Asynchronous and Fault-Tolerant Recursive Datalog Evaluation in Shared-Nothing Engines

Jingjing Wang,
Magdalena Balazinska,
Daniel Halperin

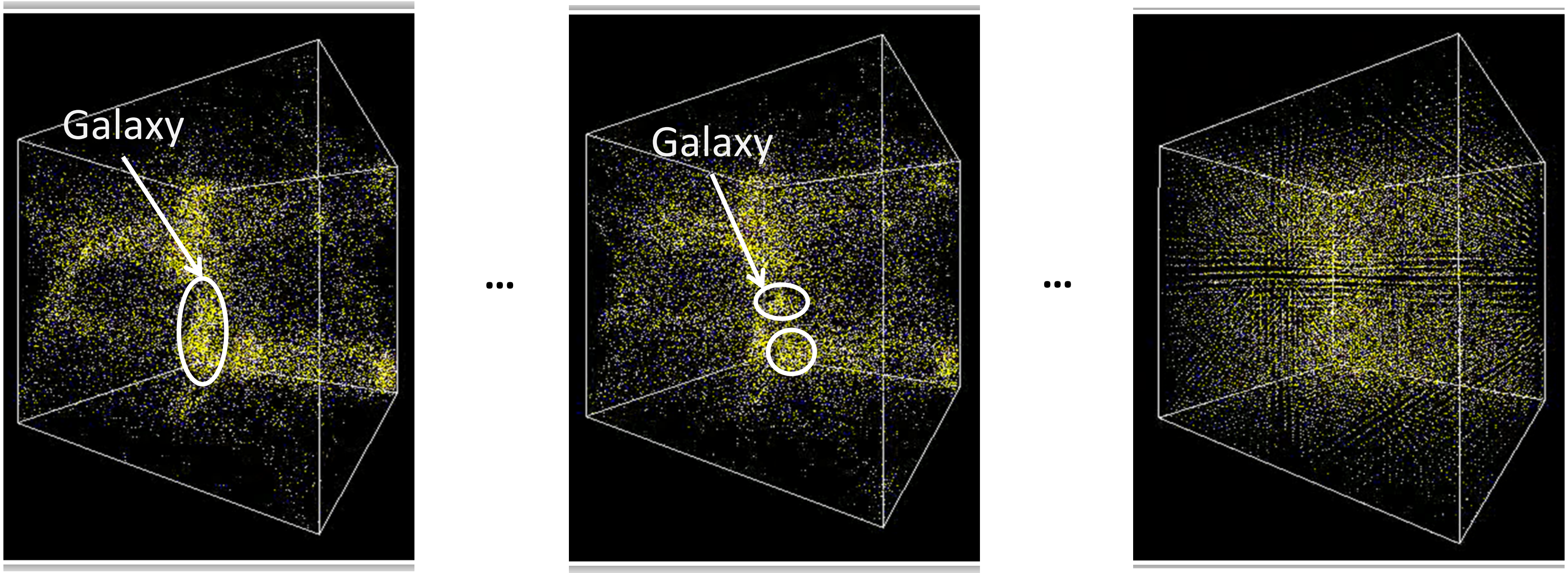
University of Washington

Modern Analytics Requires Iteration

- Graph applications
  - Graph reachability
  - Connected components
  - Shortest Path
- Machine learning
  - Clustering algorithms
  - Logistic regression
- Scientific analytics
  - N-body simulation
- …

