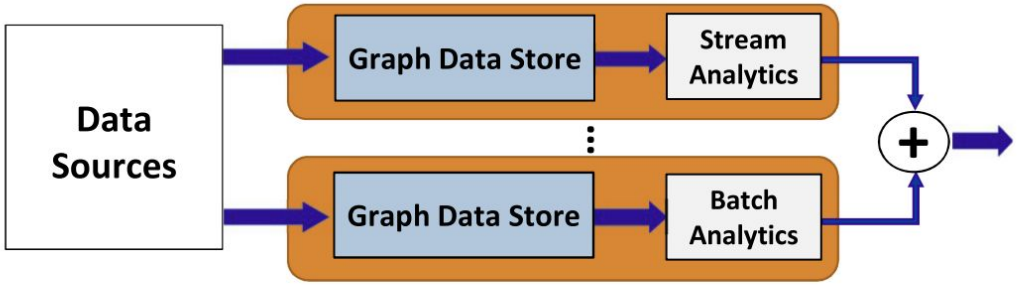
GraphOne: A Data Store for Real-time Analytics on Evolving Graphs (FAST '19)

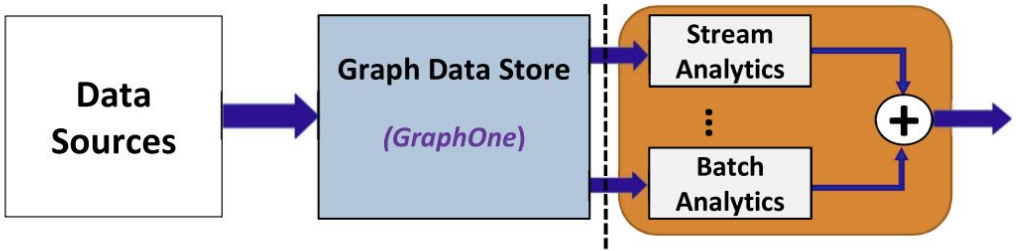
By Pradeep Kumar, H. Howie Huang
Presented by Anton Cao

Motivation

- Often need to compute both batching and streaming analytics on evolving graphs
- Prior graph stores could not efficiently provide both types of analytics with high ingestion
- Using multiple data stores requires data to be duplicated



(a) Current Graph Analytics Systems have a Private Data Store



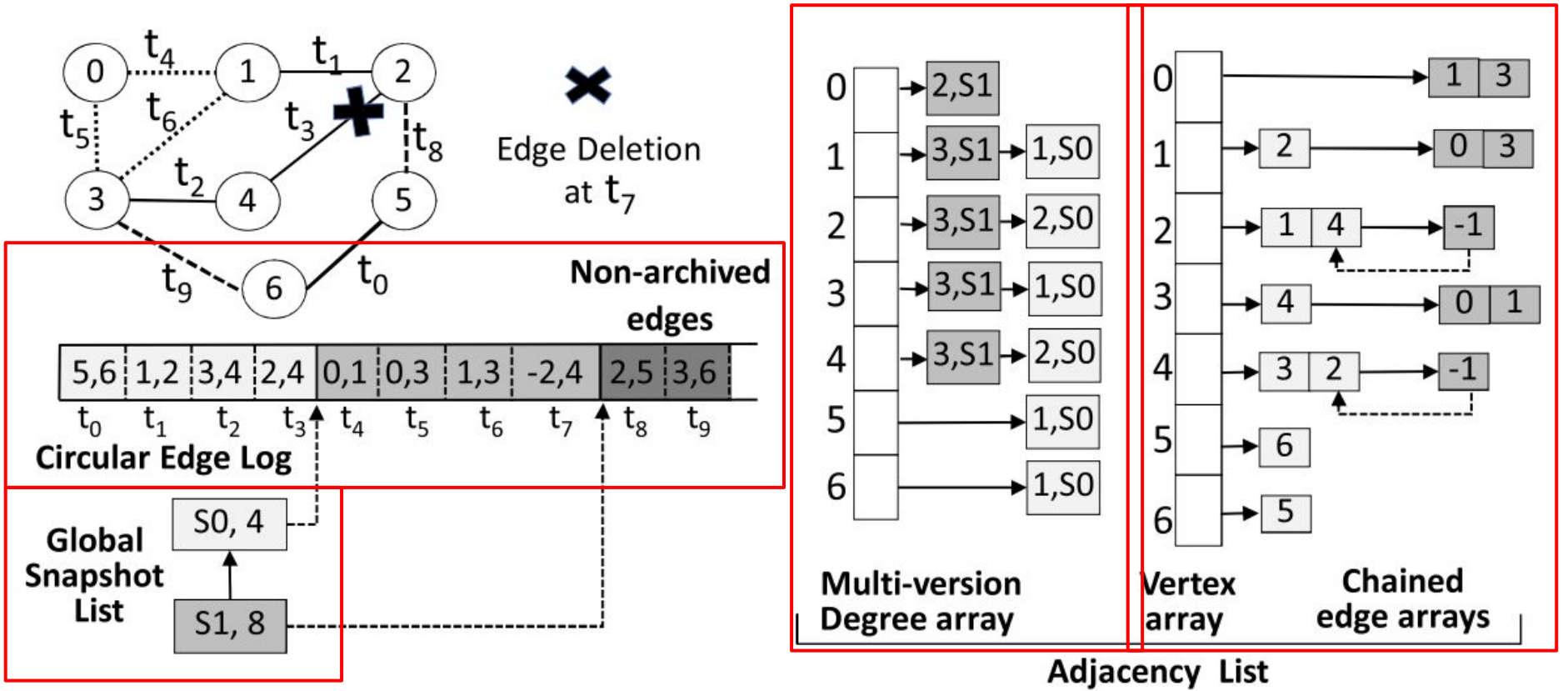
(b) GraphOne abstract Data Store away from Graph Analytics Systems

