Bimodal Multicast - Probabilistic Broadcast

Lorenzo Masciullo [203315]

Valentina Odorizzi [203314]

November 24, 2019

1 Introduction

This report focus on the implementation of the Bimodal Multicast in order to analyze the behaviour of the nodes during message exchange. Moreover, our purpose is understanding the reliability of the network using the Bimodal Multicast. The Probabilistic Broadcast, that is referred to Bimodal Multicast, is inspired to Gossip Protocol and it satisfies several properties:

- *Atomicity*, almost all or almost none, hence the protocol guarantees with high probability that each multicast reaches almost all processes and with low probability that every multicast reaches a small set of processes.
- *Throughput stability*, the variation of throughput is lower than the normal multicast.
- *Ordering*, the messages are delivered in FIFO order.
- *Multicast stability*, the protocol guarantees the stability of messages.
- *Detection of lost messages*, the processes recover the loss message through a particular procedure.
- *Scalability*, the costs are constant or grow slowly which are based on the network size.

We have tried to implement all these properties in order to get the protocol similar to the original one. The purpose of probabilistic broadcast is exchanging messages between nodes using the Gossip protocol behaviour and re-transmitting the lost messages when a transmission error occurs.

Thanks to this protocol we can analyze the throughput over time with some particular settings (e.g. Round Retrasmission Limit), the performance of the protocol and the probability of success of pbcast with different network sizes and the delivery distribution of the protocol. In details, we use several parameters in order to set up the environment obtaining the expected experiment to analyze, which are described in Section 3.

2 Implementation

The implementation is based on the agent-based model involving processes which exchange message through them. Our Probabilistic Broadcast is composed by several classes: `MessageHandler`, `Message`, `PBCastBuilder`, `Process`, `Tree` and `Node`. The main agent that is involved in our analysis is `Process`, its main goal is sending and delivering messages that are sent from the root of the spanning tree to the its children and then the messages are forwarded through the spanning tree. First of all, the PBCastBuilder agent computes the spanning trees of all processes. The number of children is chosen before starting the simulation using two parameters `Children for node (min)` and `Children for node (max)`. For each iteration, the process can perform two different actions: send message or receive message. The number of actions is limited for each iteration by a workload which is chosen randomly between minimum and maximum values set using `Workload for node (min)` and `Workload for node (max)` parameters. The processes are located within three continuousSpace. The ContinuosSpace gives us the opportunity to represent the spanning tree of a single process, the messages that are exchanging in all spanning tree in the current step and the messages that are exchanging in a single spanning tree which is related to the first continuousSpace. For each iteration is possible to see the sent messages which are represented by edges. In the continuosSpace used to represent the messages exchanged, the edges can assume different colour to indicate several situations, such as message received (green), message lost (red) and message retransmitted (yellow). Moreover, the processes have a status which can acquire four different values: QUIETE (blue), SENDER (red), RECEIVER (green) and CHILD (yellow). The architecture is not composed by a coordinator agent which triggers the event. Therefore, the Process agent triggers the pbcast through the step method.

In details, the implemented Probabilistic Broadcast Protocol is composed by two sub-protocols:

- **Optimistic Dissemination Protocol** is the first stage of the Probabilistic Broadcast and implements IP multicast or randomized dissemination protocol, in case the connection is not present, where the processes use a spanning tree in order to exchange the messages.

- **Two-Phase Anti-Entropy Protocol** is the second phase of our protocol and its purpose is recovering the loss messages since the multicast is unreliable.

2.1 Optimistic Dissemination Protocol

The Optimistic Dissemination Protocol is used by Probabilistic Broadcast in order to implement unreliable multicast. All messages use an unreliable multicast primitive using IP multicast or randomized dissemination protocol. Every process broadcasts messages using a generated spanning tree. The messages is sent to the children of the sender that deliver the message and then forward it including the tree identifier. There are several spanning trees, which are many as the number of the processes involved in the experiment. The spanning trees are realized at the beginning of the experiment and the code is implemented in the `PBCastBuilder` class. The root of each spanning trees is different. The number of the children is chosen using the parameters described above. The children are selected inspecting the grid in the continuousSpace, the process examines the space in order to add `N` children. The spanning tree is built starting from a root process which uses a distance variable in order to add the near processes as children of the tree. The processes are spread in the continuousSpace in order to avoid overlapping processes.