Galaxy Evolution: An Iterative Example

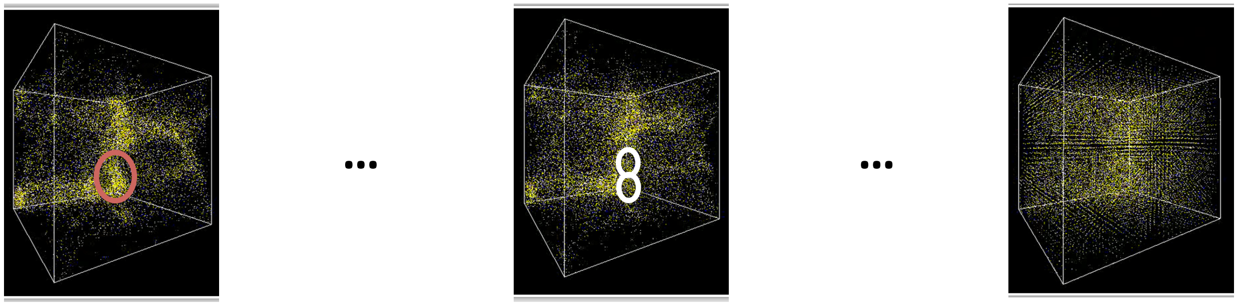
A Simulation of the Universe

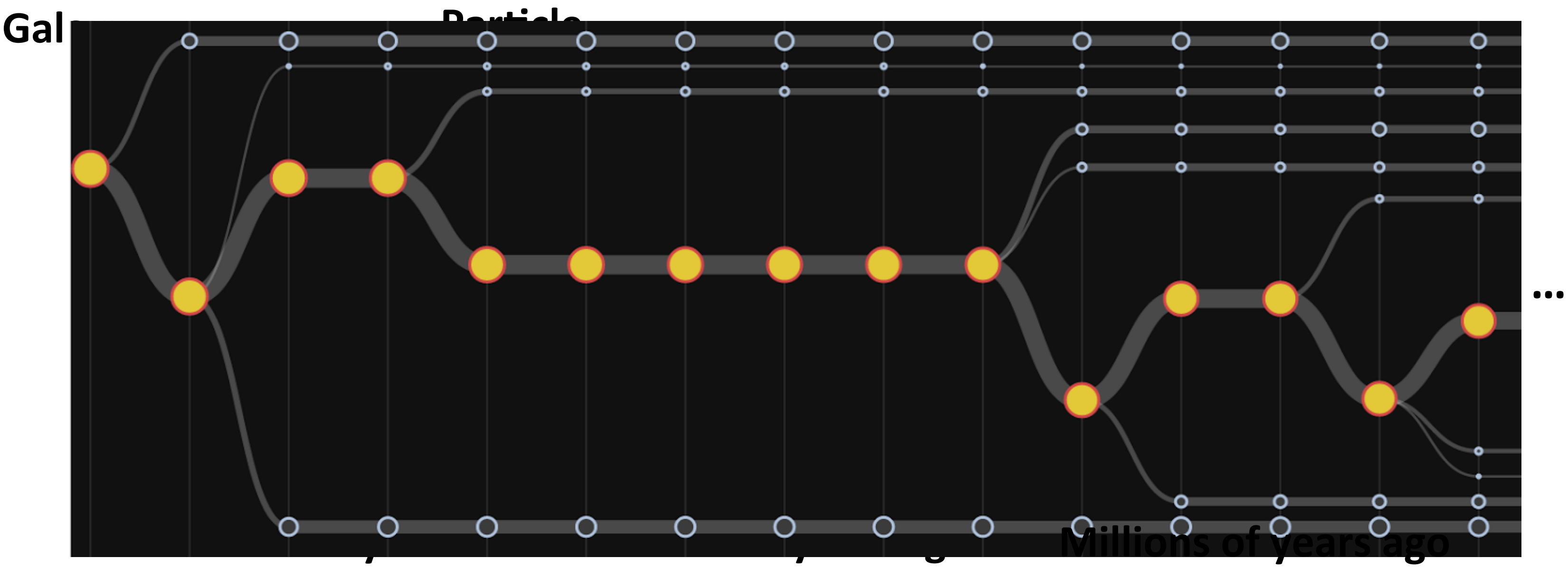


Picture from D. H. Stalder et. al. arXiv:1208.3444 [astro-ph.CO]

