Millions of Games per Hour

Ben Deane

Principal Engineer, Blizzard Entertainment

bdeane@blizzard.com, @ben_deane

30th April 2015

Outline

History & Problem Space

History

- Server lists don’t cut it any more
- Player population is large
- Game type proliferation
- e-Sports expectations

The Problem Space

- Find good games quickly
- Deal with varying game types
- Deal with large player population
- Deal with small player population

The Problem Space

- Assign games to servers
    - spread load evenly
    - fill up servers
    - deal with overload scenarios
- Deal with community preferences

Cooperative vs Competitive

Cooperative

- Drop-in, drop out
- Social matches
- Game type matching
- Party-based play

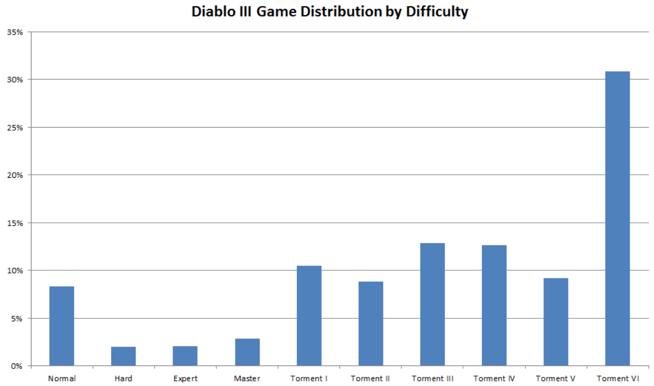
Competitive

- Join at start
- Skill-based matches
- Game size matching
- Team-based play

Evolution of players

- Players move through content over time
- Players gain skill over time
- Players return to content to farm it

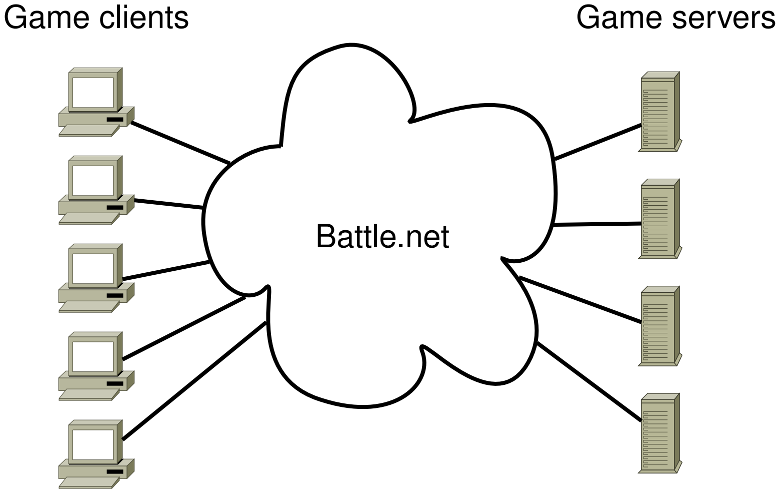
Evolution of players



Battle.net Tenets

- Keep it simple
  - Functionality comes from composability, not monolithic behavior
  - The best code is no code
- Be reliable
  - Easy configuration
  - No single points of failure
- Be game agnostic

System Overview



Abstracting Matching

What defines a game?

- A set of attributes

What defines a game?

- A set of attributes
- Partitioning attributes
  - Difficulty
  - Hardcore/Regular/Starter
  - Version

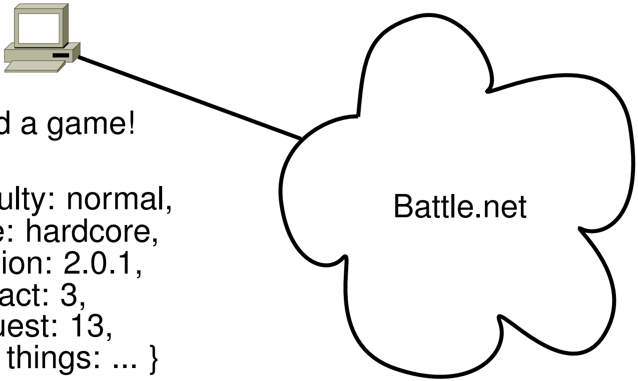
What defines a game?

