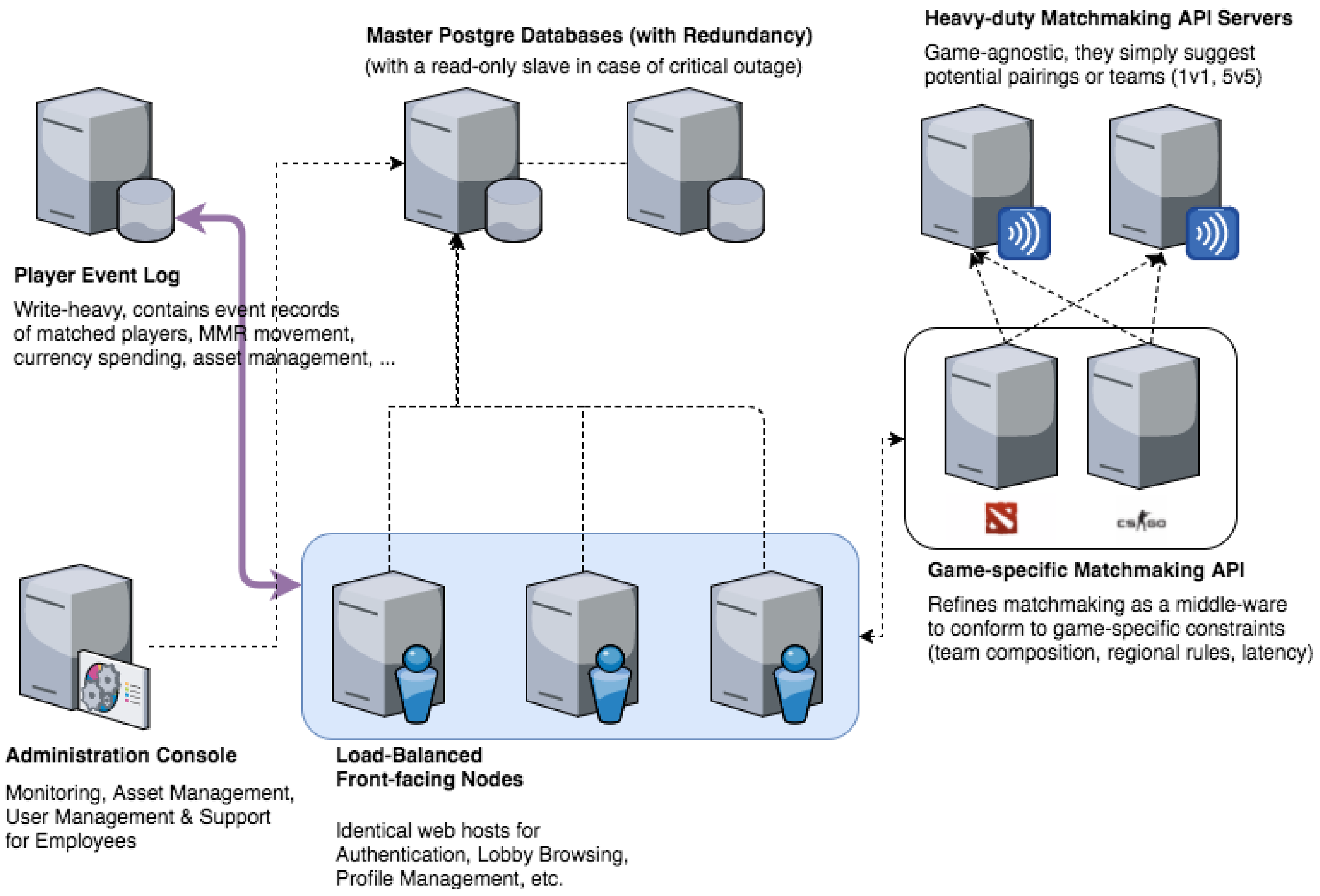
Architecture



Master Postgre Databases (with Redundancy)
(with a read-only slave in case of critical outage)
Heavy-duty Matchmaking API Servers
Game-agnostic, they simply suggest potential pairings or teams (1v1, 5v5)
Player Event Log
Write-heavy, contains event records of matched players, MMR movement, currency spending, asset management, ...
Game-specific Matchmaking API
Refines matchmaking as a middle-ware to conform to game-specific constraints (team composition, regional rules, latency)
Administration Console
Monitoring, Asset Management, User Management & Support for Employees
Load-Balanced Front-facing Nodes
Identical web hosts for Authentication, Lobby Browsing, Profile Management, etc.

Summary

Traditionally users would interact with the matchmaking platform through a web- or desktop-client, communicating with Front-facing web servers (any of Node.js, PHP, Java, Python, etc. depending on team structure). These web servers are traditionally light and they mostly use cached or pre-computed data from read-only slaves (database replicas, mongo, redis) with the exception of new data entries or currency exchanges which are handled through other APIs (internal and external).

