Microsoft

Trill: A High-Performance Incremental Query Processor for Diverse Analytics

Badrish Chandramouli, Jonathan Goldstein, Mike Barnett, Rob DeLine,
Danyel Fisher, John C. Platt*, James F. Terwilliger, John Wernsing

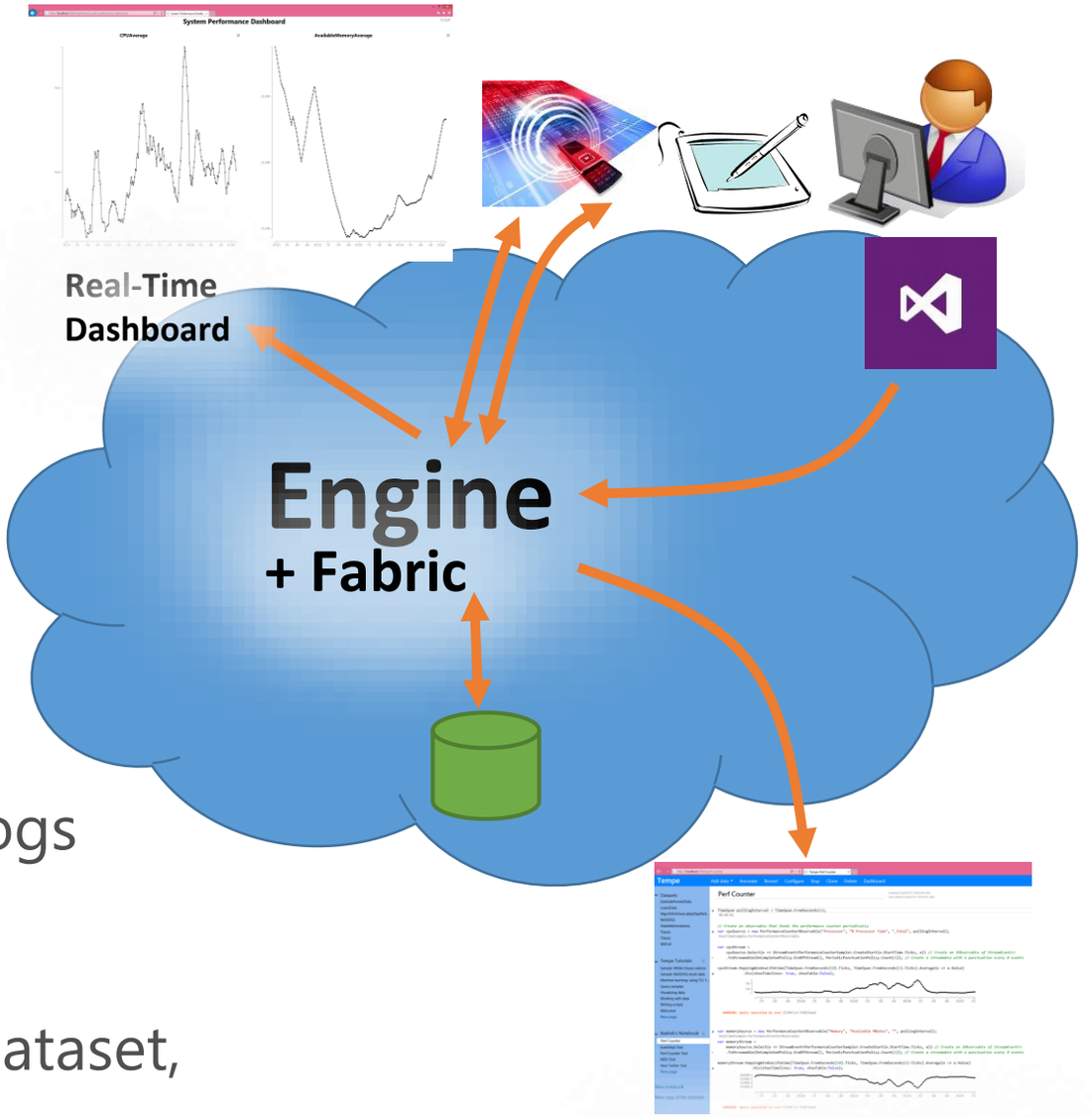
Microsoft Research

Contact: badrishc@microsoft.com Twitter: @badrishc

*Current affiliation: Google. This work was performed at Microsoft Research.