Present day **Millions of years ago** **Big Bang**

Galaxy Evolution: Iterative Lineage Tracing





Galaxy Evolution: Why It Is not Easy

- Large-scale data sizes
  - Scalability
- Iterative is the core
  - Support efficient iterative constructs
- Users are data scientists
  - Provide an easy-to-use query interface
- Shared datasets and resources
  - Within a data management system

Iterative Analytics: Where to Do

- SQL Server
  - Single-node, cannot handle huge scale
- MapReduce
  - Rigid programming model
  - Write to disk, expensive iteration
- In-memory systems such as Spark
  - Synchronous operations
- Graph engines such as GraphLab
  - Think like a vertex

No Existing System Meets All Requirements

- Synchronous iterations only
  - AsterixDB, HaLoop, Pregel, REX, Spark, PrIter, Glog, …
- Single-node
  - LogicBlox, DatalogFS, …
- No declarative language
  - Stratosphere, Naiad, Grace, GraphLab, …
- Specialized for graphs
  - GraphLab, Grace, …
- Not a data management system
  - SociaLite, …
- Theory on recursive queries
  - DatalogFS, …

Outline and Contributions

- Full-stack solution for iterative processing
  - Declarative relational query language
    - A subset of Datalog-with-Aggregation
  - Scalable and easily implementable
    - Small extensions to existing shared-nothing systems
  - Efficient iterative computation
    - Execution models and optimizations
    - Implementation and empirical evaluation using Myria

Outline and Contributions

- Full-stack solution for iterative processing
  - **Declarative relational query language**
    - **A subset of Datalog-with-Aggregation**
  - **Scalable and easily implementable**
    - **Small extensions to existing shared-nothing systems**
  - Efficient iterative computation
    - Execution models and optimizations
    - Implementation and empirical evaluation using Myria

From Datalog Programs to Asynchronous Query Plans

- Datalog: a relational query language
  - Nicely expresses recursions

```
CC(x,x)          :- Edges(x, )
CC(y,$Min(v)) :- CC(x,v), Edges(x,y)
                      :- CC(y,v)
```

- Two special operators
  - IDBController
    - Maintains state of "nonconstant" relations
  - TerminationController
  - Easy extensions to an existing engine
- Automatic compilation

```
DECLARE @id AS INT, @lvl AS INT
SET @id = 3
SET @lvl   = 2
;WITH cte (id, parent, child, lvl) AS
(
  SELECT id, parent, child,  0
  FROM t
  WHERE id = 1
  UNION ALL
  SELECT E.id, E.parent, E.child, M.lvl+1
  FROM t AS E JOIN CTE AS M
     ON E.parent = M.child
  WHERE lvl < @lvl
)
SELECT *
FROM CTE --where lvl=@lvl
--OPTION (MAXRECURSION 10)
```

Outline and Contributions

- Full-stack solution for iterative processing
  - Declarative relational query language
    - A subset of Datalog-with-Aggregation
  - Scalable and easily implementable
    - Small extensions to existing shared-nothing systems
  - **Efficient iterative computation**
    - **Execution models and optimizations**
    - **Implementation and empirical evaluation using Myria**

Iterative Computation: How Can We Do Better

- Performance impact: # of intermediate tuples
  - More tuples, more work, more resources

