Thriving in the No Man's Land between Compilers and Databases

Holger Pirk, Jana Giceva, Peter Pietzuch

Imperial College London

Prashanth Menon, DB Reading Group, Spring 2019

Motivation

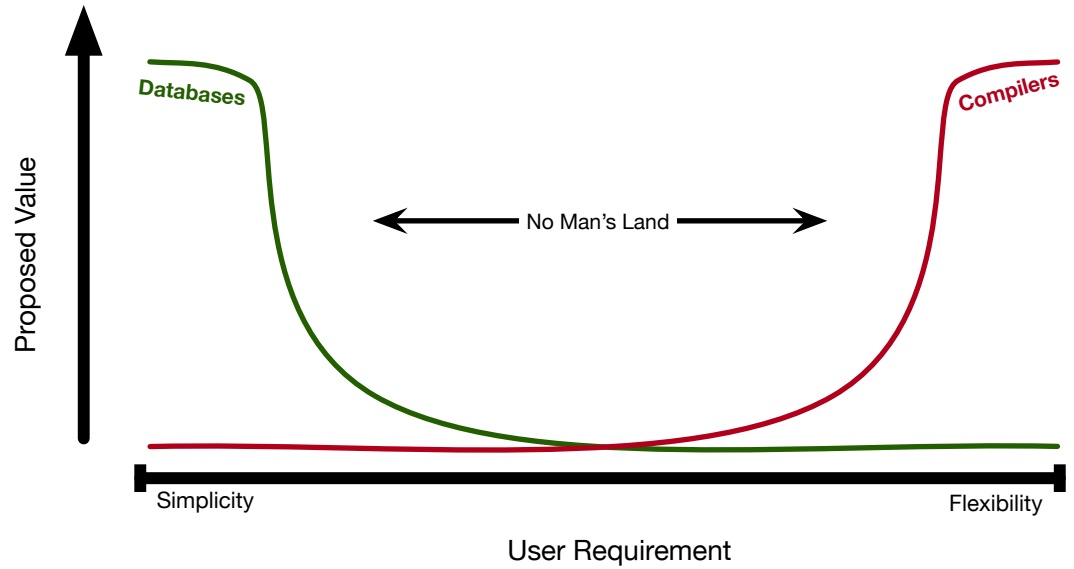
- RDBMSs initially sufficient for most applications
  - *Good out-of-the-box performance*
  - *Supports different workloads (OLTP, OLAP, HTAP)*

- Application requirements evolve
  - *Correctness and performance of SQL not enough*
  - *Users want a generic data platform to run arbitrary, stateful, complex compute*

Motivation

- Lack of DBMS flexibility results in one-off solutions
  - *Get what you need*
  - *Huge ramp-up*
  - *Questionable pay-off*

The No Man's Land



Observation

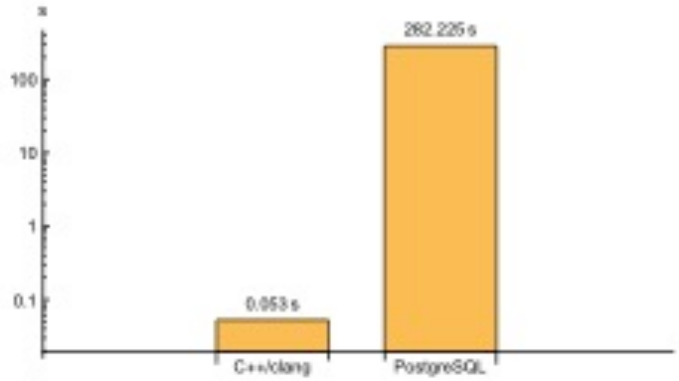
- In memory DBMSs now leverage code-generation for performance
- Compiler researchers study domain specific languages
- Both use dataflow representations internally
  - *For optimization*
- Both have notions of cost-based optimizations

Observation

- In memory DBMSs ... ration for performa...
- Compiler ... language...
- Both use ...
  - *For opti...*
- Both have notions of cost-based optimizations

Can DBMS optimizations be applied directly to a compiler's IR? Is it possible to unify the two systems?

