Stream Processing & Query Compilation

Stream Processing

- Latency vs. Throughput
- Key requirement: resource & cost efficiency
- Runtime reoptimization for minimal scale to meet requirements
- Critical operations:
  - Expensive Windowing Operators
  - Pattern Matching Queries
  - User Defined Functions

Use Cases

- Find applications with lot of data per time unit difficult
- Exception: Large local data (e.g., in cars)
- Semantics of streaming queries not well defined and difficult to optimize
  - Tuple order
  - Out of order tuples
  - Late tuples
- We need use cases from industry!
- Are new Benchmarks necessary?
  - LRB is not timely anymore?
  - Yahoo Benchmark complex enough?
  - New workloads from self-driving cars, monitoring?

Dynamic Reconfiguration

- Scale up vs Scale out?
- Problem: Often need to migrate state, introduces additional latency
- Possible Solution: Decouple compute and state (RDMA)?
- What would we even reconfigure/optimize?
  - Operator ordering
  - Multi-Query operator fusion
  - Migrate state vs. access state remotely

Compilation Approaches

- At which level should we generate code? (C, AST, IR?)
- A code generation library for databases (LLVM is a complex beast ;-))

Research Problems

- Efficiently combining UDFs with code generation is very hard
  - Inlining UDFs with classical relational operators
  - Efficiently evaluating NFAs (complex patterns)
- Stream Processing pipelines often composed of sub-pipelines in different languages
  - No holistic view to optimize for data locality
  - Different runtimes with diverse data models/formats
  - Compiler Theory enables combinations of diverse workloads
  - Database query optimizations and Compiler optimizations are combinable
- Heterogeneous hardware in the whole stack: integrated GPUs, embedded GPUs
  - More diverse processors we need to support

Sharing Artifacts and Knowledge in SPP

- Sharing Code Libraries in SPP?
- Teaching material?
- Data/Queries?
- Shared Bibliography?
- Anything else?