Every process can compute several operations which are limited by a workload in order to simulate one step in our analysis. The workload is chosen casually between minimum workload and a maximum workload parameters. Therefore, the workload limits the number of received and sent messages.

2.2 Two-Phase Anti-Entropy Protocol

The Two-Phase Anti-Entropy Protocol is executed randomly by nodes using the value of gossip_probability parameter. The node, that starts the protocol, sends a gossip message which contains the round number and the digest. In details, in our implementation the digest is composed by a list of summaries of every message in the process' history. Each summary is a string composed by the hash (e.g. MD5) of the message content and the id of the involved spanning tree. The receiver computes the digest of its history and compare it with the received one in order to verify whether it has received all messages. If the receiver does not have all messages, then it sends a solicitation message to the sender in order to retrieve the lost messages. However, our Two-Phase Anti-Entropy Protocol is able to retrieve the lost messages, but also to anticipate the delivery. Indeed, the receiver can send a solicitation message in order to retrieve the messages that do not yet reach the process since they have to pass through the spanning tree. Moreover, we assume that the retransmitted messages cannot be lost during the round.

Furthermore, the Two-Phase Anti-Entropy Protocol has several optimizations in order to limit its cost. For example, an optimization that we consider is *Round Retransmission Limit*. It is the maximum amount of messages that a process will retransmit in one round. If the process tries to send more messages, then it stops sending them when it reaches the limit.

3 Analysis

We have incurred in several problems in the analysis phase. Therefore, we describe in this report the obtained charts related to our protocol. We have tried to replicate the charts in the paper of *Bimodal Multicast* although the implementation of our protocol is based on some assumptions that are not considered in the paper, such as workload. Therefore, the behaviour is the same, but the graphic representation is different.

3.1 Parameters

Our analysis is focused on the value of different parameters which can be set from the users. The parameters are the following:

- *Children per node (Max)*: number of maximum children for every process.

  The large number of children allows to realize a spanning tree with less depth. Therefore, the workload is less balanced in the tree, so the processes need a value of workload that is closed to the number of children in order to have good performance.

- *Children per node (Min)*: number of minimum children for every process.

- *Gossip message limit*: it indicates the limit which is reached in order to remove the message from the history of each process.

  The protocol removes the messages quickly with a low gossip limit, so the messages age faster and the history queue does not store many messages. However, if the messages are deleted too soon, the protocol may not able to retrieve them, hence they are lost. The protocol with an high gossip limit may cause link saturation.

- *Message loss probability*: the probability to lose a message during packet exchange.

  The message loss probability decreases the efficiency of the protocol. However, the configuration of the Two-Phase Anti-Entropy protocol is important in order to mantain good performance.

- *Number of processes:* number of processes during the experiment.

- *Number of processes in a round*: number of processes that are select as receivers of the gossip messages.

  The high number of process per round allows to increase the probability to obtain the required message, so the the number of messages exchanged is increased.

- *Probability to send a gossip message*: the probability to send a random gossip message to round members.

  The probability to send a gossip messages affects the chances to retrieve the lost messages. The protocol with an high probability is not able to answer to the solicitation messages of the previous rounds since the process enters in a new round quickly.

- *Probability to send a pbcast*: the probability to send a pbcast through the spanning tree.

  The probability to send pbcast affects the efficiency of the protocol. Indeed, if it has a high value, then the efficiency decreases since the broadcast must reach all processes in the spanning tree.

- *Process ID to view*: shows the spanning tree of particular process where the messages are exchanged.

- *Retransmission limit per round*: the maximum number of message that a process can retransmit.

  The limit of retransmission is related to the loss probability and the probability to send gossip messages. It is useful to the Two-Phase Anti-Entropy Protocol in order to retrieve all lost messages.

- *Single pbcast*: the pbcast can sent from a single process (*0*) or from all processes (*-1* is the standard mode).

- *Workload per node (Max)*: the maximum number of operations that a process can perform in a single step.

- *Workload per node (Min)*: the minimum number of operations that a process can perform in a single step.

  The workload decreases the efficiency of the protocol with low values, since the broadcast is spread slowly.

3.2 Charts

PBCast Bimodal Delivery Distribution
A useful analysis is determining the evolution of the delivery distribution of our protocol with different network sizes and with 5% of lost messages.

We can notice in the Fig 1 that the delivery distribution follows the properties described in the Section 1. Therefore, the likelihood which almost all receive the multicast is very high. Therefore, the likelihood that a small number of processes will receive a multicast is low. Finally, the intermediate values has a low probability as we can see for 20, 25 and 30 as number of processes.

