Kineograph: Taking the Pulse of a Fast-Changing and Connected World

Raymond Cheng, Ji Hong, Aapo Kyrola, Youshan Miao, Xuetian Weng, Ming Wu, Fan Yang, Lidong Zhou, Feng Zhao and Enhong Chen

Presented by Petko Georgiev

19 February 2013

Motivation

- User Generated Content
- Rich Connections
- Fast streaming of graph updates
- Timely response to graph changes needed!

Kineograph's solution

Distributed in-memory graph storage

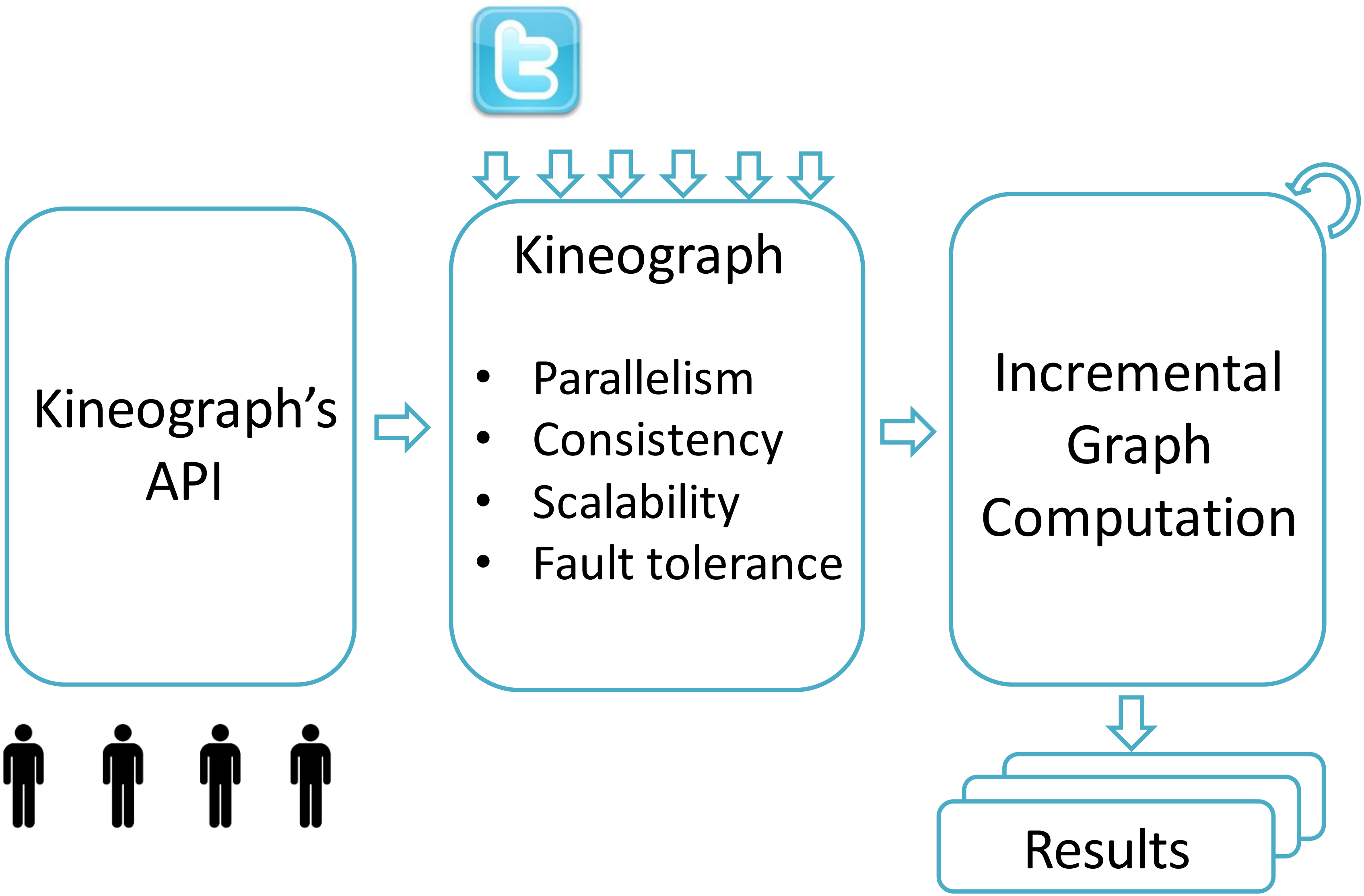
+

