16 February 2021

Master Thesis

Distributed Robust Learning

M-Krum Ring All-Reduce Implementation overview

16 February 2021

Prerequisites:

We set the following before starting the implementation (server/src/main/resources/reference.conf):

1) Number of nodes(n) = 5
2) Number of Byzantine workers(f) = 1 [n ≥ 2f+3]
3) Number of closest vectors = 2 [n- f-2]
4) M-Krum average(m) = 2 [1 ≤ m ≤ n-f-2]
5) n-f-1 tightly packed cluster

The implementation is carried out using Kompics - a message passing component model for building distributed systems and Scala programming language.

System Architecture and Bootstrapping:

In our architecture, we assume each node to be a server. A server is classified in two ways:

1) Bootstrap Server (server/src/main/scala/se/kth/rise/bootstrapping/BootstrapServer.scala)
2) Bootstrap Client (server/src/main/scala/se/kth/rise/bootstrapping/BootstrapClient.scala)

The bootstrap server will first be initiated, and it waits for the bootstrap clients to connect to it. Once bootstrap server reaches the threshold (n), i.e., minimum number of connecting workers (including itself), it initiates Ring Topology (server/src/main/scala/se/kth/rise/overlay/RingTopology.scala) by assigning predecessors and successors to each worker. For example, in fig. 1, the bootstrap server waits for requests from 4 bootstrap clients to connect before generating ring topology.

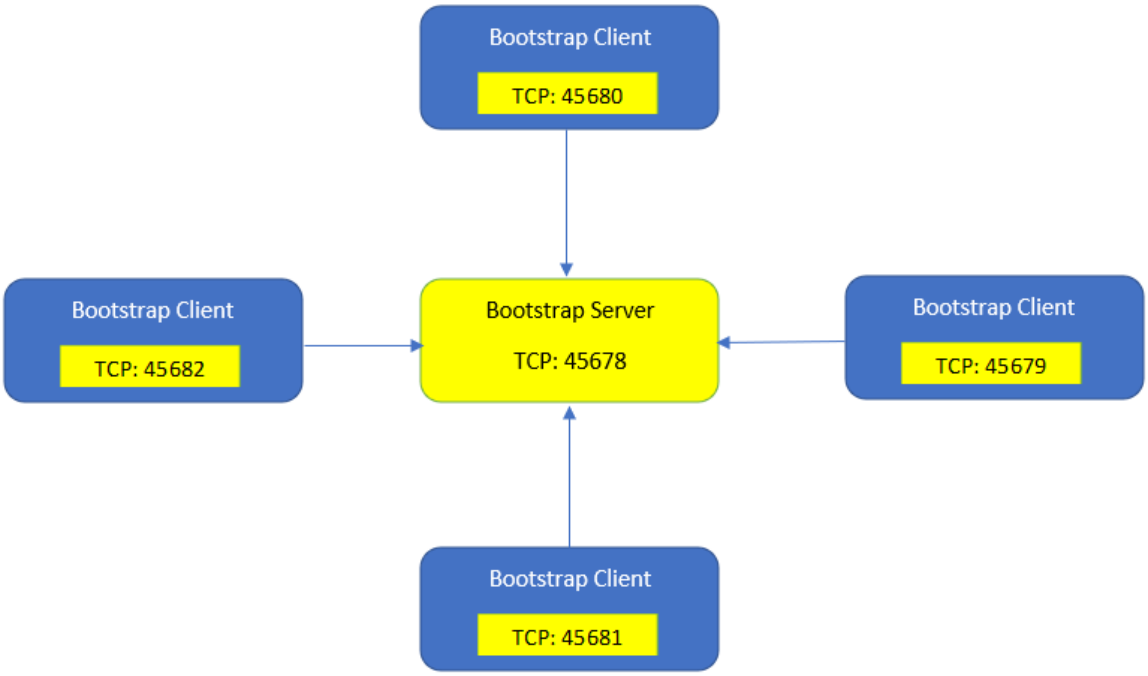


Figure 1

The bootstrap server will then share the assigned predecessor and successor information to all the connected clients including itself as shown in Fig. 2.

