Building Petabyte-Scale Postgres Clusters with Bagger

Chris Travers

June 4, 2024

Speaker Intro

(Introducing myself)

Agenda

- Why Bagger was Built at Adjust
- General Design of Adjust's Version
- Shortcomings of the First Version
- Design of the Open Source Version
- Tradeoff lessons

Life Before Bagger

- We used ElasticSearch
- Size of 1PB for 30 days of data retention
- Velocity made the system unusable

Specific ES Problems

- Noisy protocol lead to poor performance
- At that scale, ES was extremely difficult to start/restart
- GC Stop the world events would knock nodes out of cluster
- Large queries could kill the entire cluster
- We had to use fixed ES schemas to prevent major problems.

Some of these may have gotten better, but I doubt all have.

Enter Bagger

- Minimalist design
- Schemaless
- Linearly Write-scalable
- Built for write-heavy workloads
- However, read-scalability is static

Adjust's Design

- Kafla partitioning to Sharding
- Random Sharding
- Custom data shovel (Schaufel) for ingestion
- Patched Postgres
- Storing just JSONB docs
- C-language routing trigger

We Wanted to Open Source It

When I was heading the department I put some effort into open sourcing Bagger. However:

- Too many assumptions to Adjust's operations
- Statically defined partitioning criteria

Intermezzo

And then I left Adjust... fast forward 2 years

Restarting the Effort

- Several of us came together with experience on this system
- Designed a similar system
- Using the existing open source components

Design Stragegy

- Go with what we know
- Minimize unknown unknowns
- Evaluate larger changes independently later

Remember: We already had experience on a prototype that reached 10PB without difficulty.

Initial Technology Choices

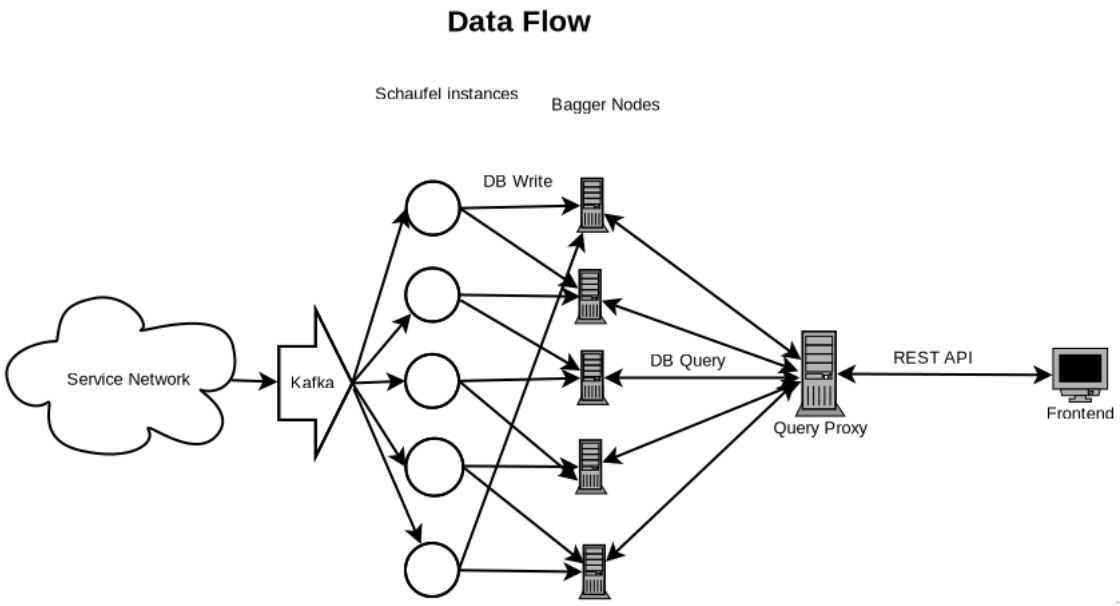
Going with What We Know

- Perl
- Moose
- PGObject
- Database-centered logic
- C-language triggers
- Perldancer
- Frontend framework to be decided
- Etcd for cluster state handling (though this is pluggable)

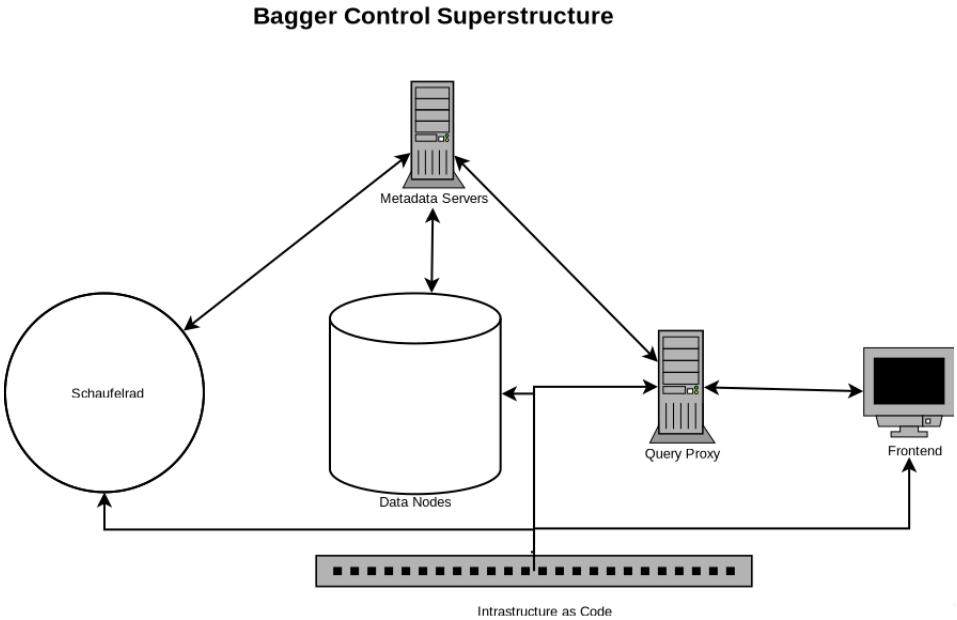
Architecture and Evolution

Will discuss the design and how it is different from Adjust's version on the next few slides.

Data Flow


Data Flow
Schaufel instances
Bagger Nodes
DB Write
Service Network
Kafka
DB Query
REST API
Query Proxy
Frontend

Superstructure



Current status

Node control infrastructure Complete

Metadata server schema/functions Complete

C-Language Trigger Not Done

PostgREST Config In Progress

Quiery Proxy Not Done

Front End Not Done

Feel free to get involved!

Beyond GA

Here is a list of some changes we plan to evaluate:

- Better language for query proxyL: Java? Golang? C?
- Integration of CitusDB? Pros? Cons?
- How can We Scale Reads?
- Replication instead of write-twice?
- Impact of compiled block size? Can we remove most TOASTing?

And many more

Tradeoff Lessons

- Scalability Tradeoffs (read/write)
- Benchmarking Everything
- Specialization generalization

Bonus learnings

- Know your requirements
- Different architectures for different needs
- Linear scaling in one dimension is easy.
- Scaling reads and writes is much harder.

Questions?

Also find us at `https://github.com/onemoredata/bagger`
Feel free to reach out to me at chris.travers@gmail.com