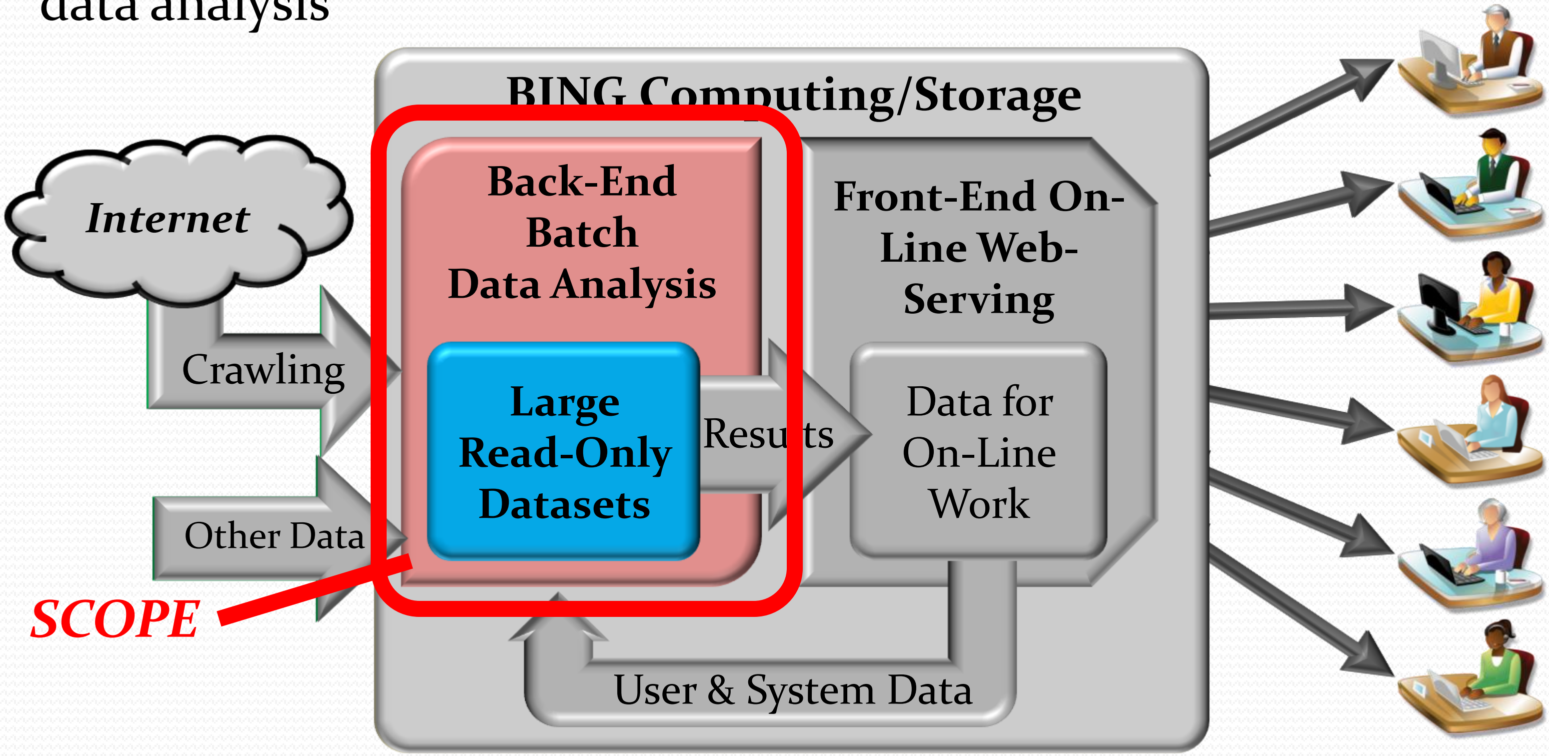
SCOPE: Query Processing in Large Data Centers