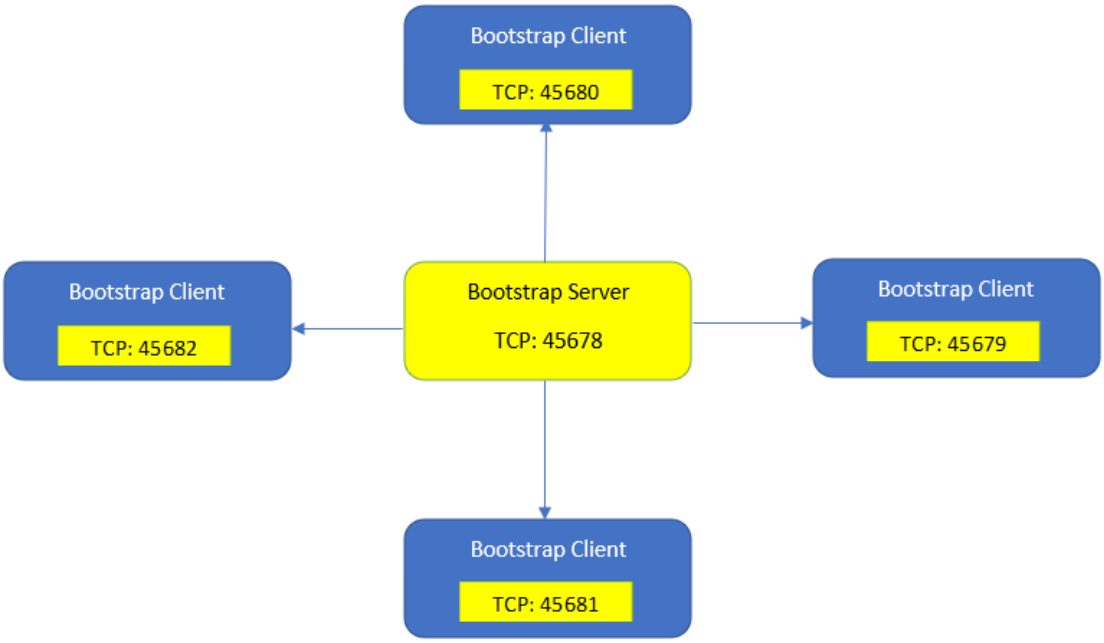


Figure 2

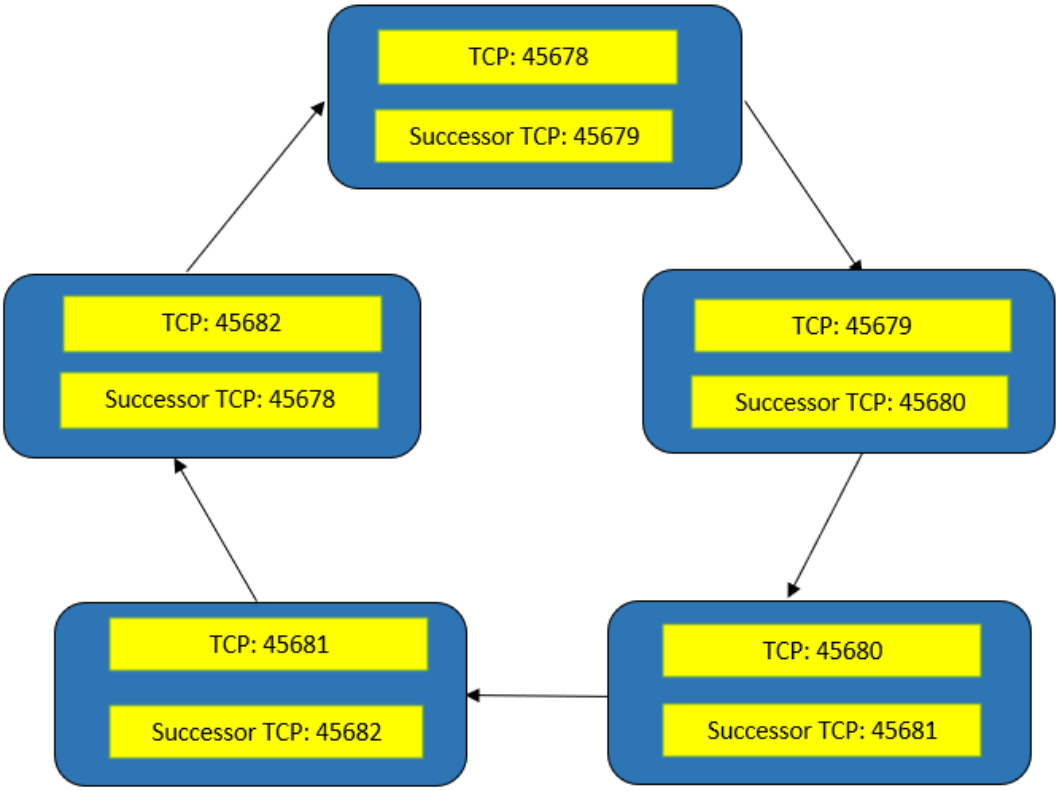


Figure 3

Once the information is sent to all workers, the bootstrapping is complete. We can assume the ring ring-topology as shown in Fig. 3 and each node in the ring can only communicate (TCP) with its successor.

Ring All-Reduce:

Before beginning the Ring All-Reduce, we generate random gradients for features in each worker. Here, we assume Node 1 to be byzantine where it generates the gradients of range 0.1 to 0.25 whereas a correct worker is assumed to generate gradients of range 0.6 to 0.75. Fig. 4 depicts the gradients generated in each worker.