- Optimization: recursive execution models
  - Synchronous vs. asynchronous

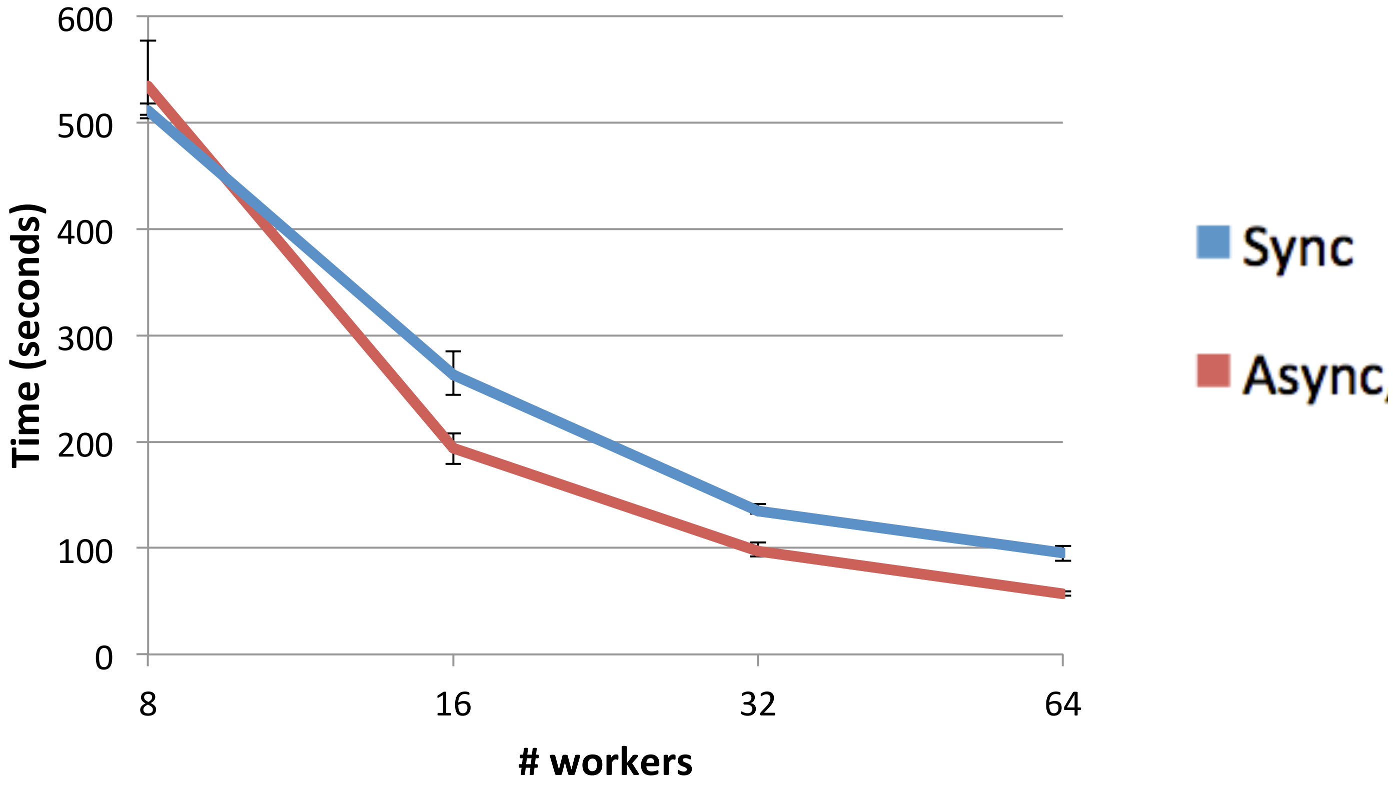
- Optimization: prioritizing tuples
  - For asynchronous model, favor new tuples vs. base tuples

Optimization: Recursive Execution Models

- Synchronous
  - Stop at the end of each iteration
- Asynchronous
  - No barrier, propagate updates when ready

