Analyzing system characteristics for graph algorithms across different graph frameworks

CS 784, Final Project, Group 10

Aakarsh Agarwal
Mohil Patel
Sweksha Shukla

(Dept. of Computer Sciences/CDIS)

Motivation

- Today we have huge graphs like social networks, web graphs, protein interaction graphs, etc
- To analyze and generate insights different graph specific algorithms are run on these graphs
- We have many different graph frameworks for running these algorithms. They all uses different techniques and partitioning schemes to process the graphs, like:
  - GraphChi - single machine framework, uses Parallel Sliding Window concept to minimizes memory consumption while ensuring low disk reads
  - GraphX - multi-node framework over spark, specialized operations for better performance and vertex-cut partitioning to minimize communication overhead
  - GraphFrames - multi-node framework designed to think of graphs as relational table with processing as queries
- Why of these are a better choice? And when?

Objective and Goals

- Comparing execution time and system characteristics of different graph frameworks
- Why graph algorithms are compared?
    - Pagerank
    - Connected Components
    - Triangle Counting
- Graph processing frameworks analyzed:

Spark	Multi-Node Framework
GraphX	Multi-Node Framework
GraphFrames	Multi-Node Framework
GraphChi	Single Machine Framework

Experimental Setup

- All the experiments are run on Cloudlab
- For Multi-node frameworks we used 3 VMs each
- And Single Machine Framework is run on 1 VM
- Each VM hardware details:

Cloudlab Machine	c220g1
CPU	Intel Xeon E5-2630
No. of Cores	5
RAM	32 GB
Disk Space	96 GB

- Each Framework was configured to utilize all the resources to max capacity

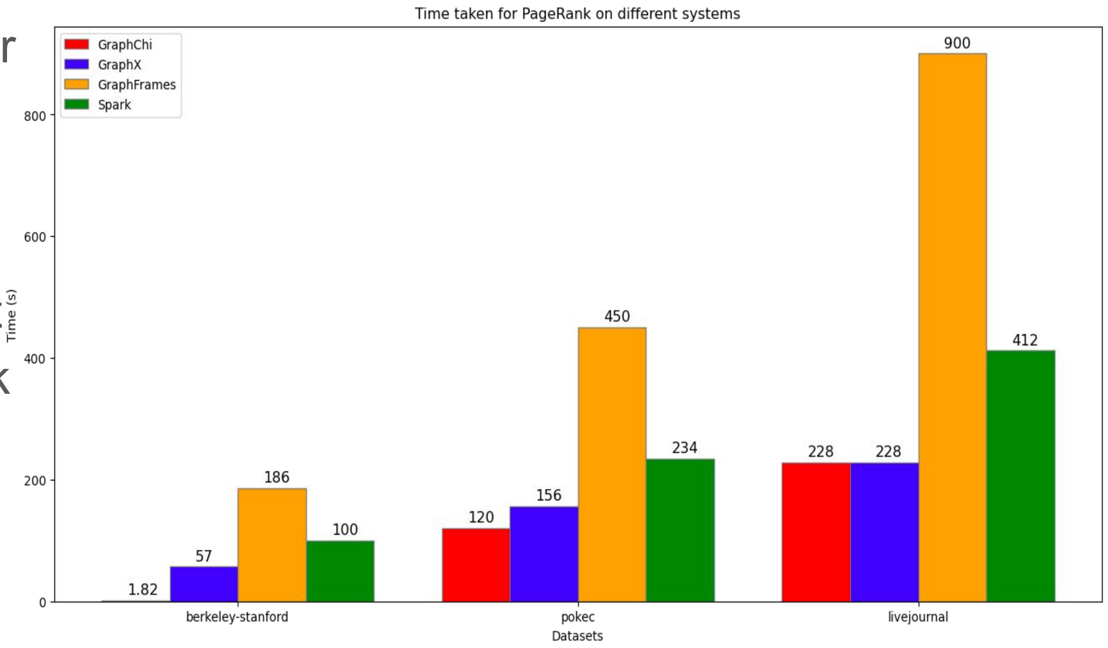
Pagerank

Datasets	Nodes	Edges
Berkeley-Stanford web graph	685,230	7,600,595
Pokec social network	1,632,803	30,622,564
LiveJournal social network	4,847,571	68,993,773

