Team Members

Nikhil Tiware :- UFID 91507670

Parikshit Tiwari :- UFID 79218564

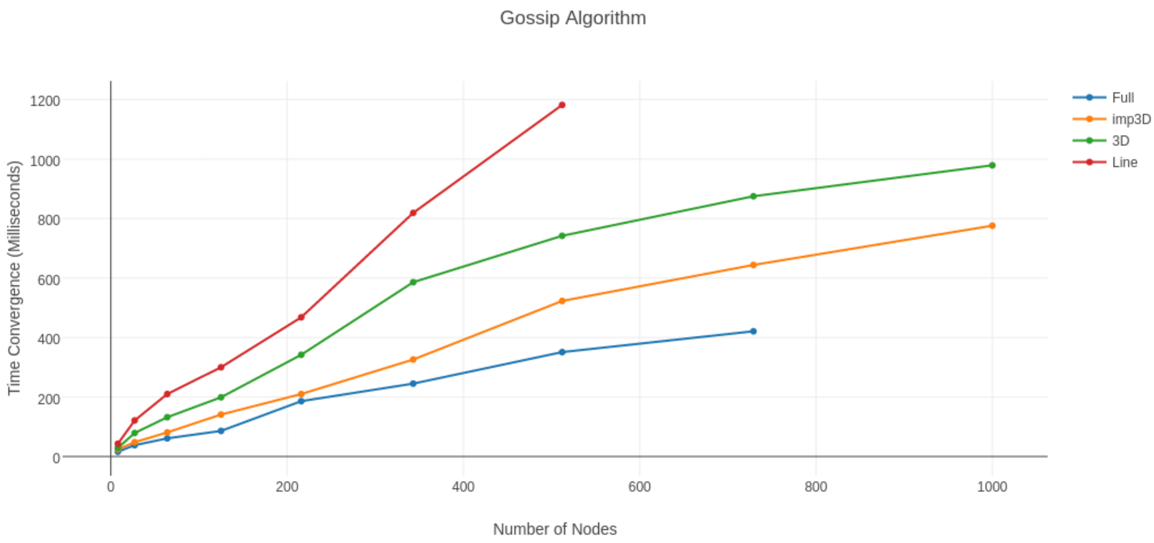
Gossip and PushSum Algorithms implementation report

Gossip Implementation: A node in gossip implementation stops transmitting once it has heard the rumour 10 times.

We have used Akka Schedulers[1] so that when a node hears a rumour for the first time, it continues transmitting the rumour randomly to its neighbours at 5 millisecond intervals, till it receives a rumour 10 times.

The following the report present the observations and inferences made while implementing the Gossip and PushSum algorithms for a 3D, Line, Full and imperfect 3D topology.