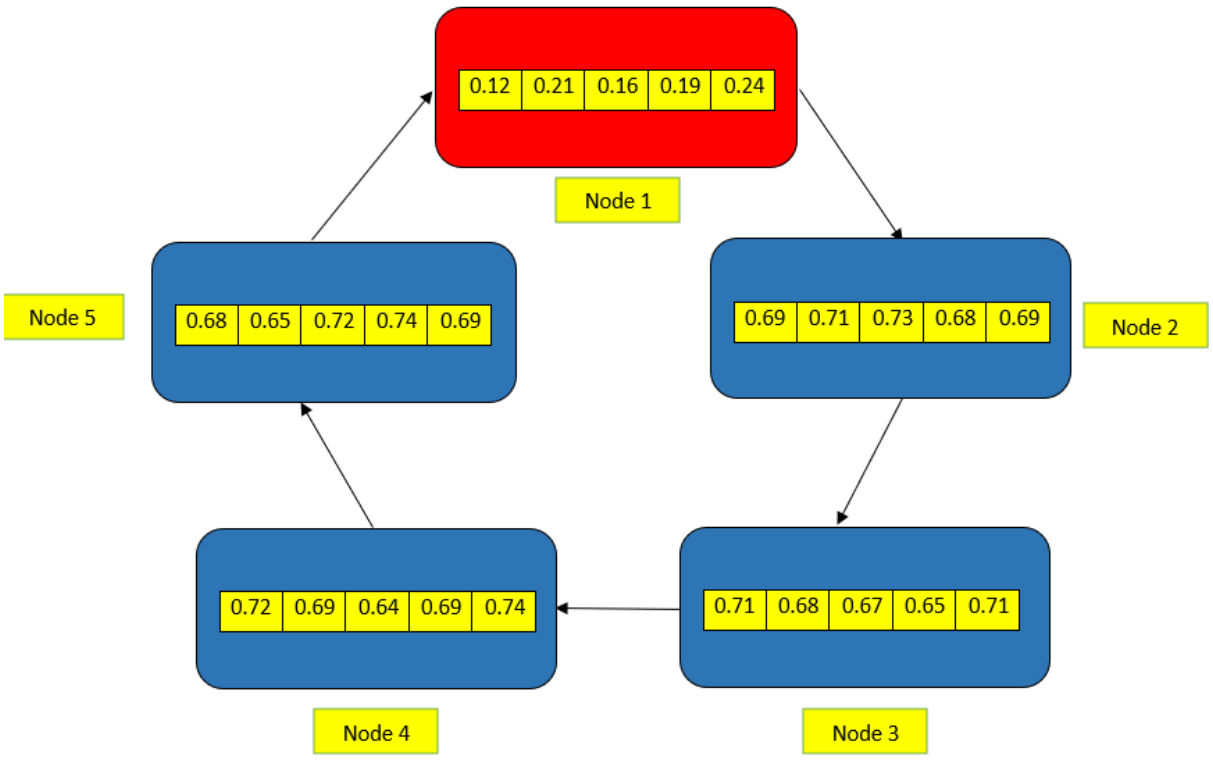
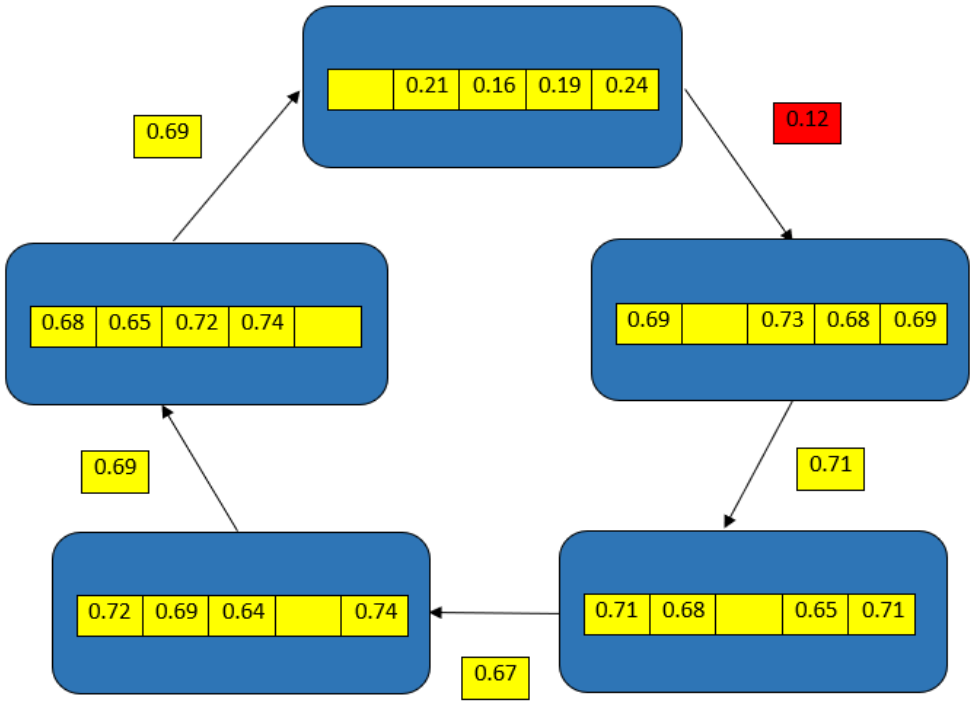


