Columnar Execution for Apache Flink Batch

公司：阿里巴巴

职位：计算平台事业部资深专家

演讲者：石春晖

About me

- Senior Staff Engineer in Alibaba MaxCompute
- Committer of Apache Drill
- Interests cover cloud infrastructure, security, big data
- Github ID: chunhui-shi

Content [内容]

- Background and Literature of vectorization [列式执行的背景]
- Vectorized Data Format [列式数据格式]
- Results and Future [效果及未来计划]

Batch SQL in Flink [现在的 batch SQL]

- No cost based optimizer to find physical plan

[没有使用基于cost的优化器来计算物理执行计划]

- Row based [基于行的]
- On top of Dataset API[基于Dataset API]

Motivations

- Two widely used techniques in SQL engines:

  Code-gen & vectorization

  Systems for vectorization: DB2 BLU, columnar SQL Server, VectorWise

  Systems for code-gen: Spark, Peloton, Hyper

- Which is a better design choice?

  The answer is obscured by implementation details: compressions,

  predicate pushdown, parallelism framework, thread model, etc.

Vectorization VS. Code-gen

Everything You Always Wanted to Know About Compiled and Vectorized Queries But Were Afraid to Ask (by Timo Kersten etc. VLDB 2018)

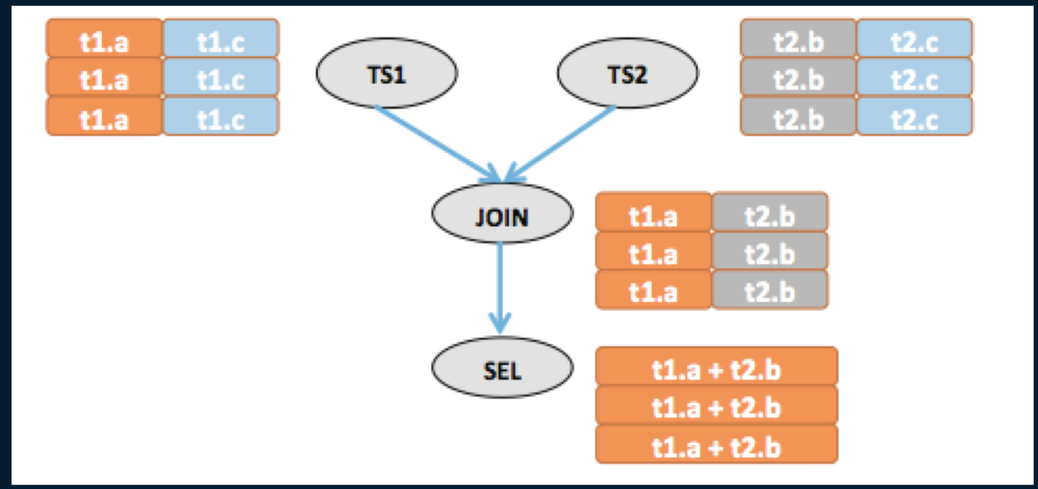
	Vectorization	Code-gen
Passing data between operators	Pull	Push
Data unit of Processing	Batch	Record
Execution code	Interpret	Code-gen
Data representation	Columnar	Row
Pros	Better for Join	
Cons	More instructions, more cache miss	

Vectorization vs. Code-gen: What is missing

- Execution engine is in JVM
- UDF in multiple languages (Python, Java, Javascript)
- Multiple/alternative Execution Engines (GPU, FPGA, native, JVM)
- CPU cache size obeys Moore's law in last ten years (L2: 256KB –> 16MB: 1MB per core, L3: 3MB –> 24MB)

Vectorized Execution in MaxCompute 2.0

- In big data scenarios, the most popular data formats are columnar: parquet, ORC, Carbondata, etc.
- Alibaba develops AliORC and contributes more than 10,000 lines back to community.
- MaxCompute operates on columnar layout since the first time data come into memory.