Combined Graph for Gossip Algorithm

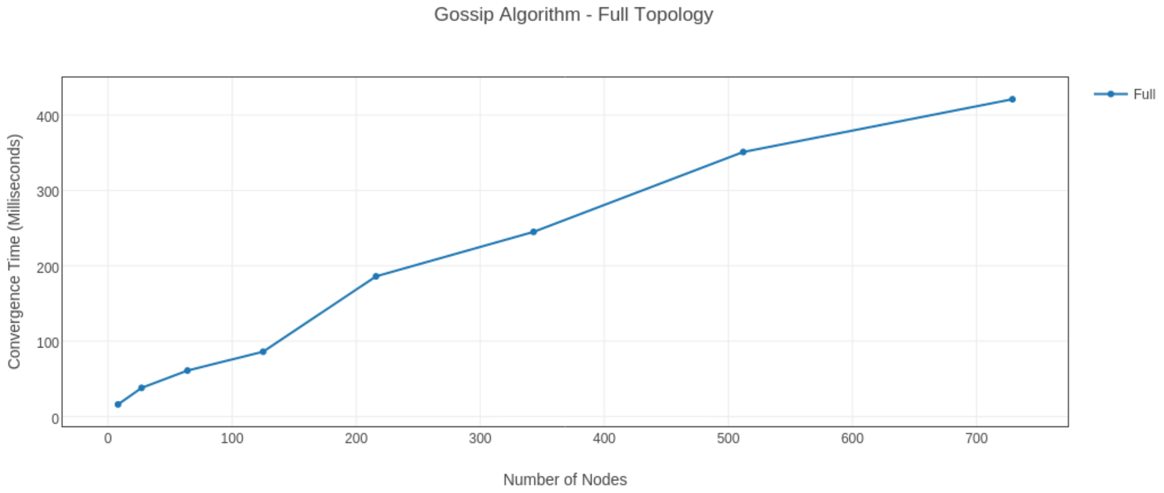


Observations

1) The above graph is based on the average Gossip convergence time of a particular number of nodes for a particular topology
2) T(Line) > T (3D) > T(imp3D) > T(Full)
   ,where T(x) is the convergence time for x topology
3) Topology with Fastest convergence time :- Full
4) Topology with slowest convergence time :- Line

Individual Graphs of Gossip

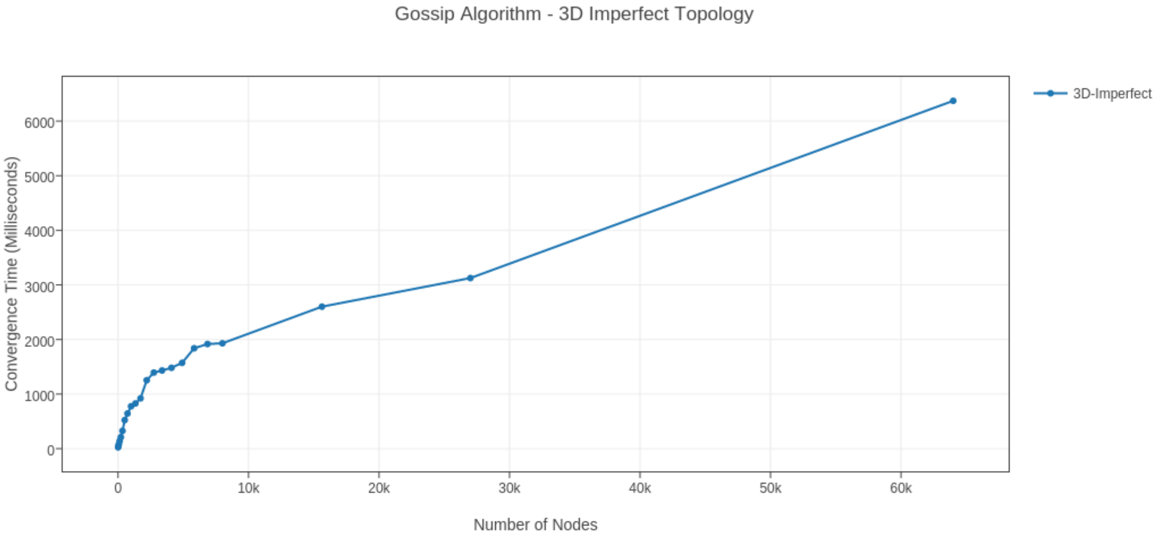
Full Topology



Observations

1) We were able to get convergence of upto 729 nodes for Full Topology in under 421 milliseconds