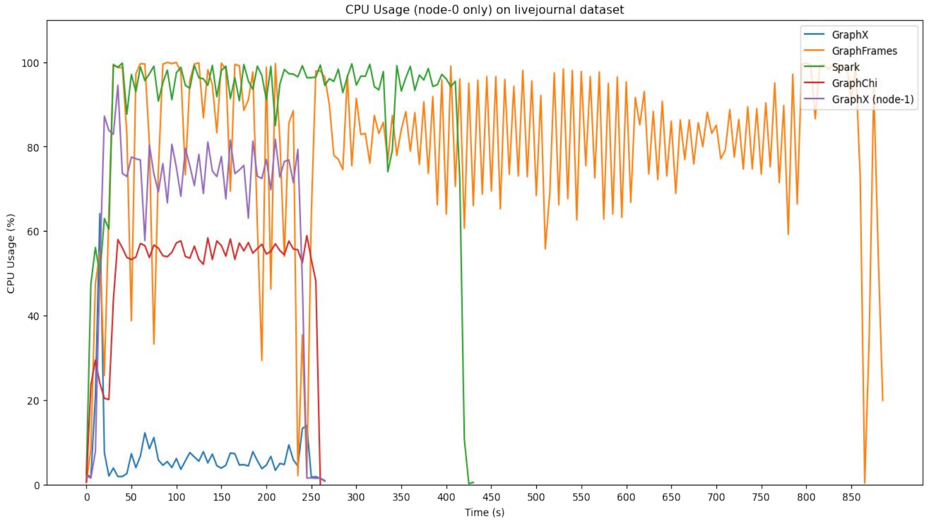
Measurements are for 50 iterations in all systems.

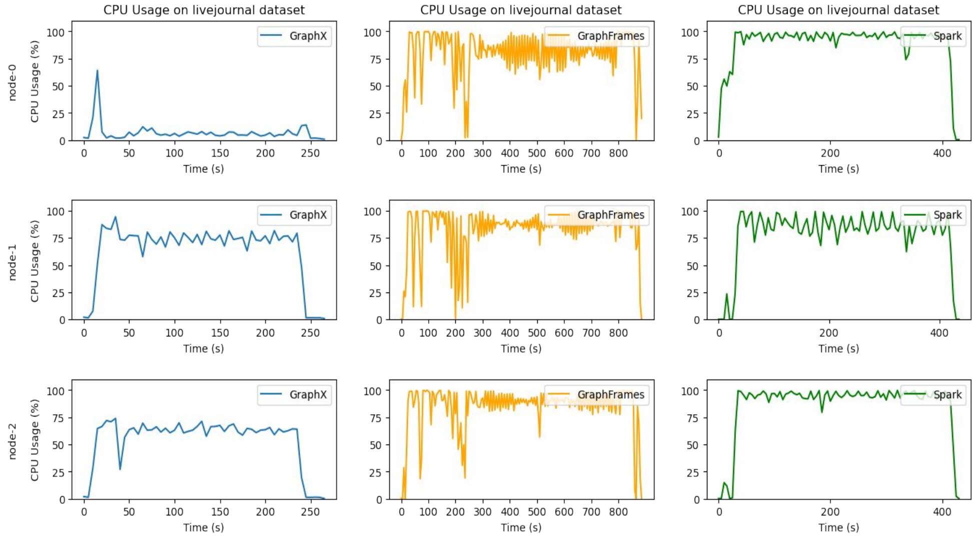
Execution Time

- GraphChi sees better timing for smaller graphs, but as graph size increases its performance is similar to GraphX.
- GraphFrames shows the worst timings, even worse than spark custom implementation.