- A set of attributes
- Partitioning attributes
  - Difficulty
  - Hardcore/Regular/Starter
  - Version
- Matchable attributes
  - Act number
  - Quest step
  - Other

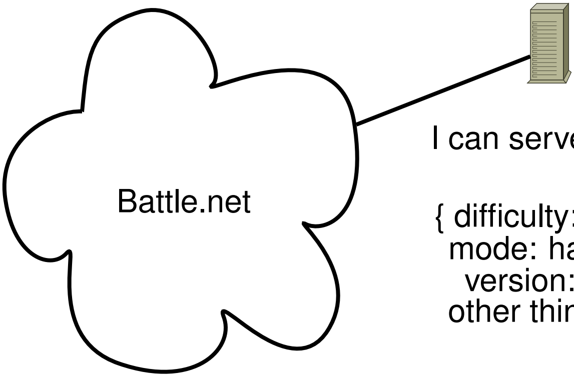
Attributes

- Attributes are key-value pairs
- Battle.net doesn't know what they mean
- Battle.net knows how to
    - Wrangle them in data structures
    - Do computations with them (hashing, sorting, comparing)

What a client does



What a server does



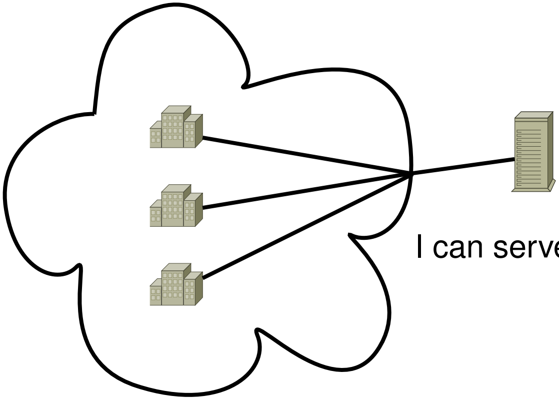
I can serve games!

{ difficulty: normal,
mode: hardcore,
version: 2.0.1,
other things: ... }

Game Factories

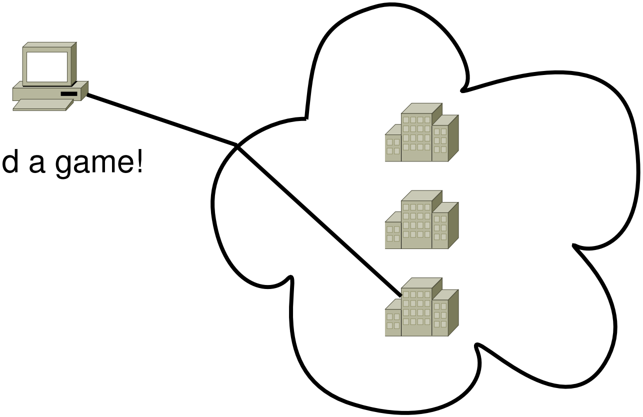
- Game factories represent partitions
    - normal-nonhardcore-v201-factory
    - hard-nonhardcore-v201-factory
    - etc
- Game factories are
    - specified in configuration
    - instantiated in response to server connections
    - combinatorial on relatively few axes

When a server connects



I'll associate you with these factories!

When a client asks for a game



I will use this factory!