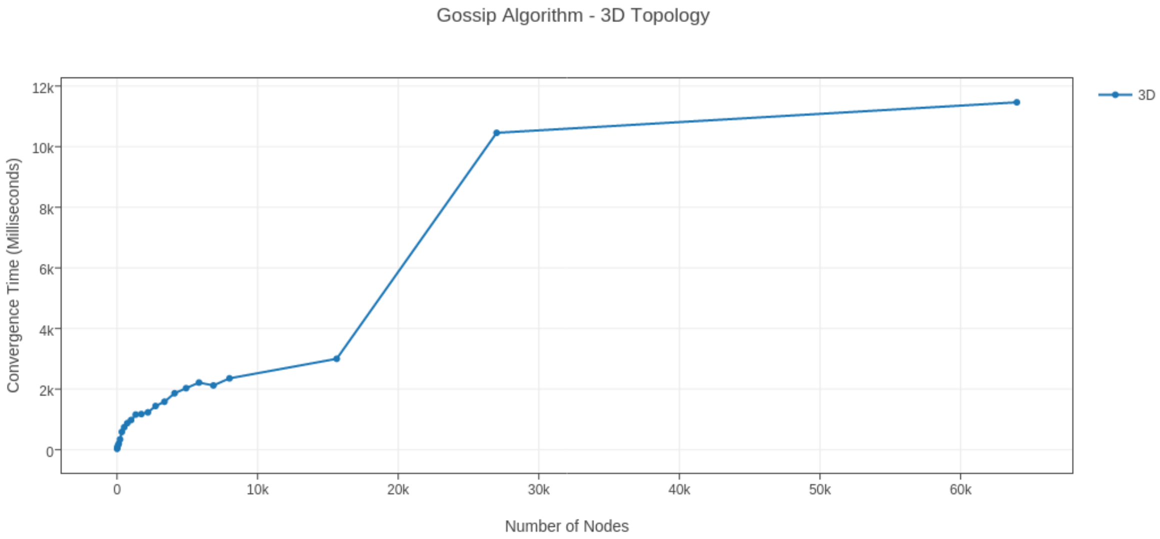
Imperfect 3D Topology



Observations

1) We were able to get convergence of upto 64,000 nodes for Imperfect 3D Topology in under 6372 milliseconds

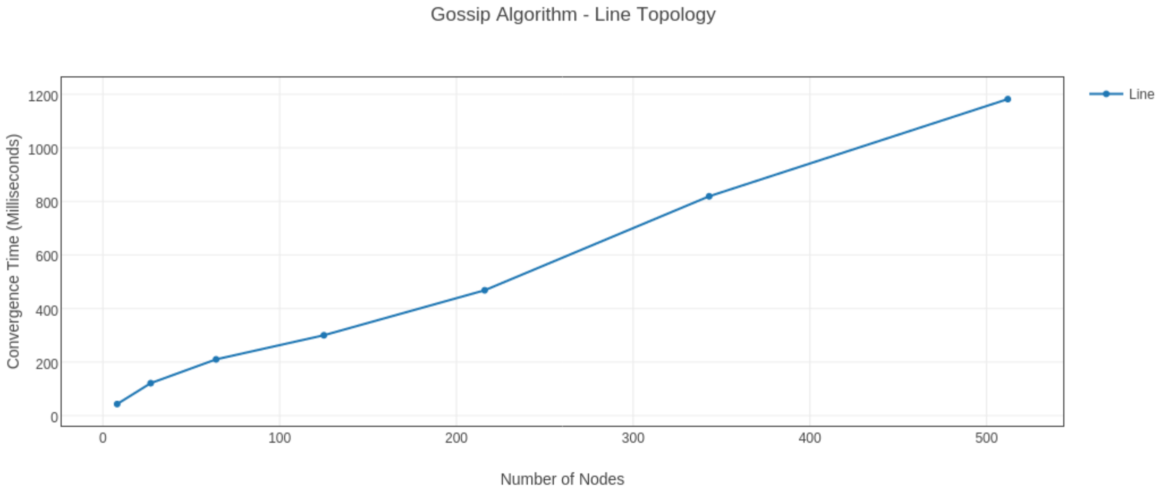
Perfect 3D Topology



Observations

1) We were able to get convergence of upto 64,000 nodes for perfect 3D Topology in under 11463 milliseconds
2) After number of nodes hits 15625 the convergence time increases suddenly up till the 25000 nodes mark after which convergence time stabilizes

