Optimizing MPICH Collective Calls using Topology info

Shobhit Sinha (20111059)

March 30, 2021

Submitted Files and execution command

The script to run the configurations is in run.sh which can be executed using **bash run.sh**. The script generates three plot files plot_Bcast.png, plot_Reduce.png, plot_Gather.png. The plot files will be generated in the same location where run.sh is present. Also, we have submitted three plots and corresponding plot file in subdirectory SubmittedPlots.

1. A brief summary of Network Topology of csewsX cluster

We were provided the topological information of the cluster in the Problem statement. From the given info, one of the ways in which the Topology can be represented is shown in Figure 1.

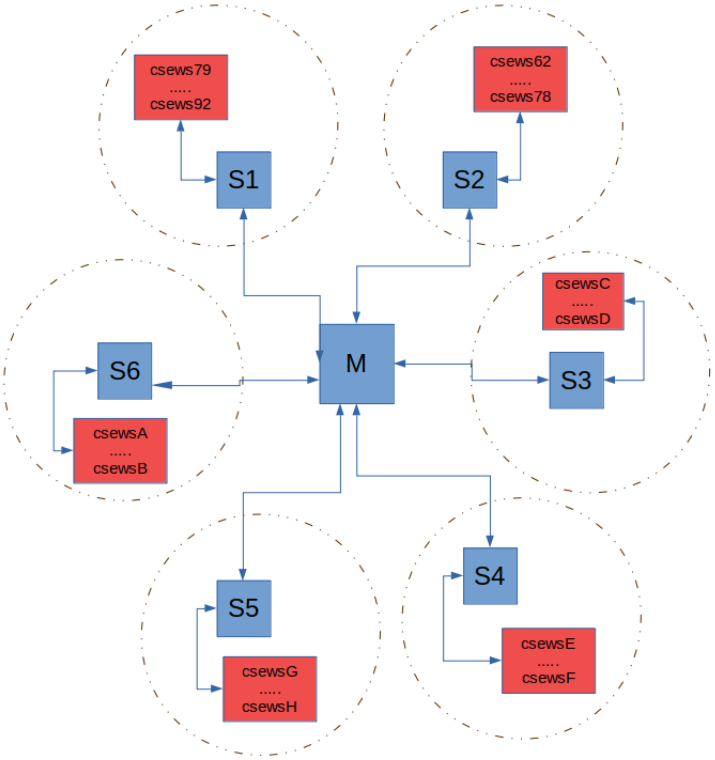

Figure 1: Assumed Network Topology of csewsX cluster

Here, $S1, \ldots, S6, M$ are switches or they can be a Routing Device. The dotted circles represent a Sub-Cluster. Two machines present in the same Sub-Cluster must use the respective Sub-Cluster switch ($S1, \ldots S6$) to communicate and the communication distance is 2 links only. If they are present in different Sub-Clusters and want to communicate, the communication distance is 4 links (double the previous distance!). Clearly, if we wish to optimize our Collectives by using topology information, we need to minimize inter Sub-Cluster communication as much as possible as it is more expensive.

2. Optimizations done :

Consider the communication pattern as shown in Figure 2. Ranks residing on the same machine/Node/Host will communicate with each other using a Communicator called sameHost_comm. This communicator can be made using MPI_Comm_Split() to split MPI_COMM_WORLD on basis of NodeID. The process with rank 0 in this new communicator is designated as the **Node Head**. Collective Communication to any rank on a different node, must go through the Node Head.
All nodes heads of a Sub-Cluster communicate with each other using a communicator called subCluster_comm. This communicator can be made using MPI_Comm_Split() to split MPI_COMM_WORLD on basis of ClusterID and splitting it again with on basis of host rank being 0. The process with rank 0 in this new communicator is designated as the **SubCluster Head**. Collective Communication to any rank on a different Sub-Cluster, must go through the SubCluster Head.