Consistent periodical graph snapshots

+

Incremental graph computations

Kineograph's solution



System Overview

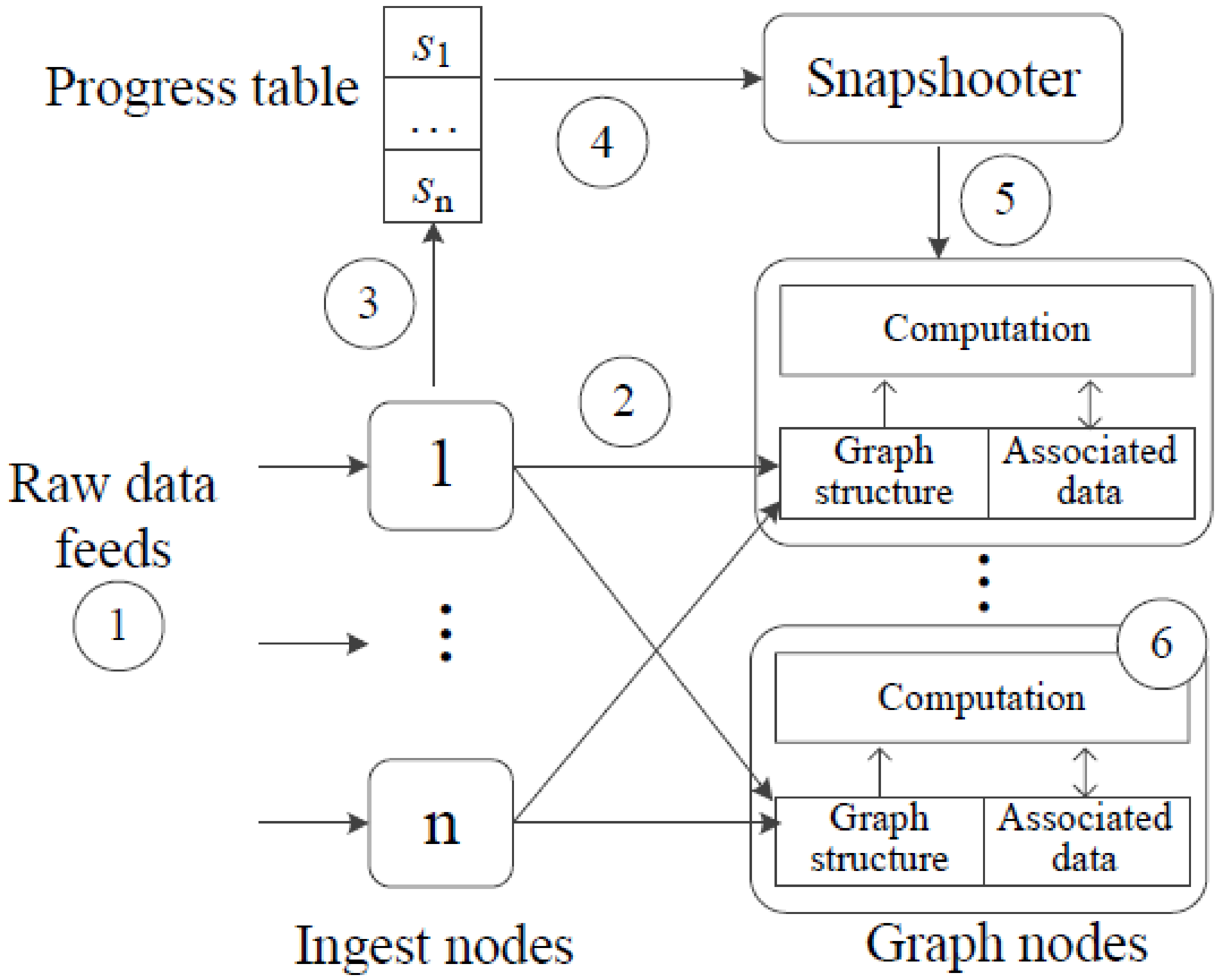
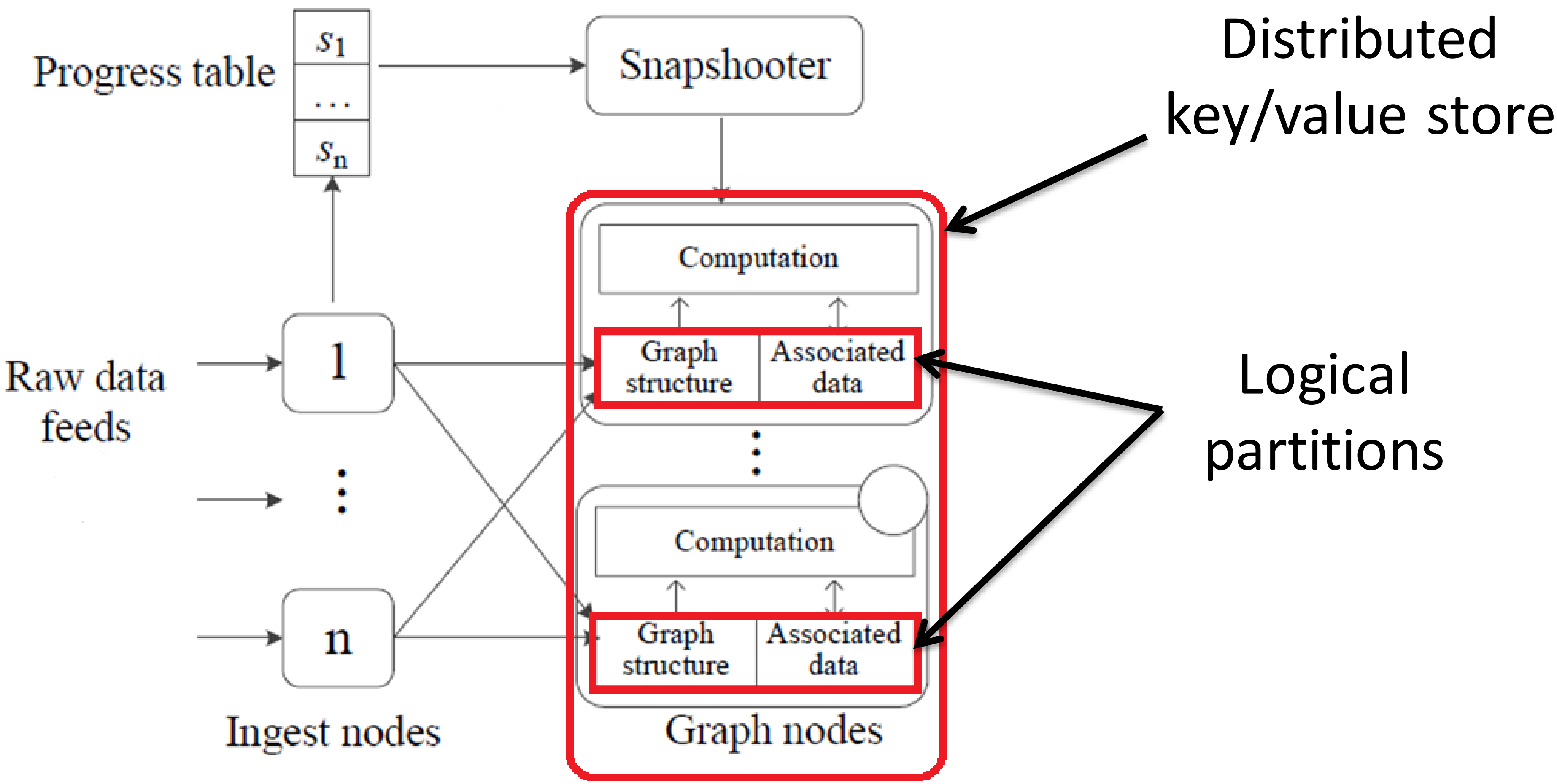
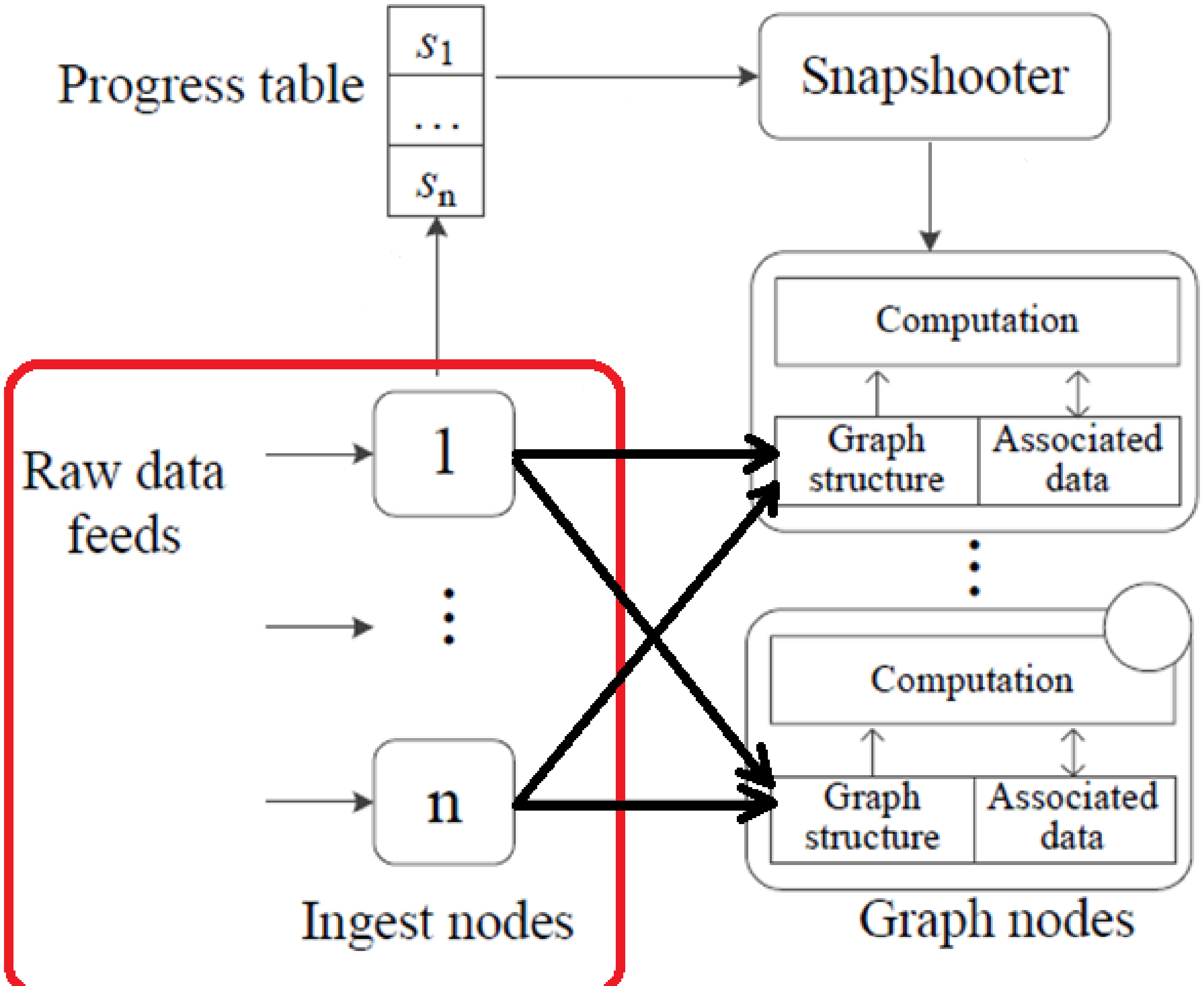