Line Topology



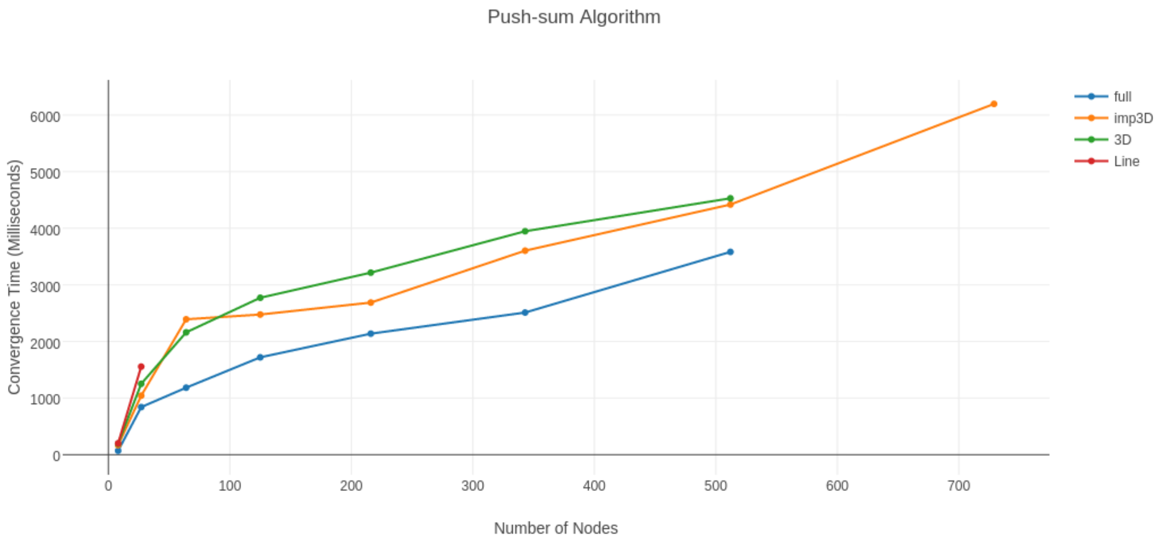
Observations

1) We were able to get convergence of upto 512 nodes for perfect Line Topology in under 1182 milliseconds

<u>PushSum Implementation:</u> A node in a PushSum implementation stops transmitting if the ratio of s/w doesn't change by 10^{-10} in three rounds.

Here as well have used Akka schedulers in a similar manner to Gossip.

<u>Combined Graph for PushSum Algorithm</u>

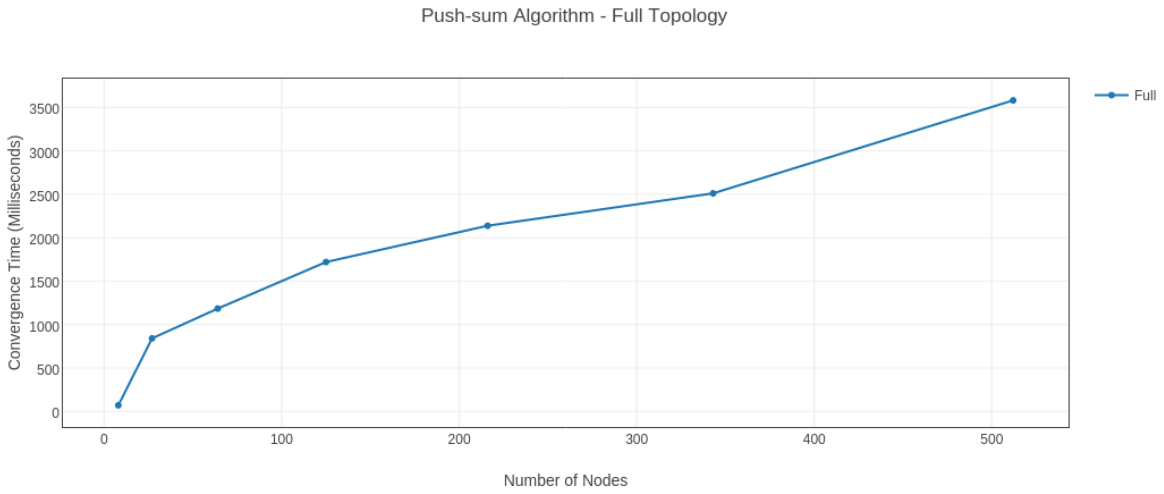


<u>Observations</u>

1) The above graph is based on the average PushSum convergence time of a particular number of nodes for a particular topology
2) For small number of nodes (< 64)there isn't much difference in the convergence times of the topologies
3) For number of nodes > 64, the convergence times begin to spread out. The convergence time of "Full" topology being the fastest. The convergence times of imperfect 3D being slightly faster than that of 3D.
4) Line topology fails to give 100% convergence for values above 27 nodes.