Game Factories

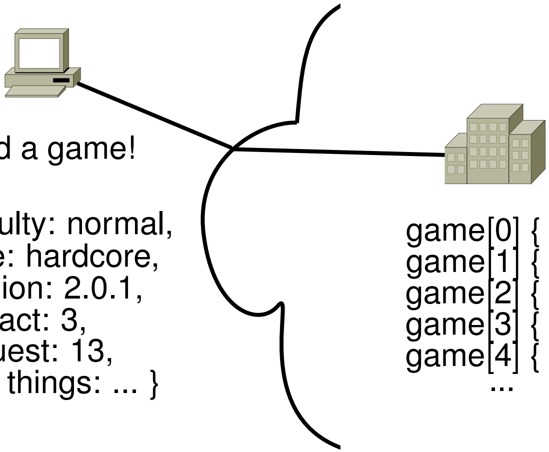
- Game factories reduce the matching problem
- Each factory matches the games it knows about
    - Based on the smaller number of matchable attributes
- Factories can use different strategies
- The factory abstraction is strategy-agnostic
    - cooperative
    - competitive

Cooperative Matching

FIND a game

- The API deliberately says FIND a game
  - not join a game
  - not create a game
- The create/join dichotomy is not part of matchmaking
- If a game cannot be matched, one will be created
  - Either way, you get into a game
- CREATE and JOIN have their place, but it's not matchmaking

Matching on attributes



Find a game!

{ difficulty: normal,
mode: hardcore,
version: 2.0.1,
act: 3,
quest: 13,
other things: ... }

game[0] { ... }
game[1] { ... }
game[2] { ... }
game[3] { ... }
game[4] { ... }
...

The most important "attributes"

- Is the game open for matching?
- Is there space in the game?
- Factories partition the open game list by number of open slots
- Players match in groups
    - individually
    - parties

Remaining problem

- We have a candidate set of games
  - that are open for matching
  - that can fit our players
  - that are associated with some attributes
- We want to match our attributes against the games

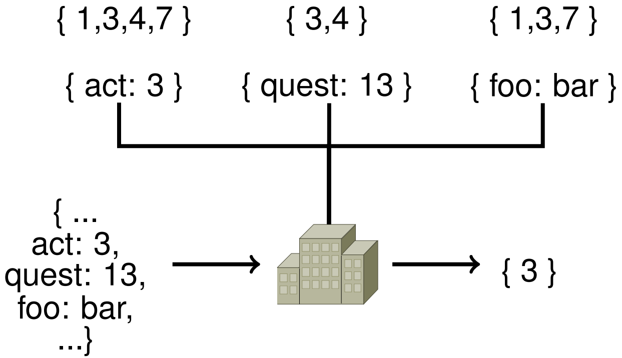
Dynamic matching

- N-dimensional nearest neighbor search?

Dynamic matching

- N-dimensional nearest neighbor search?
- Index the games list by each attribute
- Each (single) attribute lookup yields a set of games
- To find a match for all, compute the set intersection

Indexed games



Indexed stats too

- As for game matching, so for extracting stats
  - number of games
  - number of players
  - min/max/average game duration
  - etc
- Stats can be queried using the same attribute matching/indexing scheme

The Problems?

- Find good games quickly ✓
- Deal with varying game types ✓
- Deal with large player population ✓

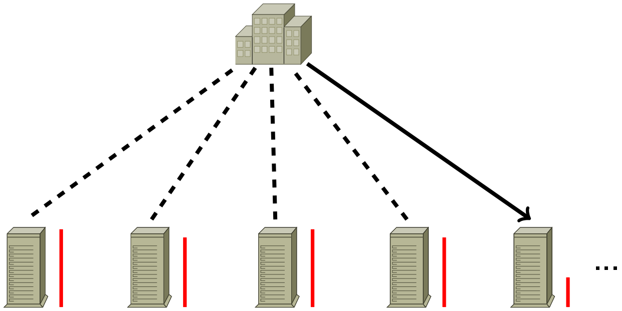
The Problems?

- Find good games quickly ✓
- Deal with varying game types ✓
- Deal with large player population ✓
- Deal with small player population ?