CPU Usage (%)

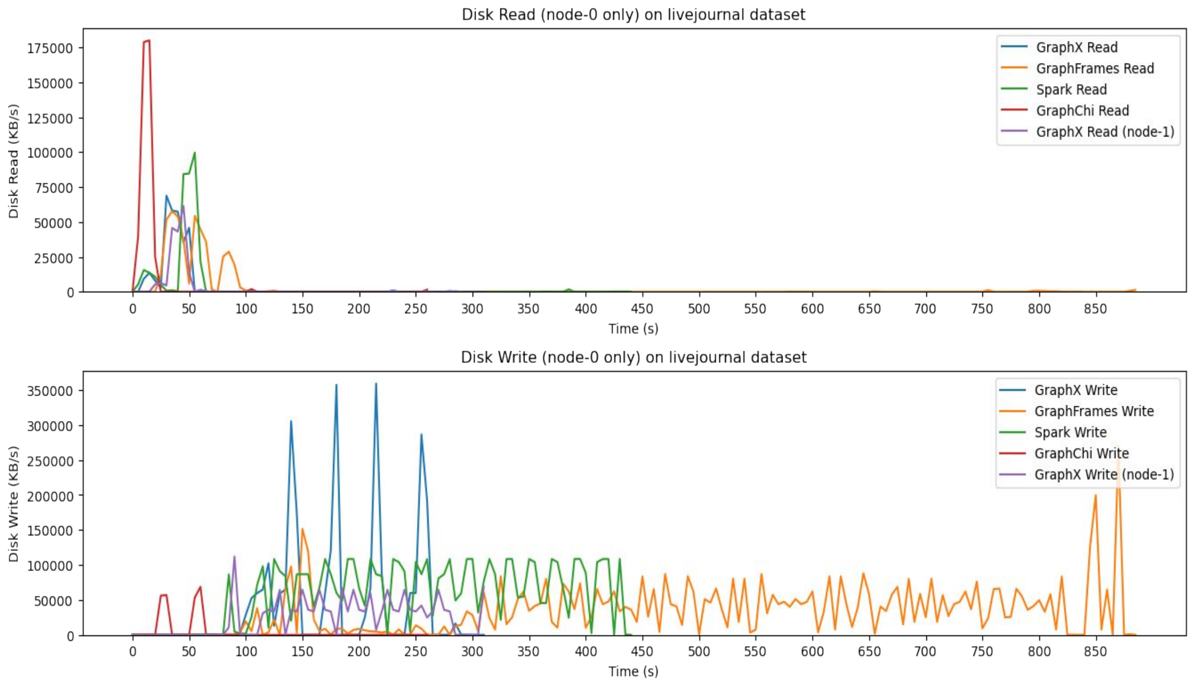




- GraphChi shows low CPU utilization even with very good performance
- In GraphX only nodes 1 & 2 are showing CPU usage. This is probably because GraphX partitioning is designed to minimize communication overhead. (We expect it to use all nodes for larger graphs)
- Spark and GraphFrames are showing high CPU usage