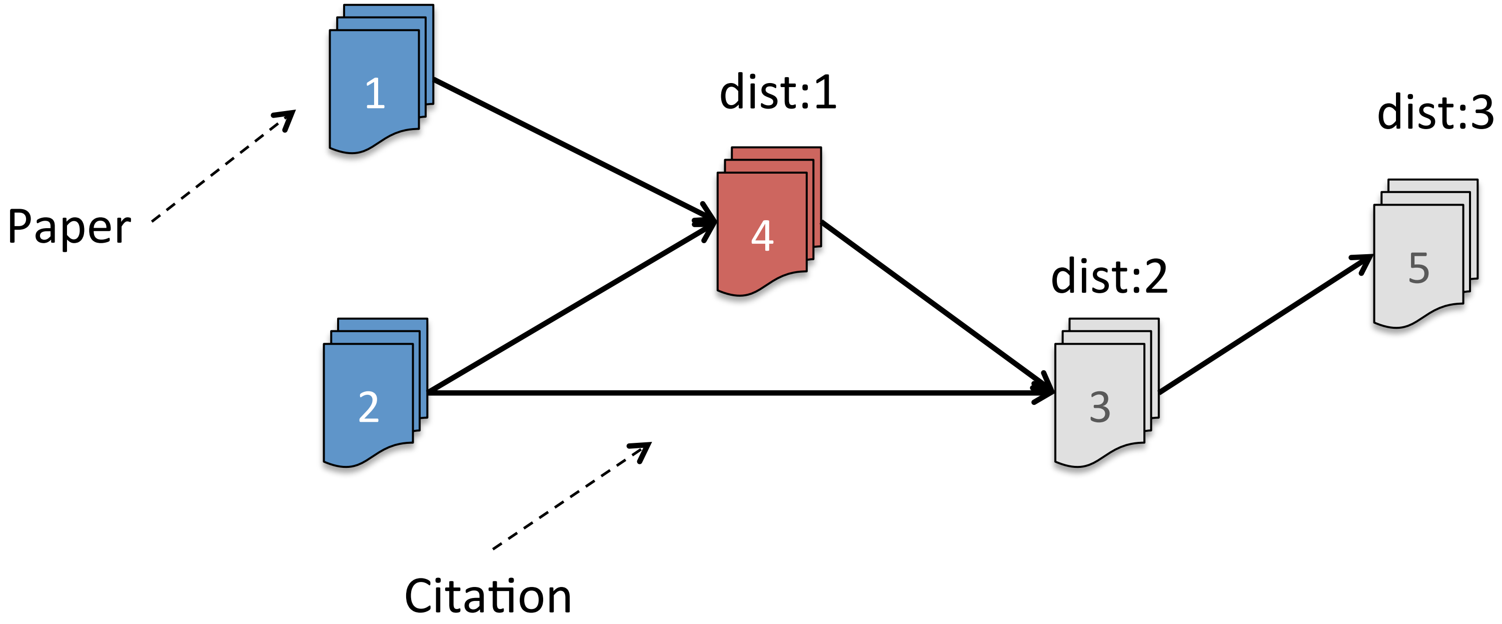
- Galaxy Evolution
  - Synchronous
    - Find all galaxies at timestep 1, then 2, …
  - Asynchronous
    - Galaxy A is a part of the evolution history
    - A shares particles with galaxy B