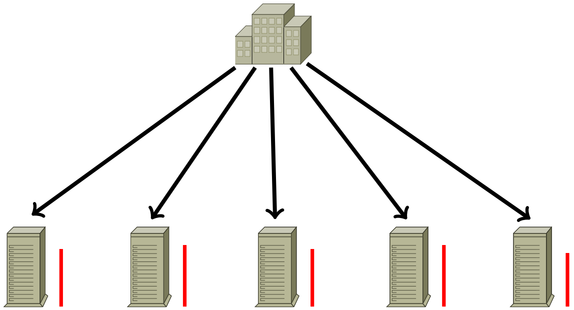
The Problems?

- Find good games quickly ✓
- Deal with varying game types ✓
- Deal with large player population ✓
- Deal with small player population ?
- Assign games to servers

Filling servers



Spreading load



Spread players vs Fill up games

- Max N players in a game
- k players in a matching group
- Just match against games with the "right" number of open slots
  - to fill, match with N-k, N-k-1, . . . 1
  - to spread, match with 1, 2, . . . N-k

Queueing and Distribution

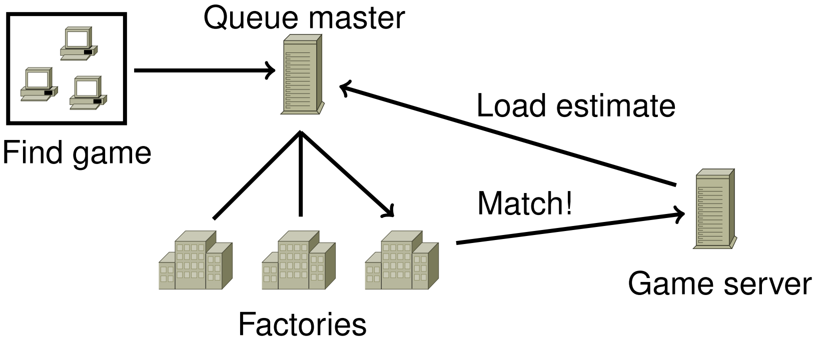
The Problems

- Take account of server load somehow
- Assign games to servers evenly
- Allow new servers to come online and get balanced
- Deal with servers being temporarily full

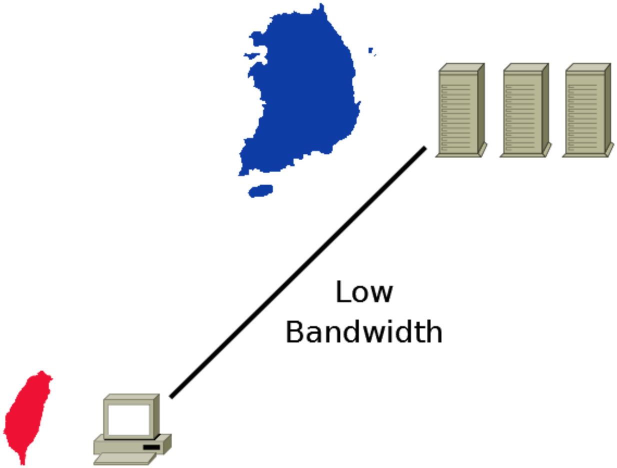
1st attempt

- One server deals with queueing
    - with fail over to another
- Poll game servers for load
- Assign a nominal load per game creation/player addition

1st attempt


Queue master
Find game
Load estimate
Match!
Game server
Factories

Extra complexity



Extra complexity

- Game server capacity isn't the only factor
- Limited bandwidth to regional data centers results in poor experience
    - very important for hardcore mode!
    - KR-TW pipe is small

Problems with 1st attempt

- Polling has a delay
  - Queue master has to anticipate load assigned during the delay period
- Single point of failure
- Queue master doesn't know what the result of matching will be
  - it deals with groups of players only
  - it doesn't know whether a game will be joined or created
  - these two scenarios have different load characteristics

Problems with 1st attempt

- Hard to reason about rate of game influx to a given server
    - hard to bring up servers
    - open beta bug
- Hard to estimate load on servers
    - queueing is by player but load is by game
    - starting games is spiky load
    - running games is smooth load

