E-Book

Build Better Experiences and a Stronger Community with Smart Matchmaking

Part 2 of a 4 Part Gaming Series

© 2021 Redis

Introduction

A strong community is essential for the longevity of any multiplayer game. When the community grows, players are engaged, monthly active users go up, and the revenue increases. But if the community is actively shrinking, then game creators need to change something fast before the game declines into obscurity—or worse, has the plug pulled.

Matchmaking is an essential part of building a strong, active community. It doesn't matter how good the gameplay and game design is. If you spend all your time staring at search screens while trying to find a game, waiting on matchmaking algorithms, or playing against wrong players, you never get to really experience the game.

Whether it's building teams, putting together battle royales, or bringing players together in social games, matchmaking services need to run their algorithms quickly and accurately. In fact, it should be so quick and accurate that it's a seamless experience for the players. This combination is what makes smart matchmaking stand out. Not only is it fast, but it makes matches based on fresh data that perfectly fit the player.

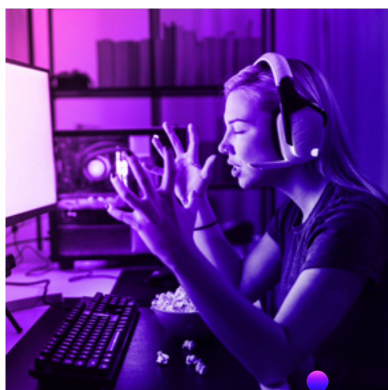
Delivering smart matchmaking is easier said than done, but with the right database, your algorithms can run at real-time speeds to give your players a low-latency, flawless matchmaking experience that keeps them coming back to play again and again.

What does real-time mean?

Research into human response indicates applications have roughly 100 milliseconds (ms)—one third of the time it takes to blink—before users feel like they're waiting for a response. To be considered real-time, a leaderboard needs to send a request, have it processed, receive the response, and present it to the player in less than 100ms.

© 2021 Redis

Players expect real-time response—and accept nothing less



Unfortunately, players don't really notice when matchmaking works perfectly. It's rare for a 5-star game review to go on about how great the matchmaking algorithm is. But if something goes wrong, you're pretty much guaranteed to hear about it. And see its results in your key game metrics like DAUs and MAUs. After all, there's only so long players will wait in the lobby for the matchmaking service to run or stare at the spinning wheel in a search bar before they jump to another game. And speed by itself isn't enough. Even if the matchmaking runs quickly, it has to be accurate. If it matches them with opponents or teammates that are too strong (or too weak), then the game isn't fun. Once again, they'll jump, and probably leave damaging reviews in the process.

In games with a PvP function, matchmaking can make or break the player experience. Whether it's team wars, battle royales, races, 1:1 duels, or social games, PvP needs to find players who are at similar power rankings and currently available to play (or similar activity levels if the PvP is asynchronous). Then they need to get rapidly sorted and slotted into matches based on these parameters. A good PvP experience quickly matches players against others who offer a challenge, but can be bested in a contest of skills or strategy. A poor experience takes a long time to load, then matches them against opponents who are too weak (removing the challenge) or too strong (removing the ability to win).

Matchmaking is also used for team recruitment. It could be called a clan, squad, house, country, or a hundred other names, it's all the same core idea: Players form up into teams of similar power and ability levels. Sometimes they're competing on leaderboards, sometimes against the computer, and sometimes against each other. In every case, the player is usually given a range of teams to choose from. They expect the team to include people with similar play styles, levels, time zones, and more. And they expect to be able to make a good team selection quickly. After all, they signed up to play, not spend all their time selecting a team.

Player selection also adds additional frontend expectations to matchmaking. Players don't want to just be shuttled into whatever game is handy. They want to be able to search for just the right match.

© 2021 Redis

Adding search to matchmaking is a great way to increase engagement, but it also introduces another place where slow or faulty matchmaking could detract from the player experience. When they search for opponents (or allies), players want to instantly see an array of good matches to choose from.

No matter what kind of game, all matchmaking use cases have the same player expectation of seamless execution. Players don't mind waiting for other people (such as waiting in the lobby area for battle royale games while enough players join), but they don't want to wait for the game itself to load. In fact, players expect the matchmaking to be so seamless that it's almost invisible.

This is where real-time matchmaking comes in. Real-time matchmaking finds the right connections between the data, quickly gives players an array of choices, backfills the players to the right server spots based on their choice, and runs through the matchmaking queue so fast that players don't even notice.

Similar is a range, not an absolute

Matchmaking algorithms can look at more than just power levels. Players that always lose can get discouraged (and players that always win can get bored), so keeping players engaged means finding a balance of challenge. This means looking at variables beyond power levels, such as the number of recent wins/losses. While players may not be aware of these additional variables, it does increase the amount of computations that have to fit within their low-latency expectations.

