Parallelization of Kempe's Influence Maximization Algorithm

EE451 Fall 2019

12/5/2019

Yi Sui, Pengcheng Li, Liheng Lin

USC Viterbi
School of Engineering

University of Southern California

Introduction to Influence Maximization

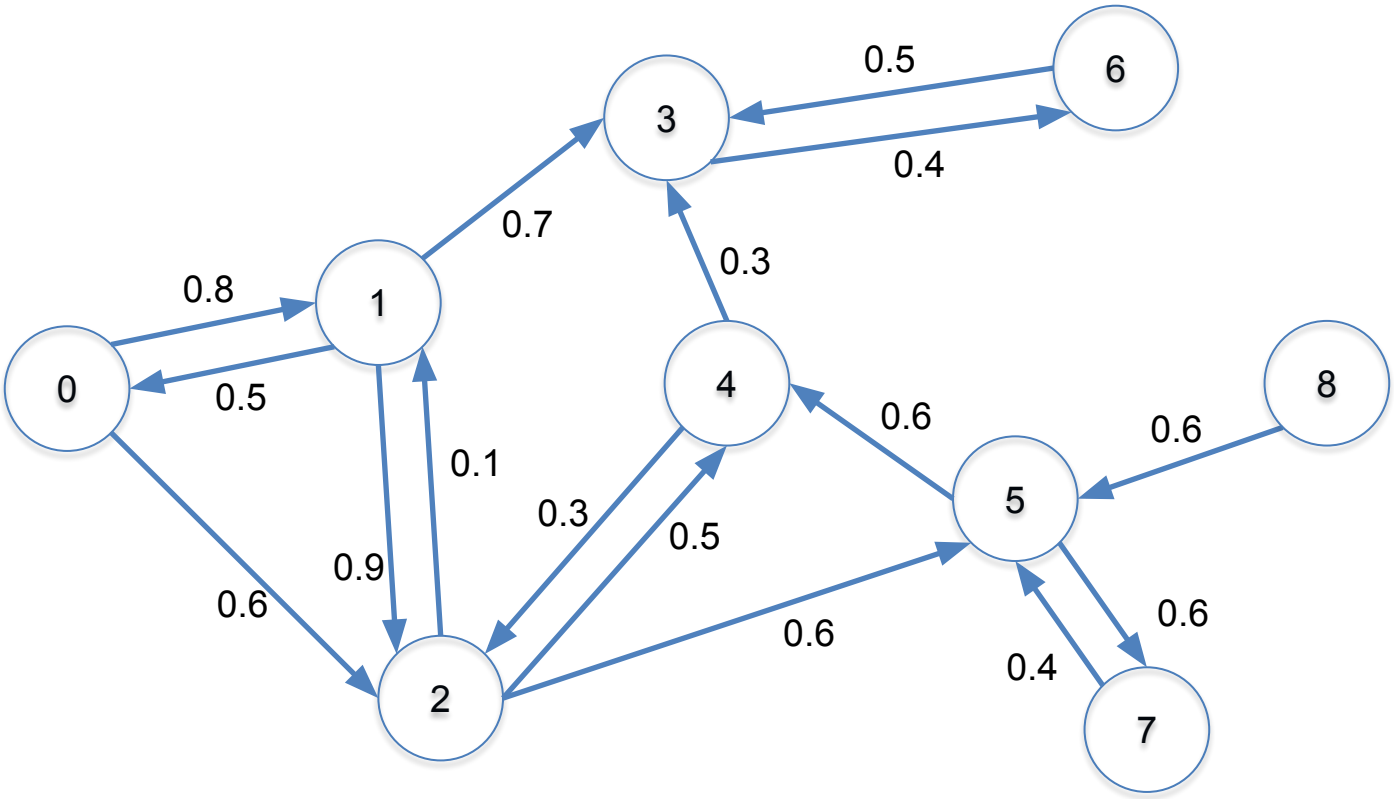
- Social Network & “Influencer”
- Spread of Information, ideas and influence
  - The use of new product
  - A new government policy
  - News
- Maximization of Information, ideas and influence
  - Different influence from one to another
  - Choose key individuals(best influencers)

Influence Maximization

- Weighted directed graph $G = (V, E, w)$
  - $0 \leq w \leq 1$
  - Weight: probability that a vertex can influence another

- Influence of a vertex v
  - A vertex influences neighbors with given probability
  - Might succeed to influence or fail
  - Successfully Influenced vertices continue to influence their neighbors
  - σ(v) = The number of vertices being influenced

Graph



Influence Maximization

- Influence of a set S
  - Every vertex in S have an influenced set
  - σ(S) = The size of union of influenced sets

- Influence Maximization:
  - Given a maximum set size k
  - Return a set of size k with maximum influence σ()

Influence Maximization Problem

- NP-hard
  - Even an approximate algorithm runs for days when there are billions of vertices!
- Large graph
  - May not fit in the memory

Need parallelization to solve the time and space issue!

