COP5615

10/1/2015

Pawel Cieslewski

Will Livesey

Project 2:

Simulating Group Communication using Gossip and Push-Sum Algorithms

Introduction

In class, Dr. Dobra mentioned how important it is to find a way to propagate information throughout a distributed system. He mentioned several different ways to this including: flooding, using a tree hierarchy, and gossip algorithms. Flooding is when a node constantly sends a message to all of its neighbors. Once a new node hears the message it'll also send it to its neighbors, this process continues until everyone has heard the message. Clearly, you can see this method is inefficient. It uses a lot computational power to continuously send the message to every neighbor. The next method Dr. Dobra covered was using trees to disseminate messages. While this approach is more efficient, it carries a lot of risk. Namely, if a node dies you lose its subtree. The final method Dr. Dobra outlined was the gossip algorithm, which is the focus of this project. Although this algorithm way seem random by nature, it has been proven to converge while being quite efficient and fault tolerant.

Implementation of Asynchronous Workers

Traditional implementations of the Gossip and Push-Sum algorithms involve nodes moving information in "rounds" where each node takes a step once per round. Our simulator has an implementation that is somewhat different and thus requires some acknowledgement.

Both Gossip and Push-Sum require a worker to transmit messages after they have received a message. The classical way is to have the manager coordinate the task. Instead, our workers wait for a “reply” message. See figure one.

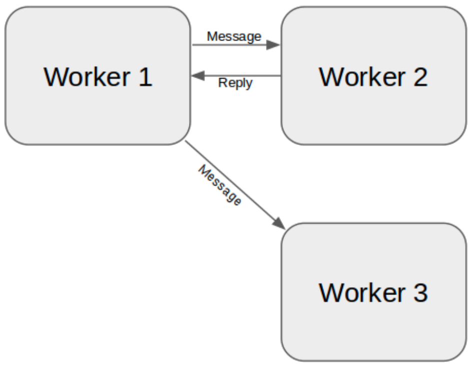

Figure 1. Relaxed Step-Based Worker Communication

Worker 1 will send a message to Worker 2 and Worker 2 will send back an empty “reply” message regardless of its current state. The next action that Worker 1 will take, such as messaging Worker 3, will occur when Worker 2 has sent back a “reply.” This allows Worker 1 to know that the message it sent last was processed.

A flawed implementation could be for Worker 1 to send itself a message as a reminder to take more steps. However, this causes irregular behavior as some actors may end up taking many more steps than others.

While our implementation does not guarantee that all workers in the system will have taken the same number of steps, it produces a similar trend. We call this “relaxed step-based communication”.

Analysis of Convergence Rates

After running a variety of simulation, we have compiled the aggregate results in table 1.

Number of Nodes	Imp3D (ms)		3D (ms)		Full (ms)		Line (ms)	
	Gossip	Push-Sum	Gossip	Push-Sum	Gossip	Push-Sum	Gossip	Push-Sum
27	20	87	31	55	20	41	45	80*
343	96	198	120	281	29	75	65	249*
2744	128	1378	144	1131	91	308001	1163	358*
27000	448	44638	561	24261	N/A	N/A	21182	260*

Table 1. Results of the algorithms for the respective topologies

Figure two depicts the data shown for the gossip algorithms in table one. Please note the y-axis is logarithmic due to the high variance of data points for the respective topologies.