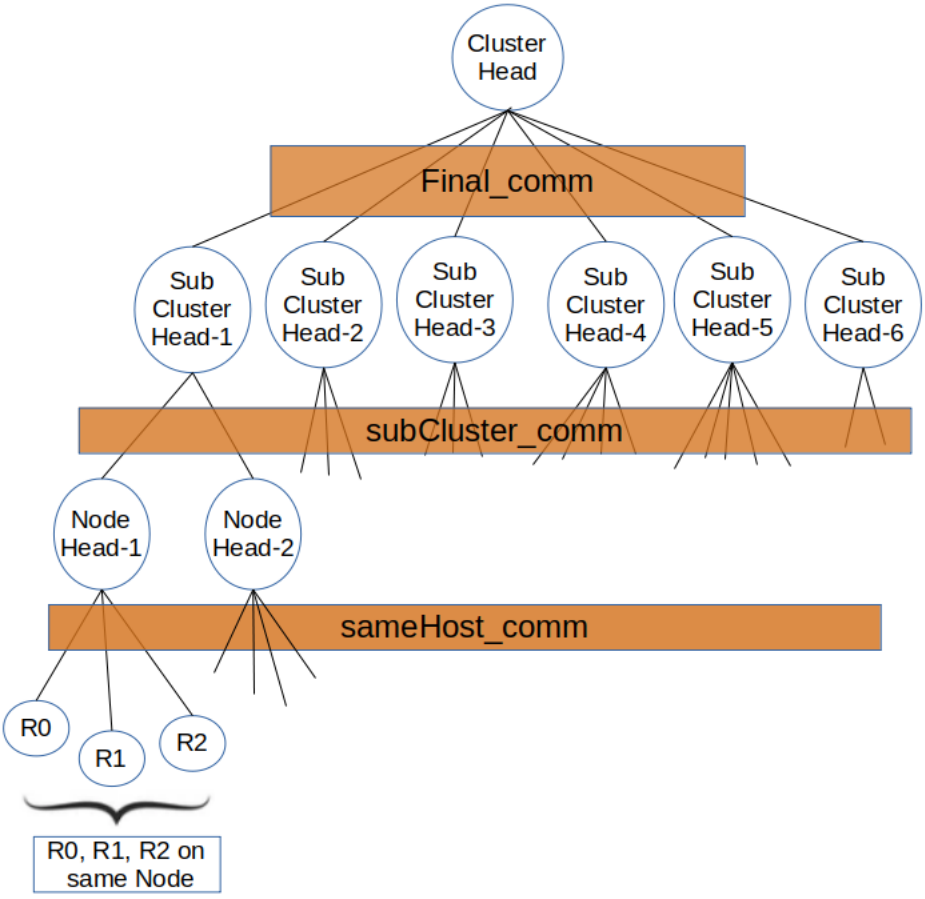

Figure 2: Communcation tree to minimize inter Sub-Cluster communication

Finally, each SubCluster Head can communicate to a **Cluster Head** using a communicator called Final_comm. The Cluster Head is Rank 0 of MPI_COMM_WORLD. In act, all the heads are rank 0s of their respective communicators a level below them! This is to ensure that a head always exist because if a communicator is not MPI_COMM_NULL then at least a rank 0 must exist inside it.

2.1 Optimization of MPI_Reduce

The flow of data is upwards in the communication tree (Fig. 2) in case of optimized MPI_Reduce. Initially, the normal call is performed at the Node Level with all ranks on the same node performing the operation and reduction results are available at Node Head. Then, the call is performed at SubCluster level among the Node Heads and the reduction result is available at SubClusterHead. Finally, in the end the the subClusterHeads perform the Reduction and the result is available in Cluster Head. In case, the initial MPI_Reduce() required results to be available at a different rank, Cluster Head can then communicate the result directly to that rank using Point to Point Communication (we have used simple MPI_Send and MPI_Recv) to do this.

2.2 Optimization of MPI_Bcast

The flow of data is downwards in the communication tree (Fig. 2) in case of optimized MPI_Bcast. Initially, the root of the Bcast communicates it's data to the Cluster Head. The Cluster Head

broadcasts it's data to Sub-Cluster Heads which in turn broadcast downward to Node heads. The Node heads finally broadcast their data to Ranks on the same Node.

2.3 Optimization of MPI_Gather

The optimization is very similar to MPI_Reduce as in both cases the data is flowing up the communication tree (Fig. 2).