Kempe's Influence Maximization Algorithm

- Approximation Algorithm

- Greedy Strategy
  - Maintain a seed set
  - In each step, add a vertex that helps maximize the influence
  - Stop until maximum seed size is achieved

Kempe's Influence Maximization Algorithm

```
S = ∅
while |S| < k:
    for each vertex v in V-S:
        simulate σ(S ∪ {v}) by sampling for a large number of times
    end for
    v *= argmax σ(S ∪ {v})
    S = S ∪ {v*}
end while
return S
```

This algorithm needs a large amount of computation for a mid-size graph

Kempe's Influence Maximization Algorithm

S = ∅

while |S| < k:

for each vertex v in V-S:

simulate σ(S ∪ {v}) by sampling for a large number of times

Independent Work: Parallelizable!

S = S ∪ {x}

end while

return S

Parallelization

3 parallelization approaches:

- Static Mapping
- Dynamic Mapping
- Graph Partitioning

Mapping

```
S = ∅
while |S| < k:
    for each vertex v in V-S:
        simulate σ(S ∪ {v}) by sampling for a large number of
times
    end for
    v *= argmax σ(S ∪ {v})
    S = S ∪ {v*}
end while
return S
```

MPI - Static Mapping

- K vertices and P processes, each process is responsible for a block of K/P vertices
- Each process stores the entire graph locally
- Parallel:
  - Each process begins simulation from one vertex v in its block
  - Calculate average influence of v
  - Find out v with max influence in its block
- Gather P vertices with max avg influence from P processes
- Add the vertex with global max influence into Seed

MPI - Dynamic Mapping

- Hypothesis:
  - Imbalanced workload due to different probability
  - Dynamic Mapping for balanced workload to reduce runtime
- Master-worker model
- Instead of evenly dividing the vertices, master assign new vertices(a block) to workers whenever they finish current work
- Communication cost increases but idle time decreases

MPI - Graph Partitioning

- K vertex and P processes, each process is responsible for a block of K/P vertices
- Only read the assigned part of graph into memory
- Communicate when influencing a vertex stored in other processes
- Communiation time increases but memory access time decreases
- Better performance in larger graphs

Platform

- MPI Model
  - Suitable for communication
  - Mapping: distribute influence simulations to workers
  - Graph-partition: not memory sharing model
  - While loop: not for OpenMP

Challenges

- Unfamiliar with MPI non-blocking send/receive
  - Synchronization issues
- Graph Partitioning
  - Partition & Synchronization
- HPC(testing & debugging)
  - Slow resources allocation

Raw Dataset

- *Bitcoin OTC trust* network example from the Stanford Large Network Dataset Collection
  - A trust network of Bitcoin investors from the online trade platform Bitcoin OTC
  - Buyer & seller can rate each other after a transaction
    - range: ± 10 (edge weight)
- Weighted signed directed temporal network
- 5,881 vertices and 35,592 directed edges
- 89% edges are positive

