CS265 Spring 2016 research project
Self-designing Data Systems

Keywords: self-designing data systems, iterative optimization, join operator, join algorithms, genetic algorithms, Bayesian optimization

Problem: Matching a scientific or commercial application with its perfect data system is a time-consuming task that not only requires expertise in the area of databases, but also a willingness to compromise. Often, off-the-shelf solutions will only provide suboptimal performance. However, building a custom-tailored system for the task at hand is an expensive endeavor. Modifying an existing system under today’s monolithic implementations is extremely complex, while designing and building a new data system from scratch requires expertise and tenths of man-years worth of time.

Rather than chasing changes in workload and hardware by continually designing and implementing new systems from scratch, or forcing end-users to settle for suboptimal solutions, we envision self-designing data systems that smoothly and autonomously navigate the design space to quickly generate the optimal solution for a given application. Self-designing data systems would relieve both system designers and end-users of data management headaches, culminating in greater productivity. Moreover, a self-designing system may discover new architectures that researchers would have never even considered by synthesizing new solutions out of existing ones, mimicking the natural process data system architects are performing manually.

Project: In this project we explore the design space of database systems focusing on two crucial decisions: (i) what should the fanout of a b-tree be, and (ii) what join algorithm should we use, both for a given workload.

The goal of the project is to build an automatic iterative process that reaches this decision. In particular, the student should use either genetic algorithms or Bayesian optimization, in order to reach the final decision. Hence, here we define four different and sub-projects, that each student is expected to take one. For a given dataset, and workload:
(1) Use genetic algorithms to decide the fanout of the b-tree
(2) Use genetic algorithms to decide which join algorithm to use
(3) Use Bayesian optimization to decide the fanout of the b-tree
(4) Use Bayesian optimization to decide which join algorithm to use

Both genetic algorithms and Bayesian optimization operate as an iterative way to make an increasingly better guess leading to the “right” decision. Every iteration picks potentially a new fanout, or a new join algorithm before it converges to the one we should be using. For the join, the pool of possible algorithms should contain nested-loop join, block nested-loop join, hash join, radix join, and the optimization phase should include picking the write parameters.

A typical genetic algorithm requires: (a) a genetic representation of the solution domain, and (b) a fitness function to evaluate the solution domain, (c) genetic operators to create the new generation of solutions (crossover and mutation). At every step, a number of solutions are evaluated with the fitness function, and then new solutions will be generated by the genetic operators. The search typically stops either when we reach a plateau in performance, or when we have used a budget.

A typical Bayesian optimization process requires (a) a “prior” which is an initial probability distribution over the initial space which represents our belief about where the solution should be, (b) a “posterior” which is the probability after we have started taking into account evidence, (c) an “acquisition” function which will guide us to decide which should be the next experiment from which to gather evidence.

There are numerous variations as well as other open topics in this general direction. Students may explore any open topic in consultation with the instructor.

References

Supervisor: Prof. Stratos Idreos (stratos@seas.harvard.edu)

Responsible collaborator(s): Lukas Maas (maas@seas.harvard.edu)
Manos Athanassoulis (manos@seas.harvard.edu)