# Domino: An Incremental Computing Framework in Cloud with Eventual Synchronization

Dong Dai, Xuehai Zhou, Dries Kimpe, Rob Ross, Yong Chen
Texas Tech University
University of Science and Technology of China
Argonne National Laboratory

# Outline

- Motivation
- Domino Model
- Design and Implementation
- Evaluations
- Q & A

# Motivation

- Continuous data streams are more and more important today.
- To process these streaming data, event-driven models have emerged, including Percolator, Oolong, etc.
- Event-Driven tasks are distributed into multiple servers and run in parallel.
- **How to synchronize them becomes a big challenge**

# Motivation - Cont.

- Multi-iteration trigger-based applications requires synchronization between iterations
- We show two typical strategies to synchronize
  - Fig.(a): Sync needs to wait all triggers to finish. In streaming case, this is not possible as we do not know when previous triggers will finish.
  - Fig.(b): Each sync waits for a time window to start. This may introduce lots of unnecessary syncs.
- Our goal:
  - ***trigger-based programming model with flexible sync mechanism for streaming data to avoid these limitations***


(a) Wait-based synchronization




(b) TimeWindow-based synchronization

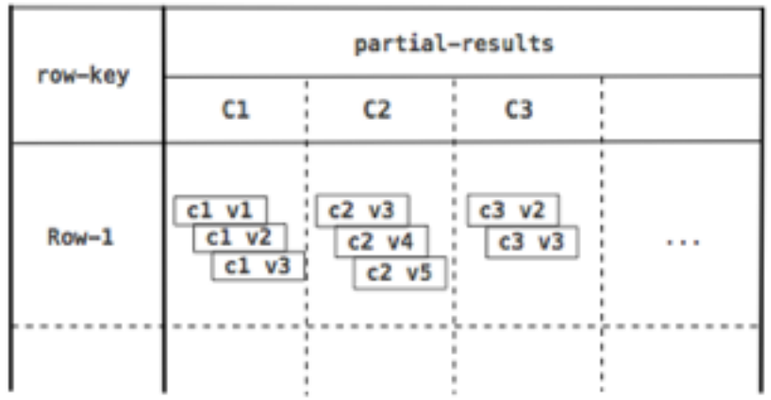
# Domino Model

- Sparse table
  - Data streams are viewed as *insert* or *update* on table
  - Similar with Bigtable with multi-columns in different column families
  - Each data cell stores multi-version data
- Programming model
  - **Event**: Generated from the data modifications; Applications need to declare which columns, column-families, or table they are monitoring to detect the events.
  - **Condition**: Filter events to control the execution of triggers. It is user-defined function that returns true or false to denote whether current event should be processed or not.
  - **Action**: The real logic of application. It consumes the events and write the results back into sparse table persistently. Actions always run locally.
- **Event + Condition + Action => A Trigger**

# Trigger Types

- ***Plain trigger***
  - The simplest case. It responses to new data streams and executes independently in different servers in parallel.
- ***Async accumulator trigger***
  - Accumulate partial results from other triggers in an async way.
  - Partial results arrive and activate the execution of accumulator triggers without any coordination.
- ***Sync accumulator trigger***
  - Similar with Async accumulator trigger
  - Provide a self-managed eventual synchronization mechanism between partial results.
  - Avoid global blocking and make progress all the time.

# Accumulator, Sync




- **Accumulator trigger**
  - For each accumulator trigger, Domino implicitly creates a table.
  - The *partial-results* is automatically monitored by accumulator trigger
  - Previous results will be written into this table under the same column-family and trigger actions
  - For the sync accumulator, Domino will use version management to guarantee the right results
- **Version Management**
  - The sparse table stores multi-version data in each cell, which allows us to trace the execution status of each data
  - We introduce two ids to trace the data: one represents the iteration round; one indicates the external data version.
  - The actions also have version information inherited from those two ids.
  - Whenever an action is triggered, it only can access the data with lower version number comparing the version number of itself.

# PageRank Example

- Sparse table *WebRepo* stores the web repository.
- Application has two triggers:
  - ***Plain trigger***.
    - Monitor the *Meta* column-family (*Event*);
    - If the changes are large enough (*Condition*);
    - Calculate all the out-edges' weights (*Action*);
    - Write the out-edges' weights to accumulator trigger (*Key=edge.dst, Value=weights)*
  - ***Sync accumulator trigger***.
    - Monitor the implicit table, which stores the *urls* and the incoming edges' weights (*Event*);
    - Accumulate all weights for each *url* to form a new page rank value (*Action*)
    - Write new page rank value to *Meta:Rank* column to activate next round.