Raw Dataset

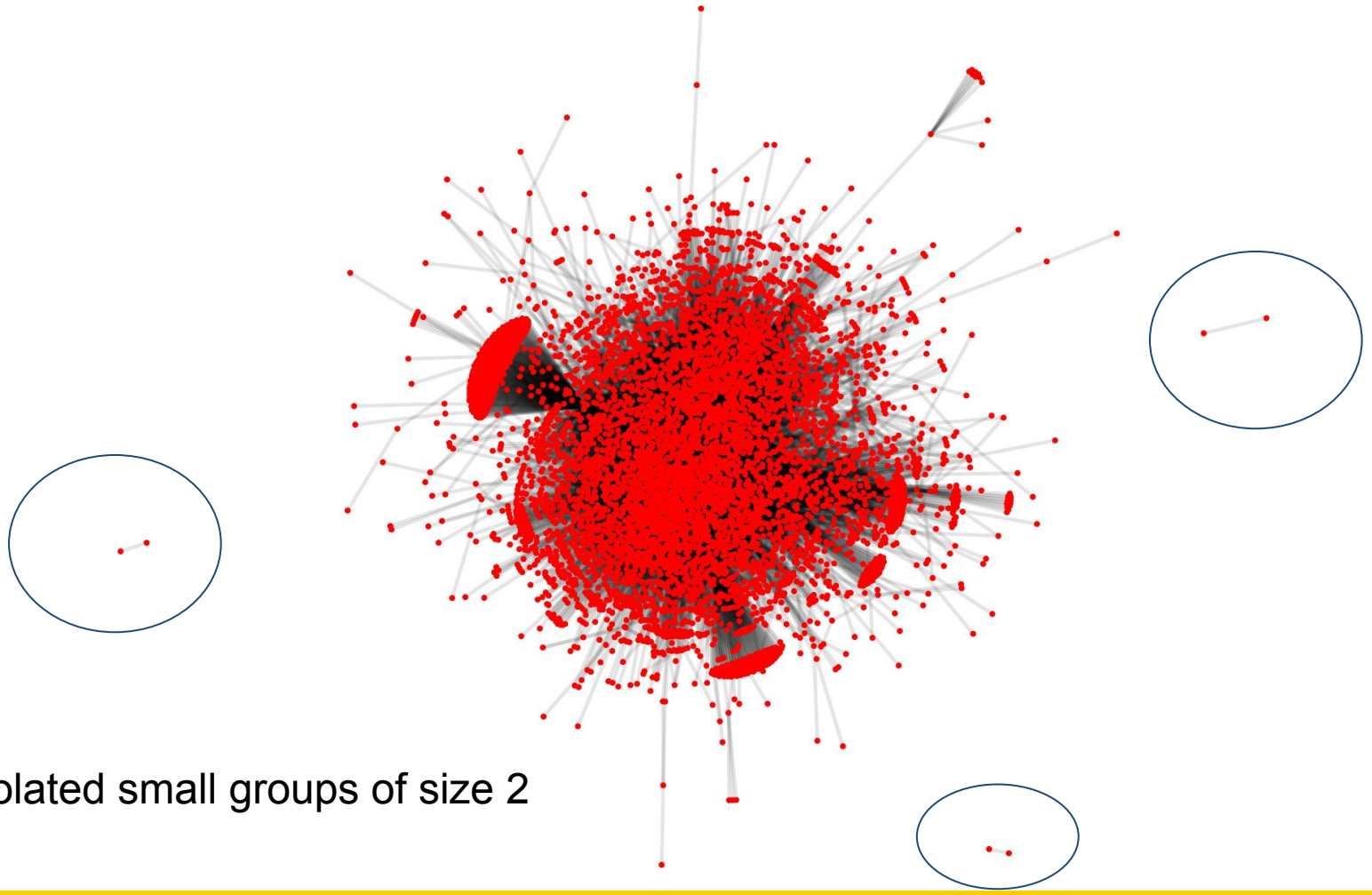
```
source,target,rate,timestamp
6,2,4,1289241911.72836
6,5,2,1289241941.53378
1,15,1,1289243140.39049
4,3,7,1289245277.36975
13,16,8,1289254254.44746
13,10,8,1289254300.79514
7,5,1,1289362700.47913
2,21,5,1289370556.80938
2,20,5,1289370622.21473
21,2,5,1289380981.52787
21,1,8,1289441411.46365
21,10,8,1289441438.4426
21,8,9,1289441450.59006
```

Dataset

- Processed into a weighted directed network
- 5,881 vertices and 35,592 directed edges
- Shift & normalize edge weight(rate) from ± 10 to [0, 1]