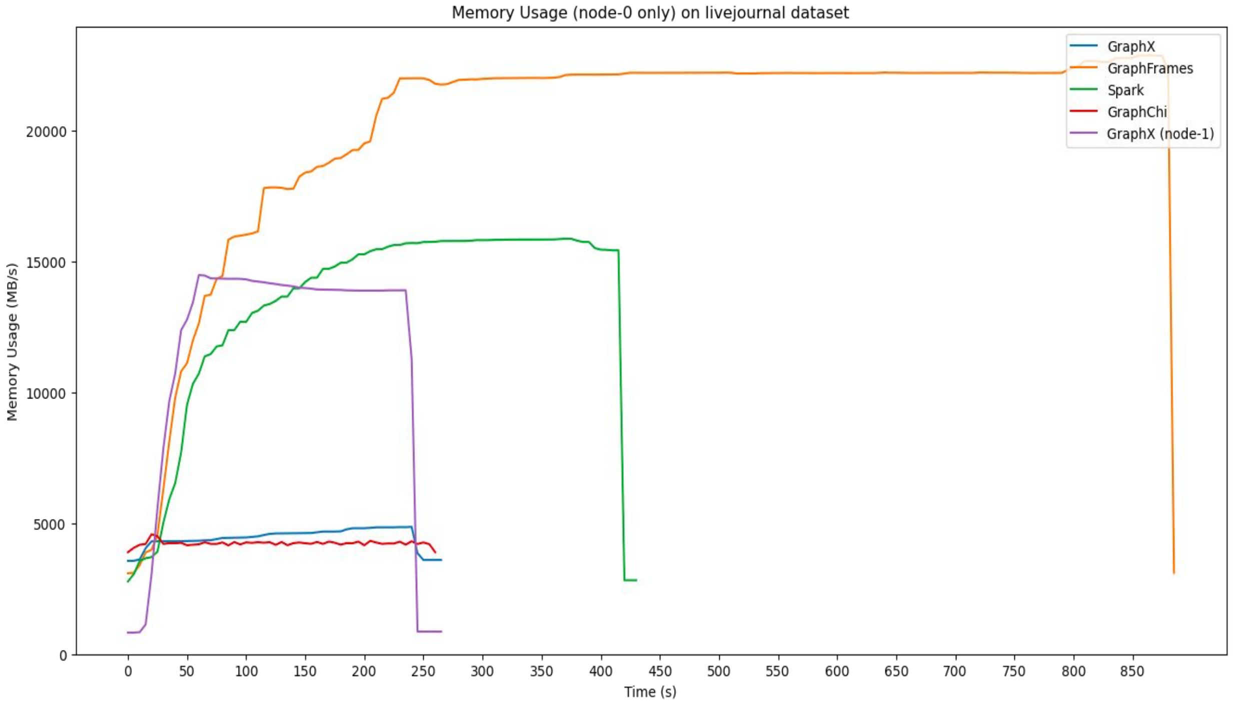
Disk Throughput

- GraphChi shows the highest Disk Read Throughput as it is designed to have sequential reads.
- In Disk writes, Spark and GraphFrames show consistent behavior. Whereas in GraphX we see huge spikes in node-0.



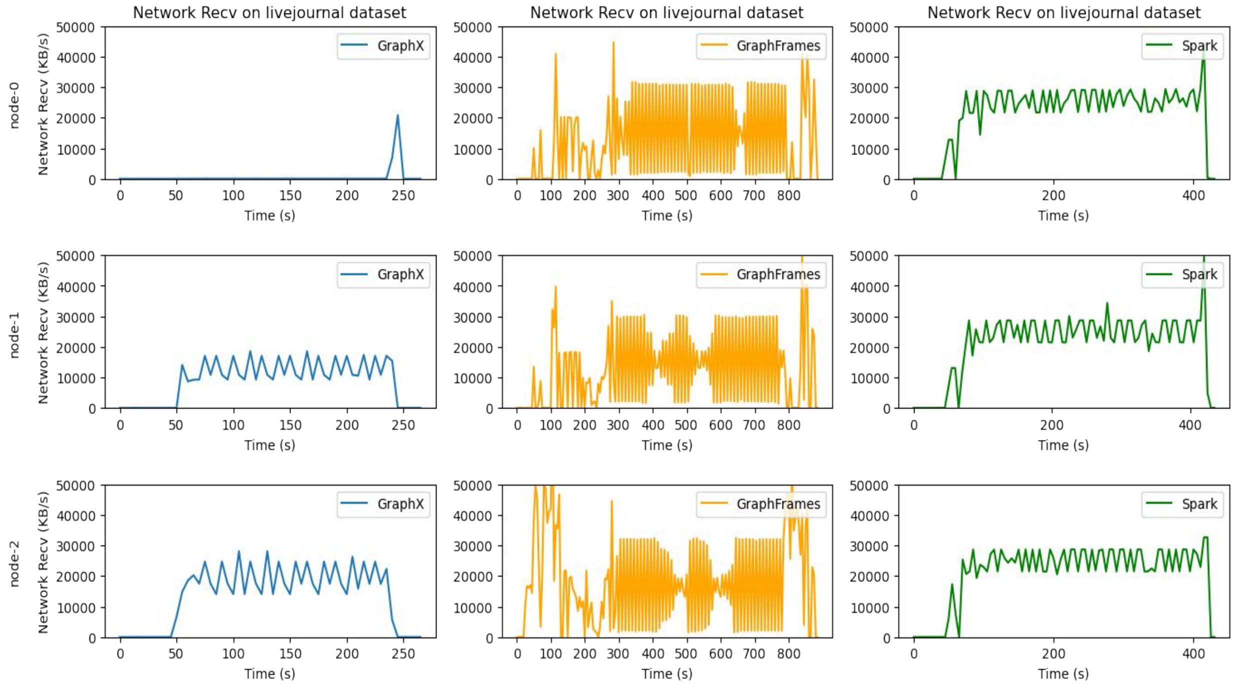
Memory Usage

- GraphChi has lowest memory usage which is expected as it is designed to have low memory usage
- GraphFrames shows the highest memory usage.
- GraphX (node 0 & 1 together) shows lower memory usage than Spark. This is probably due to its optimal partitioning and indexing optimizations.