Diverse Scenarios for Analytics

- Real-time
  - Monitor app telemetry (e.g., ad clicks) & raise alerts when problems are detected
- Real-time with historical
  - Correlate live data stream with historical activity (e.g., from 1 week back)
- Offline
  - Develop initial monitoring query using logs
  - Back-test monitoring query over historical logs
- Progressive
  - Non-temporal analysis (e.g., BI) over large dataset, stream data, get quick approximate results



Three Key Requirements

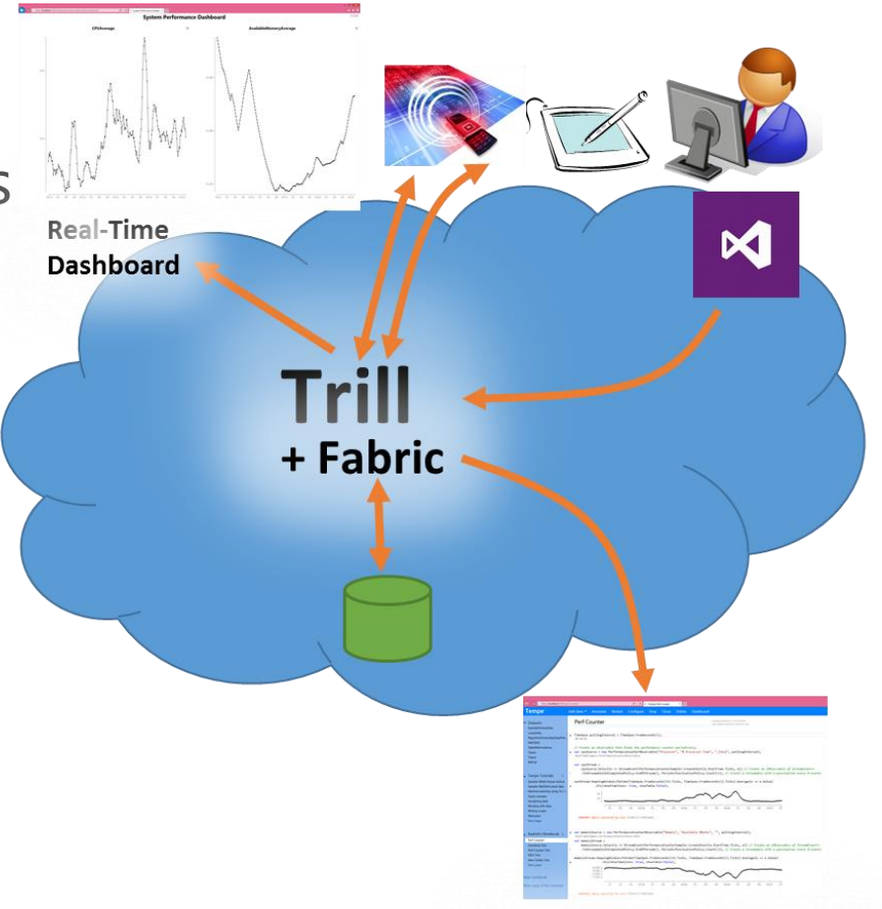
- Performance
  - High throughput: critical for large offline datasets
  - Low latency & overhead: Important for real time monitoring
- Fabric & language integration
  - Cloud app/service acts as driver, *uses* the analytics engine
  - Need rich data-types, integrate custom logic seamlessly
- Query model
  - Need to support real-time and offline data, temporal and relational queries, early results for exploratory offline queries

Scenarios

- monitor telemetry & raise alerts
- correlate real-time with logs
- develop initial monitoring query
- back-test over historical logs
- offline analysis (BI) with early results