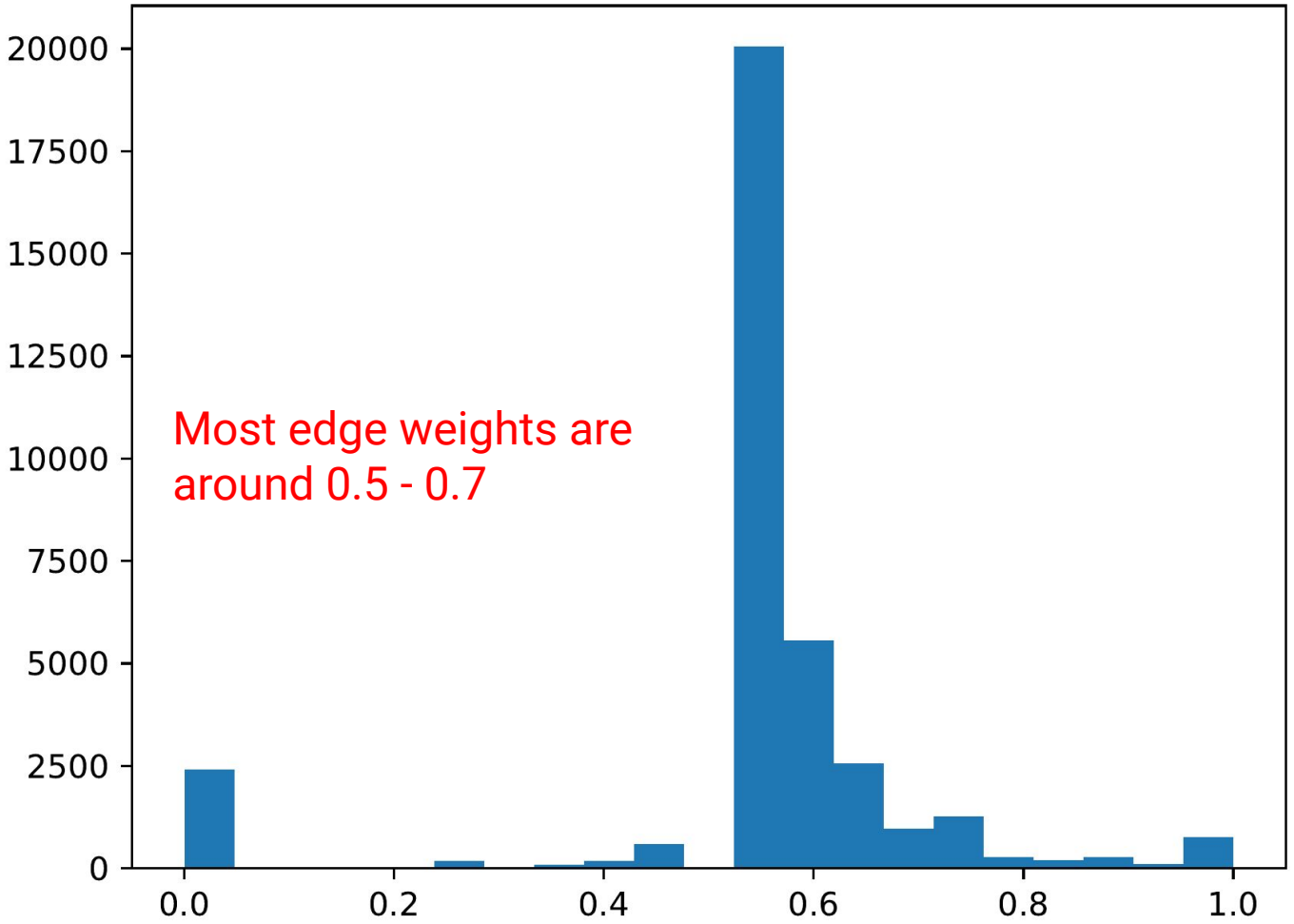
```
source,target,weight
1,2,0.9
1,3,0.8
1,4,1.0
1,5,0.7
1,6,0.9
```