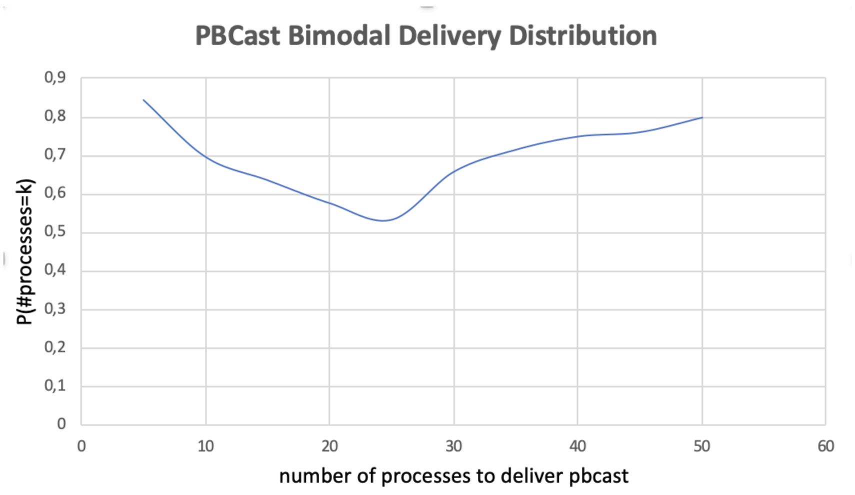

Figure 1: PBCast Bimodal Delivery Distribution

Throughput with/out Round Retransmission Limit
Another important aspect of the Protocol is the optimization where a process can retransmit a maximum number of messages. The charts are quite different from those in the paper due to the configuration used. Indeed, we do not have the possibility to set an established number of messages exchanged per second as in the evaluations of the original paper since we prefer to maintain a random environment. We have tested the network with 35 processes with a single sender, hence the messages are flooded in a unique spanning tree and the other processes forward the messages without generating new ones. The number of messages exchanged is not constant and changes over time.

The Table 1 shows the value of parameters that are used in the analysis.

Number of processes	35	35
Parameter Name		
Children	4	4
Gossip limit	2	2
Loss probability	0,3	0,3
Processes in a round	10	10
Send gossip	0,1	0,1
Send pbcast	1	1
Retransmission limit	-1	10
Single pbcast	0	0
Workload	100	100

Table 1: Parameters' value used in the evaluation of Round Retransmission Limit

The charts below show the messages sent from every process over the time, hence we consider also the messages forwarded by the children of the single sender.

In the Fig 2, we can notice the behaviour of the Protocol without Round Retransmission Limit, where the Protocol reaches some peaks due to the retransmission of the messages. Therefore, the process sends as many messages as those required. In details, the behaviour of the Two-Phase Anti-Entropy protocol without Round Retrasnmission Protocol produces many fluctuations.

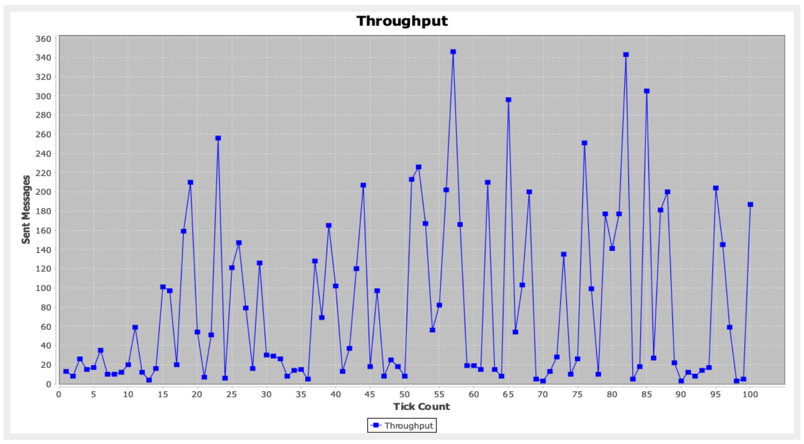

Figure 2: Throughput over time without Round Retransmission Limit

However, in the Fig 3, the behaviour is more limited due to the Round Retransmission Limit. Therefore, the messages sent over the network are few.

