CS 744: BIG DATA SYSTEMS

Shivaram Venkataraman

Fall 2018

ADMINISTRIVIA

- Assignment 2 grades: Tonight
- Midterm review session on Nov 2 at 5pm at 1221 CS
- Course Project Proposal feedback

EFFICIENT SQL ON MODERN HARDWARE

MOTIVATION

Query Model

- Need to handle diverse queries
- Real-time streaming, temporal queries on logs, progressive queries etc.

Language Integration

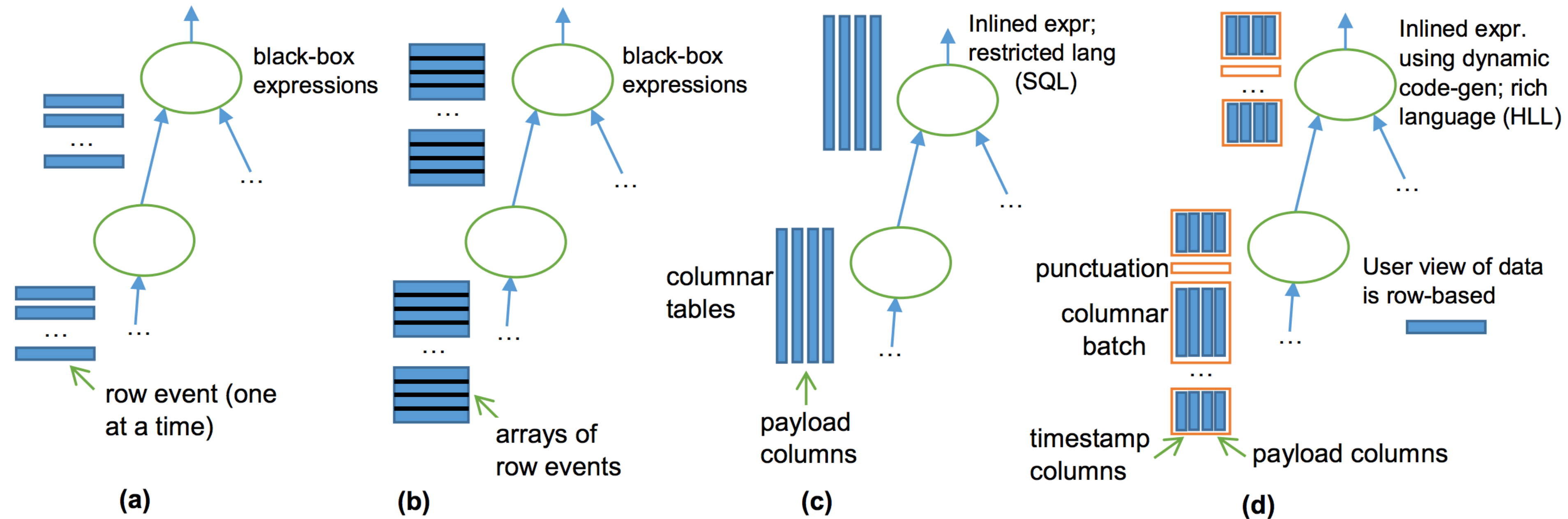
- Support for High-level Language (HLL)
- SQL *Library*

Performance

APPROACH

1. Temporal Logical Data Model
2. DAG of operators (Volcano, Spark, DryadLINQ etc.)
3. Performance
    i. Data batching
    ii. Columnar processing
    iii. Code Generation
    iv. Efficient Aggregation

ARCHITECHTURE



DATA MODEL, QUERY

LINQ style queries

- Similar to SparkSQL, DryadLINQ

Includes *timestamp* by default

Support for windowing, aggregation

```
struct UserData {
    long ClickTime;
    long UserId;
    long AdId;
}
```

```
var str = Network.ToStream(e => e.ClickTime, Latency(10secs));
var query = str.Where(e => e.UserId % 100 < 5)
 .Select(e => { e.AdId })
 .GroupApply(e => e.AdId,
 s => s.Window(5min).Aggregate(w => w.Count()));
```

DATA BATCHING

Why is batching important ?

Vectorized operations, better throughput

Implementing batching

Group a set of events together, each having sync time

Aadaptively choose batch size

Insert punctuation to enforce batch gets flushed

Example: Punctuation every 5min, batch contains 500 tuples

Throughput is 1000 tuples/sec → 600 batches each punctuation

COLUMNAR PROCESSING: LAYOUT

```
class DataBatch {
 long[] SyncTime;
 long[] OtherTime;
 Bitvector BV;
}

class UserData_Gen : DataBatch {
 long[] col_ClickTime;
 long[] col_UserId;
 long[] col_AdId;
 }
```

- Separate into control, payload fields
- BitVector to indicate absence
- Each of these has columnar layout
- Payload generated from user struct

COLUMNAR PROCESSING: OPERATORS

Operators → nodes in query DAG

Chain operators together with On()

Tight-loop from code-gen

Further optimizations:

Copy-on-write,

Zero-copy pointer-swing

```
void On(UserData_Gen batch) {
 batch.BV.MakeWritable();
 for (int i=0;i<batch.Count; i++)
   if ((batch.BV[i] == 0) &&
      !(batch.col_UserId[i] % 100<5))
   batch.BitVector[i] = 1;

 nextOperator.On(batch);
}
```

COLUMNAR PROCESSING: OTHER

Serialization

- Store data in column batches
- Code generation of serialization/deserialization

String Handling

- Bloated string representation in Java/C#
- Encode multiple strings into MultiString
- stringsplit, substring – operate directly on MultiString

GROUPED AGGREGATION

Temporal Data Model

- Each event belongs to a data window or interval
- Aggregates can be stateless or stateful (more in next 3 lectures)

other_time

- When other_time > sync_time, represents interval
- When other_time is infinity, start at sync_time
- When other_time < sync_time, end at sync_time

GROUPED AGGREGATION

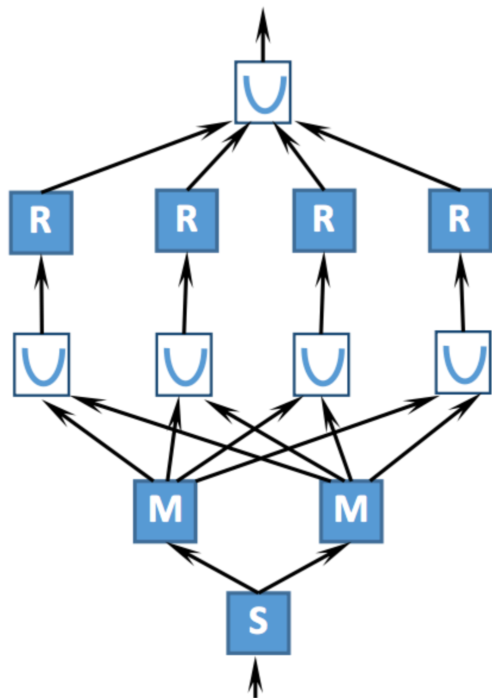
API for user-defined aggregation functions

Efficient implementation using three data structures

Example for count:

```
InitialState: () => 0L
Accumulate: (oldCount, timestamp, input) => oldCount + 1
Deaccumulate: (oldCount, timestamp, input) => oldCount - 1
Difference: (leftCount, rightCount) => leftCount - rightCount
ComputeResult: count => count
```

MAP-REDUCE ON MULTI-CORE



cascading binary merge

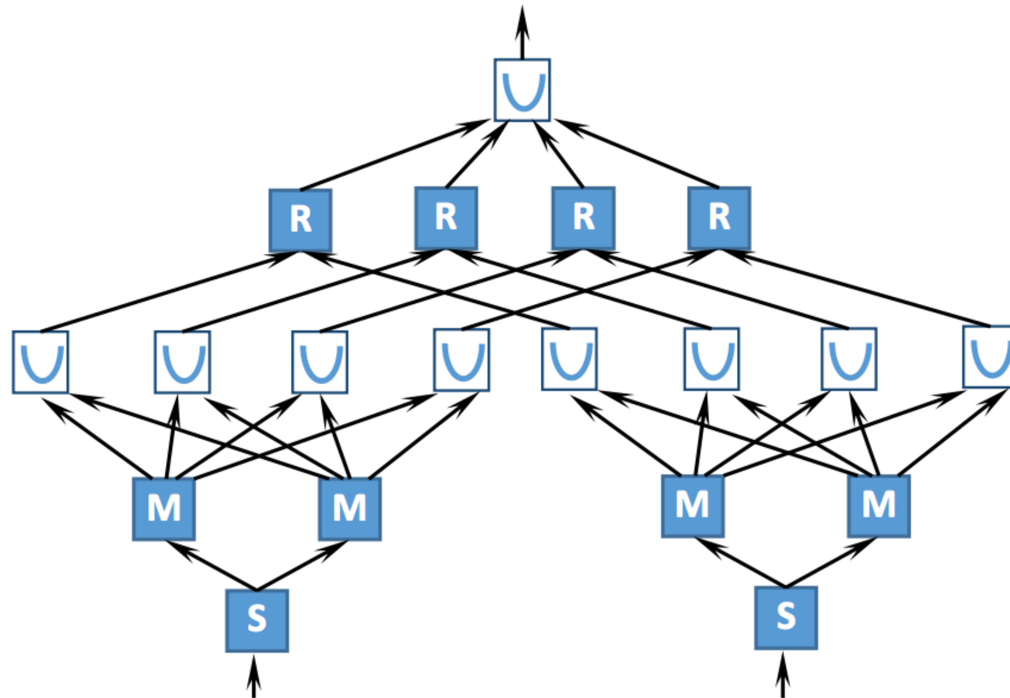
reduce

cascading binary merge

shuffle

map

spray



SUMMARY

Flexible SQL library to handle workload patterns

Integration with high-level language

Efficient execution through

- Batching
- Columnar processing
- Code generation