Graph storage formats

- Edge list
  - Pros: fast ingestion and good temporal locality
  - Cons: bad spatial locality (edges of vertex not located together)
- Compressed sparse row (CSR)
  - Pros: good spatial locality
  - Cons: expensive to insert edges
- Adjacency list
  - Pros: fast ingestion, good spatial locality
  - Cons: bad temporal locality

Hybrid Store

- Combine edge list and adjacency list for good spatial and temporal locality
- Also provides coarse and fine grained snapshotting


0
1
2
3
4
5
6
t_0
t_1
t_2
t_3
t_4
t_5
t_6
t_7
t_8
t_9
Edge Deletion at t_7
Non-archived edges
5,6
1,2
3,4
2,4
0,1
0,3
1,3
-2,4
2,5
3,6
Circular Edge Log
Global Snapshot List
S0, 4
S1, 8
2,S1
3,S1
1,S0
2,S0
Multi-version Degree array
Vertex array
Chained edge arrays
Adjacency List

Versioning

- Adjacency list provides coarse grained snapshotting with epoch number
  - Adjacency list is append only
- If application desires fine grained snapshotting, can select specific edge within an epoch

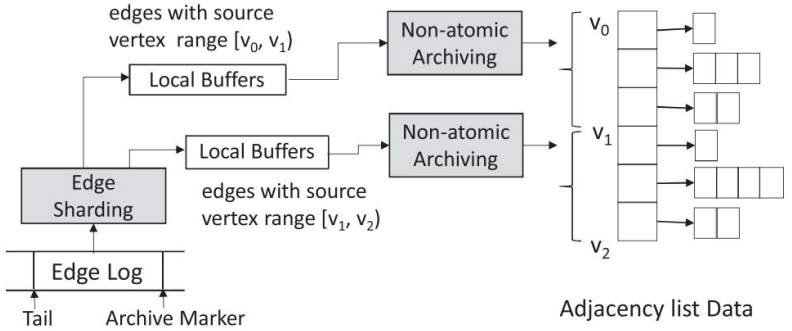
Data Management

1. Logging: appends updates to edge log
2. Archiving: moves updates to adjacency list
3. Storing: stores data on non-volatile storage
4. Compaction

Data Management - Logging

- Appends updates to edge log
- Addition/deletion stored in MSB of source
- Log rewind parity stored in MSB of destination to guarantee atomicity
- Begin archiving and storing to durable storage when number of non-archived edges exceeds threshold

Data Management - Archiving



- Move non-archived edges (epoch) to adjacency list
- Shard edges into buffers by source vertex in parallel
  - Preserves ordering using two scans
- Non-atomic archive of independent shards
  - Greedily assign shards to workers to balance the workload
- Once complete, create edge block in adjacency list, and nodes in degree array and global snapshot list