A user accesses one of these through a load balancer and his main objective is to orchestrate his finances and arrange matches with suitable opponents. Assuming that several incompatible game options (e.g., moba, shooters, duels) are available, different (minor) variations of matchmaking API servers are needed to handle their respective specialized domains (ie.

additional validation for teams with certain compositions, matching people of the same cs:go league, etc.). It is plausible that to assume that gameplay volume and populations differ vastly by game so it is easier to keep them separated and ready in case more API nodes of a certain type are necessary to handle larger traffic.

These game-specific APIs sign the player up to lobbies with the heavy-duty matchmakers. These will likely face heavy data traffic and CPU constraints so it is advised to keep game domains (csgo, mobas, dota) away as much as possible and simply focus on emitting potential pairings or suggested larger teams to their respective domain API nodes. At high volumes and/or populations it would be advisable to maintain these heavy-duty nodes in a language or platform that supports threading, allows for easy and robust matrix manipulation (Java, Go, perhaps an optimized node/python/php/php-hack cluster with in-memory databases). If low latency is desired, these two types of nodes can be connected through messages queues (sqs, redis, rabbitmq) to allow users to receive immediate feedback through websockets or short polling.

All events emitted from this ecosystem, be it matchmaking orchestration, match results, currency transactions would then be logged in a write-heavy database for safe-keeping. Think of it as a giant bank account where game matches, wins/losses, currencies and MMR movements are logged historically like transactions in a bank account.

It is important that the Matchmaking Infrastructure is as independent as possible (with read-only copies, slaves, caches) from the front-facing nodes to maintain low latency and minimal disruption at high load (either due to high traffic, large volume of games, code errors or infrastructure failure). These caches would likely have to be periodically primed from the event database and the master database for use on the matchmaking servers.

Matchmaking MMR

Placement matches from Dota 2

Assumptions

- Both FB MMR and Dota 2 MMR are ELO-esque so it is plausible to assume that if Player A and Player B differ by 50 Dota MMR in the 2500 (2%) range that they will also differ by 60 FB MMR in the 3000 range (2%). The error in this increases at the extremes but will hold for most players in the middle of the pack.
- All the players who are matched with new player Dave have at some point had a Dota 2 MMR upon entry into the FB system.
- FB MMR operates with 5 leagues. Assuming a gaussian distribution, the middle 3 will be the most populated with heavy MMR discrepancies in the lower and upper outer

brackets. It’s possible to merge the bottom-most leagues together to form a large noob- or pre-league but this may carry a negative stigma to players that end up in these brackets.

- All 5 leagues are denoted with a strict FB MMR bracket (ie. [1500, 1800]). Demotion and promotion processes are not relevant for the purpose of this example.
- There are enough FB players in the system to track a reasonable normalized conversion table between Dota 2 MMR and FB MMR (ie. historical records of the past 3-6 months), e.g. Dota 2 MMR [1500-1800] converts to [1600-1850] FB MMR which is Silver League.

Modus Operandi

1. Deliberately match player A with Dota 2 MMR *dMMR* with players P1, P2, P3, ... , P10 of significantly higher or lower dMMR skill level.
2. We track player A’s accuracy γ [0.0, 1.0], starting at 0.5. If player A wins games against opponent Px with lower dMMR and loses games against opponent Py with higher dMMR we increase the accuracy by 0.1 per game. We decrease it by 0.1 for losing against lower dMMR opponents and winning against higher dMMR opponents.
3. We continue this process until 10 matches have been played, avoiding opponents of very similar dMMR (if possible).
    a. If it is not possible to find much lower/higher rank opponents, it is possible to match pairs with much closer dMMR ratings, but the adjustments to accuracy will likely have to be smaller, e.g. 0.05 or less.
4. At the end of the placement matches player A with dMMR will have accuracy γ.
    a. If γ is below 0.2, we will treat this player according to point 6.
    b. If γ is between 0.2 and 0.6, we find the respective FB MMR rating for this player based on his dMMR from the aforementioned conversion table (see Assumptions) and assign him one league lower as stated in the table.
    c. If γ is above 0.6, we find the respective FB MMR rating for this player based on his dMMR from the aforementioned conversion table and assign him to the league from the table.
5. Once players are placed into leagues, their FB MMR is set to the league’s current median value. If a player has not been placed into a league due to rule 4.a, see 6.
6. If player A has not been placed into a league due to low accuracy γ, or if not enough players in the system have ever finished this process (thus having no FB MMR records to show), place the player into the 3rd (middle) league with 1500 FB MMR (assuming FB MMR is ELO-esque with movements between 700 and 3000).