2.4 Optimization of MPI_Alltoallv

The flow of data in this case is bi-directional and happens in two phases. First, we gather all the data to Cluster Head (upward flow) using optimized MPI_ScatterV. We then, scatter the data (downward flow) using optimized MPI_ScatterV. However, we were unable to successfully implement MPI_Alltoallv. This was mainly due to a lot of book-keeping involved (maybe C++ STL could have helped) and our implementation was buggy.

3. Experimental Results and Observations :

We have calculated the Speedup as $\frac{\text{Execution Time for Normal Call}}{\text{Execution Time for Optimized Call}}$

3.1 MPI_Bcast vs Optimized MPI_Bcast

P	ppn	D	Min Speedup	Max Speedup	Mean Speedup	Median Speedup
4	1	16	0.47	1.33	0.98	1.00
4	1	256	1.02	49.18	5.90	1.09
4	1	2048	0.08	4.39	1.23	1.02
4	8	16	0.90	1.48	1.15	1.10
4	8	256	0.79	1.55	1.10	1.08
4	8	2048	2.28	3.00	2.61	2.56
16	1	16	0.64	1.37	0.99	1.00
16	1	256	0.62	1.46	1.02	1.03
16	1	2048	1.28	2.00	1.64	1.64
16	8	16	1.10	3.13	2.09	2.05
16	8	256	0.80	1.26	1.01	1.03
16	8	2048	1.43	1.85	1.63	1.62

Table 1: Speedup values for MPI_BCast for different configurations

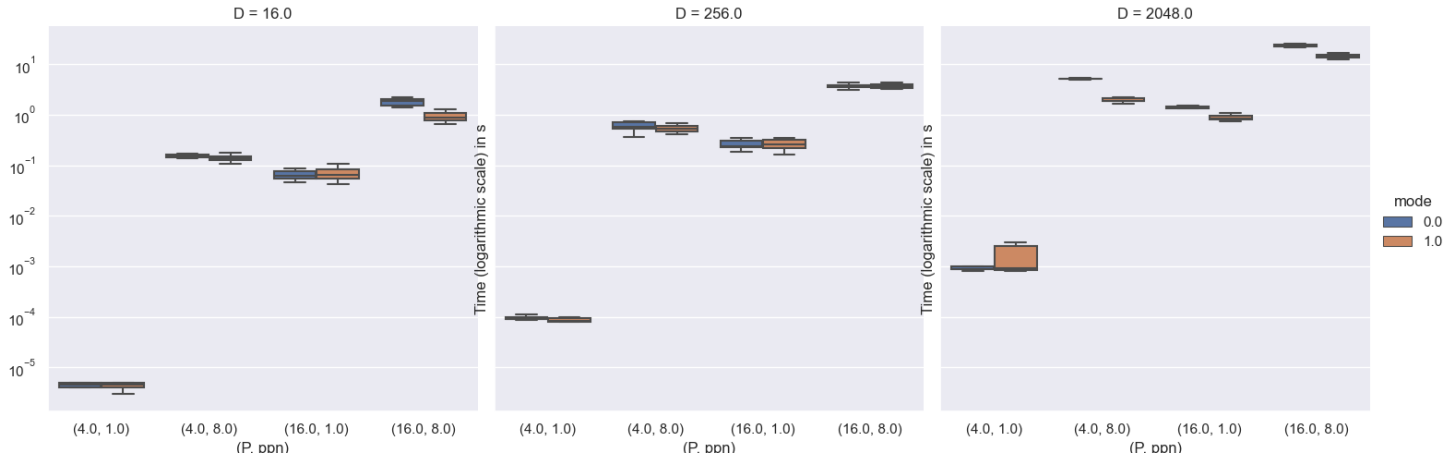

Figure 3: MPI_BCast (Mode 0) Vs Optimized MPI_BCast (Mode 1)

Table 1 shows statistics on speedup achieved (based on 10 sets of values for each {P, ppn, D} configuration. We note that for fixed {ppn, D} as P is increased, the speedup increases. This was expected as more nodes (hence, more Sub-Clusters) bring the Topology based optimizations in the picture. Similarly, we expected that for fixed set of {P, ppn } as D increases, the speed up increases as more bytes have to be transferred. However, this is not noticeable for lower number of processes if all the nodes belong to the same Sub-Cluster. As number of processes increase, they are on different Sub-Clusters and more speedup is there as our optimizations minimize inter Sub-Cluster communication.
We have also plotted the 10 sets of values for each {P, ppn, D} configurations in Fig. 3. Please note that y-axis is in logarithmic scale.