Trill: Fast Streaming Analytics Library

- Performance
  - 2-4 **orders of magnitude** faster than traditional SPEs
  - For relational queries, comparable to best DBMS
  - User-controlled latency specification
    - explicit latency vs. throughput tradeoff
- Fabric & language integration
  - Built as high-level language (HLL) library component
  - Works with arbitrary HLL data-types & libraries
- Query model
  - Extended LINQ syntax based on tempo-relational query model
  - Supports broad & rich analytics scenarios (relational, progressive, time-based)



Trill's Use Cases

- Azure Stream Analytics Cloud service
- With Scope for Bing Ads
- With Orleans for Halo game monitoring & debugging
- …

- Key enabler: performance + fabric & language integration + query model

Example (simplified)

- Define event data-type in C#

```
struct ClickEvent { long ClickTime; long User; long AdId; }
```

- Define ingress

```
var str = Network.ToStream(e => e.ClickTime, Latency(10secs));
```

Application time

Latency specification

- Write query (in C# app)

```
var query =
 str.Where(e => e.User % 100 < 5)
    .Select(e => { e.AdId })
    .GroupApply(  e => e.AdId,
                  s => s.Window(5min).Aggregate(w => w.Count()));
```

Lambda expression

Grouping key selector

Operator

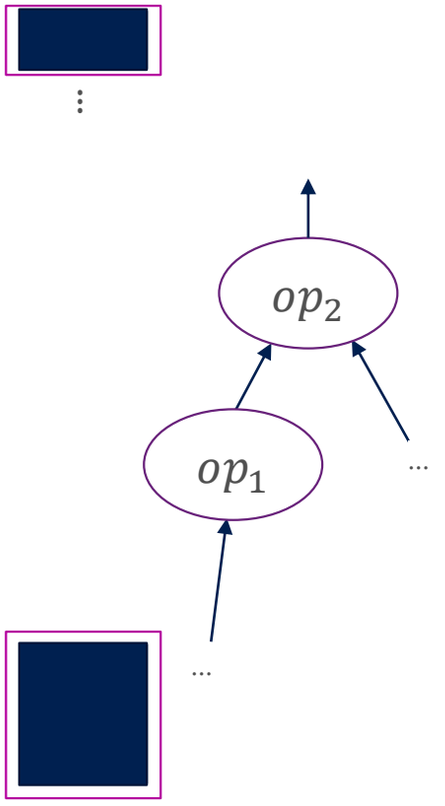
Grouped sub-query

- Subscribe to results

```
query.Subscribe(e => Console.Write(e)); // write results to console
```

Latency-Throughput Spectrum

- Data organized as stream of batches
  - Purely physical (no impact on query results)



- Users specify latency constraint (10 secs)
  - Batch up to 10 secs of data
  - Small batches → low latency
  - Large batches → high throughput
  - More load → larger batches → better throughput

+ Columnar

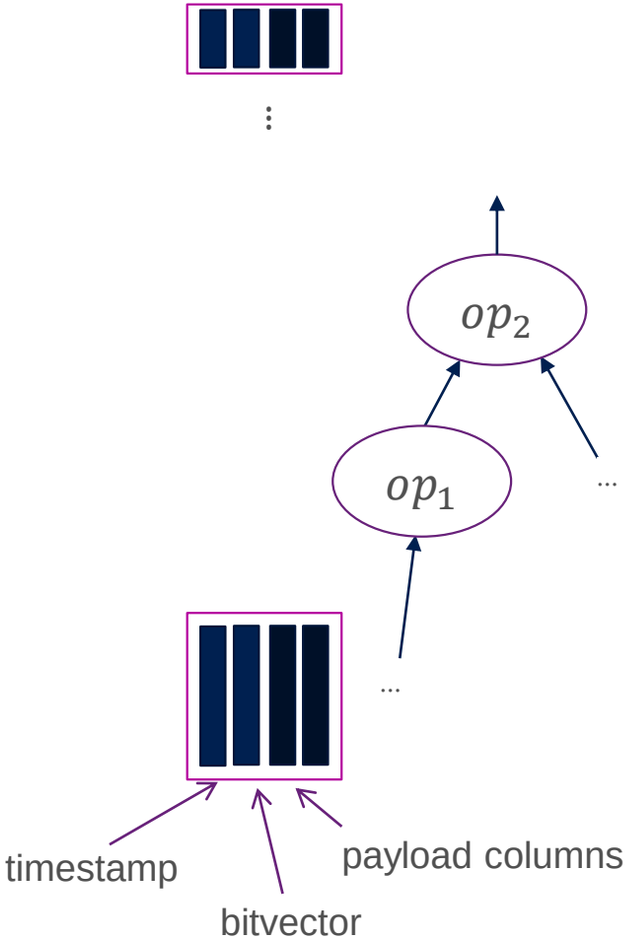
- Columnar format within each batch
  - Timestamps as arrays
  - Bitvector to indicate row absence