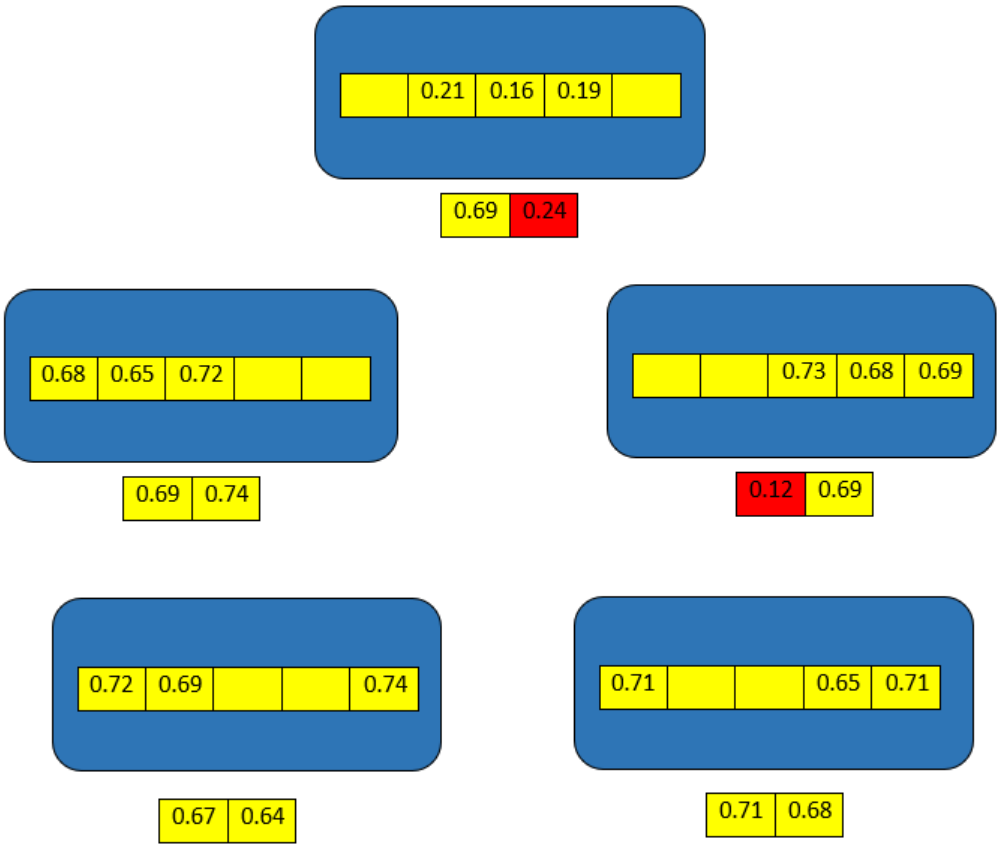
Figure 4

16 February 2021

Share (1):

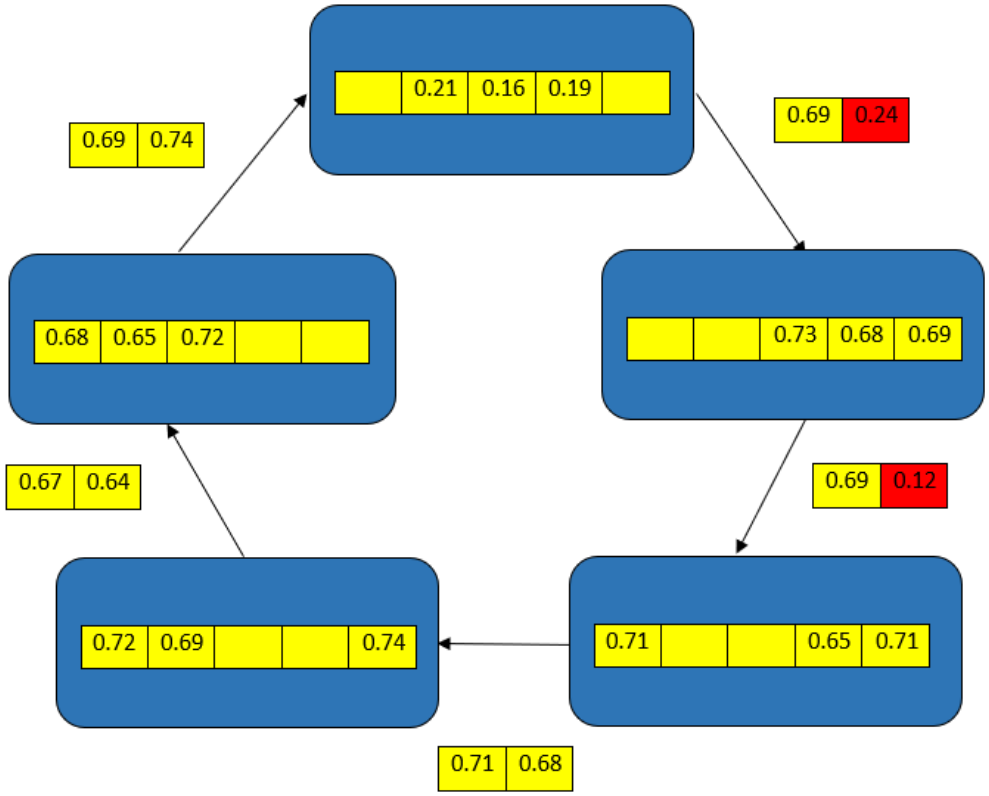


Reduce (1):