Dataset - Graph



3 isolated small groups of size 2

Dataset - Distribution



Explanation of Parameter

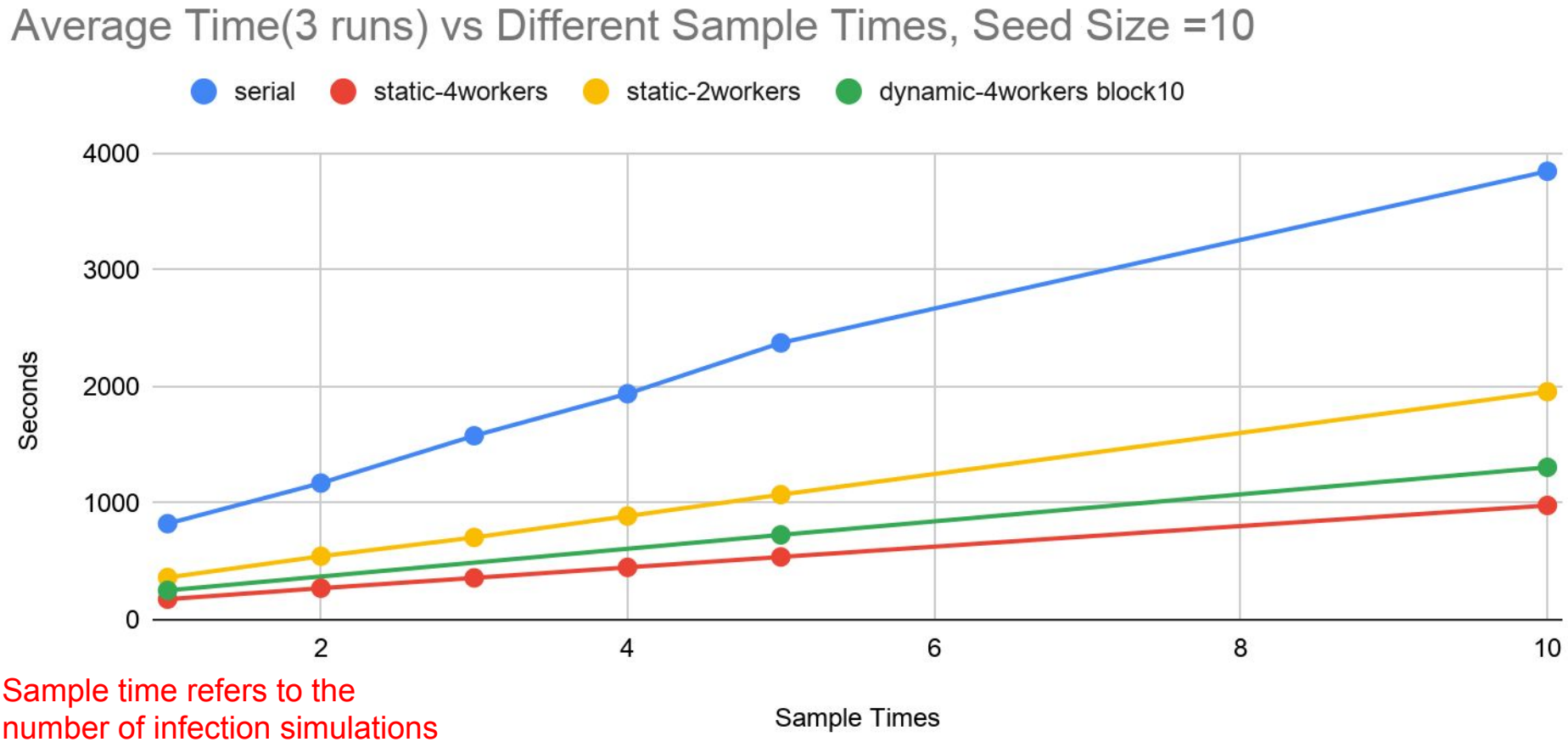
- Seed size(Input parameter)
  - How many vertices to pick and return?
- Sample Times(Input parameter)
  - Repeat the influence process to improve accuracy
- Block size in Dynamic Mapping(parallelization parameter)
  - Balancing workload vs communication cost
- Number of workers(parallelization parameter)
  - Using 2 vs 4 workers

Testing Platform

- USC HPC
  - Serial: allocate 1 computing node
  - Parallel: allocate 2/4 computing nodes