3.2 MPI_Reduce vs Optimized MPI_Reduce

P	ppn	D	Min Speedup	Max Speedup	Mean Speedup	Median Speedup
4	1	16	0.91	2.39	1.22	1.00
4	1	256	0.96	52.12	7.57	0.99
4	1	2048	0.15	4.11	1.50	0.59
4	8	16	1.11	1.61	1.47	1.54
4	8	256	1.19	1.79	1.39	1.32
4	8	2048	0.83	2.09	1.38	1.34
16	1	16	1.03	2.14	1.64	1.65
16	1	256	1.02	1.99	1.55	1.60
16	1	2048	0.80	2.32	1.53	1.55
16	8	16	1.29	1.54	1.46	1.47
16	8	256	1.11	1.58	1.35	1.34
16	8	2048	1.17	1.42	1.26	1.23

Table 2: Speedup values for MPI_Reduce for different configurations

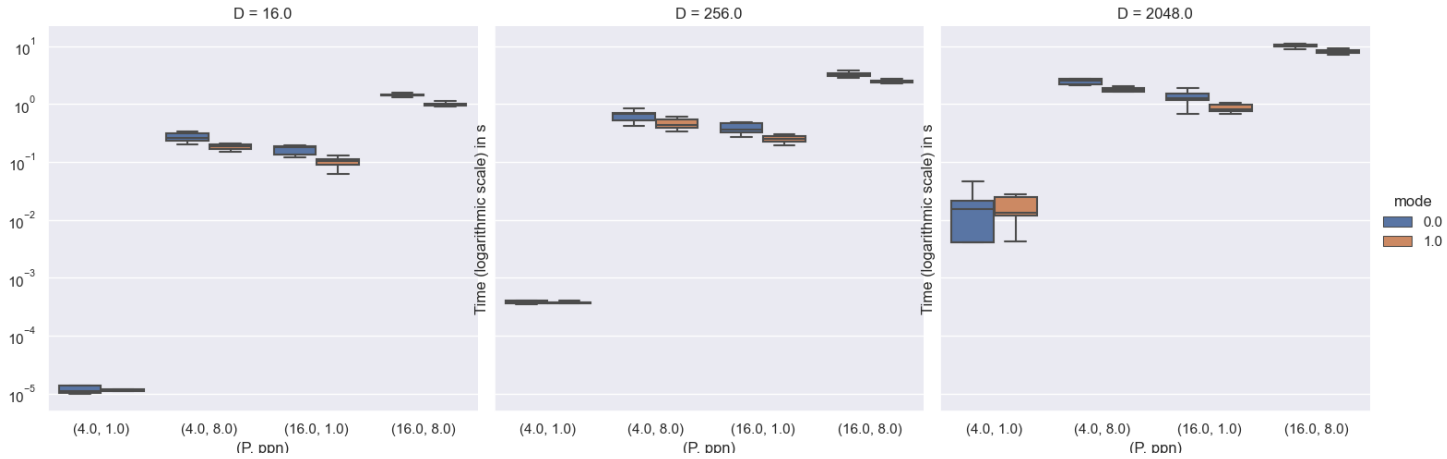

Figure 4: MPI_Reduce (Mode 0) Vs Optimized MPI_Reduce (Mode 1)

Table 2 shows statistics on speedup achieved (based on 10 sets of values for each {P, ppn, D} configuration. We notice that for fixed set of {P, ppn } as D increases, the speed up decreases. This is unexpected and is in direct contrast with what we observed in MPI_Bcast. Maybe the upward flow of data in MPI_Reduce is creating bottlenecks as D is increasing and more messages are in transit. We have also plotted the 10 sets of values for each {P, ppn, D} configurations in Fig. 4. Please note that y-axis is in logarithmic scale.