**Table 1: WebRepo Table**

| RowKey | *Meta:Rank* | *Meta:OutEdges* | *Cot:En* |
|---|---|---|---|
| $url_1$ | 1.0 | $url_{11}, url_{12}, ...$ | ... |
| $url_2$ | 1.0 | $url_{112}, url_{21}, ...$ | ... |
| ... | ... | ... , ... | ... |

**Table 2: Accumulation Table**

| *RowKey* | *Partial-results:c1* | *Partial-results:c1* | ... |
|---|---|---|---|
| $p_1$ | 0.1 | 0.15 | ... |
| $p_2$ | 0.25 | 0.32 | ... |
| ... | ... | ... , ... | ... |

# Implementation

- Current tightly integrated with HBase,
    - Can work on any storage layers that support *spare table* and *multi-version persistent storage*.
- Event detector
    - Detects the updates on spare table
    - Intercept the core execution path of WAL (write-ahead-log) appending in HBase.
- Gathered I/O
    - Encapsulates all the data accesses in action function into a delegate to accelerate I/O operations.

# Evaluations

- 12-node local cluster and 64-node EC2
- Word-count and PageRank
- Compare with MapReduce, Mimic Percolator, Oolong.

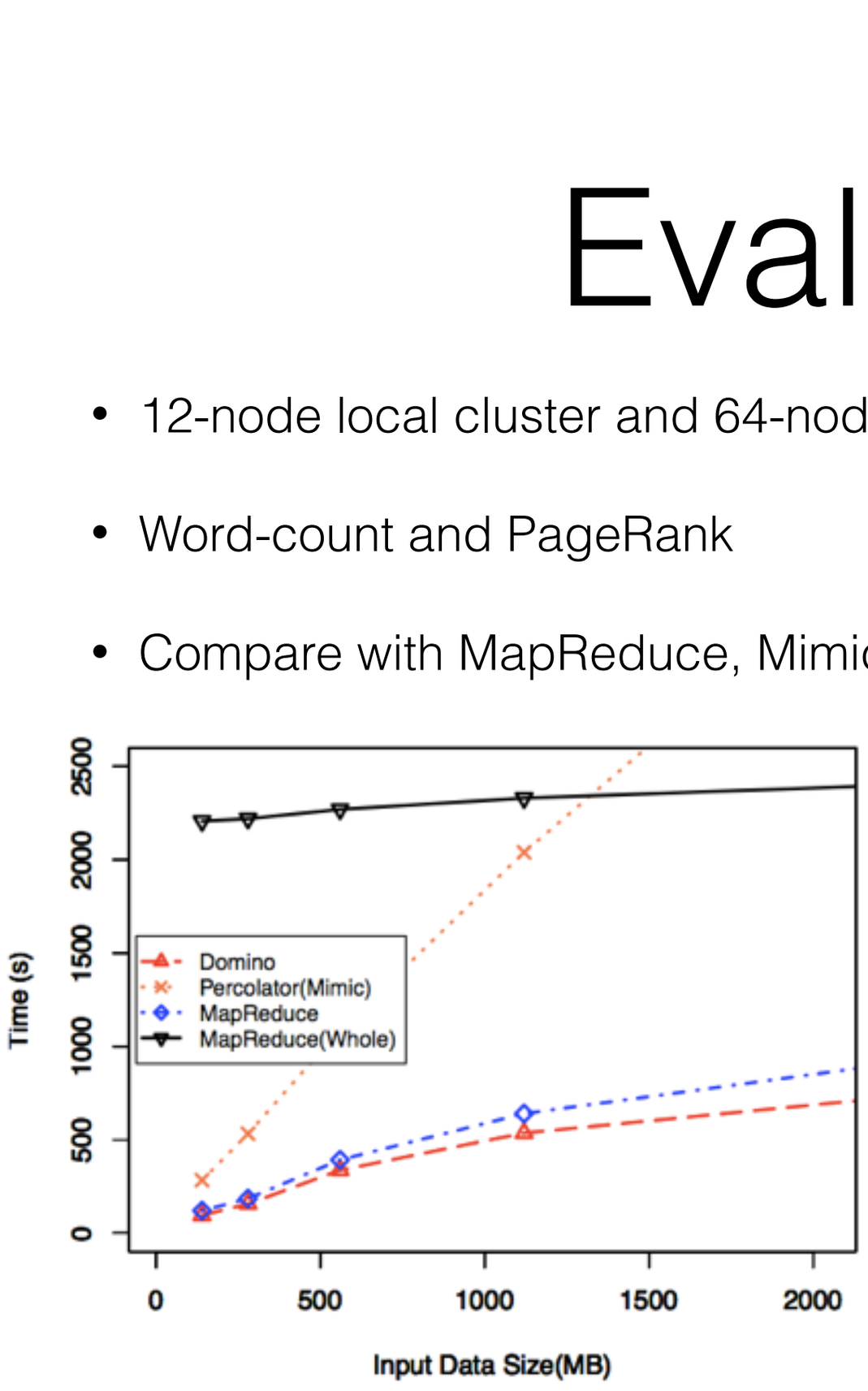



**Figure 3: WordCount under different sizes.**

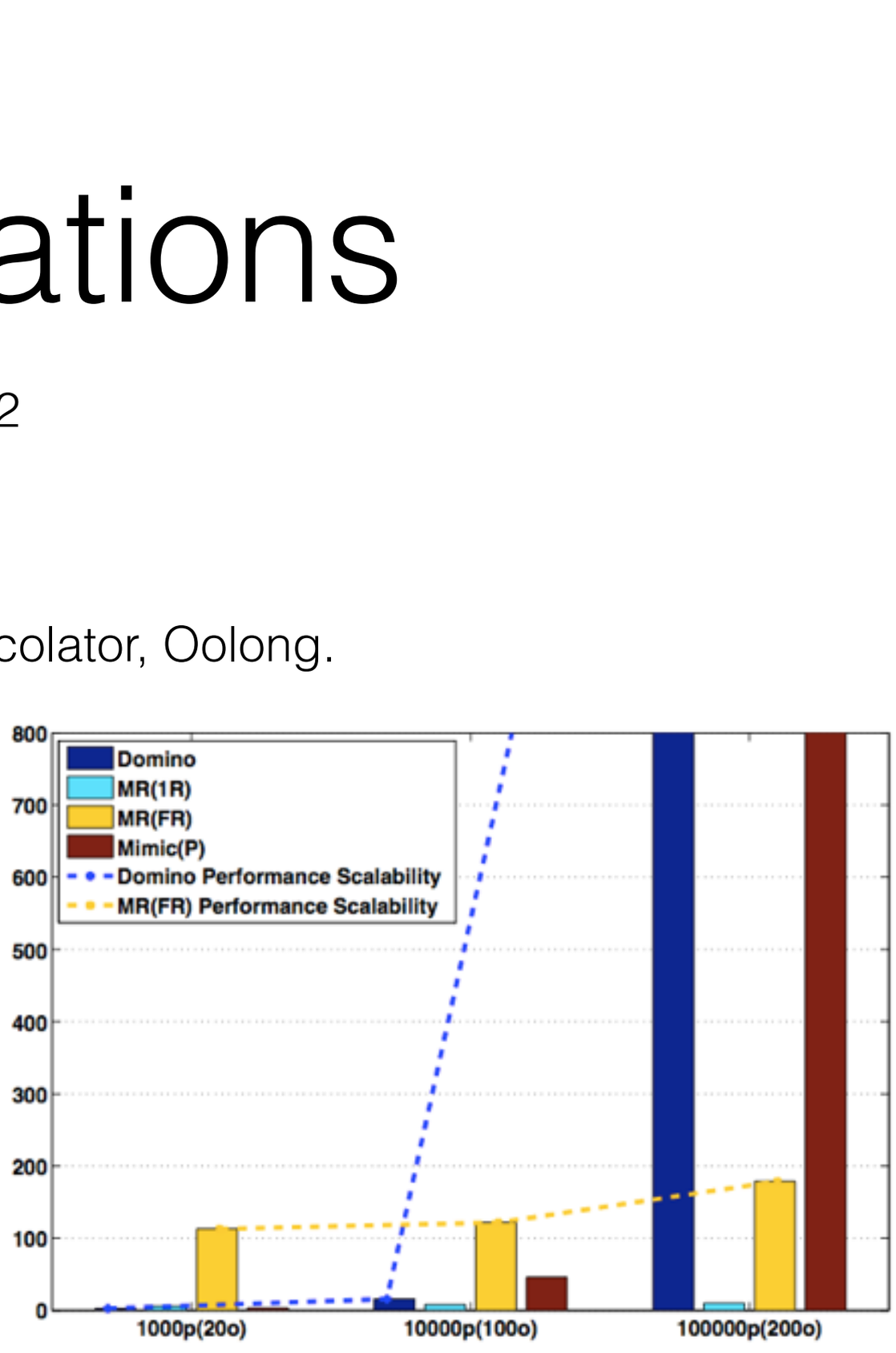



**Figure 4: PageRank under different changing sets.**

# Thanks! Q & A