Galaxy Evolution: Execution Model Does Not Matter Much

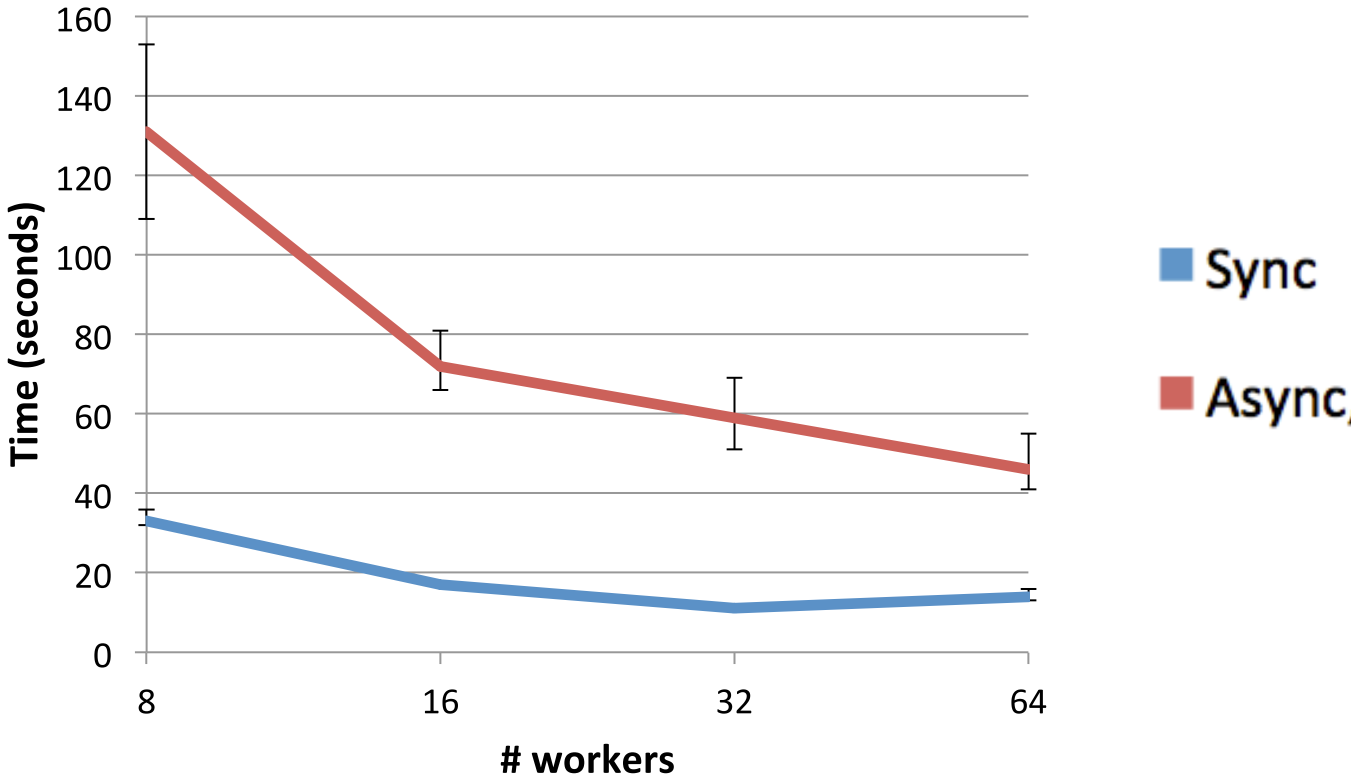


80GB, 27 snapshots
16 machines

Another Application: Least Common Ancestor



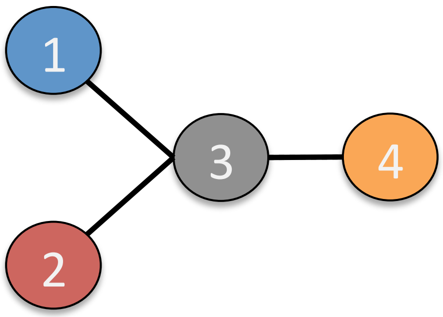
LCA: Asynchronous Can Be Much Slower Than Synchronous