Figure 1. System overview.

Storage Layer



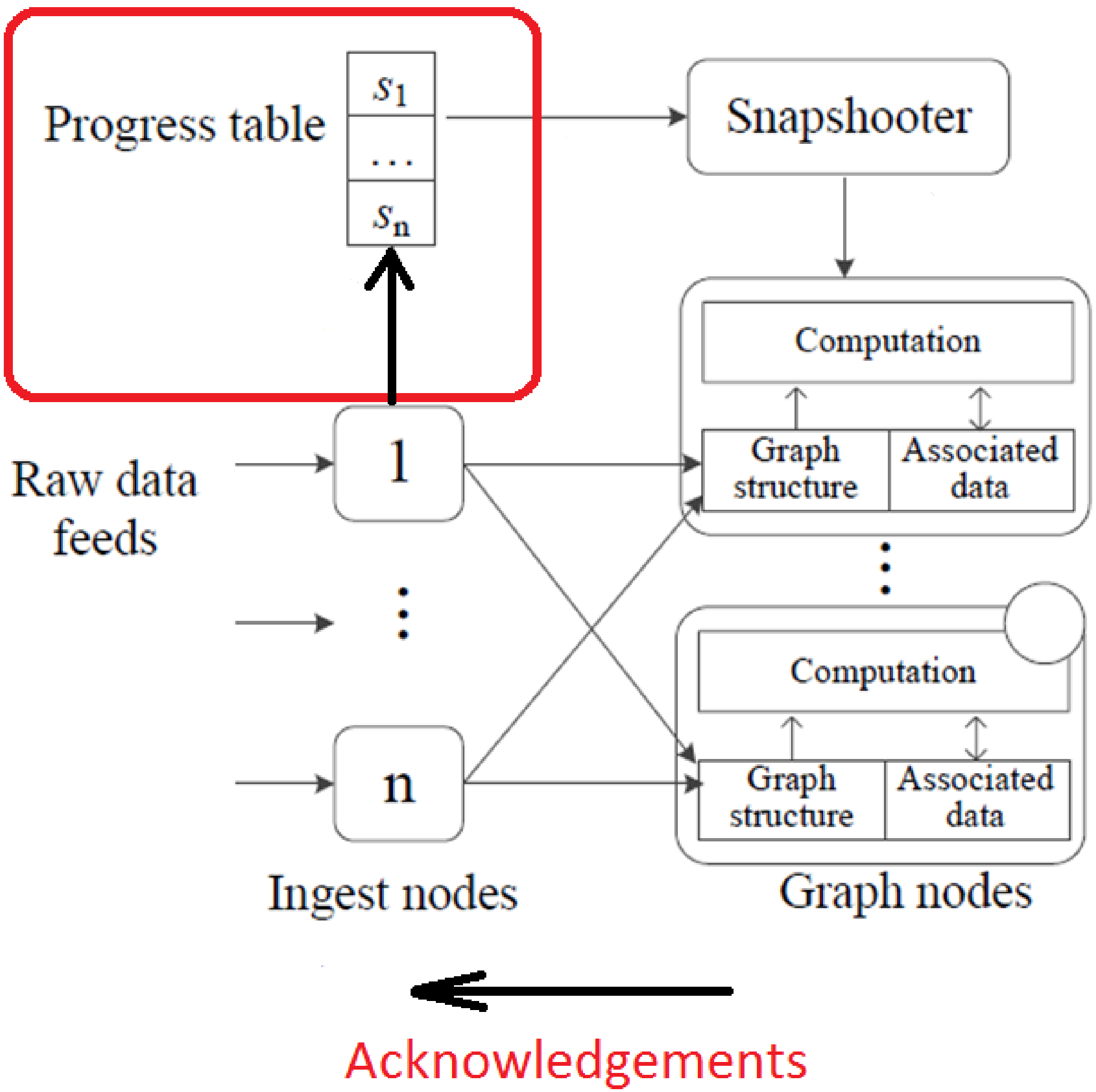
Ingest Nodes



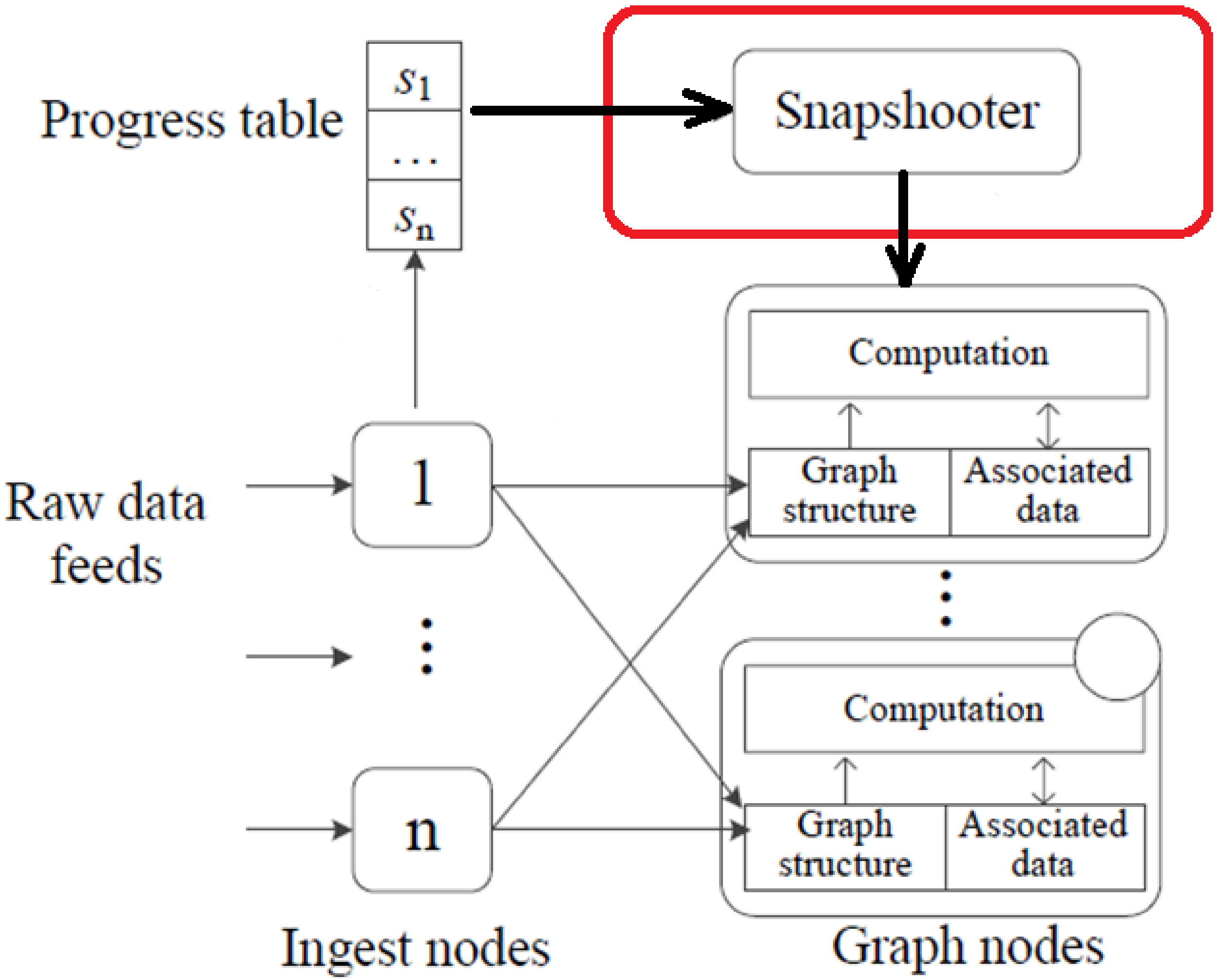
Transactions of graph updates

Might span multiple logical partitions

Progress Table



Epoch Commit



Periodically take snapshots

Incoming updates are not blocked

Atomicity is guaranteed

Computation Layer

- Vertex-centric approach

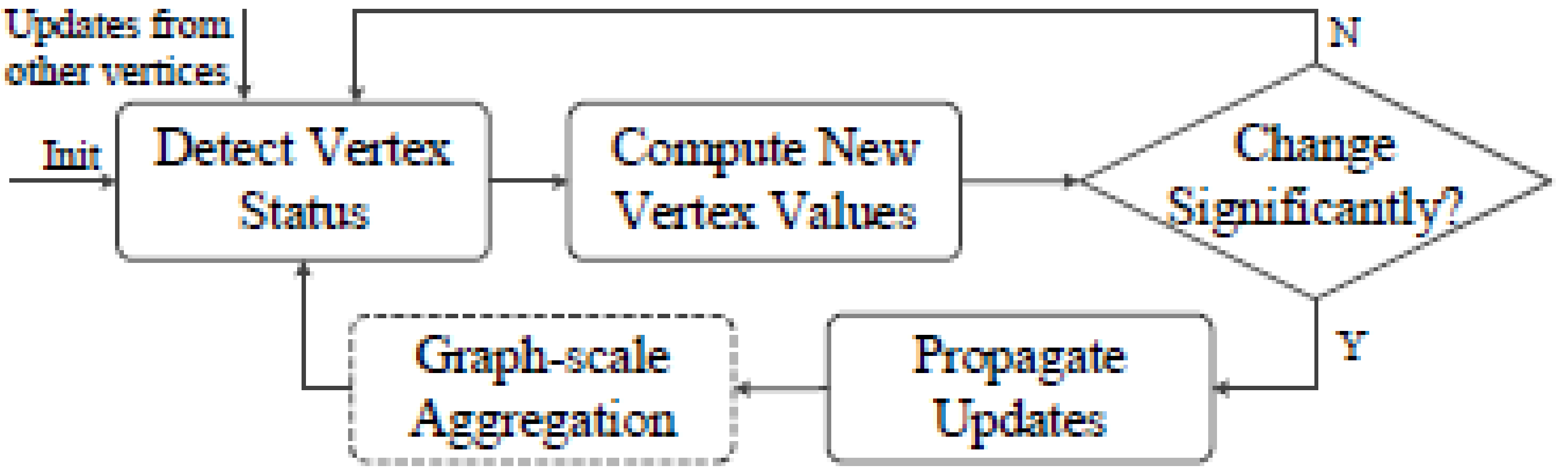


Figure 3. Computation overview.

Push vs. Pull Model

Push

```
value_0: T

initialize

updateFunction(vertex)
```

Pull

```
value_0: T

initialize

updateFunction(vertex,
        List[readonly-vertex])
```

```
trigger(oldval: T, newval: T): boolean

accumulator(accumValue: T,update: T): T