Jingren Zhou

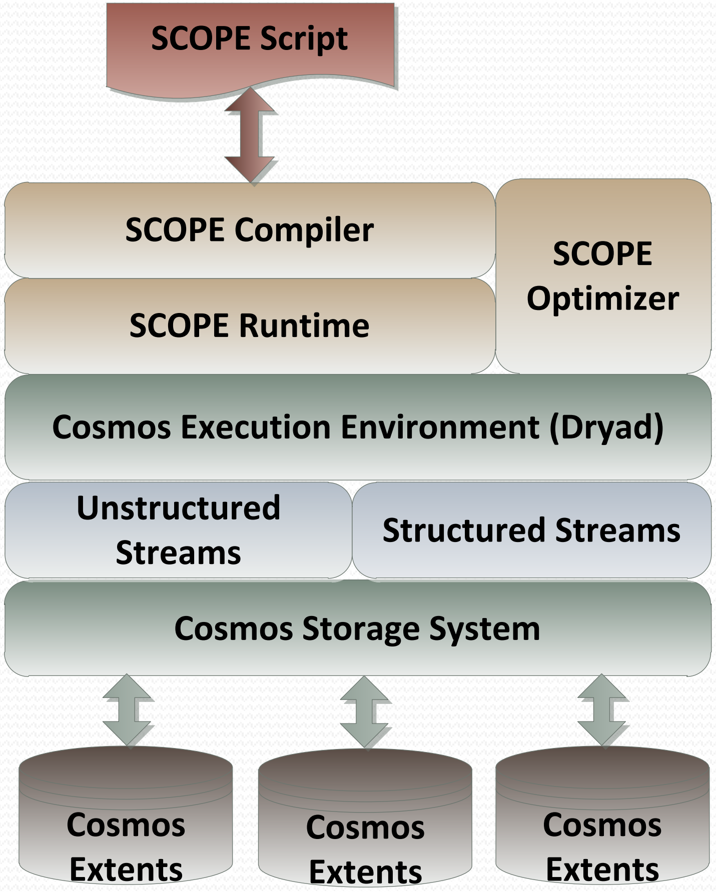
Microsoft Corp.

Microsoft Bing Infrastructure

- BING applications fall into two broad categories:
  - **Back-end**: Massive batch processing creates new datasets
  - **Front-end**: Online request processing serves up and captures information
- SCOPE/Cosmos provides storage and computation for Back-End Batch data analysis



SCOPE / Cosmos



- A hybrid of parallel database and MapReduce system
- SCOPE
  - A SQL-like declarative language
  - Fully integrated with .NET framework
  - Highly extensible and flexible
- Cosmos Storage System
  - Append-only distributed file system for storing petabytes of data
  - Optimized for sequential I/O
  - Data is compressed and replicated
- Data comes in two formats
  - Unstructured streams
  - Structured streams

SCOPE (VLDB'08)

- ***S***tructured ***C***omputations ***O***ptimized for ***P***arallel ***E***xecution
- A declarative scripting language
  - Easy to use: SQL-like syntax plus MapRecuce-like extensions
  - Modular: provides a rich class of runtime operators
  - Highly extensible:
    - Fully integrated with .NET framework
    - Provides interfaces for customized operations
  - Flexible programming style: nested expressions or a series of simple transformations
- Users focus on problem solving as if on a single machine
  - System complexity and parallelism are hidden

An Example: QCount

Compute the popular queries that have been requested at least 1000 times

Scenario 1:

```
SELECT query, COUNT(*) AS count
FROM “search.log” USING LogExtractor
GROUP BY query
HAVING count> 1000
ORDER BY count DESC;

OUTPUT TO “qcount.result”
```

Scenario 2:

```
e = EXTRACT query
    FROM “search.log” USING LogExtractor;

s1 = SELECT query, COUNT(*) AS count
      FROM e GROUP BY query;

s2 = SELECT query, count
      FROM s1 WHERE count> 1000;

s3 = SELECT query, count
      FROM s2 ORDER BY count DESC;

OUTPUT s3 TO “qcount.result”
```

SCOPE Optimizer (ICDE’10)

- A transformation-based optimizer based on the Cascade framework
- Reasons about a rich set of logical/physical operators
- Employs traditional database optimization techniques
- Chooses an optimal plan based on cost estimates
- ***Goals***:
    - *Seamless generate both serial and parallel plans*
    - *Reasons about partitioning, sorting, grouping properties in a single **uniform** framework*

SCOPE Execution

- SCOPE Runtime
  - Provides a rich class of composable physical operators
  - Operators are implemented using the iterator model
  - Executes a series of operators in a pipelined fashion
- A SCOPE query plan
  - A DAG of SCOPE vertices
  - Each vertex consists of a serial of runtime operators
  - It relies on the job manager to schedule vertices at runtime

Structured Streams

- Structured streams have well-defined schema
  - Data is transparently partitioned
  - Local index on each partition is maintained
- Structured streams offer many performance benefits
  - Rich structural properties for optimization
    - Avoid unnecessary partitioning, sorting, etc.
    - Rich data access methods (through local index)
  - Column-wise optimization
  - Dynamic management of partitions
    - Automatically deal with data skewness and adapt to changing data distribution
  - Efficient and flexible physical design

Conclusions

- SCOPE/Cosmos is a hybrid system of MapReduce and traditional parallel database
  - Extensively used in cloud-scale data centers at Microsoft Bing
  - Optimization greatly improves query performance
    - Systematically reasons about structural properties (partitioning, grouping, and sorting), functional dependencies, and their interactions
    - Seamlessly integrates optimization of both serial and parallel plans into a single uniform framework