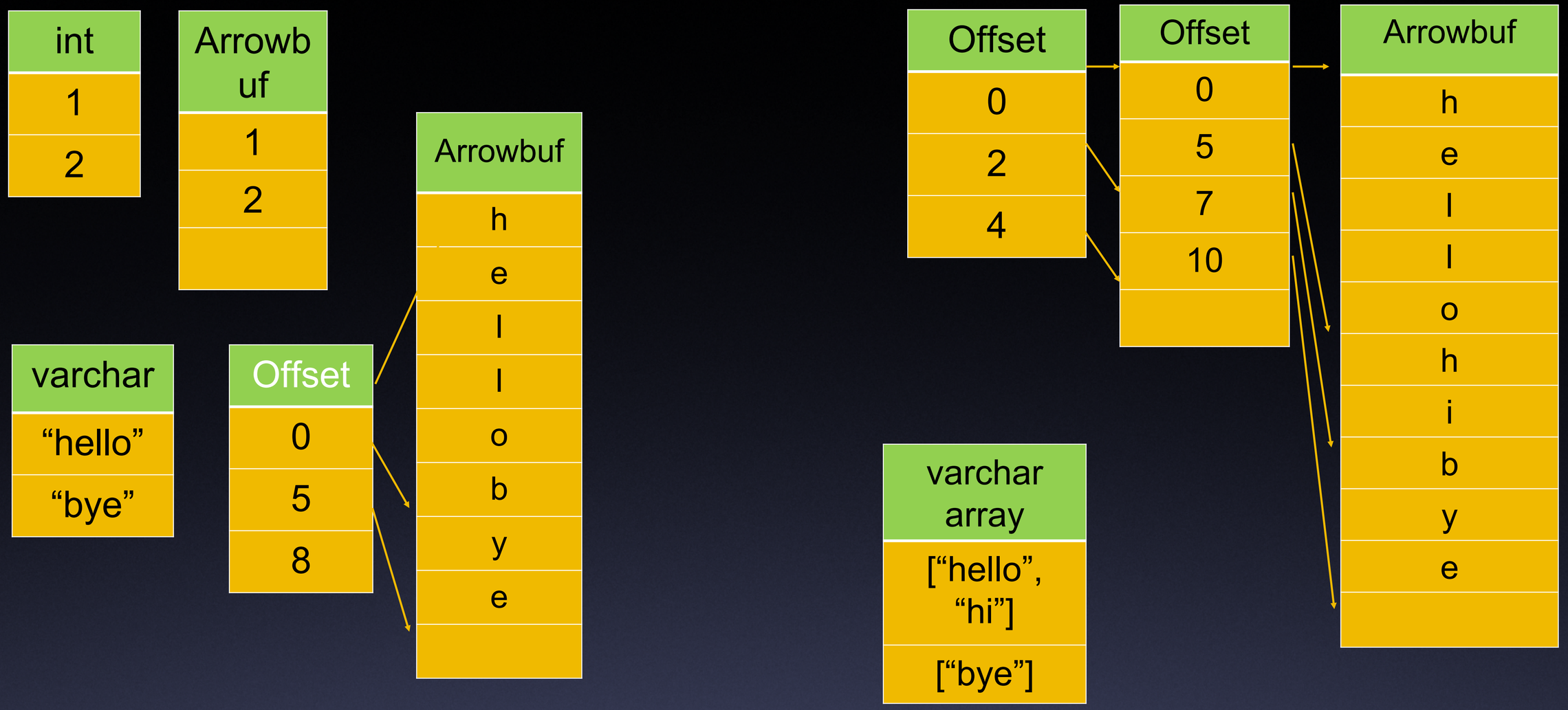
Batch: Flink vs Spark

- Community version is not comparable. Fundamental refactoring is required:
- Compiler: Queries won’t be compiled to Dataset API, but will convert to StreamOperators
- Optimizer: improvement: Apply Calcite to generate physical plan
- Execution Engine improvements: Vectorized layout, batch execution

Vectorized Execution Engine Design

- In memory layout: off-heap vs on-heap
- Extensions to StreamOperator interface
- Vectorized expression Evaluation
- Vectorization specific Improvements

列式内存：ValueVector



Enhancements to Arrow Library

- Support both off-heap and on-heap vectors.
- Optimize buffer access to vectors: remove extra function calls and checks in Arrow library.