```

TunkRank Push Model Example

- **`graph:`** user mentions
- **`initialize:`** for new out edges mark vertex
- **`updateFunction(vertex):`**
  - send difference of new and previous rank to neighbors
- **`accumulator:`** sum operation
- **`trigger(oldval, newval):`**

```
abs(oldval - newval) > ε
```

Implemented Applications

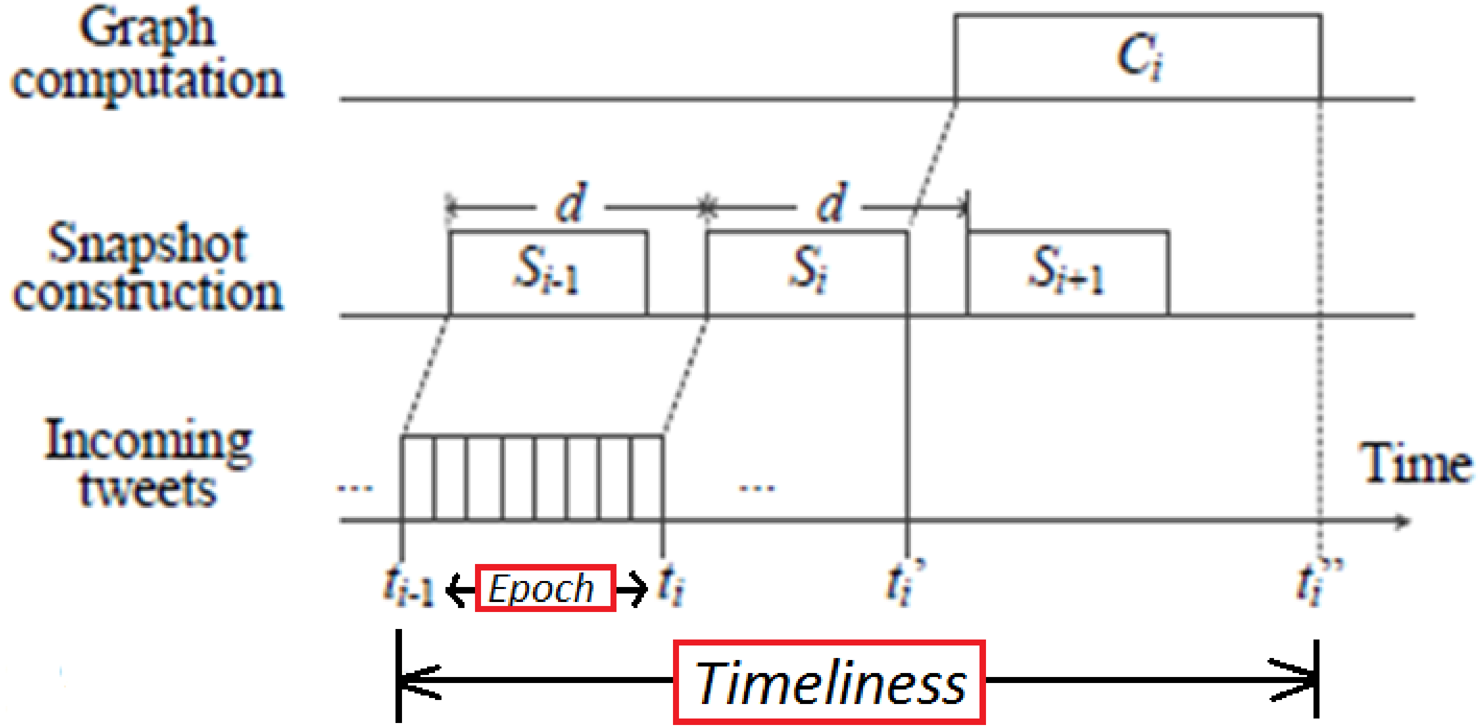
- TunkRank (*push*)
- Shortest Paths (*push*)
- K-exposure (*pull*)

Fault Tolerance

- Ingest nodes – incarnation numbers
- Storage layer replication of logical partitions
- Computation layer
  - Roll back and re-execute on failure
  - No computation on replicas!
  - Primary/backup replication for results