Data Management - Storing

- Write edge log data to disk
- May be fetched if non-archived edges get overwritten
- Optionally checkpoint adjacency list data

Data Management - Compaction

- Remove deleted data from adjacency list and combine edge blocks
- Maintain reference count of number of views that are using each snapshot version
- Can compact data up to last retired snapshot (reference count of 0) and update degree array

Memory

- Adjacency list is biggest overhead (excessive chaining of edge blocks, each needs to store size and pointer to next block)
- Optimizations:

Table 2. Impact of Two Optimizations on the Chain Count and Memory Consumption of the Edge Arrays of the Adjacency Store on the Kronecker (Kron-28) Graph

Optimizations	Chain Count		Memory Needed (GB)
	Average	Maximum	
Baseline System	29.18	65,536	148.73
+Cacheline memory	2.96	65,536	47.42
+Hub Vertex Handling	2.47	3,998	45.79
Static System	0.45	1	33.81

Cache line Memory optimization

- Allocate edge blocks in multiples of cache line (64 bytes) instead of dynamically based on edges in epoch
- 10x reduction in chain length and 3x reduction in memory consumption

Hub Vertex optimization

- Some vertices have very large degrees
- Allocate page-aligned (4KB) instead of cache-aligned (64B) edge blocks for vertices that exceed a threshold of neighbors
- Greatly reduces max chain length, little effect on memory footprint

GraphViews

- Interface that GraphOne provides to analytics applications
- Supports multiple views on the same data store with little overhead (no data duplication)
- **Static View** for batch queries
- **Stream View** for stream processing

Static View

Table 3. Static View APIs

snap-handle	create-static-view(global-data, simple, private, stale)
status	delete-static-view(snap-handle)
count	get-nebr-length-{in/out}(snap-handle, vertex-id)
count	get-nebrs-{in/out}(snap-handle, vertex-id, ptr)
count	get-nebrs-archived-{in/out}(snap-handle, vertex-id, ptr)
count	get-non-archived-edges(snap-handle, ptr)

- get-nebrs needs to add non-archived edges to adjacency list
- **simple=True** creates in memory copy adjacency list of non-archived edges, speeds up get-nebrs
- **private=True** creates local copies of non-archived edges and degree array, does not interfere with archiving/storage, useful for long-running analytics
- **stale=True** ignores non-archived edges

Stream View

Table 4. Stateless Stream View APIs

stream-handle	reg-stream-view(global-data, window-sz, batch-sz)
status	update-stream-view(stream-handle)
status	unreg-stream-view(stream-handle)
count	get-new-edges-length(stream-handle)
count	get-new-edges(stream-handle, ptr)

- **window-sz:** size of data store that should be available (defaults to beginning of stream)?
- **batch-sz:** number of new updates before next iteration triggered
  - **update-stream-view** blocks

Stateful Stream

Table 5. Stateful Stream View APIs

sstream-handle	reg-sstream-view(global-data, window-sz, v-or-e-centric, simple, private, stale)
status	update-sstream-view(sstream-handle)
status	unreg-sstream-view(sstream-handle)
bool	has-vertex-changed(sstream-handle, vertex-id)
count	get-nebr-length-{in/out}(sstream-handle, vertex-id)
count	get-nebrs-{in/out}(sstream-handle, vertex-id, ptr)
count	get-nebrs-archived-{in/out}(sstream-handle, vertex-id, ptr)
count	get-non-archived-edges(sstream-handle, ptr)