Network Usage

- GraphX shows smallest network usage due to its optimal vertex partitioning which minimize communication overhead.
- GraphFrames shows high network utilization but is also very jittery.
- Spark shows constant network usage across all 3 nodes.

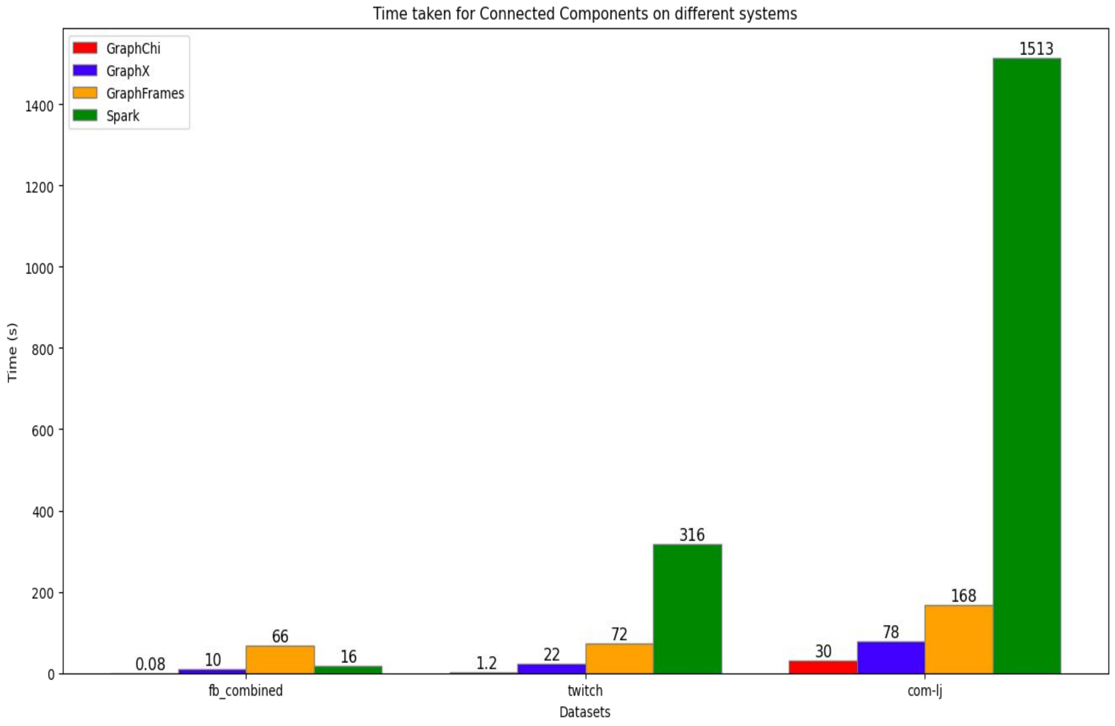


Connected Components

Datasets	Nodes	Edges
Social circles: Facebook	4,039	88,234
Twitch Gamers Social Network	168,114	6,797,557
LiveJournal network	3,997,962	34,681,189

