Cloudflare's Graph-Based Maintenance Scheduler

I always find it fascinating to see how major infrastructure players solve their own internal tooling problems. Cloudflare recently shared how they built their automated maintenance scheduler, and the evolution of their architecture is a textbook example of hitting the limits of naive implementations.

The problem is classic constraint satisfaction: With data centers in 330+ cities, you can't just turn off a router for maintenance whenever you feel like it. You have to ensure that doing so doesn't isolate a region or break a specific customer's routing rule (like their "Aegis" product, which binds traffic to specific IPs).

The "Naive" Approach vs. The Graph

Their first attempt was to load all the data—server relationships, product configs, health metrics—into a single Worker. Unsurprisingly, this hit memory limits immediately. You can't fit the state of a global network into a single ephemeral function's RAM.

The fix was to shift mental models from "loading a dataset" to "traversing a graph." They adopted an object-association model inspired by Facebook's TAO. Instead of "give me all Aegis pools," the query becomes "give me the Aegis pools associated with *this* specific datacenter."

This reduced response sizes by 100x, but introduced a new problem: the "N+1 query" issue, or what they call the "thundering herd" of tiny requests.

Middleware as the Hero

To solve the request volume, they built a fetch pipeline middleware that handles:

- **Deduplication:** Merging identical in-flight requests (similar to Go's `singleflight`).
- **LRU Caching:** A tiny in-memory cache for the immediate execution context.
- **CDN Caching:** Caching GET requests at the edge with careful TTLs (1 minute for real-time data, hours for static infrastructure data).

The result was a ~99% cache hit rate. It's a great reminder that when you move to micro-requests, the network overhead becomes your new bottleneck, and caching strategy becomes your primary optimization lever.

What I take from this

The most interesting part for me wasn't the scheduler itself, but the pivot to **Parquet files on R2** for historical analysis.

Querying months of Prometheus TSDB blocks from object storage is painfully slow because of random reads. By converting that data to Parquet (a columnar format), they could issue precise range requests, fetching only the specific columns needed. They report a 15x performance improvement. It's a strong argument for using big-data formats even in operational tooling when the scale gets large enough.

Contributor: Alessandro Linzi