Femtocode: querying HEP data

Jim Pivarski

Princeton University – DIANA

April 17, 2017

Distributed server

Scaling up to full datasets

Exploratory data analysis requires human-scale time to completion: seconds at most. If a query server takes much longer than this, physicists will go back to private skims.

Scaling estimates for one query:

- Several dozen samples, totaling $\mathcal{O}(10\ \text{TB})$.
- Every query runs over all events; a single query rarely uses as much as 1% of the *columns*. (Popularity distribution is steep.)
- Worst cases observed in early implementation: 30 ms/MB.
- Implies 3000 core-sec for that query: 3 seconds for 1000 cores.

Scaling up to full collaborations

diana hep

Scaling estimates for multiple users:

- Most analyses have significantly overlapping needs. Evidence: home-grown skimming frameworks select the same 10% of CMS MiniAOD (Bacon, Pandas, Cms3, TreeMaker).
- File I/O is more expensive than processing: ∼40 ms/MB versus ∼2 ms/MB. Everyone wins if users *share* cache.
- 10% of 10 TB of samples is 1 TB, which easily fits in RAM on a cluster of 1000 cores (hard to fit on one user's machine).
- Short-lived queries are less likely to use resources at the same time, so shortening latency also reduces contention.

The parameters of the final system depend on the hardware allocated for it, but improving software can steepen the performance per price.

Dividing the problem

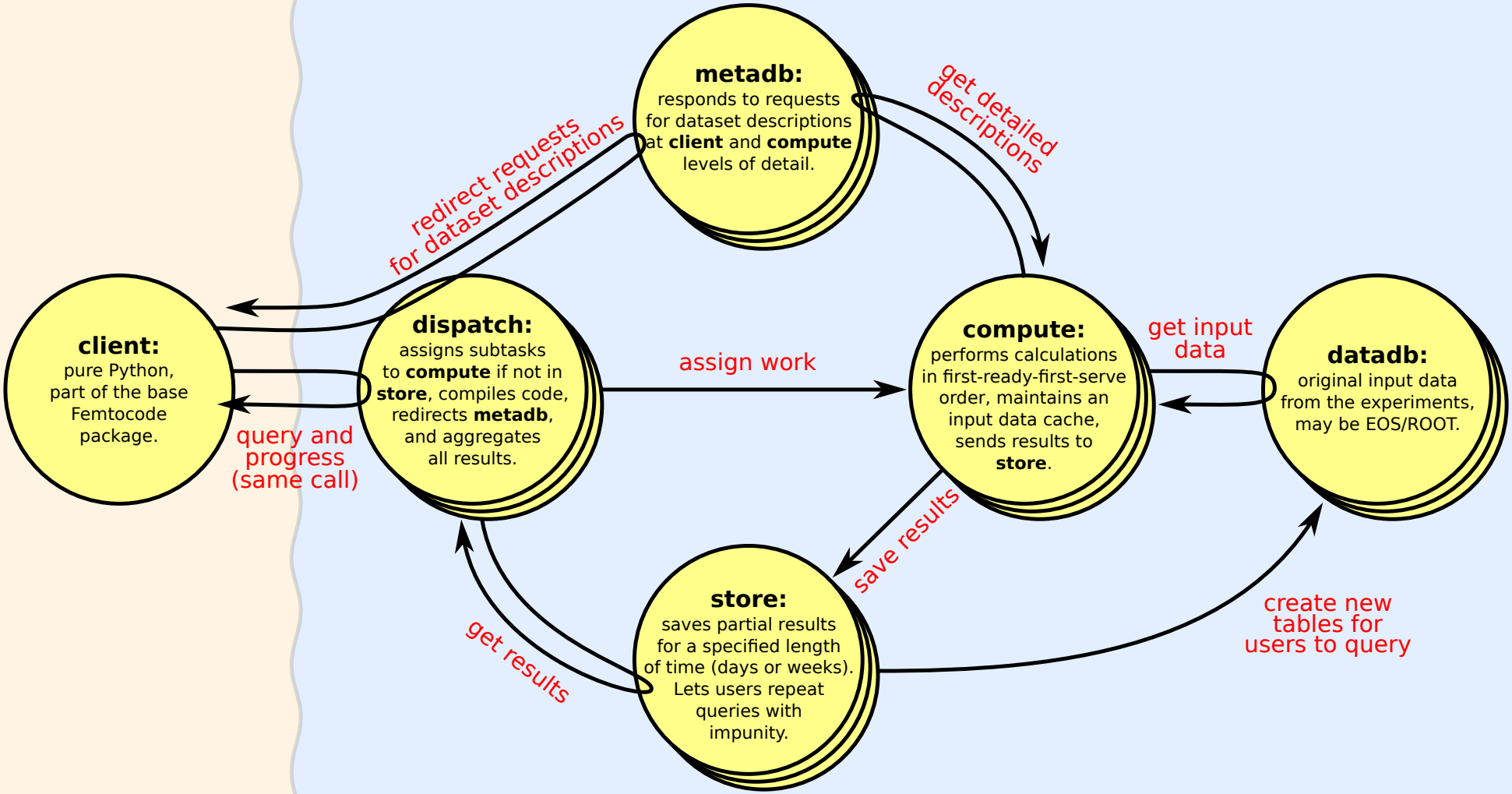
Femtocode's design philosophy is to do work up-front to streamline the event loop. In the distributed server, managing subtasks is part of this up-front work. Time to completion could be summarized as

$$\text{time} = C_1 + C_2(n_{\text{cores}}) \cdot N_{\text{subtasks}} + \frac{C_3}{n_{\text{cores}}} \cdot N_{\text{events}}$$

- C_1 is a constant, dominated by 70 ms of JIT-compilation time,
- $C_2(n_{\text{cores}})$ is the time spent managing subtasks, a complex concurrent processes affected by Amdahl's law.
- C_3 is the compiled and embarrassingly parallel part that executes the user's query.

The order parameter in this problem is N_{events}. We get to choose $N_{\text{subtasks}}/N_{\text{events}}$ and can simply make partitions larger if the Pythonic "data management" part becomes an issue.

Distributed system layout



Not a batch system

Although incoming jobs are scattered to compute nodes, computed in parallel, and then gathered, this differs from a batch system in important ways.

1. No "job id" or attempt to send results back to the user.

   Identical queries on the same dataset will always yield identical results, so jobs are identified by a hash of the query itself.

   Client polls for updates and may break/reestablish connection before it's done. Dispatch checks the "store" for partial results, rather than re-running.

   Therefore, when users refresh their analysis scripts or run tutorial examples, they don't stress the computation engine.
2. Subtasks are assigned to "compute" nodes according to a hash function on required input data. Subtasks requiring the same input data go to the same nodes and are more likely to find it in cache.

Conclusions

Conclusions