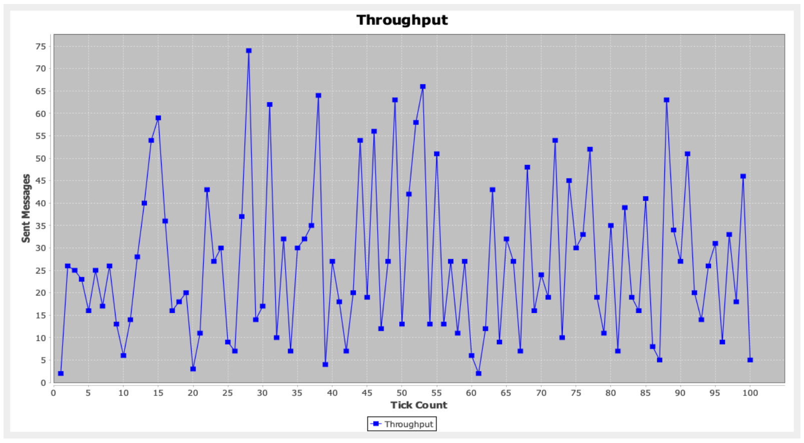

Figure 3: Throughput over time with Round Retransmission Limit

The change in behavior is clearly visible in Fig 4 where it considers both throughput with and without Round Retransmission Limit. We can notice that the throughput changes drastically when we set a Round Retransmission Limit of 10. Therefore, there are less fluctuations in the throughput like in the original paper.

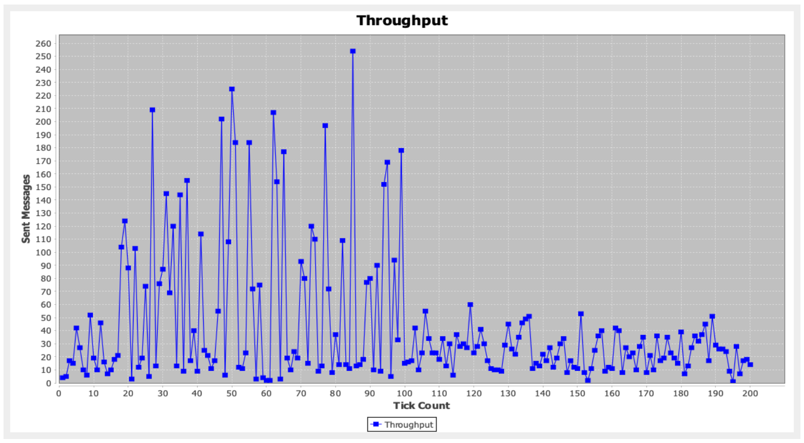

Figure 4: Throughput over time without Round Retransmission Limit (0-100 Tick Count) and with Round Retransmission Limit (100-200 Tick Count)

In conclusion, our protocol has the same behaviour of the original one, since it does not reach big peaks using the Round Retransmission Limit. We are not able to see a constant behaviour in the chart with Round Retransmission Limit since the throughput also considers the messages forwarded by the other processes. However, the fluctuations are decreased.

Performance and probability of success

The performance and the probability of success of pbcast are important to verify the quality of our protocol. The performance is ratio between the mean of processes that have received a pbcast (on each sent pbcast) and the total number of processes involved. The performance is measured in percentage. Instead, the probability of success of pbcast is the percentage of processes that are delivered the pbcast, hence the pbcast is successful. We consider in our analysis the charts with 15, 35 and 50 as group of members and 30% as probability of losing messages.

The Table 2 shows the value of parameters that are used in the analysis. In the Fig 5, 6 and 7, we can

Number of processes	15	35	50
Parameter Name			
Children	2	5	7
Gossip limit	5	5	5
Processes in a round	5	10	25
Send gossip	0,5	0,5	0,5
Send pbcast	1	1	0,5
Retransmission limit	-1	-1	-1
Workload	100	100	100

Table 2: Parameters' value used during the analysis

notice that initially the protocol has an exponential progress since the messages should flow through the spanning tree, and then it is constant. The protocol has a good performance since the mean percentage of it is above the 90% for different size of group members. Moreover, the probability of success of pbcast is around 80%. We can notice that the probability of pbcast is different changing the network size. Since, the protocol guarantees stability and follows the delivery distribution shown above in first evaluation.