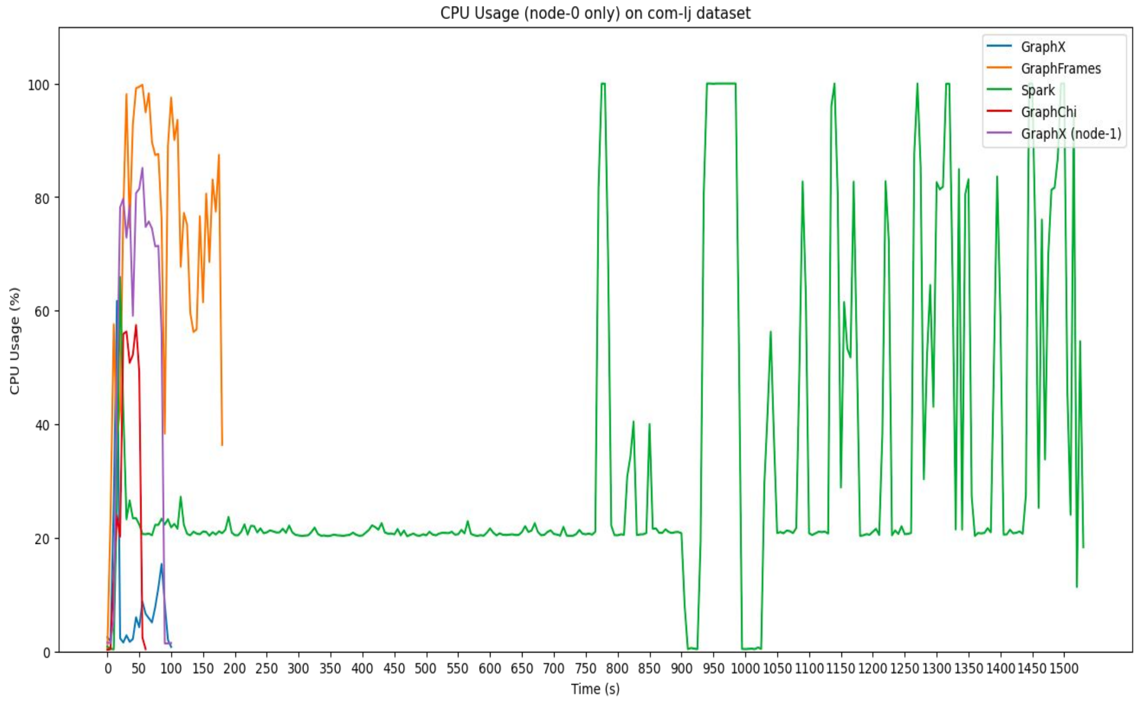
Execution Time

- GraphChi has the best execution time
- GraphX has fastest execution in multi-node systems that we compared..
- Spark only code implements large-small star algorithm which has worst performance.



CPU Usage (Master node)

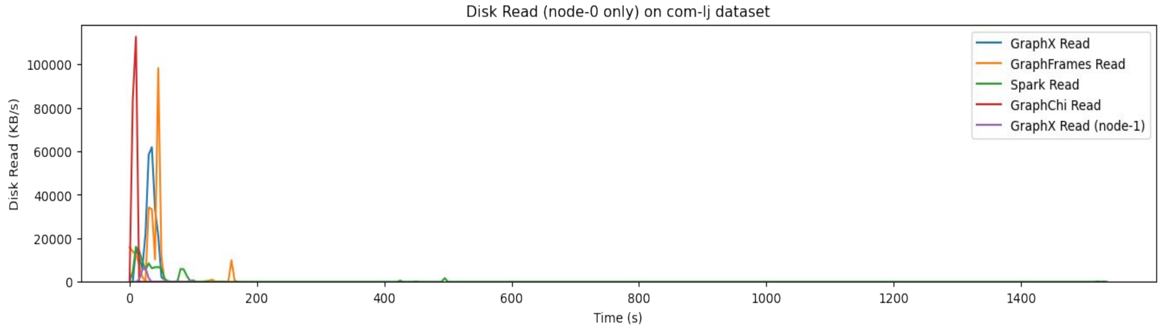
- GraphChi CPU usage at 60% but has the best performance
- GraphX shows less CPU usage for node-0 but shows a good utilization in node-1
- Spark has constant cpu usage during the time it reads data, has peaks after that.