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

  - One array per payload field

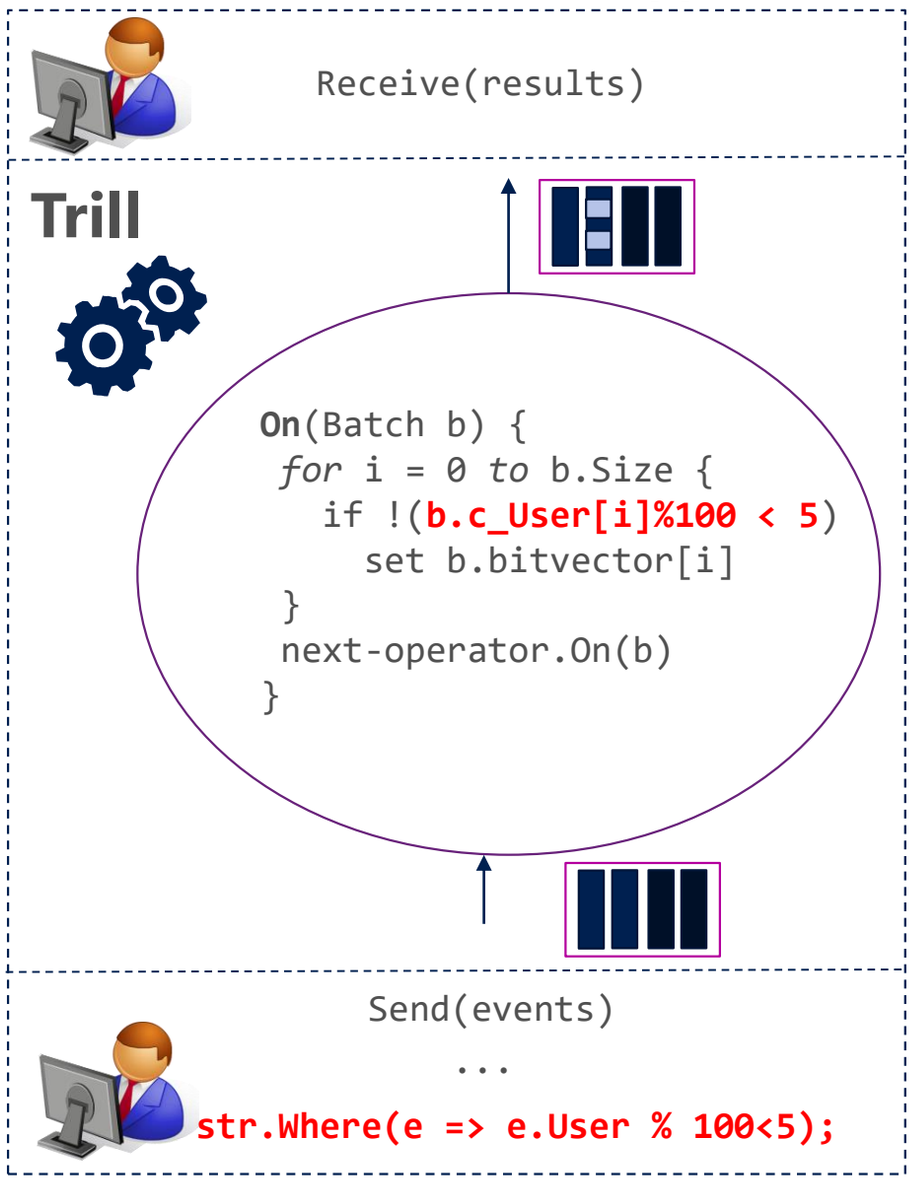
```
class UserData_Gen : DataBatch {
        long[] c_ClickTime;
        long[] c_User;
        long[] c_AdId;
}
```