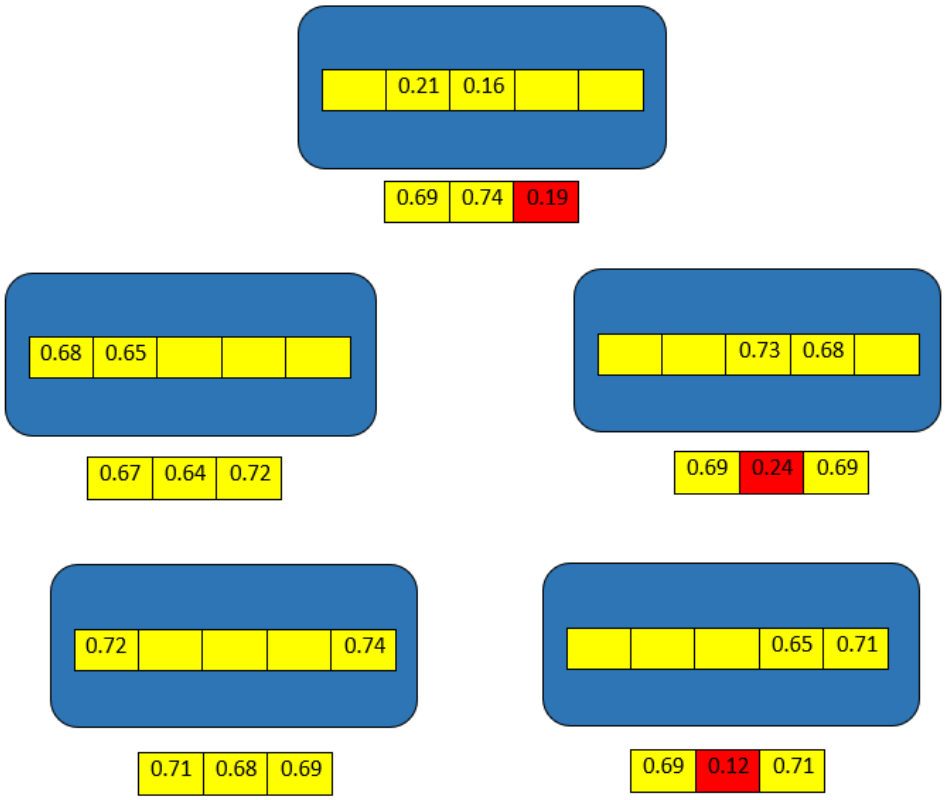


16 February 2021

Share (2):

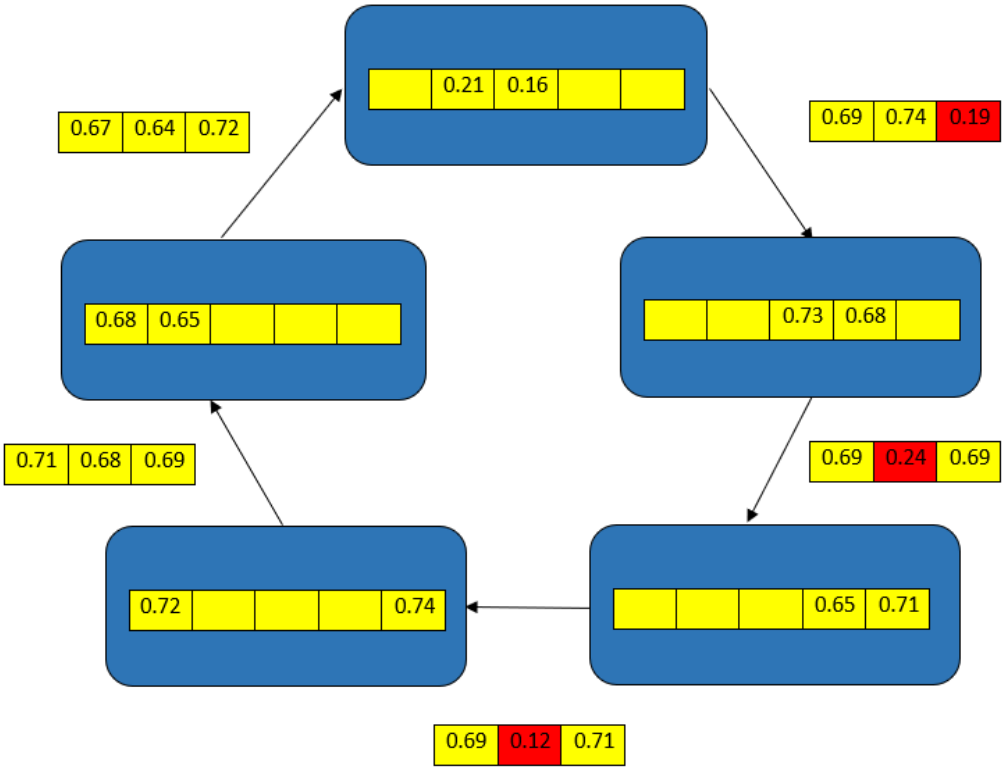


Reduce (2):