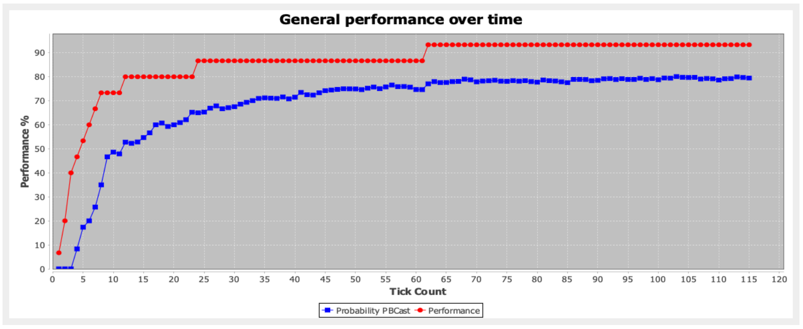

Figure 5: Performance over time with 15 processes

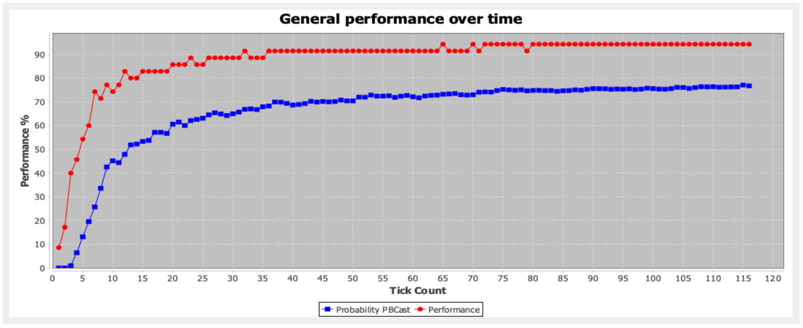

Figure 6: Performance over time with 35 processes

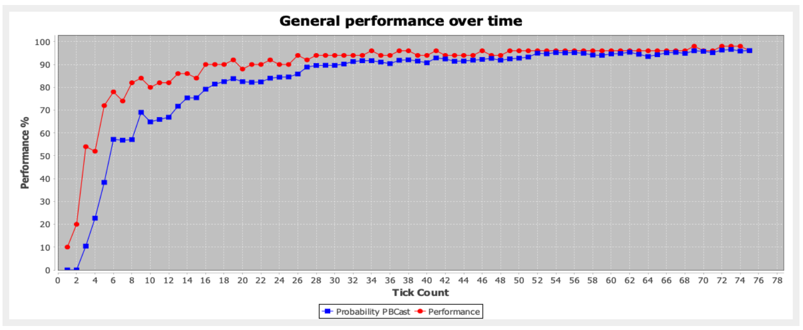

Figure 7: Performance over time with 50 processes

In conclusion, we can consider the performance of our protocol quite good with 30% probability of losing messages.

4 How to run the simulator

Once download the `setup_pbcast.jar` from the link placed in the repository's `README`, it's possible to follow the wizard steps in order to install all the resources needed to execute the model simulator. Basically, you can execute the following steps in order to run the simulator correctly.

- Make sure *Java RE 1.8* is installed in your system.
- Run `setup_pbcast.jar` using system GUI or typing `java -jar setup_pbcast.jar` on the terminal.
- Follow the wizard steps in order to accept licence and choose the installation directory, as you prefers.
- Launch the simulator executing the `start_model.bat`. A Java application will showed on your screen. Warning: if you use an Apple MAC then you have to run `start_model.command`.
- Set each parameter from the `Parameters` window. If you don't see it, you can reach it clicking on *Window* menu and take it from the *Show View*. Once you have chosen parameters, it's possible to save them in order to not rewrite them on the follow time.
- Finally, you can launch the simulation doing click on the "Initialize Run" button placed on the upper bar. At this point, you can see spaces and graphs in the main window of the simulator. You can start simulation doing click on "Start Run" button and observe results. If you want, it's possible perform simulation slower from "Run Options" window.

5 Conclusion

In conclusion, our protocol is similar to the original one. Its performance is also good in a scenario with 30% of lost messages. Moreover, it follows the most properties of the original protocol as we have seen in the analysis Section. It is atomic (i.e. almost all or almost none), hence the broadcast is reached by almost all processes with high probability and the broadcast is reached by a small set of processes with low probability. Furthermore, it is scalable, since the performance and the quality of protocol are not affected by the number of processes. Finally, our protocol is able to detect the loss messages sending a gossip message randomly to several processes. In this way is possible to guarantee the detection of the lost messages.