Evaluation

- Throughput (# tweets per second)
- Timeliness



Evaluation: Graph Update Throughput

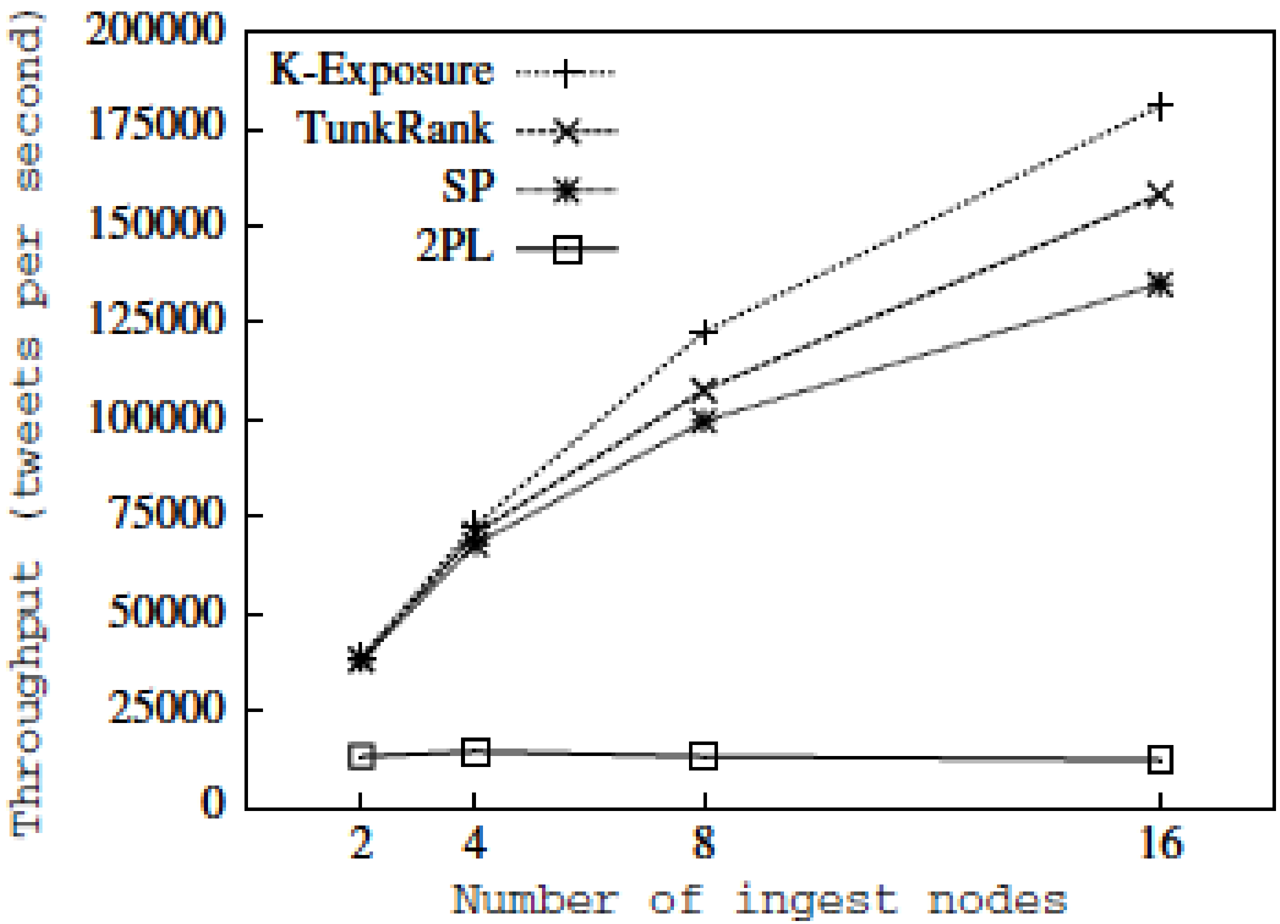


Fig. 9. Graph update throughput
(32 graph nodes, 10-second snapshots)