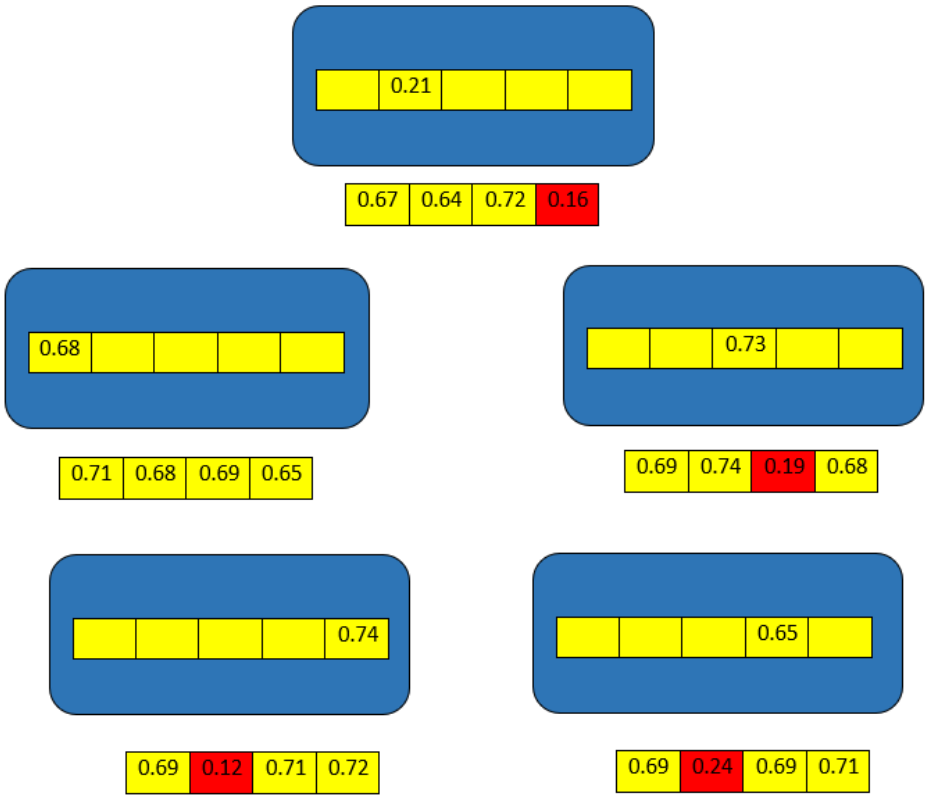


16 February 2021

Share (3):

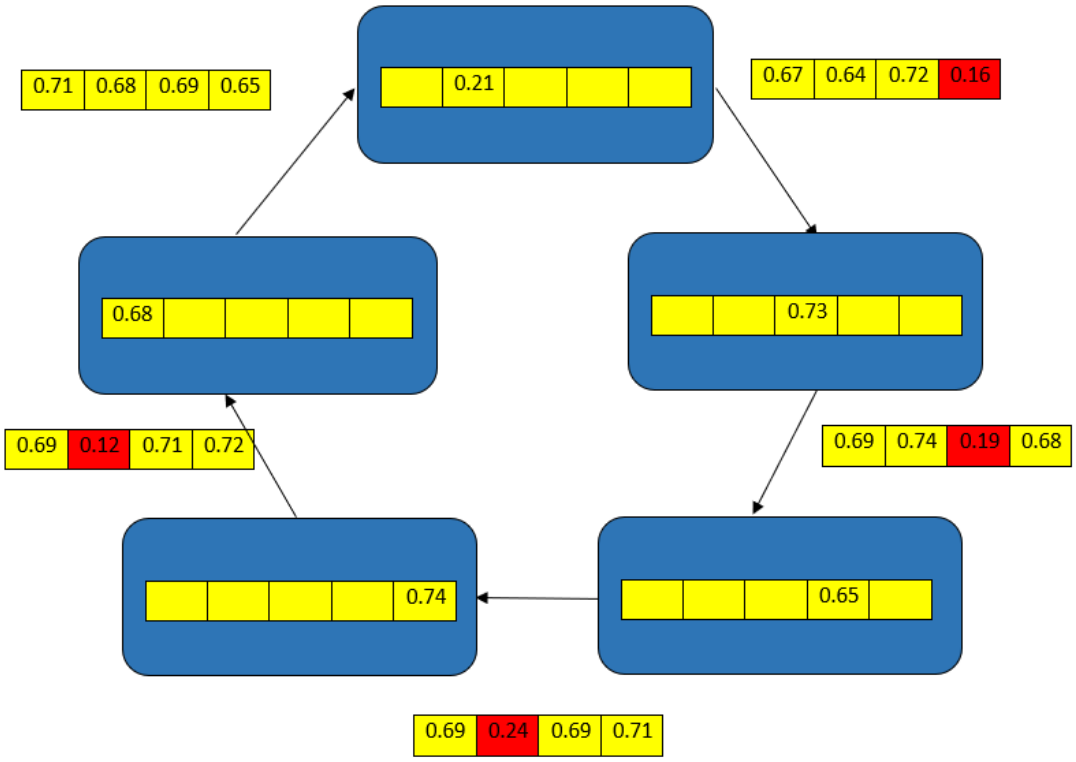


Reduce (3):