Individual Graphs of Push Sums

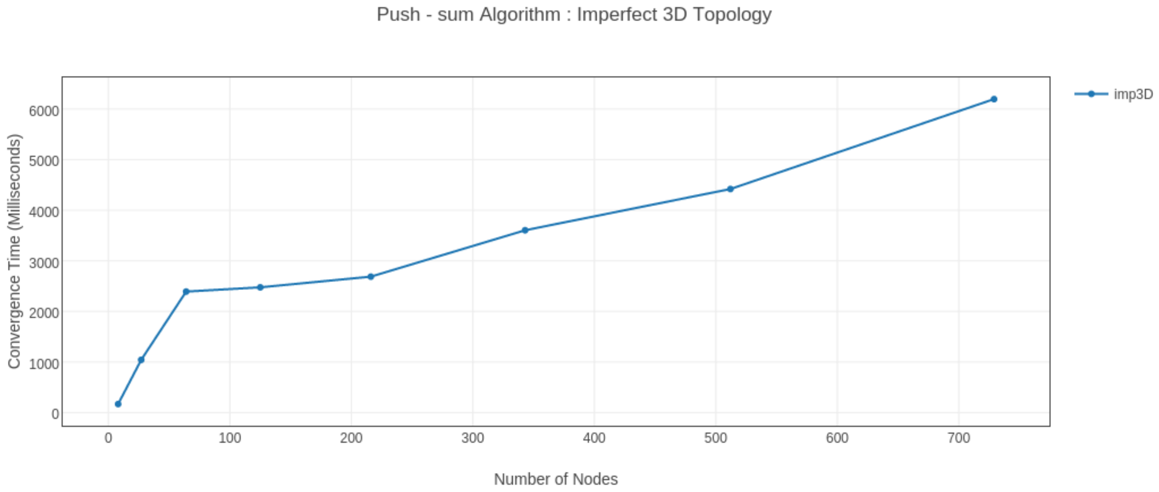
Full Topology



Observations

1) We were able to achieve convergence of 512 nodes in just over 3582 milliseconds

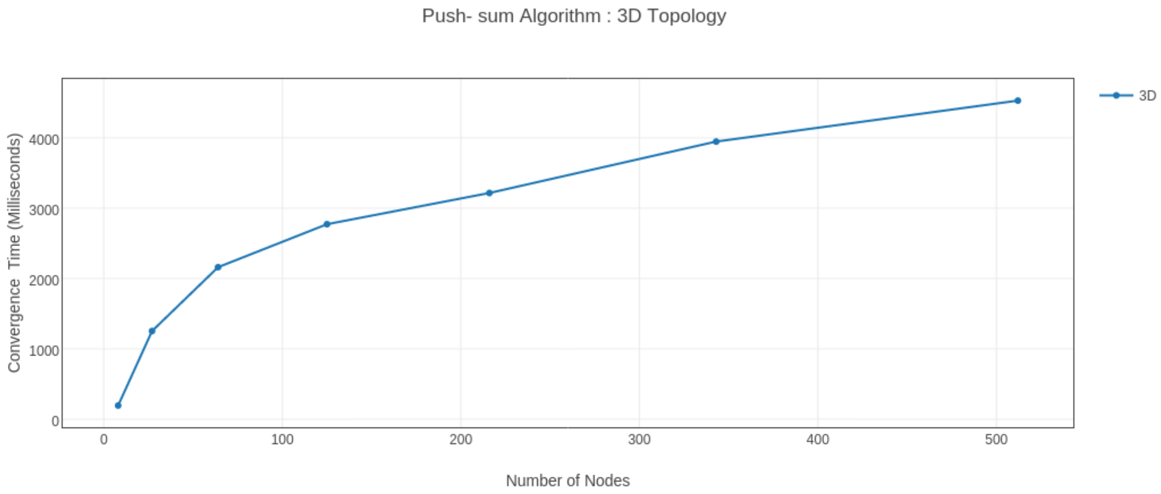
Imperfect 3D Topology



Observations

1) We were able to achieve convergence of 729 nodes in just over 6196 milliseconds