  - Batch classes are generated & compiled on-the-fly (under the hood)
  - Enables efficient QP & serialization



+ Fabric & Language Integration

- But, user view is row-oriented
  - Dynamically generate and compile HLL code for operators
- Our example: filter (where)
  - User writes `str.Where(e => e.User % 100 < 5)`
- General technique
  - Generate tight loops over batches, with inlined expressions (done using reflection)
  - Avoid method calls within loops
  - Timestamps are columns – accessed only if needed
    - Pointer-swing where possible

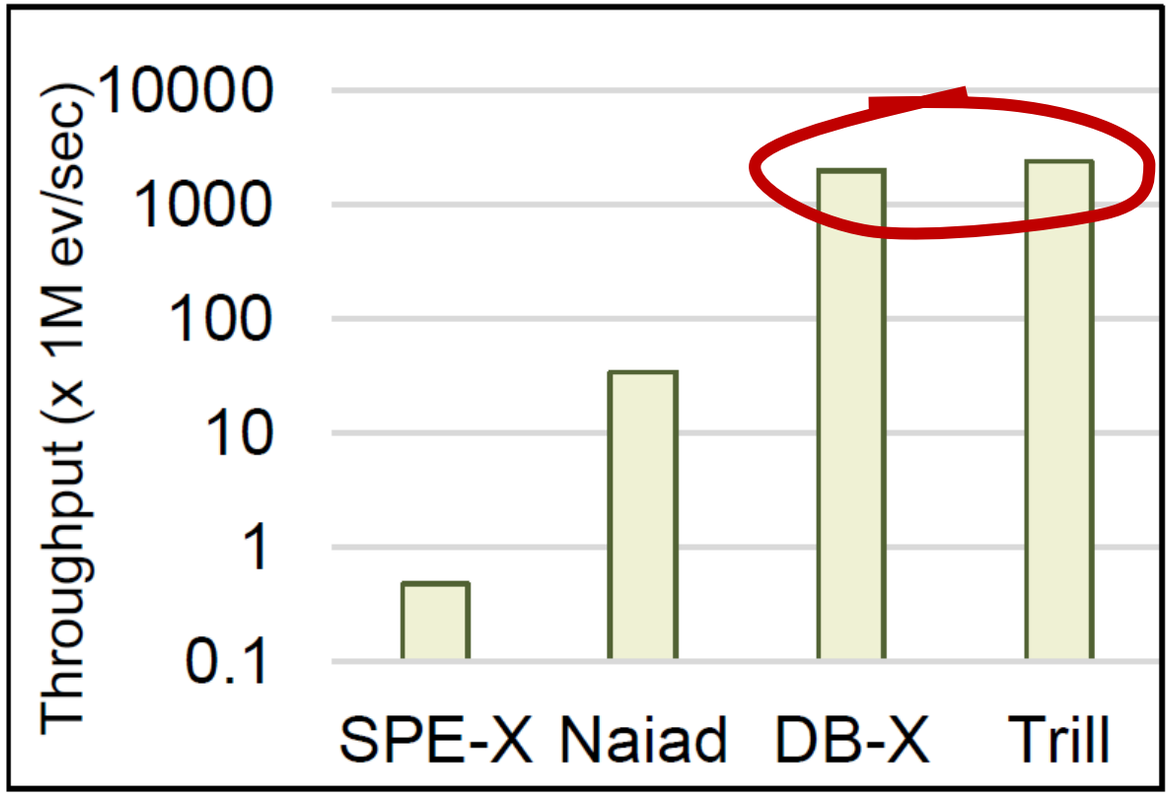


See Paper …

- Property-based operator codegen specialization
- Grouped & batched operator algorithms
- Library-mode & multi-core scheduler
- Efficient serialization & string support
- Rich query language
  - SQL queries with progressive (early) results
  - Temporal queries
    - Sliding, hopping, tumbling, data-dependent session windows
    - Temporal joins, set difference
    - Powerful high-perf expression-based user-defined aggregate framework