Evaluation: Timeliness

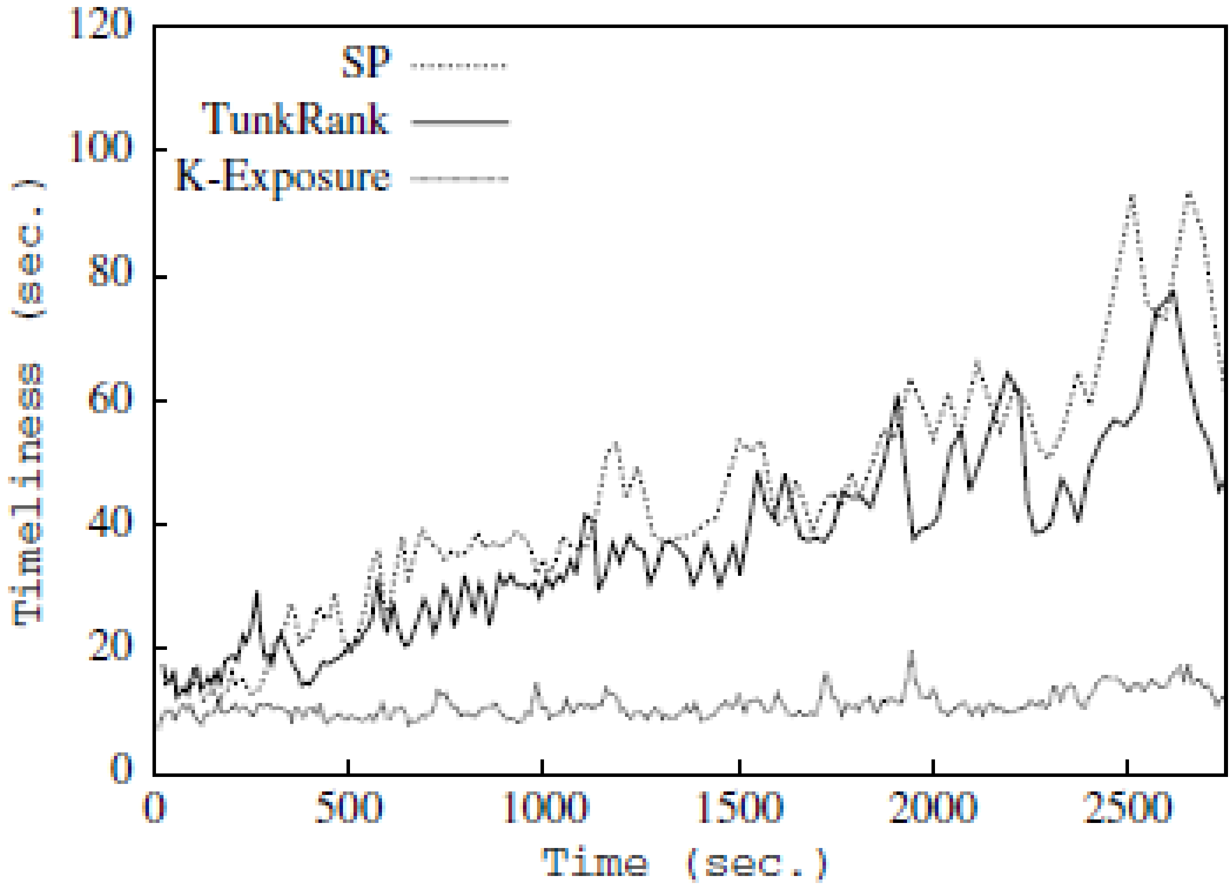


Figure 10. Data timeliness for different applications with 2 ingest nodes and 32 graph nodes.

Evaluation: Timeliness

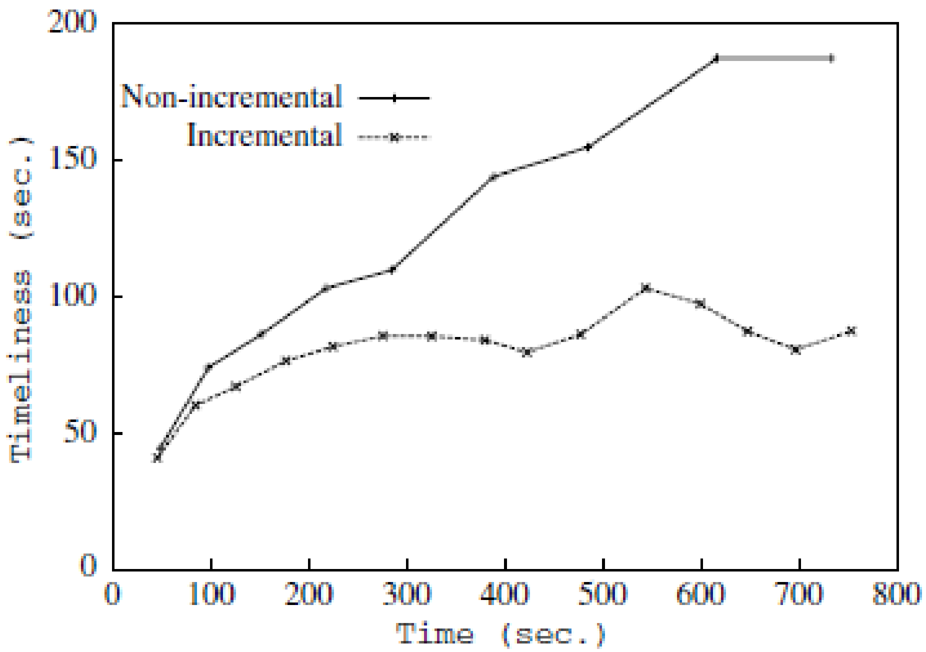


Figure 11. Timeliness changes over time for incremental and non-incremental graph computation with TunkRank, 4 ingest nodes, and 32 graph nodes.

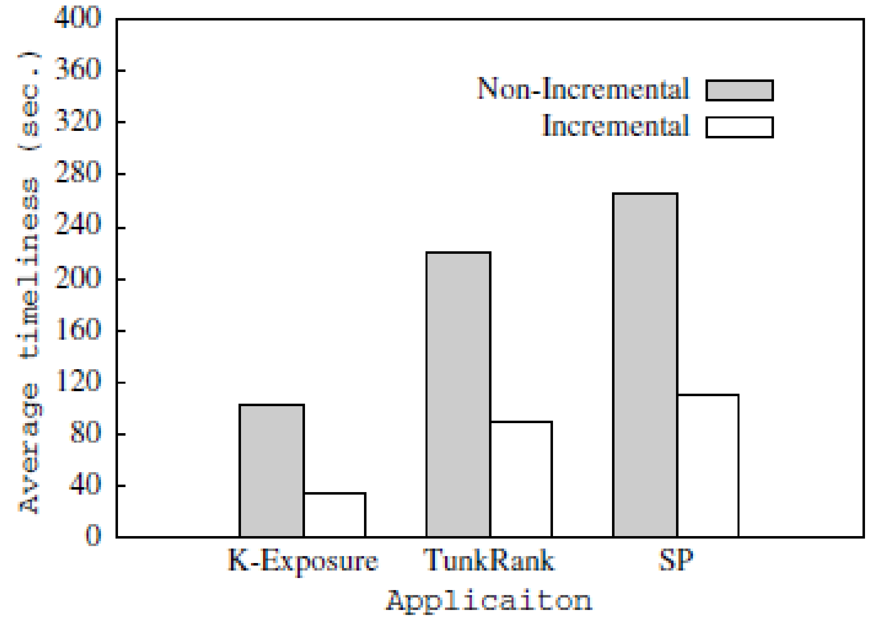


Figure 12. Average timeliness improvement of incremental applications under 4 ingest nodes and 32 graph nodes.