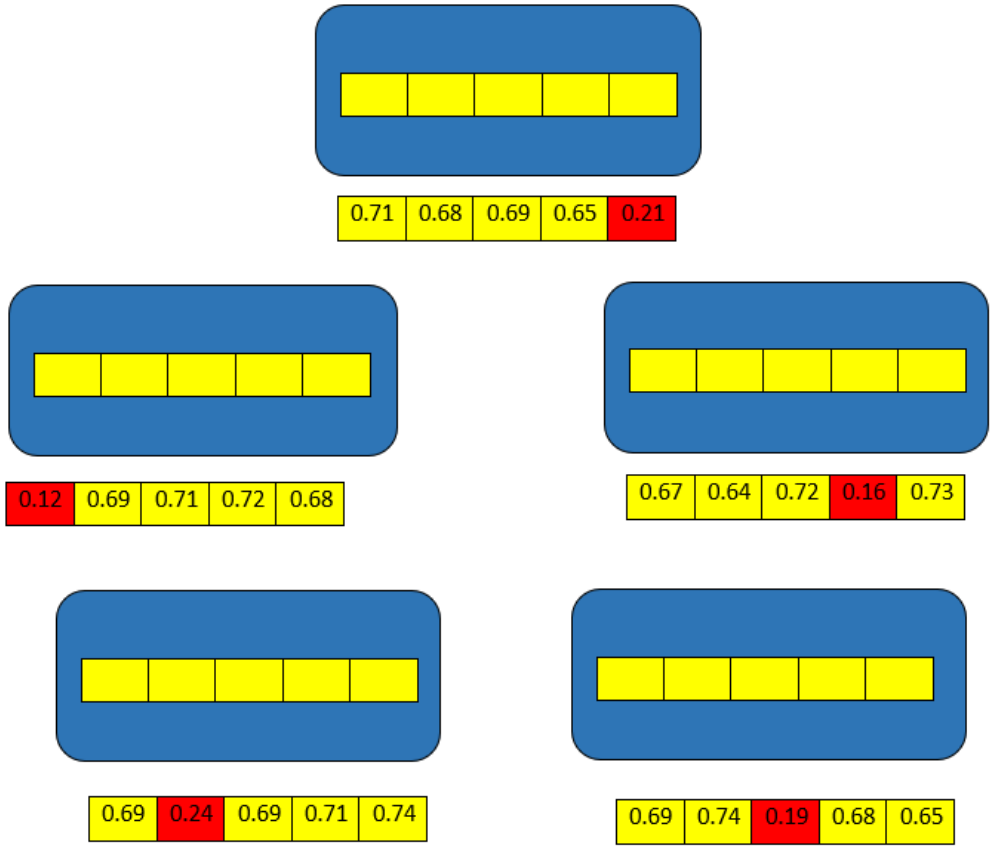


16 February 2021

Share (4):



Reduce (4):



16 February 2021

At the end of share-reduce phase, each worker will hold a list of gradients computed by all workers for a specific index. For example, in Reduce (4), Node 5 will hold all the information belonging to index 0 of each worker.

M-Krum:

(server/src/main/scala/se/kth/rise/byzantine-resillience-algorithm/Multi-Krum.scala)

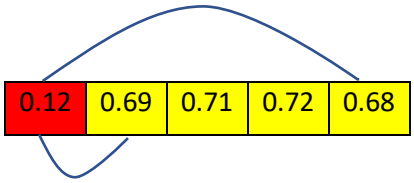
Once share-reduce phase is completed, we begin M- Krum algorithm in each worker by giving the received gradients as input. For example, in worker 5 we give below as input.

0.12	0.69	0.71	0.72	0.68

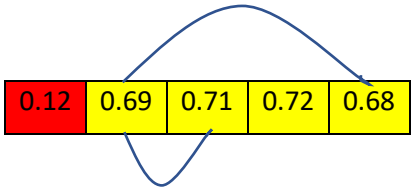
Here, we obtain 2 (n-f-2) closest gradients for each gradient. For example, the closest gradients for 0.12 is 0.69 and 0.68 in our example. Post obtaining the closest for all other gradients in the list, we start computing scores for each of them. The score can be computed using the following:

$$s(i) = \sum_{i \to j} \|V_i - V_j\|^2$$

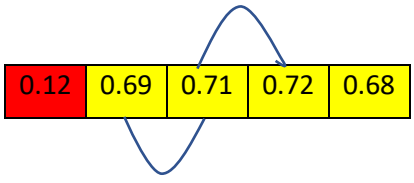
Below are the scores,



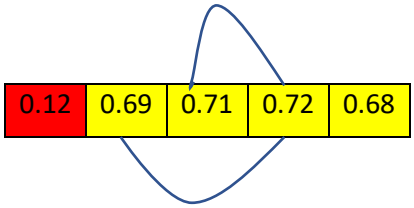
$s(0) = (0.69 - 0.12)^2 + (0.68 - 0.12)^2 \Rightarrow 0.3249 + 0.3136 \Rightarrow 0.6385$



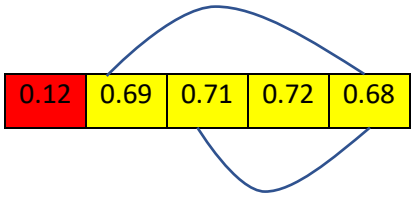
$s(1) = (0.69 - 0.68)^2 + (0.71 - 0.69)^2 \Rightarrow 0.0001 + 0.0004 \Rightarrow 0.0005$