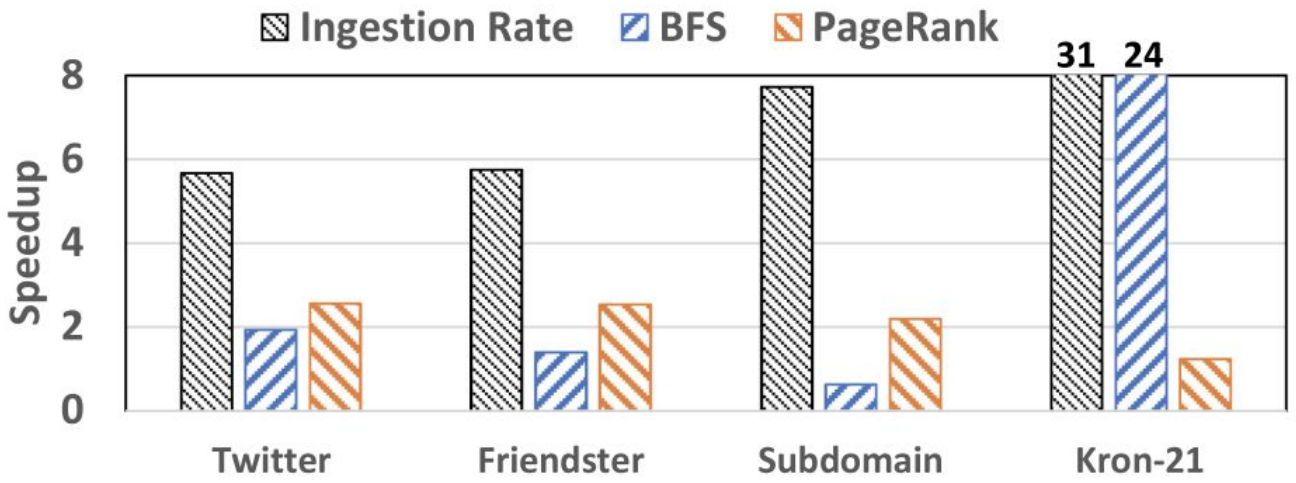
Evaluation

- 16K loc (C++)
- Evaluate on data ingestion, BFS, PageRank, and 1-Hop query (access edges of random vertices)
- Real-world and synthetic graphs

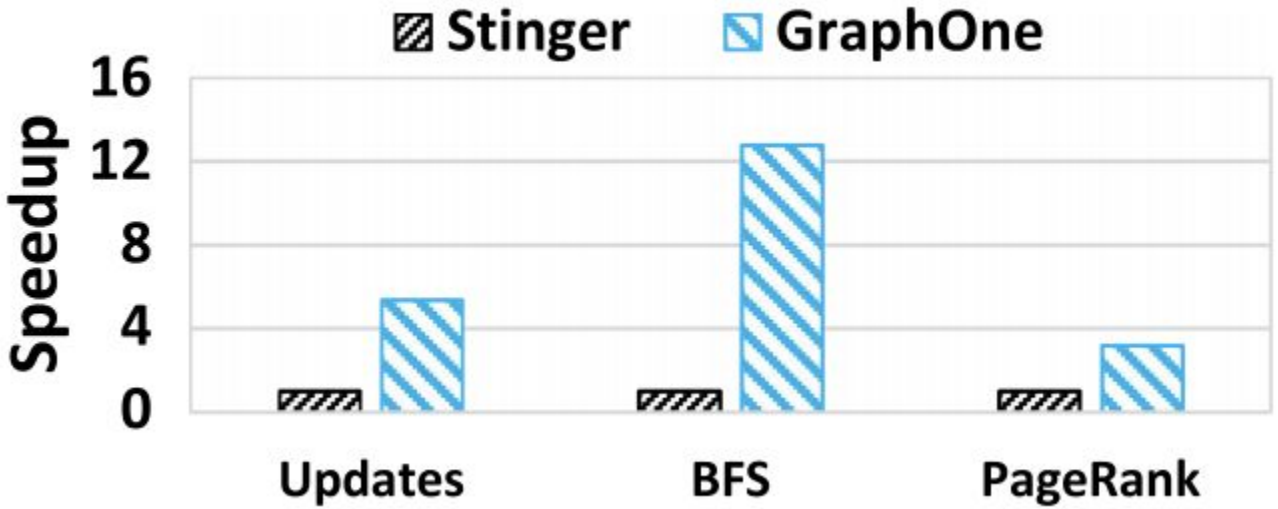
Table 8. Different Data Management Rates in Millions Edges/s (M/s) on Various Graph Dataset

Graph Name	Individual Phases (M/s)		In-Memory Rate (M/s)		Ext-Memory Rate (M/s)		Compaction Rate (M)
	Logging	Archiving	Ingestion	Recovery	Ingestion	Recovery	
LANL	35.98	28.91	26.99	30.23	25.26	29.48	41.85
Twitter	82.62	47.98	66.39	71.28	61.13	71.87	541.71
Friendster	82.85	49.32	60.40	95.78	58.35	95.44	520.65
Subdomain	82.86	43.43	68.25	180.75	61.54	151.96	444.84
Kron-28	79.23	43.68	52.39	116.18	49.70	107.61	798.91
Kron-21	78.91	78.40	58.31	90.44	57.02	66.66	1011.68

