Server and Client Roles

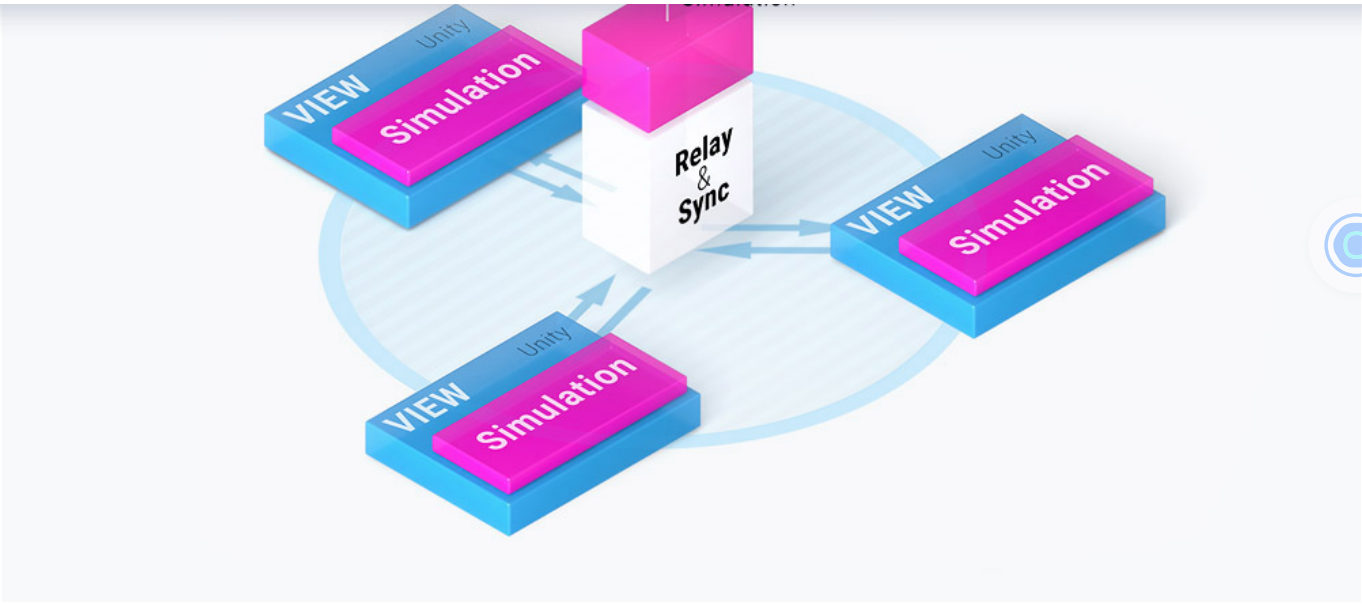
Introduction

In deterministic systems, game clients only exchange player input with the simulation running locally on all clients.

In the past, a lockstep approach was necessary for all game clients to wait for the input from all other players before updating each tick/frame of the simulation. *Quantum's advanced predict-rollback system* makes this a relic of the past and allows for realtime deterministic simulations.

In Quantum, the predict system allows game clients to advance the simulation locally using input prediction, and the rollback system takes care of restoring the local game state and re-simulating any mispredictions.

A game-agnostic authoritative server component (photon server plugin) manages input latency and clock synchronization, clients therefore never need to wait for the slowest one to rollback/confirm the simulation forward.



In Quantum, deterministic input exchange is managed via game-agnostic server logic. This prevents game clients with bad network from interfering with the experience of players on good networks.

Server

Quantum is a server based solution. The Quantum server plugin exist in two flavours, a vanilla version that is completely game-agnostic and a custom version. The former is identically for all games running on the ***public cloud***, and the latter can be deployed on a dedicated ***photon enterprise*** cluster.

The ***configuration*** for the current game session is defined by the `DeterminsticConfig` sent to the server by the client who created the photon room; this includes information such as the update rate and hard tolerance for input acceptance which will be used by the server to manage the session in question.

By Default

The default plugin running for **every** Quantum App ID does everything required relay and synchronise the client inputs and ensure the clients run a deterministic simulation locally. This includes:

- input management (timing and validation)
- clock sync
- player index assignments

photon

Input and Verify Frames

An often asked question is: *Who signs/authenticates **verified** frames?* . The answer is: technically no-one verifies a frame; rather it is a frame which meets the following 2 criteria:

1. all validated inputs for frame/tick N have been received from the server; and,
2. all previous frames/ticks N-1 have been simulated using their respective validated input.

If both these conditions are fulfilled, then a frame is qualified as **verified** . For more information on how to check a frame's status, please refer to the *Frames* document in the manual.

