., "How to rate players skills for fun and competitive gaming," (2006).

[10] HowlingPress,, "Elo rating system,".

[11] Manweiler, J., Agarwal, S., Zhang, M., Roy Choudhury, R., and Bahl, P., in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11 (ACM, New York, NY, USA, 2011) pp. 71–84.

[12] Minka, T. and Cleven, R., "Trueskill 2: An improved bayesian skill rating system," (2018).

[13] Minka, T. and Zaykov, Y., "Trueskill ranking system," (2005).

[14] Moser, J., "The math behind trueskill," (2010).

[15] Shi, L., "Game clan matchmaking," (2008).

[16] Véron, M., Marin, O., and Monnet, S., in *Proceedings of the 24th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV 2014* (2014).

[17] Zhang, N., Lee, Y., and Balan, R. K., in *Proceedings of the 2Nd Workshop on Mobile Gaming*, MobiGames '15 (ACM, New York, NY, USA, 2015) pp. 31–36.