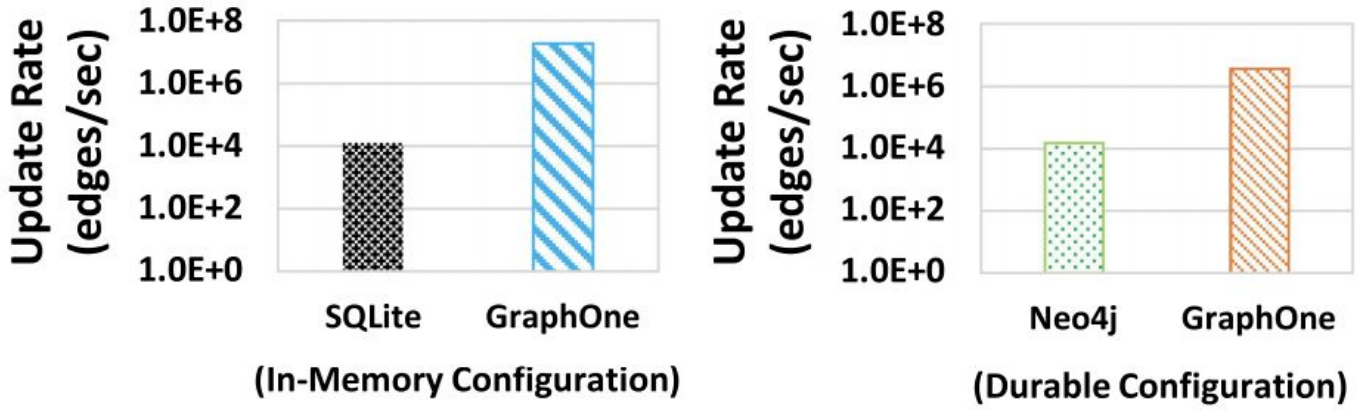
Vs LLAMA (snapshot based)



Vs Stinger (fine grained updates)



Vs SQLite, Neo4j (databases)



Vs Kickstarter (streaming)

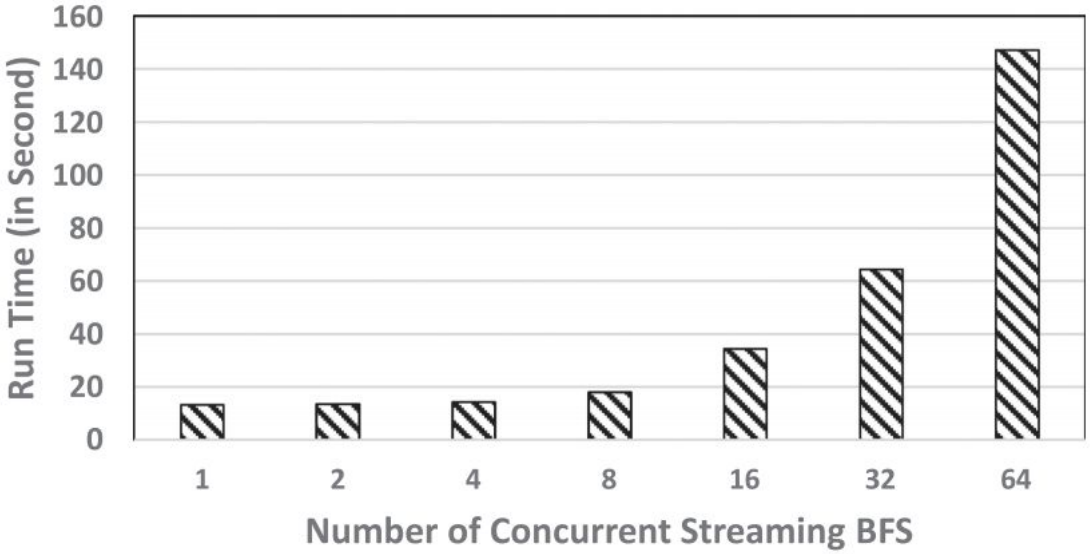
Kickstarter was too slow at updating the adjacency store (2000x)