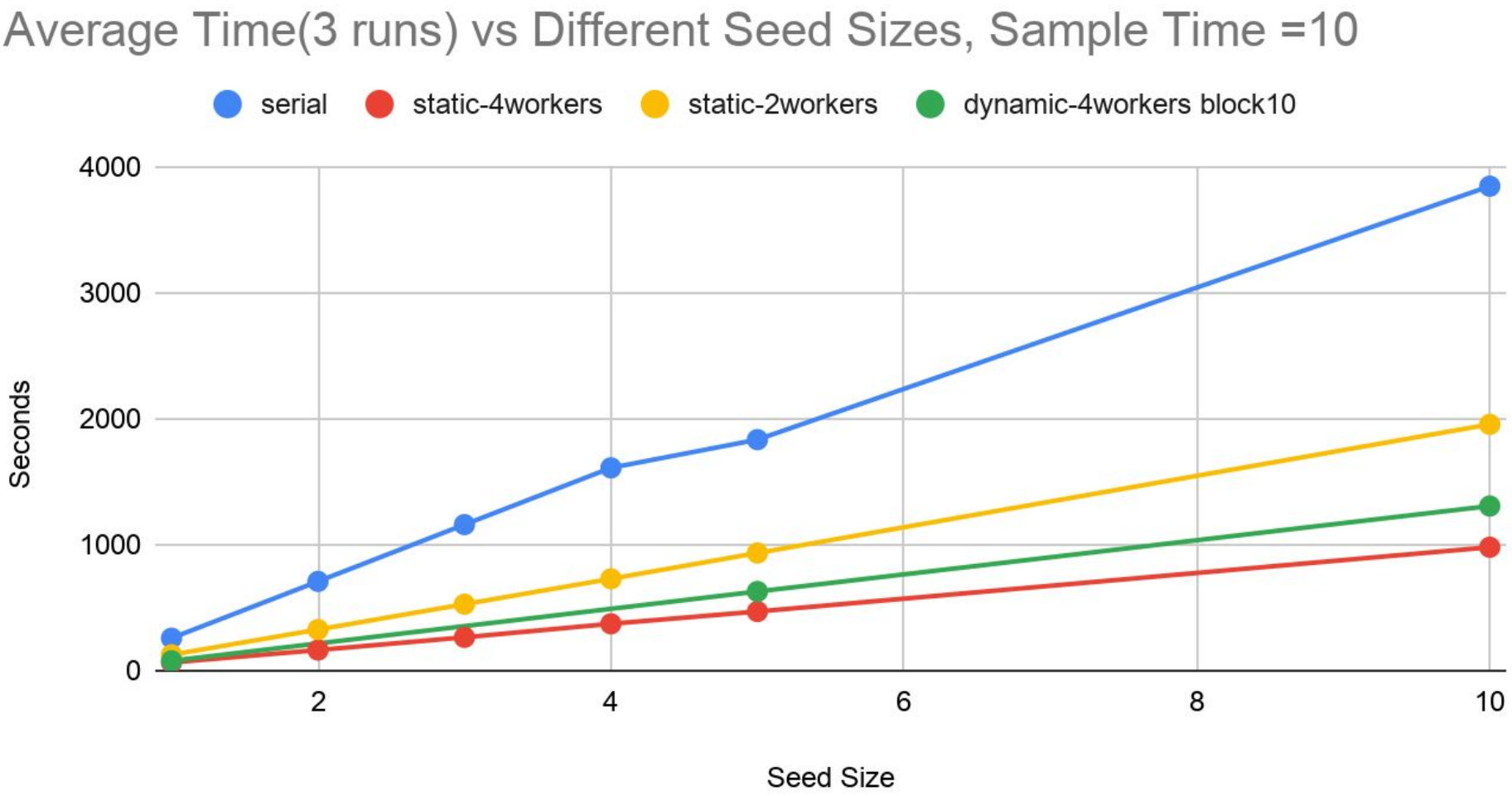
  - #SBATCH --ntasks-per-node=1
  - #SBATCH --nodes=1/2/4