$s(2) = (0.71 - 0.69)^2 + (0.72 - 0.71)^2 \Rightarrow 0.0004 + 0.0001 \Rightarrow 0.0005$

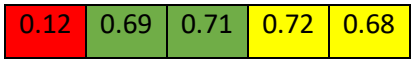


$S(3) = (0.72 - 0.71)^2 + (0.72 - 0.69)^2 = 0.0001 + 0.0009 => 0.001$



$S(4) = (0.69 - 0.68)^2 + (0.71 - 0.68)^2 = 0.0001 + 0.0009 = 0.001$

After computing the scores, we can sort the values obtained in the ascending order to obtain the least 2 (m) scores. The order will be $s(1) < s(2) < s(3) < s(4) < s(0)$, therefore we can pick s(1) and s(2). Hence, gradients 1 and 2 will be selected for being averaged ignoring the byzantine gradient.



The average of both would be 0.70 (G(0) in Node 5). Now, we start the All-Share phase as shown in fig. 5 where we also share the computed (G(1) in Node 1, G(2) in Node 2, G(3) in Node 3 and G(4) in Node 4) with all other workers.

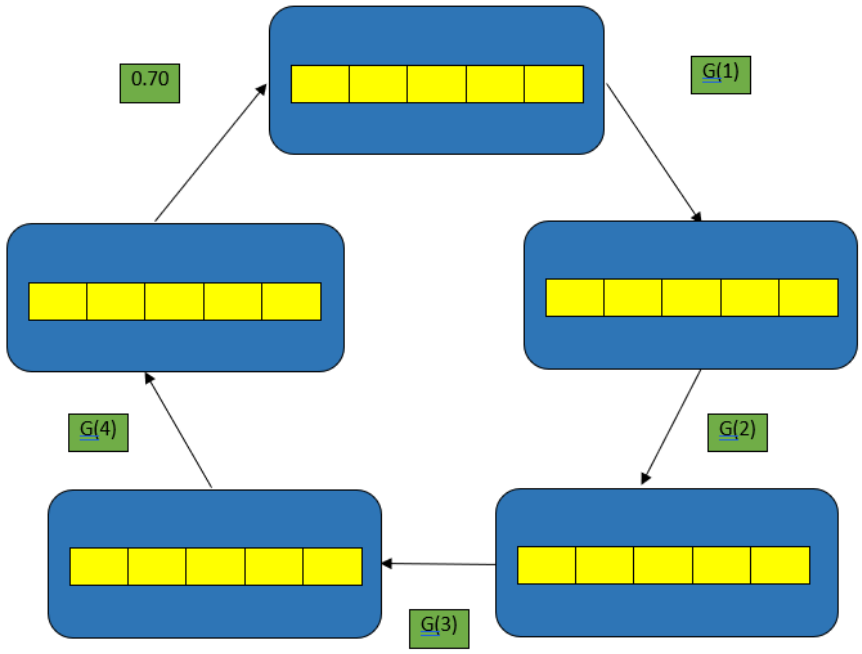


Figure 5

16 February 2021

Final gradients for all features:

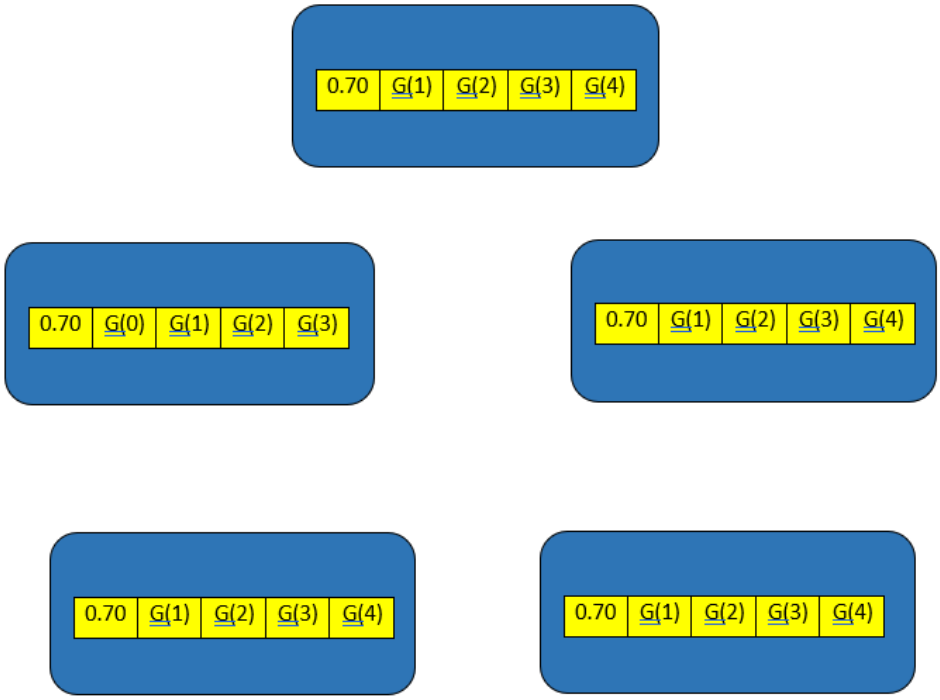


Figure 6