Parallel archiving and streaming

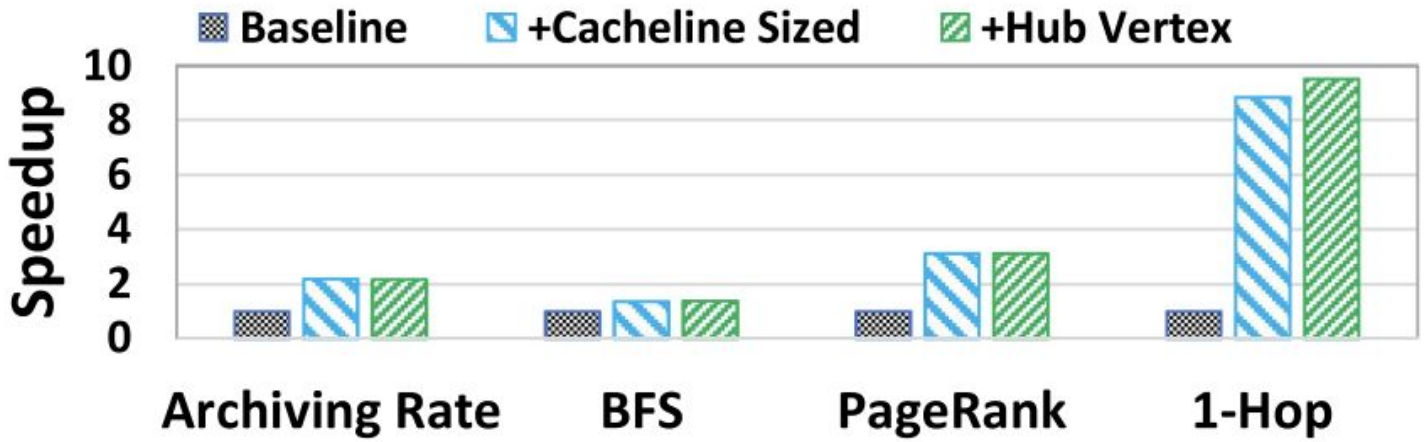
Table 10. Impact of Separating Archiving Phase out of Analytics Workflow

Graph Name	Baseline GraphOne	GraphOne
Twitter	987.45	764.58
Friendster	1236.78	1032.35
Subdomain	975.74	817.90
Kron-28	2687.60	1990.68
Kron-21	19.37	13.87

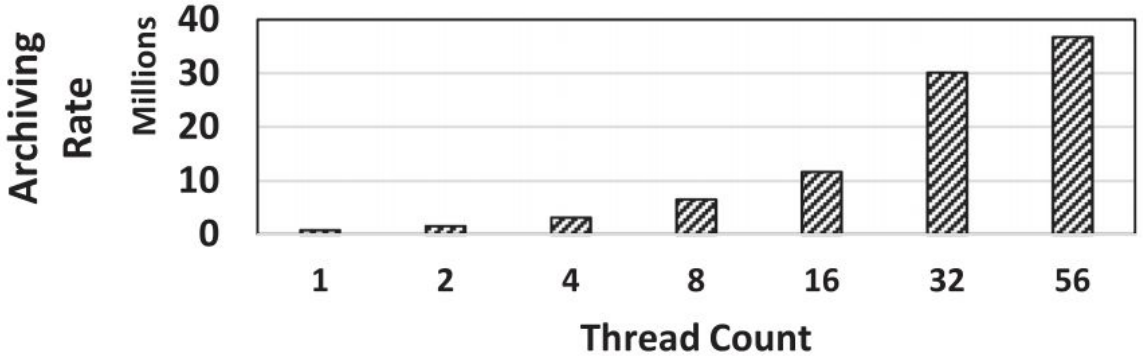
Support for concurrent streaming



Effects of optimizations



Scalability



Summary

- GraphOne abstracts data store away from analytics
- Novel hybrid data store allows efficient batch and stream queries
- At least 3x speedup compared with existing frameworks like LLAMA and Stinger
- Well organized paper, extensive evaluation

Discussion

- Can the idea of a hybrid data store be applied in other domains?
- Is the 2000x result against Kickstarter reasonable?
- It seems the archive step uses a thread aware work balancing scheme, would it be possible/helpful to use a thread agnostic framework like Cilk?
- How was GraphOne received, and is it the current state-of-the-art?