Evaluation: Timeliness

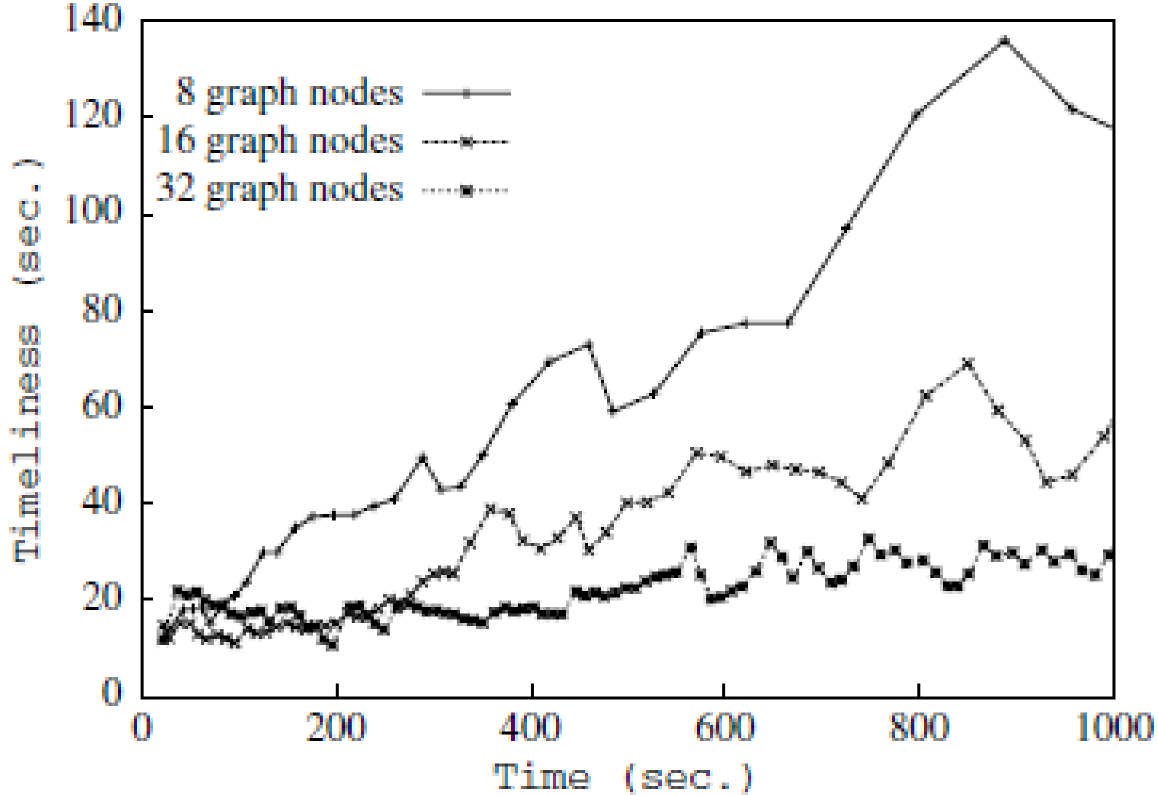


Figure 13. Scalability of TunkRank with different numbers of graph nodes and 2 ingest nodes.

Evaluation: Timeliness

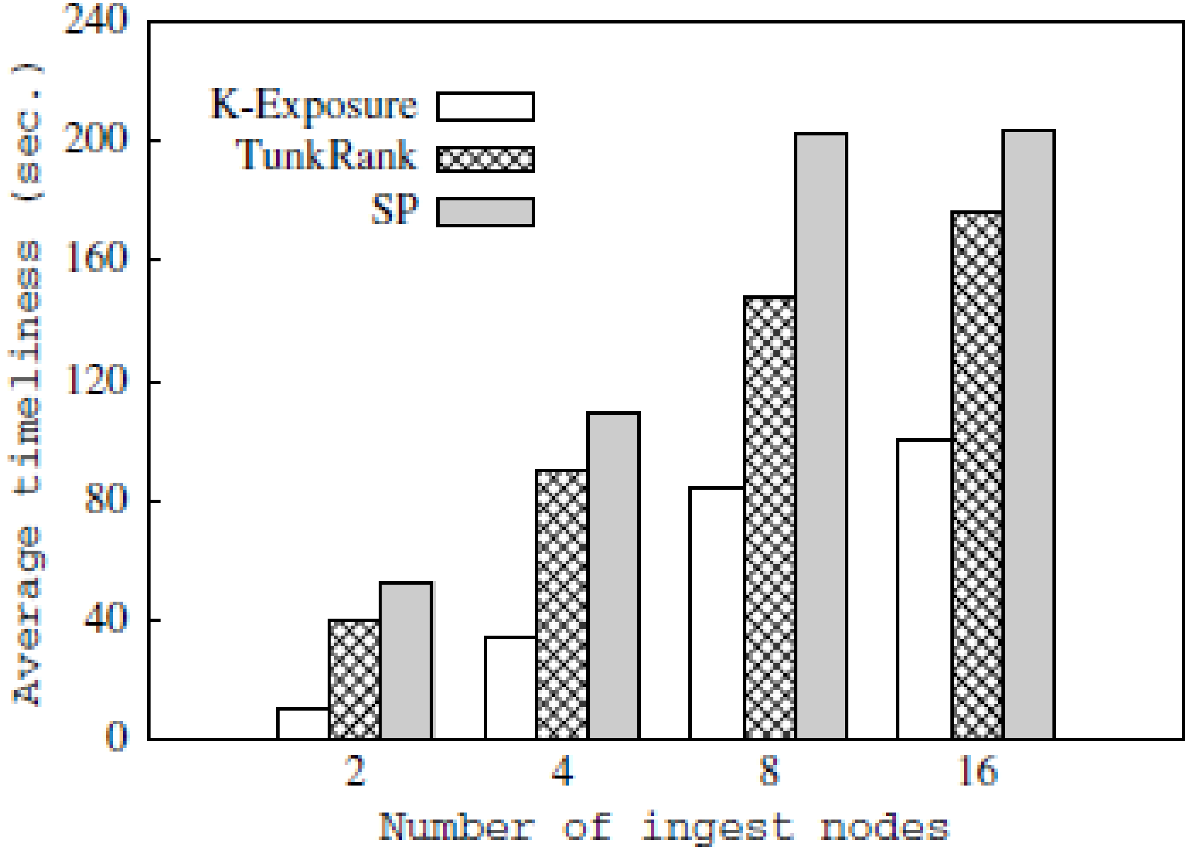


Figure 14. Average data timeliness with different number of ingest nodes and 32 graph nodes.

Difference from Existing Work

- Streaming of *graph* updates
- Incremental computation on a *global* snapshot of a *graph* model (vs. MapReduce, databases)
- Kineograph does not use locks (unlike Google Percolator)
- Vertex-based processing model (like Pregel, GraphLab) but with *incremental* computation

Critique and Future Work

- A nice combination of ideas
- Decaying not implemented and not evaluated
- Locality sensitive hashing?
- Choice of snapshot interval – any more concrete justifications? Why exactly 10 seconds and not 12 or 8?
- Exact time for applying updates upon epoch commit?
- How many snapshots backwards are stored?