Selection Vector Enhancement

- Compact and Sparse Selection Vector

  {1, 7, 23, 1024}

  {1,-1, -1, 4, …}

- Allow selection vector to be passed and used in next operators.

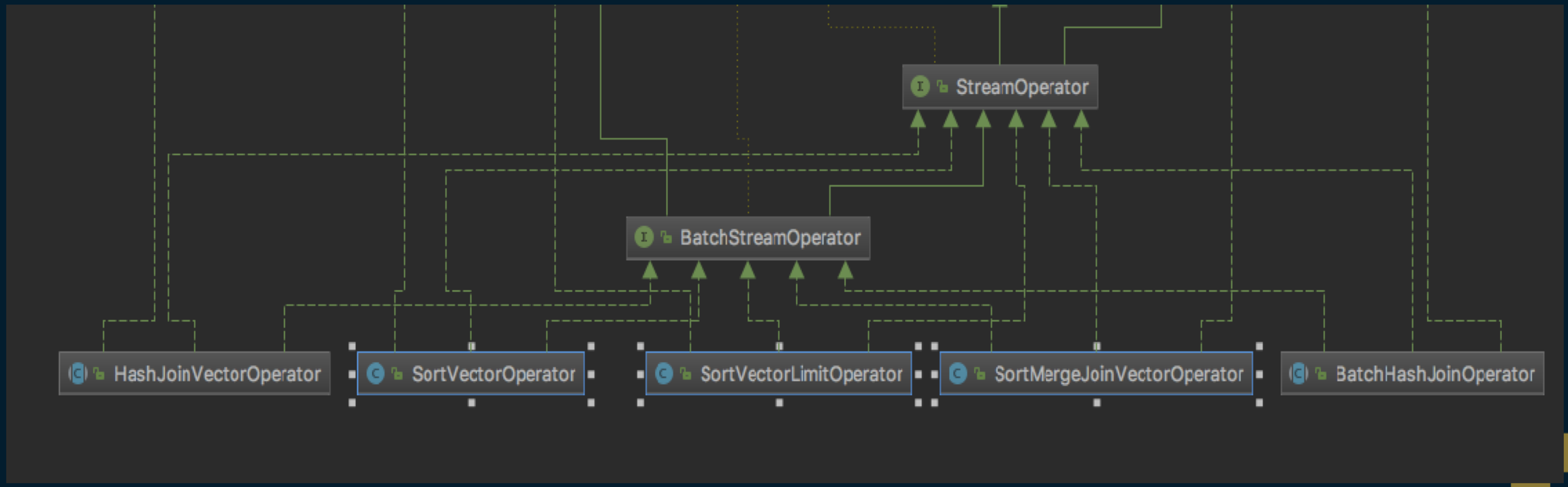
Lazy evaluation of vectorized expression

- Introduce ‘Virtual Vector’ in batch that has not been materialized

  The vector carries only the expression and required vectors.
- Materialize only when needed in serialization or evaluation.

Interface extended to StreamOperator

- Added interfaces, abstract classes to StreamOperator



Performance and resource gain

- TPCH benchmark improvements

  Achieve significant time reduce comparing with the best number a row based batch implementation running on the same cluster.

- Less resource consumption

  Memory, network consumption are saved since the data is transferred and handled in batch.

Future of this vectorized execution engine

- Vectorized and Arrow based data layout makes it easy to interact with multiple languages
- Native implementation of operators and expression evaluation

  - LLVM codegen for expression (Gandiva initiative from Arrow)
- Vectorized UDF framework: Python first!


THANKS