The server just "signs" an official stream of accepted inputs and forwards those validated inputs to the clients in the session. Since the inputs are pivotal to advance the local simulations, the validated the inputs associated with a specific tick/frame are enough for all clients to locally simulate a deterministic **verified** frame.

Custom Plugin

Dedicated servers (*Photon Enterprise*) let you go beyond the basics. Using the *Quantum Custom Plugin* you can add custom controls for input interception, configuration data, and much more to customize the session management by the server.

NOTE: Developing and deploying a custom plugin does NOT require any have to changes to your gameplay/client code. It is an optional step that can be done incrementally after the core game has been finished.

Authority

If you are aiming to retain game authority, you can do any of the following with the custom plugin:

- push the full input stream from the server to a backend.
- check the game match results from all clients that finished the game.
  - If they all match, just accept the results as-is.
  - If there are mismatches, temporarily accept majority vote -i.e. the result send by most clients-, and mark the match for post-fact validation
- automated post-fact validation; just run the saved replay with quantum's console-runner. This does not require Unity and a full match can be simulation within a few seconds.

photon

Server-side Simulation and Cheating

It is technically possible to run the full simulation on the server, there are no barriers to it. However, it does not make much sense from a practical and cost perspective.

Usually this question comes up in an effort to combat cheaters. Should that be your driving concern, then there is no problem.

The game being deterministic, there is no way to "cheat" in the sense it is usually understood. In the case of validating match results or responding to complains, you can simply use the official input stream that has already been validated by the server to fully confirm a match result on your end - ***see previous section***. This can be done without running a master-copy of the sim on server.

Client

The local client runs the game (simulation in Quantum + view in Unity). Its job is to:

- send input to the server in time. If the input does not reach the server in time, it is replaced.
- predict frames based on the synced clock.
- receive a continuous stream of input confirmations from the server, rollback (if necessary) and verify frames. See the previous section on *Input and Verify Frames* for an overview of the process.

The local simulation of **verified** frames runs at a fixed update rate. The amount of **predicted** frames varies and the prediction will continue as soon as the simulation has accumulated enough delta time in the local session.

Assuming frequent rollbacks, this means a 30hz simulation rate can actually run at 300hz for predicted frames. Although this may appear insanely high, it simply translates to 1 rollback per 1 verified frame + 10 predicted frames. The thing about predicted frames is that they are simulated, then on a rollback, simulated AGAIN.

The update rate for the view in unity is independent from the simulation rate. To simplify the example let's assume a 30hz Unity update, this means each Unity update:

- advances 1 verified tick (1 full heavy frame simulated + synced events are dispatched); and,
- rolls back all predicted ticks and re-simulates them starting from the latest verified up to SYNCED-clock (proportional to RTT/2).

predicted ticks are "light weight" enough to run at the aforementioned rates; one way to lighten the load is to use the build in prediction culling which helps predict entities around the camera, i.e. in the view of the player.

Input, Ping and Latency

The server uses the `HardTolerance` in `DeterministicConfig` as the timing-limit for accepting inputs. If the server does not receive the input within the defined time frame, it will replace the client's input.

The **max** length of a client's local rollbacks is: `hard-tolerance + RTT/2 + jitter`. The server enforces this limit for every client. This guarantees a consistent experience no matter the number of players, because it ties the quality of a player's experience to their **OWN** network quality; in other words, if one client has a bad connection only their own experience will be affected and everyone else's will stay smooth.

The more players are participating in a game session, the more important it becomes as the probability of at least ONE of them having network issues grows.

TL;DR:

These are the basic building blocks of a Quantum game:

Client:

- Game Client Simulator: communicates with Quantum server, runs local simulation, performing all input prediction and rollbacks.
- Custom Gameplay Code: developed by the customer as an isolated pure C# simulation (decoupled from Unity) using the Quantum ECS. Besides providing a framework for how to organize high-performance code, Quantum's API offers a great range of pre-built components (data) and systems (logic) that can be reused in any game such as deterministic 3D vector math, 2D and 3D physics engines, navmesh pathfinder, etc.

Server:

- Quantum Custom Plugin: can extended the server plugin to integrate with customer-hosted back-end systems (matchmaking, player services, etc).

Predicted vs Verified Frames:

- `predicted` : the client simulates CPU-light "predicted" frames at any frame rate based on the inputs predicted by the local client.
- `verified` : the client simulates CPU-heavy "verified" frames at a determined frame rate when a server validated input stream is received; misprediction are rolled back and predicted again.

Back to top

photon

Connect

Public Discord
YouTube
Facebook
Twitter
Blog
Contact Us

Languages

English
日本語
한국어
简体中文
繁体中文