The Pliny Database — PDB

Chris Jermaine

Carlos Monroy, Kia Teymourian, Sourav Sikdar

Rice University

PDB Overview

PDB: Distributed object store + compute platform

In Pliny project:

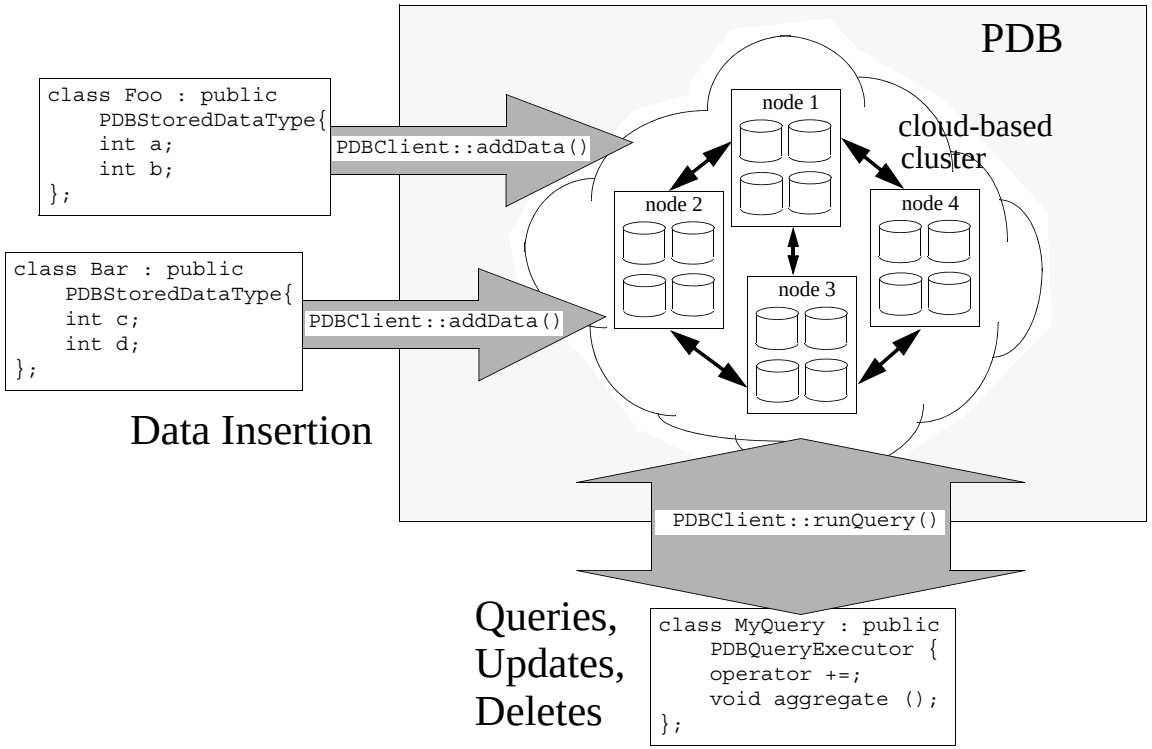
- Used to store processed source code
- Used to store large volume of program analysis results
  - Call graphs
  - Parse trees
  - Bags of system calls
  - Text mining results for comments
  - ...
- Used to perform new program analyses at scale
  - Program element at-a-time
  - As well as batch (such as ML)

Design Goals

Want a system that combines

- Flexibility of a schema-less system (Hadoop, Spark)
- High performance of a database system

Programming Model



User defines data types in host language (C++, at least initially)

System stores data persistently in distributed compute cluster

PDB manages object inserts, deletions, updates like a DB

Data manipulation via code in the host language

Programming Model

No data manipulation language (SQL/XQuery/etc.)

- Easy to write new query types...
- ...Machine learning computations
- ...Top-k queries (fundamental to Pliny!)
- ...Graph analyses

All access to stored objects via user-defined methods

Why Don't Existing Systems Work for Pliny?