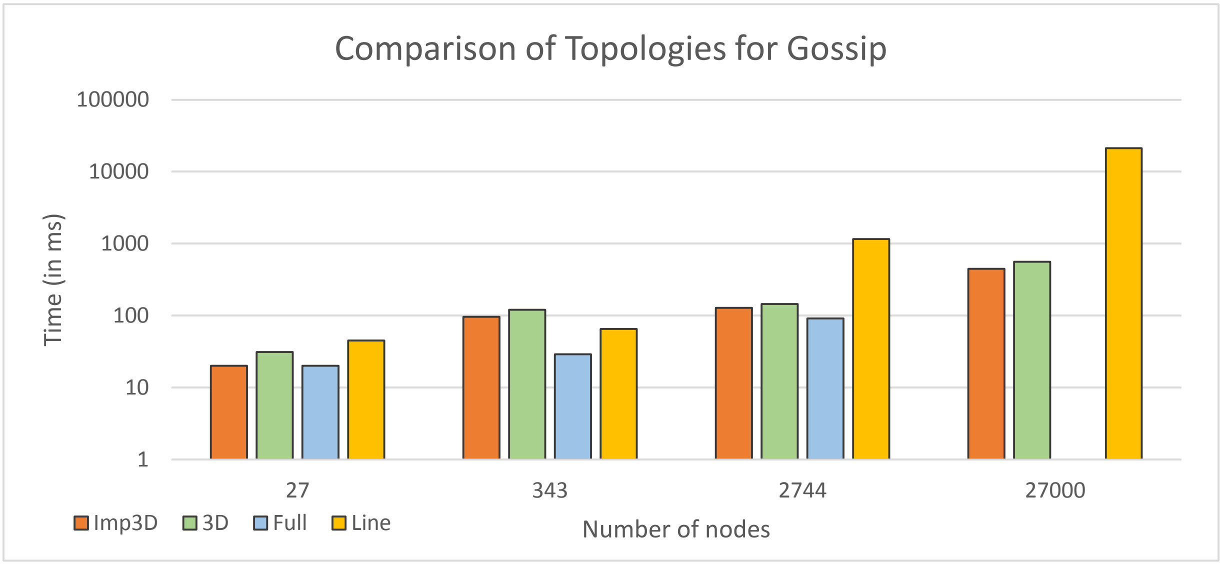

Figure 2. The results of the gossip algorithm. Note that for 27,000 nodes the line topology was unable to converge due to the high number of connections needed (27,000 * 27,000) to initialize it

3D Topologies:

Both of the 3D topologies follow the same quadratic trend. However, the imperfect 3D topology exhibits a slight increase in performance. This is due to the fact that messages can jump long distances on the network, rather than just propagating through their neighbors.

Full Topology:

This topology has the best performance; however, it also has the most overhead. We were unable to calculate the convergence when there were 27,000 nodes in the system because of this. Each node has to be able to communication with every other node in the system, meaning 27,000*27,000 connections have to be established before the algorithm can even begin. It's also likely that an out of memory exception can occur due to the large number of references that have to be generated.

Line Topology:

It is intuitive and evident in the data that the line topology is the worst performing topology. In three out of four cases it was slower than the other three topologies. The line topology performance is hindered by the fact that it can't have several different areas working simultaneous. For example, in the 3D topology the system can reach multiple nodes in the same step; whereas, the line topology is limited to a single direction.