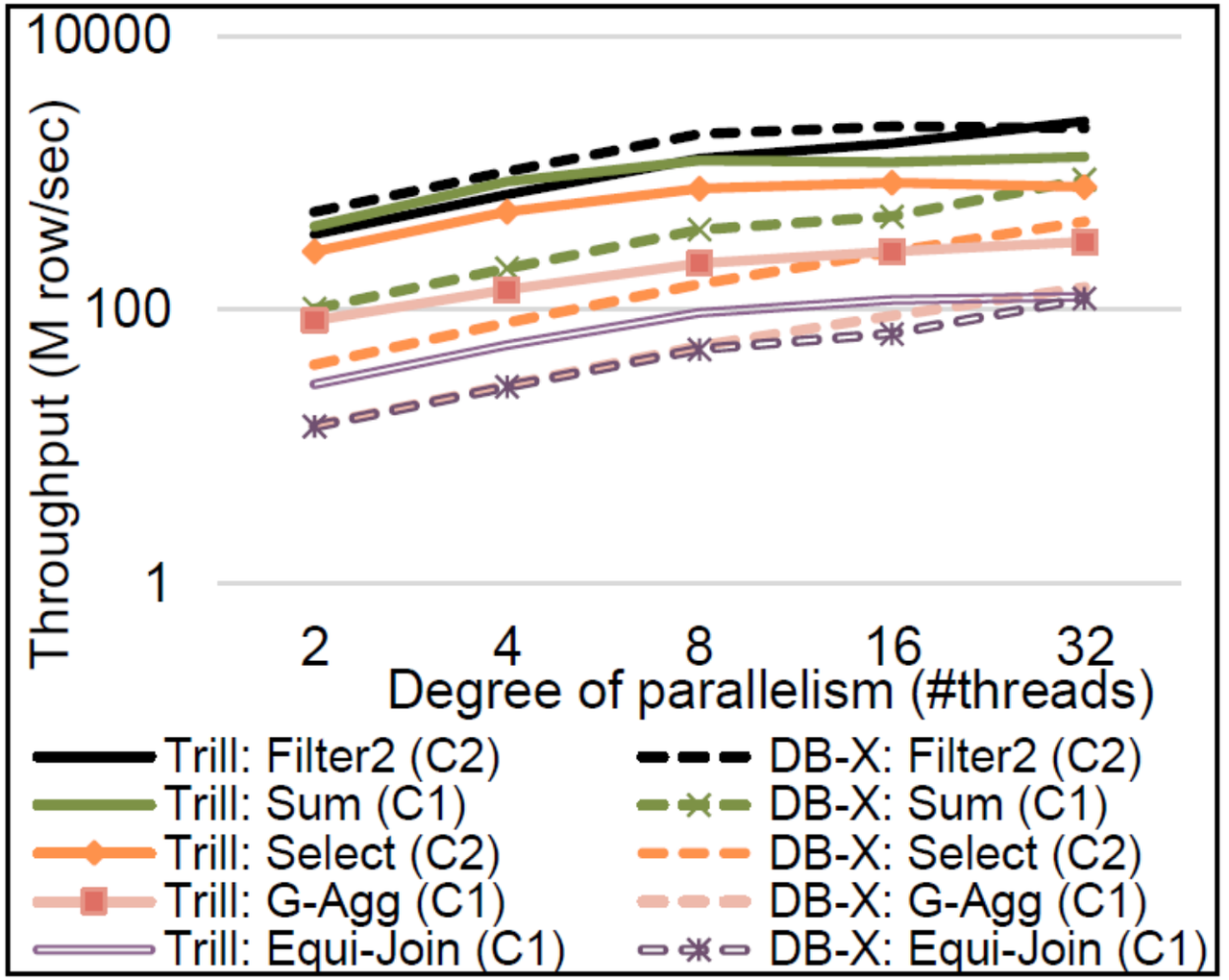
Evaluation (sample)