Classical RDBs not a good fit

Not easy to store program analysis results as flat tables

- Trees
- Graphs
- Lists

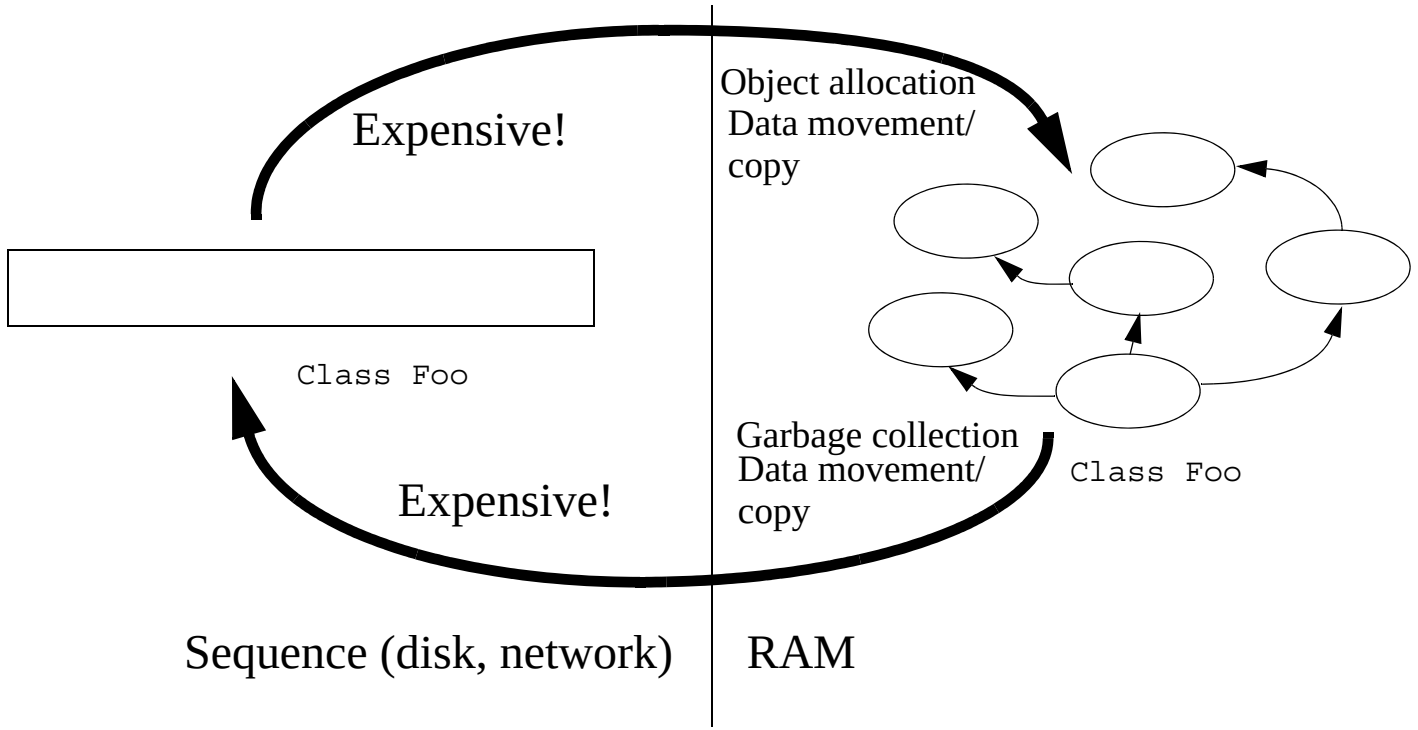
Difficult to implement program analysis in classical DML

- Want to allow arbitrary user-defined computations
- Hard to do in SQL/XQuery/etc.

Why Don't Existing Systems Work for Pliny?

Existing Big Data systems not a good fit

- Hadoop/Spark are schema-never: BIG performance hit
- Excellent flexibility, cost of performance/complexity



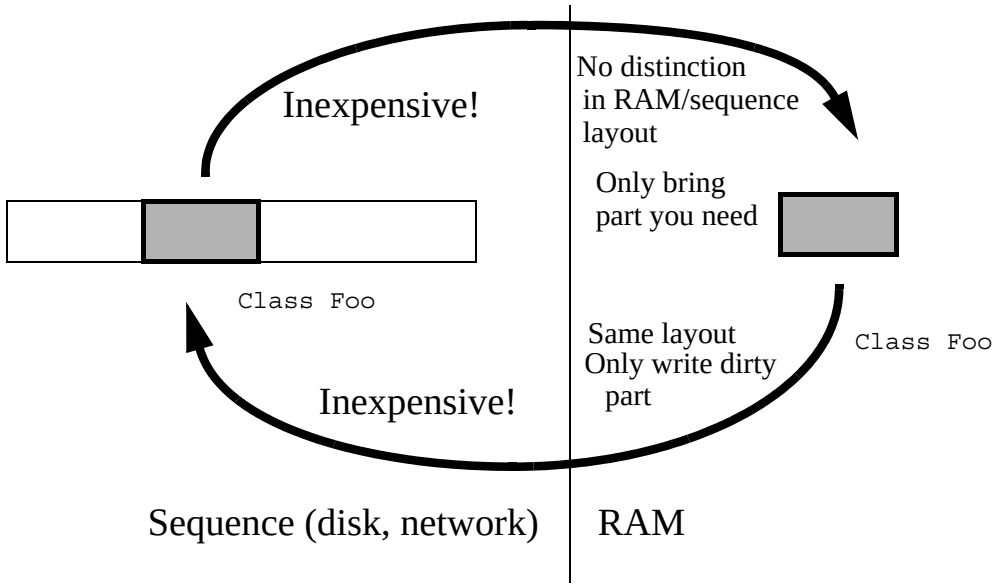
Performance/Scalability Goals

Scale to 100s of machines

Performance/Scalability Goals

10X faster than Spark for most batch computations. How?

- Pliny data DOES have structure
- Just not structure well-served by traditional data models
- Like DB: control data movement, layout in RAM and on disk
- Don't delegate to user, or 3rd party tools



Performance/Scalability Goals

100X faster than Spark in special cases

- How?
- Leverage classical DB techniques
  - Indexing
  - Fast parallel join algorithms
  - Smart buffer management

Preliminary Performance Results

Task

- (1) Load up 20,000 text documents
- (2) Build a dictionary of most frequent 10,000 words
- (3) Add a sparse, 10,000 entry TF vector to each document
- (4) Close down, re-load data
- (5) Run a top-k query to find 12 closest docs to query doc

Run on a single, 4-core Amazon EC2 machine

Spark vs. PDB

Task	Spark Time (sec)	PDB Time (sec)
(1)	0.6	3.5
(2)	146	2.8
(3)	85	6.7
(4)	0.5	1.8
(5)	17.2	0.7
Total	249	15.5