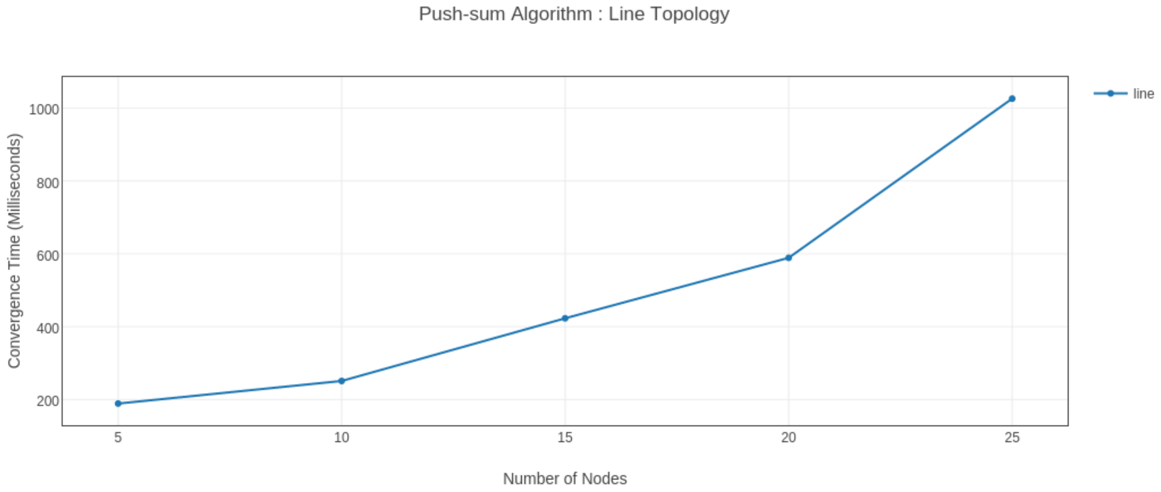
Perfect 3D Topology



Observations

1) We were able to achieve convergence of 512 nodes in just under 4529 milliseconds

Line Topology

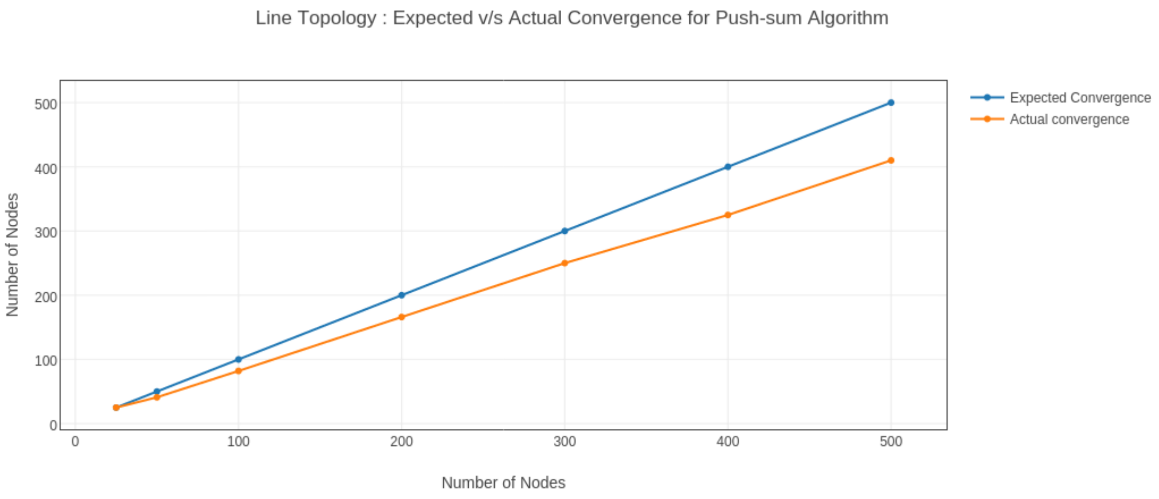


Observations

1) We were able to achieve convergence of 25 nodes in just over 1182 milliseconds
2) Beyond 25 nodes line topology rarely shows 100% convergence in its nodes

Interesting Observations

1) Convergence of Gossip is higher than that of PushSum
2) In case of Gossip for all topologies the nodes converge with near 100% accuracy. For full topology the lowest conversion value was 63998 out of 64000 nodes converging. An accuracy of about 99.997%
3) In case of PushSum convergence rate is 82%. The convergence is worst for Line topology with only 410 nodes converging from 512 nodes. Below is the graph of ideal vs real convergence rate of the Line topology for PushSum



4) As we can see from the above graph as number of nodes goes on increasing the convergence goes on decreasing
5) For Full and Line topology the maximum number of nodes we could get was 729 and 512 respectively. Beyond this we were getting "Java lang out of memory error" The solution to this is to increase heap size of JVM. If we had more time we could look into this issue

References:

[1] http://doc.akka.io/docs/akka/2.0/scala/scheduler.html