- Pre-loaded datasets in main memory
- 16-core machine
- Streaming filter



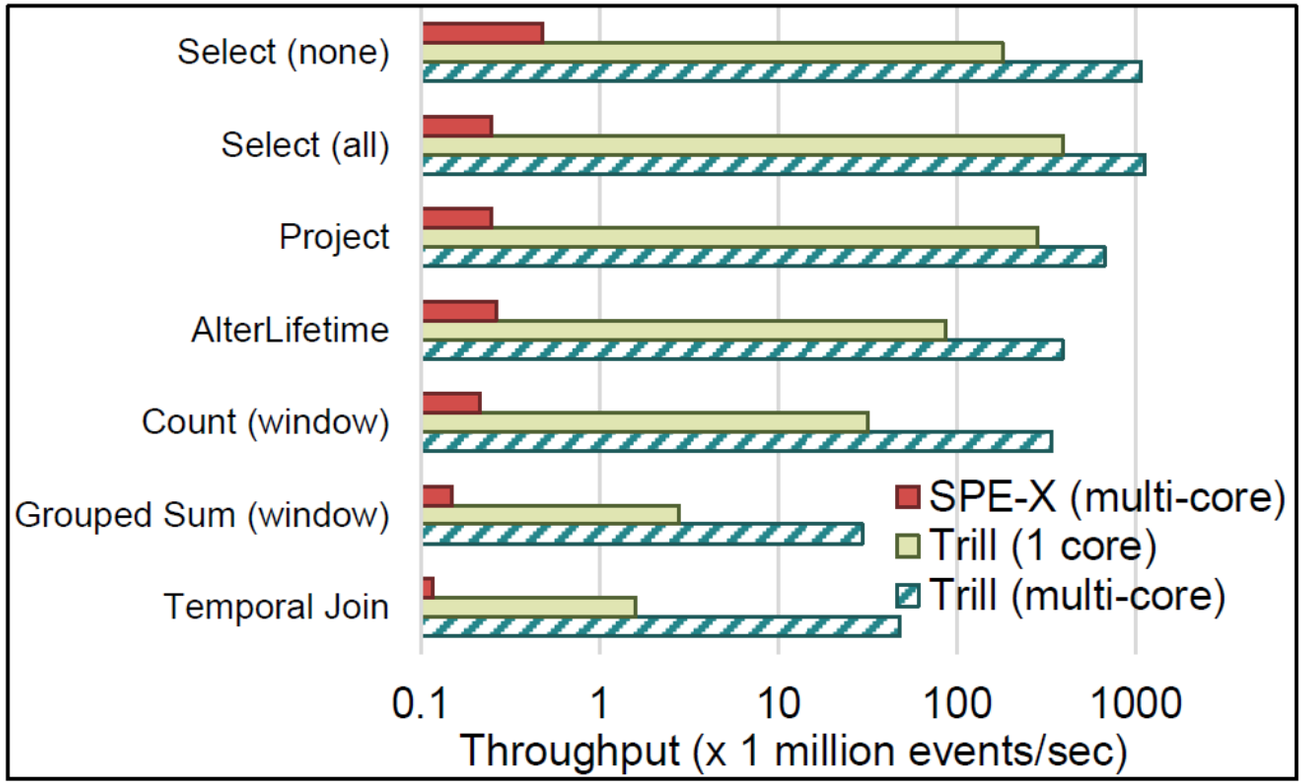
Evaluation (sample)

- Pre-loaded datasets in main memory
- 16-core machine
- Relational queries



Evaluation (sample)

- Pre-loaded datasets in main memory
- 16-core machine
- Temporal queries



Conclusions

- Trill is a fast & expressive library for analytics
  - 2-4 orders-of-magnitude faster than traditional streaming engines
  - Comparable to columnar databases for offline SQL queries
    - But with progressive (early result) support
  - Expressive query language and support for HLL type-system & code
  - Library that can be easily embedded in a variety of settings (distributed fabrics, servers, Cloud applications, devices, …)
- Trill is being used across diverse analytics scenarios
- More info @ http://aka.ms/trill