Example



- Linear least-squares classifier with online retraining
  - *C++ and PostgreSQL UDF implementations*
- UDF incurs ~5000x overhead

General Purpose Languages (GPL)

- Not a silver bullet
  - ***Correctness***
  - ***Performance***
  - *Flexibility*
  - *Security*
  - *Heterogenous hardware and portability*
  - *Ease of development*

GPL Correctness

- DBMSs use declarative SQL to express computation
  - *Separation of declarative front-end and execution enables reasoning about correctness*
  - *Correctness defined independently of implementation*
- GPL (and frameworks) hard-code execution plan
  - *Too low-level*
  - *Undecidable correctness*

GPL Performance

- DBMSs have spent decades becoming hardware-conscious
  - *PAX, vectorized execution, pre-fetching, SIMD*
- DBMS knows algorithms, data access, and distribution
  - *Compiler IR is too low-level*
  - *DBMS will generate better code than compiler*

Unifying DBMSs and Compilers

	DMS	Compiler	Lib	HW
Transactional Isolation	✓	✗	✓	✓
Transactional Atomicity	✓	✗	✓	✓
Transactional Consistency	✓	✗	✓	✗
Transactional Durability	✓	✗	✓	✗
Binary ABI	✗	✓	✗	✗
Cost-based Optimization	✓	✗	✗	✗
Indexing	✓	✗	✓	✗
Adaptive Indexing	✓	✗	✗	✗
Runtime Re-Optimization	✓	⊙	✗	✗
Defined memory layout	✗	✓	✗	✗
Data Independence	✓	✗	⊙	✗
Persistence	✓	✗	✓	✓
Declarative Interface	✓	✓	✓	✗
Access Control	✓	✗	✗	✓
Crash Recovery	✓	✗	✗	✗
Explicit State Management	✗	✓	✗	✗
Intermediate Operator State	✗	✓	✓	✗
Fallback Language	✓	✓	✗	✗
Implicit Resource Mgmt.	✓	✗	✓	✗
Explicit Resource Mgmt.	✗	✓	✗	✗
Unit Testing	✗	✗	✓	✗
Pay-as-you-go cloud pricing	✓	✗	✓	✗

Table 1: Comparing (some) Features of Execution Platforms

Challenges To Unification

- Three main challenges:
  - *Model of intermediate state*
  - *Model of computation*
  - *Model of persistence*

Challenge #1 - Intermediate State

- For DBMSs, the state model is a relation

- Compilers have multiple options:
  - *Registers; too small*
  - *Heap; difficult to reason about*
  - *Stack; appears like a reasonable choice*

Challenge #2 – Model of Computation

- DBMSs supported bounded loops, variables, conditions
  - *Suffer with recursive functions*
  - *Unboundedness makes optimization difficult*
- Compilers have sophisticated models
  - *SSA, polyhedral (very difficult and slow), continuation-passing style (CPS)*
- CPS appears like a reasonable choice

Challenge #3 – Model of Persistence

- DBMS has single persistence model
- GPLs/Compilers can use multiple models
  - *Raw disk, third-party libs (i.e., Protobuf, Thrift etc.)*
  - *No consistency guarantees, but possible to generate*

Opportunities For Unification

- State recovery of general programs
  - *Not all data created equal*
  - *Annotate what data should be recoverable by DBMS*
  - *Annotate what data should be recoverable by app. logic*
- Cost as a first class citizen
  - *Need a notion for the overall cost of a program*
  - *Especially important for PaaS providers to cost-estimate generic programs*

Opportunities for Adaptivity

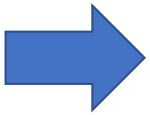
- DBMS adapts between query executions
  - *Gather knowledge about data (e.g., LEO)*
  - *Built indices (e.g., cracking)*
  - *Similar to profile-guided optimization*

- JITs use runtime adaptivity to selectively compile functions

Example: Compression

Ungrouped Aggregation

```
int process(int* input, unsigned long size) {
  int result = 0;
  for(auto i = 0; i < size; i++)
    result += input[i];
  return result;
};
```



Ungrouped Aggregation on RLE input

```
extern int* runs;
extern int* lengths;

int process(int* input, unsigned long size) {
  int result = 0;
  for(auto i = 0ul; i < size; i++)
    result += runs[i] * lengths[i];
  return result;
};
```

- Rewriting aggregation to RLE data is non-trivial
- Don’t operate on source, use LLVM IR
- Take DBMS knowledge, implement into compiler framework

Conclusions

- Databases and compilers have a lot in common
  - *Should perform some knowledge transfer*
- Mastering both will allow applications to evolve without being tied to any one technology
- Requires unification in three areas:
  - *Model of intermediate state*
  - *Model of compute*
  - *Model of persistence*