© 2021 Redis

Modern matchmaking is more complex than ever

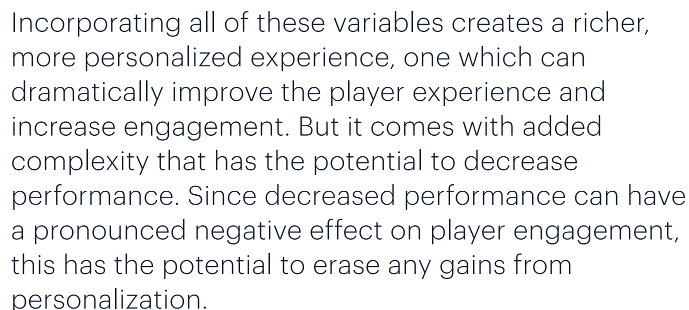
Early online matchmaking was much simpler. Are people online? Great, put them on the same server, spit out a list of all available games for them to wade through and let them sort it out themselves. But this approach left way too much of the player experience up to chance—with potentially disastrous consequences. Modern matchmaking is sophisticated and complex, going that extra mile to make sure that as many players as possible have engaging experiences that make the game enjoyable.

To pull off that degree of personalized sophistication, matchmaking services have to process a massive amount of data. Player compatibility can be determined by variables like player rank, current inventory, location, recent win/loss ratio, and more. For potentially millions of players. It's a lot of data to run through the queue, sort, and then backfill into servers.

It gets further complicated when you add player choice into the mix. Take an online racing game. Players choose a racer, car, then track. These choices will have to also be included in the matchmaking, and processed fast enough to not slow down the players' experience. After all, how frustrating would it be if you choose your favorite racer, the perfect car, and the exact track you want, then lose your spot because matchmaking took too long to process your request? Similar player-led variables pop up in a wide variety of use cases, such as game modes for first-person shooters (like Capture the Flag or Free for All), player character types (for games that only allow 1 of each type), a selection of game maps, or tourneys where victory is a condition for advancement.

Showing players those choices presents its own technological demands. When a player goes to select a game, they're performing a specialized kind of search, even if it doesn't look like it on their end. But it's not enough for the search to just return an unsorted list of results. The results need to weigh all the factors above, then be sorted based on the best possibility of matches. For example, a player searching for a trivia game to join will need to be served options that best match their social graph, then sorted by things like number of available spots or time until game begins.

© 2021 Redis



Incorporating all of these variables creates a richer, more personalized experience, one which can dramatically improve the player experience and increase engagement. But it comes with added complexity that has the potential to decrease performance. Since decreased performance can have a pronounced negative effect on player engagement, this has the potential to erase any gains from personalization.

Fortunately, there are many design and architecture elements to optimize a matchmaking service for this increased complexity. For example, storing data using a graph data model could make it simpler and faster to find relationships between records than it would be with a relational database. Streamlined search functionalities also help increase responsiveness. Instead of having to run results through a separate search database, integrated capabilities directly interact with the same core database.

So yes, while the expectations from players demand faster and more complex matchmaking services than ever before, it's also possible to build services that complete the request within real-time speeds. But doing so creates specific technical requirements for the database and architecture behind the matchmaking service.

© 2021 Redis

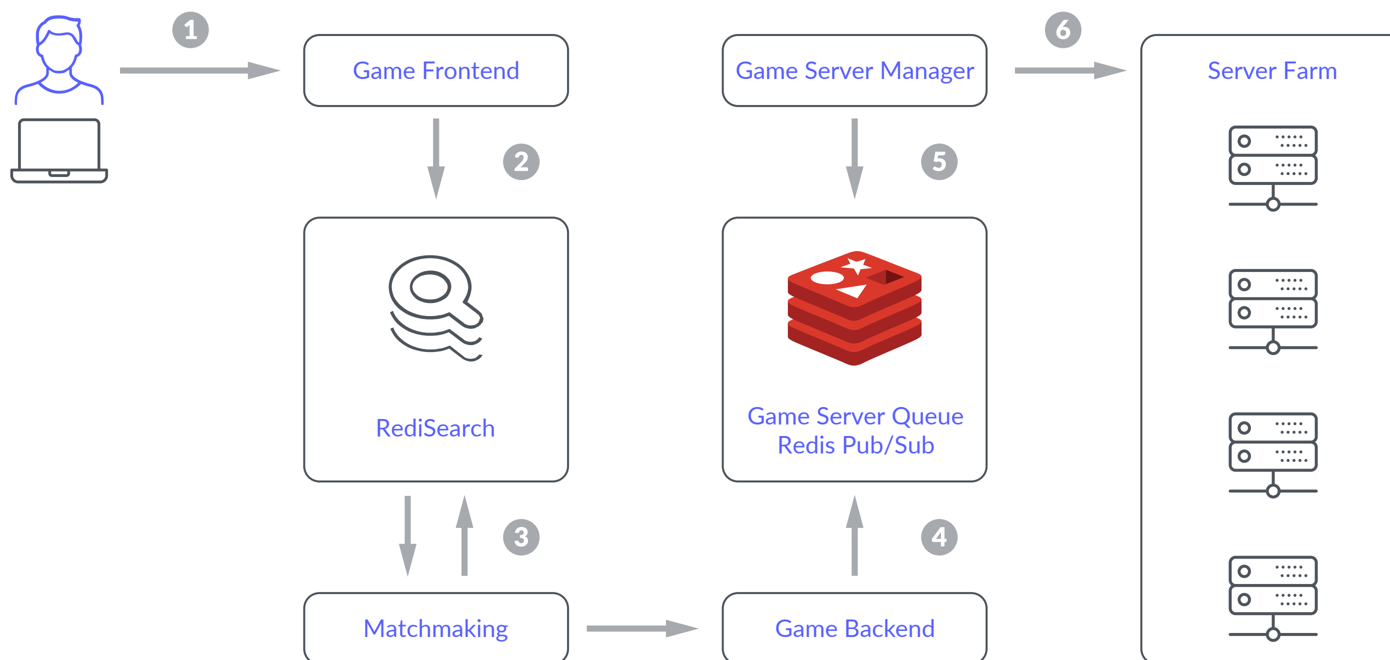
Technical requirements for a smart matchmaking database