3.3 MPI_Gather vs Optimized MPI_Gather

P	ppn	D	Min Speedup	Max Speedup	Mean Speedup	Median Speedup
4	1	16	0.60	1.40	0.71	0.64
4	1	256	0.66	12.64	1.88	0.68
4	1	2048	0.16	30.10	4.00	0.47
4	8	16	0.87	1.40	1.07	1.05
4	8	256	0.83	1.33	1.13	1.19
4	8	2048	0.58	1.51	1.03	1.05
16	1	16	0.85	1.39	1.07	1.01
16	1	256	0.53	1.83	0.92	0.86
16	1	2048	0.40	1.66	1.07	1.05
16	8	16	0.79	1.19	1.02	1.01
16	8	256	0.93	1.27	1.05	1.05
16	8	2048	0.90	1.26	1.14	1.17

Table 3: Speedup values for MPI_Gather for different configurations

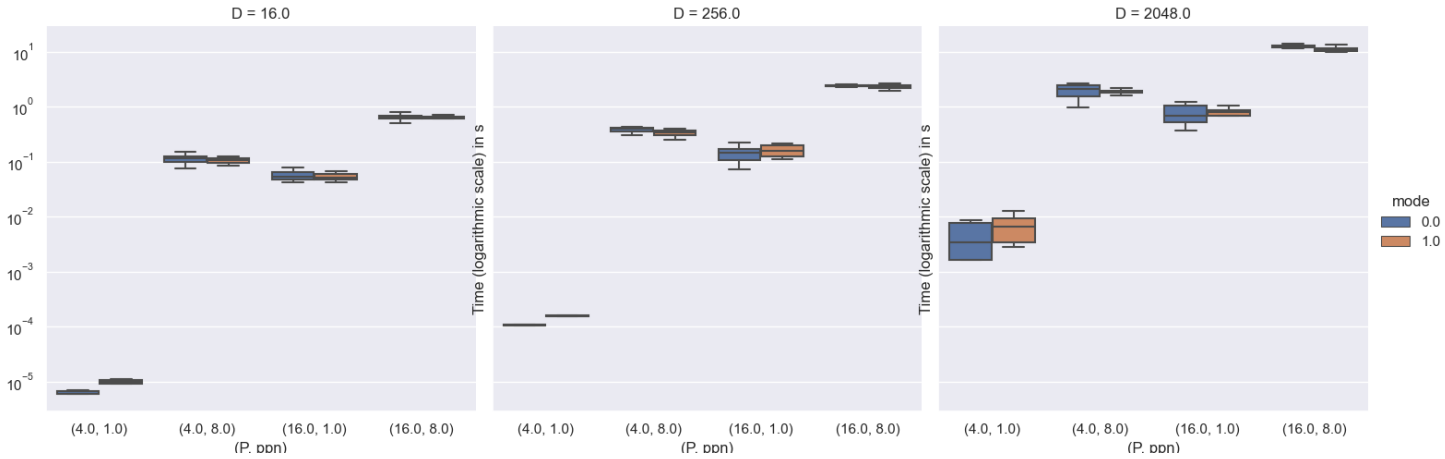

Figure 5: MPI_Gather (Mode 0) Vs Optimized MPI_Gather (Mode 1)

Table 3 shows statistics on speedup achieved (based on 10 sets of values for each {P, ppn, D} configuration. We note that for fixed {ppn, D} as P is increased, the speedup increases. This was expected as more nodes (hence, more Sub-Clusters) bring the Topology based optimizations in the picture. Similarly, we expected that for fixed set of {P, ppn } as D increases, the speed up increases as more bytes have to be transferred. However, this is not noticeable for lower number of processes if all the nodes belong to the same Sub-Cluster. As number of processes increase, they are on different Sub-Clusters and more speedup is there as our optimizations minimize inter Sub-Cluster communication.
We have also plotted the 10 sets of values for each {P, ppn, D} configurations in Fig. 5. Please note that y-axis is in logarithmic scale.

4. Conclusion

We have implemented Network Topology Optimizations for MPI_BCast, MPI_Reduce, MPI_Gather and have attained mostly positive Median Speedups.
These speedups (sometimes 1.05 times or even sometimes 1.20 times) indicate that knowledge of Network Topology can decrease the execution time of MPI_Collective Calls. This speedup is more noticeable when number of processes is more and D is also large. Also, for all the three calls, the mean and median Speedup values are very close to each other in most of the configurations, possibly implying an almost symmetric Distribution. While we could not correctly implement MPI_Alltoallv we expect it to speedup similarly like other implemented collective calls.