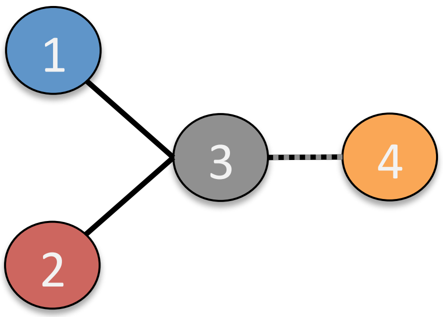


2 million papers
8 million citations

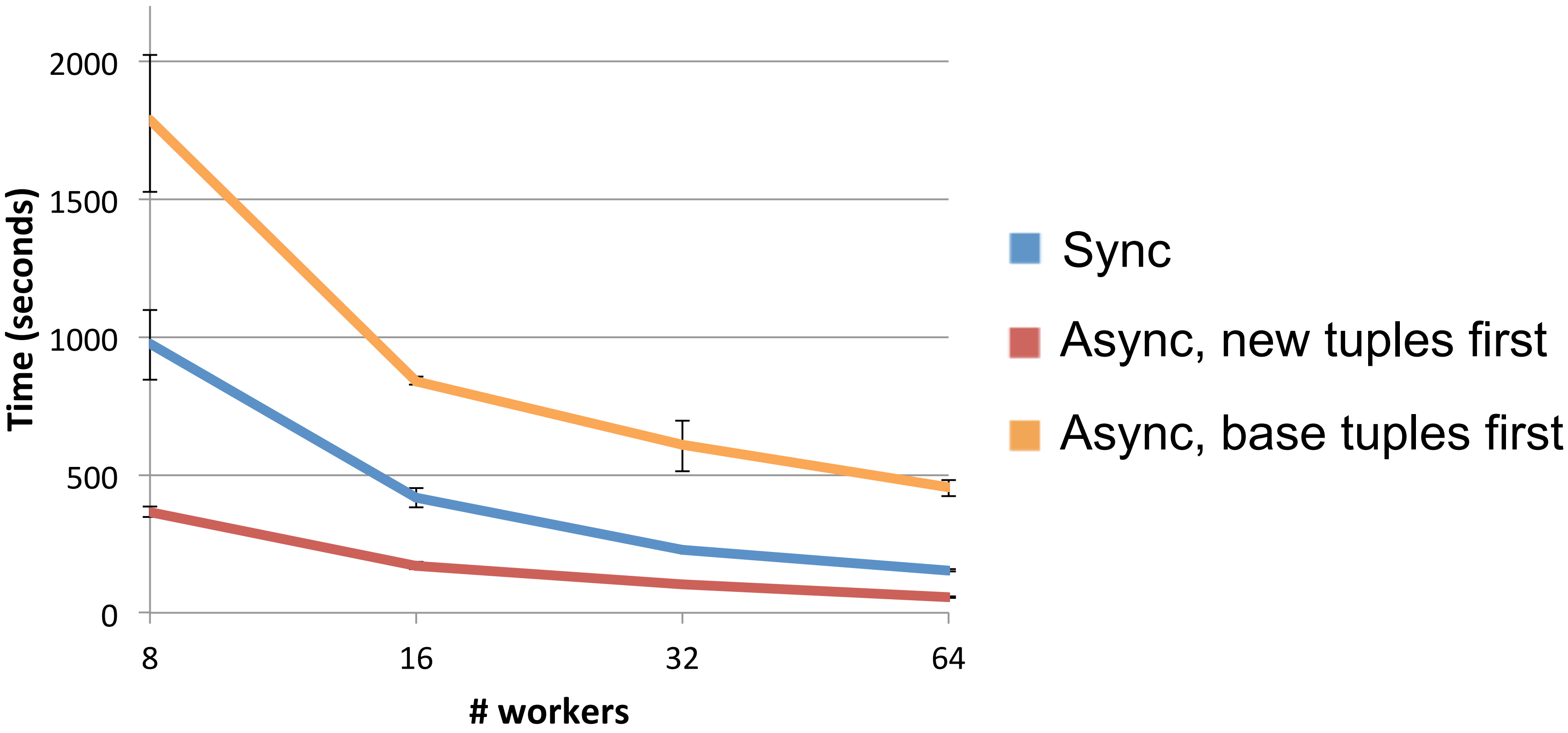
Optimization: Prioritizing Tuples

- For asynchronous processing
  - Choice: favor new tuples vs. base tuples
- Example: connected components





Connected Components: Pull Order Impacts Run Time



21 million vertices
776 million edges

Conclusion

- Full-stack solution for iterative big-data analytics
  - A declarative language
  - Small extensions to existing shared-nothing engines
  - Efficient iterative execution
  - Failure handling methods
  - More details in the paper
- Empirical evaluation of various models
  - No single method outperforms others
  - Future work: an adaptive cost-based optimizer