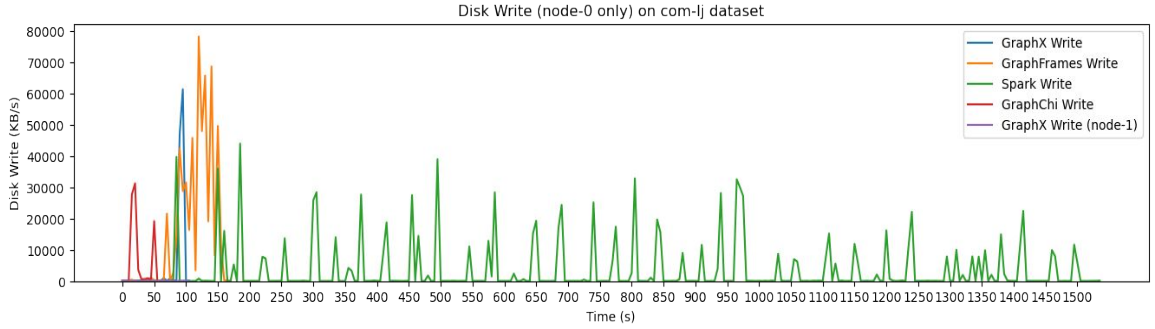


Disk Throughput

- All the systems show expected read behavior. With each system loading data at the start.

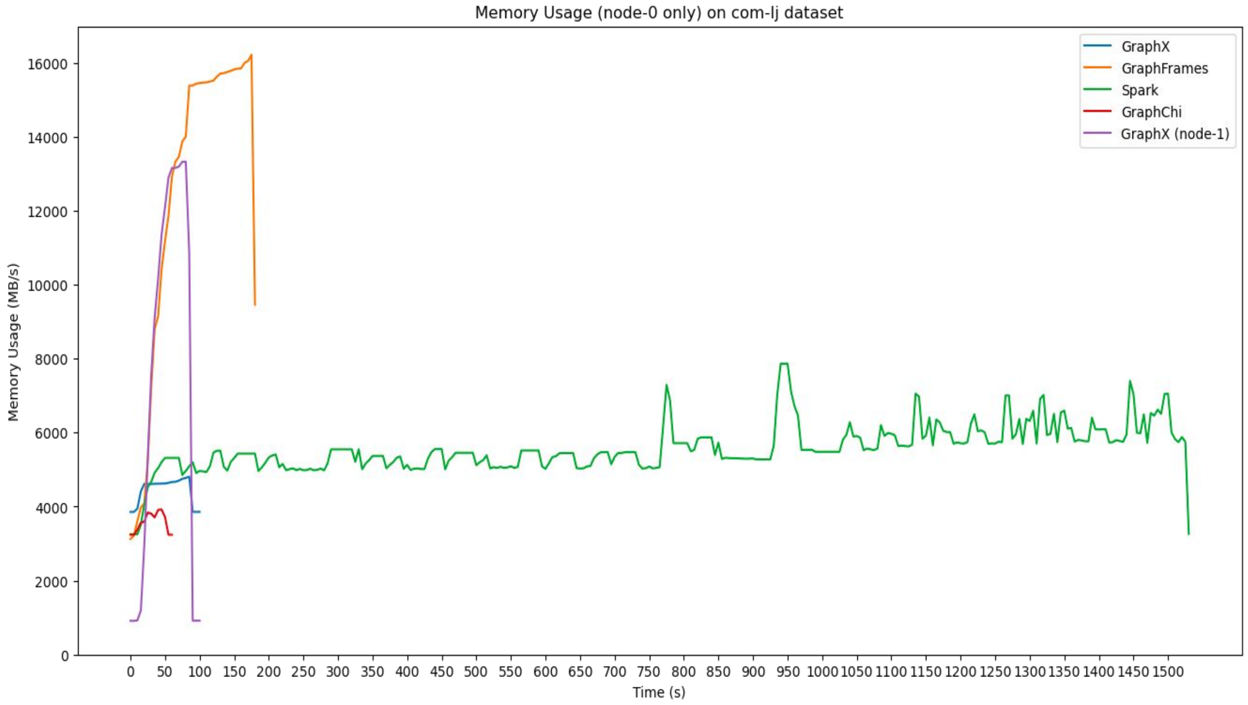
- GraphFrames shows high disk throughput towards the end.
- GraphX node-0 also showing a spike in disk write towards the end.