Results

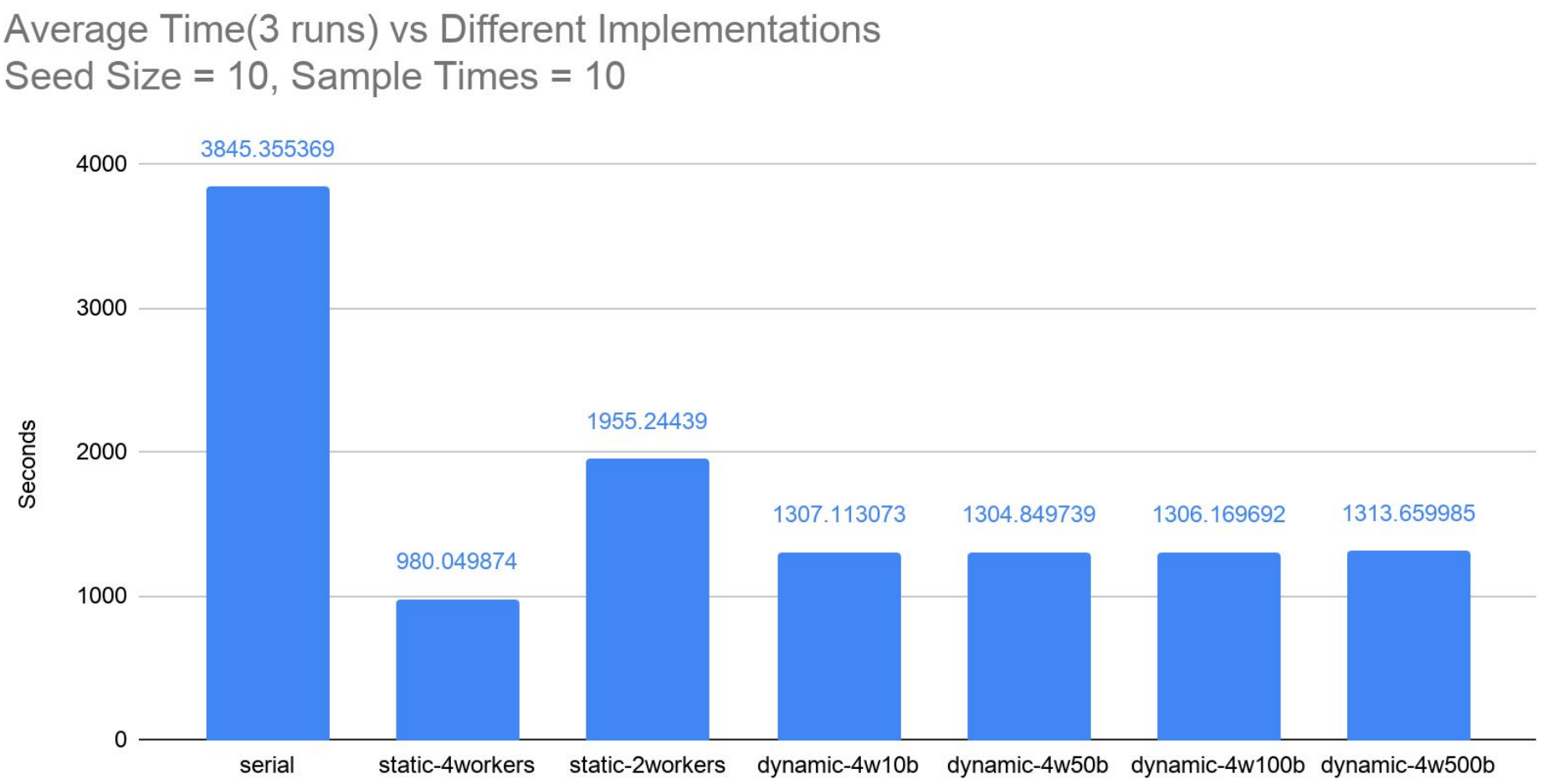


Sample time refers to the number of infection simulations in Kempe's algorithmm

Results


Average Time(3 runs) vs Different Seed Sizes, Sample Time =10
serial
static-4workers
static-2workers
dynamic-4workers block10
4000
3000
2000
1000
0
Seconds
2
4
6
8
10
Seed Size

Results


Average Time(3 runs) vs Different Implementations
Seed Size = 10, Sample Times = 10
Seconds
4000
3000
2000
1000
0
3845.355369
980.049874
1955.24439
1307.113073
1304.849739
1306.169692
1313.659985
serial
static-4workers
static-2workers
dynamic-4w10b
dynamic-4w50b
dynamic-4w100b
dynamic-4w500b

Result - Graph Partitioning

- Implementation is complete
  - Results are correct, but performance can be improved
  - Communication & synchronization costs
    - Barrier for synchronization
    - Influence steps moving between vertices in different processes - extra communication

Analysis

- Both Static Mapping and Dynamic Mapping are faster than the serial version
  - Static w/ 4 workers: 3.9x Speedup
  - Static w/ 2 workers: 2.0x Speedup

- Dynamic w/ 4 worders: 2.9x Speedup

Analysis

- Dynamic Mapping is slower than Static Mapping
  - One process serves as Master
  - Master does not compute

- Dynamic Mapping with different block sizes:
  - Time difference was ignorable
  - Communication cost doesn't affect much in testing
  - Possible causes:
    - Workload is not very unbalanced
    - Graph is not large enough to show the difference

Conclusion

- We have implemented:
  - Serial version of an Influence Maximization Problem
  - Several parallelized versions

- We learn…
  - MPI Programming

Question?