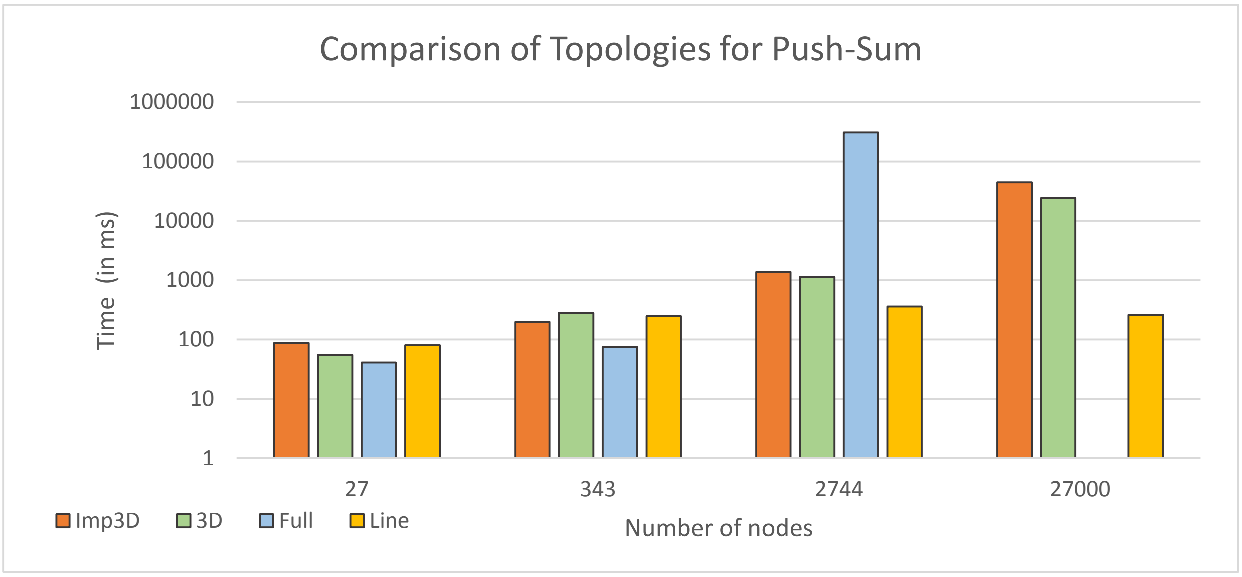

Figure 3. The results of the push-sum algorithm for the respective topologies

3D Topologies:

We should point out that while the imperfect 3D topology was better in the gossip algorithm, it is now worse than the 3D topology for the push-sum algorithm. This is due to the fact that the 3D topology is more prone to local convergence. The random connection in the imperfect 3D topology caused sudden changes in sum estimate. Therefore while the 3D topology can converge with a gradient throughout the system, the imperfect 3D topology will favor total system convergence.

Full Topology:

The full topology has the worst performance of any once again due to the need for total convergence. Every node is connected to every node and therefore the system must wait until the ratio converges with all of them. Interestingly, if you watch the nodes during the simulation of the full topology, you will notice a sudden burst where a majority die. This is the point at which the system reaches convergence.

Line Topology:

The line topology for the push-sum algorithm is essentially meaningless. As you traverse the nodes in the line, the probability that a crucial node will die increases. Thus, it is improbable that the message can even reach all of the nodes in the system using this topology.

Difficulties of Implementation and Future Work

One of the hurdles of this project was understanding how to synchronize and coordinate tasks in a fully asynchronous system. At first we ran into several problems where work would be unequally distributed resulting in a waste of computational power and introducing some memory leaks. Some workers would have an infinitely growing mailbox due to lack of coordination.

Another challenge was accurately timing the simulation while accounting for the time it takes to build the topologies and ensuring that the end time was the time of the last heartbeat. Fortunately, fixing this issue only required more coordination from the manager.

One of the ways that this project could be extended is to compare how it would run on a fully synchronous actor system or a strictly step-based model. We hypothesize that strictly step-based models would have a larger overhead impacting the speed, but it is not for certain.

Conclusion

Even though the gossip and push-sum algorithms rely on selecting random nodes, they have been proven to be useful. Both algorithms don't depend on a centralized receiver, both are very fault tolerant, and both realize that the data in the whole network is more important than the data at any individual node.

Now that we know that the gossip and push-sum algorithms are a very dependable for group communication and aggregate communication, the real decision is deciding on the topology. While figures two and three may show the speeds of convergence for the respective algorithms, they don't show the initial overhead to connect the nodes. For example, if you use the full topology (for gossip) each node in the system must have a way to contact every single other node in the system. This causes an extremely high overhead for this topology, which is why it couldn't handle having 27,000 nodes in the system. Disseminating the information wasn't the problem, it was establishing the connections. All in all, the ideal topology for your system depends on how much overhead you want to have. In the end everything has tradeoffs, including gossip protocols.

Reference

The math behind the rate of convergences for gossip and push-sum can be seen here (Dr. Dobra's paper): http://www.cs.cornell.edu/johannes/papers/2003/focs2003-gossip.pdf.