Above, we defined real-time response as less than 100ms for receiving, processing, and returning the request. If a database is going to power the above complexity and have the latency for that 100ms speed, it needs to meet some very exacting technical requirements:

- **High database concurrency** – With potentially millions of match requests all hitting at the same time, the database needs to have the ability to handle concurrent operations without slowing down performance.
- **Low latency** – As a general rule of thumb when running applications in production, every millisecond spent in the database means 100 milliseconds felt at the application level, which means the database needs to be capable of <1ms latency to hit the 100ms goal.
- **Scalability with consistent performance** – The volume of matchmaking requests can surge or drop in massive spikes. The database needs to scale and deliver the same performance whether it's the top of the spike or the bottom of a trough.
- **Matchmaking data model compatibility** – Data models like graphs can drastically increase the speed of matchmaking, which means any real-time matchmaking database needs to include support and compatibility for those data models.
- **Integrated capabilities** – By integrating search or other capabilities into the database, you cut down on the number of transitions between services, which helps the matchmaking service run faster and the frontend experience more seamless.
- **Flexible deployability** – The matchmaking services, and its database, needs to be able to deploy wherever the games are run, whether it's in the cloud, on-prem, or in a hybrid cloud environment.
- **High global availability** – It's not enough for the matchmaking service to run fast. It needs to be reliably available whenever (and wherever) your players are online, which means your database also needs to have high availability while running on a global scale.

© 2021 Redis

As you can see, there's a lot that needs to happen between a matchmaking request and slotting the player into a specific game on a specific server. And it can only happen as fast and as accurately as your database can reliably respond to requests.



© 2021 Redis

Smart matchmaking needs a real-time database

So if real-time response is <100ms, why does a real-time database need <1ms latency? This comes back to the rule of thumb above. It can easily take 50ms for the request to move through the network, then another 50ms for server and infrastructure time, leaving somewhere between 0 and 1ms for database latency. Unfortunately, this low latency isn't possible with traditional disk-based relational databases. To get that kind of speed, you're going to need to look at in-memory NoSQL databases. For example, Redis Enterprise is an in-memory NoSQL database capable of 50 million operations per second at <1ms latency.

But as you saw in the technical requirements enough, speed alone isn't enough. A true real-time database needs to be able to deliver real-time responsiveness at any scale, with high availability, across the globe. And it has to be able to do it within real-world architectures, which means data model compatibility and the right additional frontend capabilities for real-time matchmaking from start to finish. With Redis, this is done through modules. The RedisGraph module provides graph database functionality, while the RediSearch module provides frontend search capability.

All told, the real-time database behind any real-time matchmaking service needs to have the capability for matchmaking search and data models, automated resharding for scaling, 99.999% availability, and the ability to deploy in any environment.

© 2021 Redis

Conclusion

Real-time matchmaking can make or break your game, even if the players never know what's going on behind the scenes. Make sure your matchmaking increases engagement, DAUs/MAUs, and game revenue by powering it with a real-time database.

Find out other ways a real-time database can level up your game and increase player engagement. Go to redis.com/gaming or download the *Level Up Your Gametech with a Real-time Database* white paper.

© 2021 Redis

About Redis

Data is the lifeline of every business, and Redis helps organizations reimagine how fast they can process, analyze, make predictions, and take action on the data they generate. Redis provides a competitive edge to any business by delivering open source and enterprise-grade data platforms to power applications that drive real-time experiences at any scale. Developers rely on Redis to build performance, scalability, reliability, and security into their applications.

Born in the cloud-native era, Redis uniquely enables users to unify data across multi-cloud, hybrid and global applications to maximize business potential. Learn more about Redis at redis.com and sign up for your free trial.