2nd attempt

- Drop regional bandwidth requirement
- Allow servers to advertise the number of games they can take
- Slots apportioned to factories by popularity
- Queue can be distributed across hardware

Competitive Matching

Competitive Matching

- Join at start
- Skill-based
  - Elo-like player rating
- 1v1 or NvN

Game agnosticism

- Battle.net doesn’t know about player skill per se
  - Stats and logic are down to the game
  - Abstracted as a single player score

Hearthstone

- Batch players
- Sort batch by skill
- Make games according to threshold
- Easy!

Hearthstone

Batch	Sort	Form Games	
{phineas: 1.5}	{ candace: -0.7}	{ candace: -0.7}	♥
{ferb: 1.8}	{doof: -0.5}	{doof: -0.5}	
{isabella: 1.7}	{vanessa: 0.3}	{vanessa: 0.3}	💔
{doof: -0.5}	{phineas: 1.5}	{phineas: 1.5}	
{vanessa: 0.3}	{isabella: 1.7}	{isabella: 1.7}	♥
{ candace: -0.7}	{ferb: 1.8}	{ferb: 1.8}	

StarCraft II

- More options make it harder
  - teams/random NvN/free-for-all
  - map selections
  - different player ping times
- Simple sorting doesn't work
  - hill-climbing optimizer

Hill Climbing

1. Take a batch of players, assemble some games
2. Swap something around
3. See if the games are better
4. If you still have time, goto 2
5. Start the games that are viable
6. Put leftovers in the next batch and relax constraints

Performance Issues

- Hill-Climbing algorithm is $O(n!)$
- Fewer players means you need to work harder
- More players means it's easier to make viable games
- With appropriate selections for batch size the system is self-regulating

Statistical Issues

- Player score starts out uncertain
  - 1v1 requires ~25 games to focus
  - Other modes require more games
- Unbalanced teams are hard to match
  - Teams of friends are often variable
  - e.g. Experienced player + novice
- Aggregating team scores is tricky

Statistical Issues

- Very good players can't get good matches
    - as in any sport
- Players like their rating to increase
- Players get better, then leave

Competitive Design Issues

- If your game is 1v1 and your matchmaker is perfect:
    - 50% of players will lose the first match
    - 25% of players will lose the first two matches
- You need to make the game fun even when players lose
    - Progression
    - Achievements

Abuse Issues

- Achievements incentivize loss-botting
- Disconnection = loss

Thanks for listening

- Factories allow abstraction of strategies
- Queueing to manage load
- Competitive and Cooperative are different animals
- Reduce problems by design

Ben Deane
bdeane@blizzard.com, @ben_deane

Testing

- Bonus section!
- A little bit about how I tested. . .

The need to test

- I can’t run at scale on my desktop machine
- I need to be sure that the system runs at scale

Possibilities

- Use repurposed ("spare") hardware
- Use Amazon EC2 or similar
- Or I could just figure out how to test on my machine

TDD

- I had already built the parts with unit tests
- I/O was separated out
- Configuration was dependency-injected
- Matchmaker logic was separated out

Unit testing for performance

- Absolute perf not so good
    - my machine isn't a production machine
    - my machine can't simulate a million players
    - unit tests are supposed to be fast

Unit testing for performance

- Modified unit test framework
  - Call tests N times and time the result
  - Vary N from 2^a to 2^b
  - Divide results to obtain complexity order
- Algorithmic complexity tests
  - No hidden $O(n)$ or worse algorithms
  - Everything is $O(log_2 n)$
- I didn’t do any statistical analysis: good enough is good enough

Thanks for listening (more)

- Factories allow abstraction of strategies
- Queueing to manage load
- Competitive and Cooperative are different animals
- Reduce problems by design
- Test at scale somehow

Ben Deane
bdeane@blizzard.com, @ben_deane