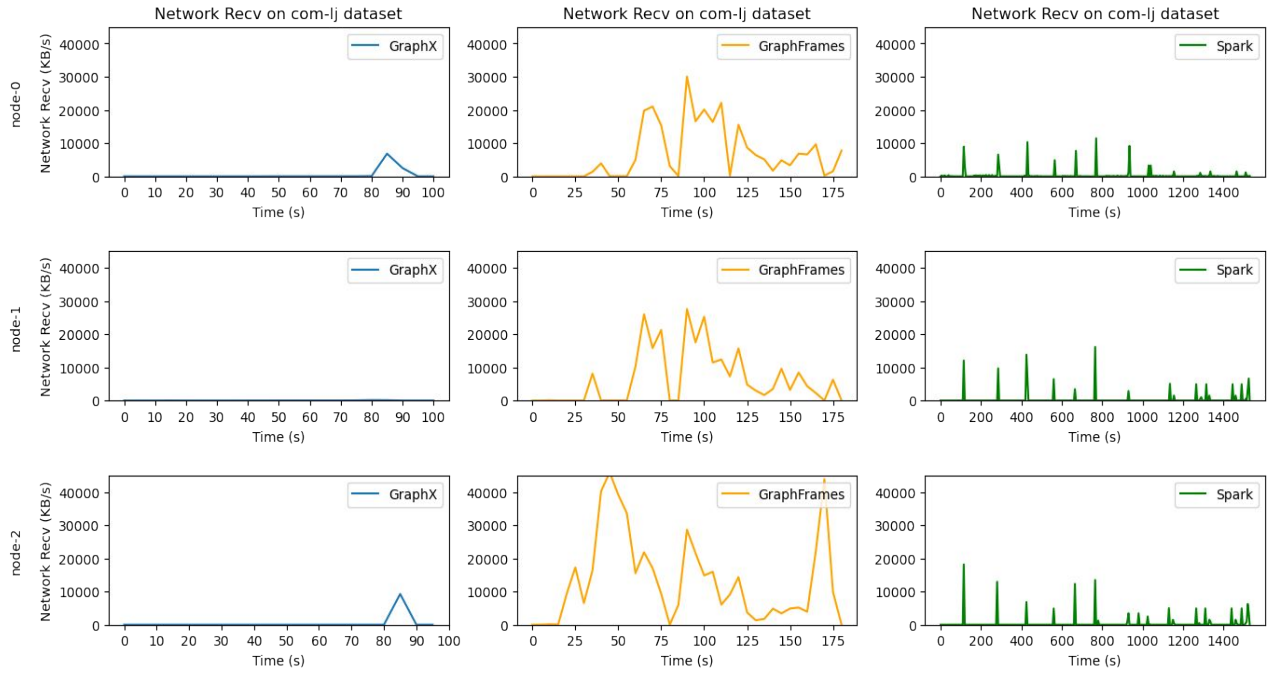
Memory Usage

- GraphChi is the fastest system and it has very nominal memory usage as expected
- GraphFrames is having highest memory usage followed by GraphX
- Spark is having a constant memory usage throughout the job.



Network Usage

- GraphX has lowest network activity across all systems.
- GraphFrames on the other hand has high network activity suggesting its partitioning might not be optimal.
- Spark's behavior is in spikes at regular intervals.



Triangle Counting

Datasets	Nodes	Edges
Social circles: Facebook	4,039	88,234
Twitch Gamers Social Network	168,114	6,797,557
LiveJournal network	3,997,962	34,681,189

- Skipping this considering time constraints
- Similar behavior as pointed out till now
- More details in the report

Conclusion

- GraphChi seems to be the fastest system for all graph size. Except for pagerank where large graphs might perform better on GraphX.
- GraphChi is designed for low memory usage, which is reflected in its plots. But due to this design, it needs to read data from disk more frequently which might be the reason for its CPU usage bottleneck to around 60%.
- GraphX shows lowest network activity across multi-node systems. This is because it’s vertex partitioning and data sharing is designed to have low communication overhead.
- GraphFrames has high network usage which might suggests that its data partitioning might not be most optimal. Also it has highest CPU and memory usage.

Thank You