SAPIENZA
UNIVERSITÀ DI ROMA

DIPARTIMENTO DI INGEGNERIA INFORMATICA,
AUTOMATICA E GESTIONALE "ANTONIO RUBERTI"

Performance Evaluation of Broadcast Communication

DEPENDABLE DISTRIBUTED SYSTEMS

Student:
Angelo Casciani
M.N. 2022406

Academic Year 2022/2023

Contents

1 Introduction

1.1 Broadcast Communication

As distributed applications become more complex, the traditional point-to-point communication protocols supporting the interactions between two processes (as client–server model) are no longer enough.
In this scenario, *broadcast* communication protocols were developed to support a more articulated coordination among the participants of a distributed system: they allow a process to send a message to the others (figure 1), ensuring the agreement on the message they deliver ([1]).

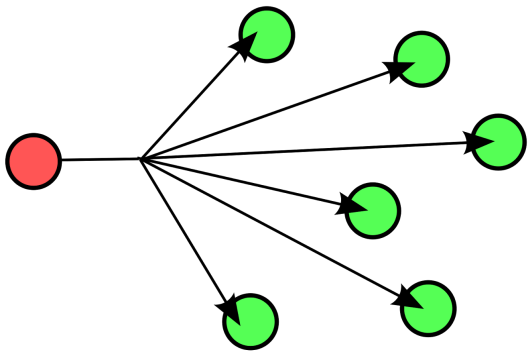

Figure 1: Broadcast Communication in a Distributed System.

Depending on the properties guaranteed by the protocol, broadcast algorithms can be broadly categorized in:

- *reliable*, ensuring a safe delivery of the message, on which all processes agree, and avoiding the duplication of that message or the creation of different ones on behalf of the communication channels;
- *probabilistic*, which are cheaper in terms of exchanged messages and data structures to maintain (always satisfying no creation and no duplication properties) but are not fully reliable, since they guarantee the delivery of the message by every process only statistically.

Anyway, differently from the simple point-to-point communication, broadcast primitives have to face additional challenges.
An example can be the fact that messages sent by a sender which fails by crash can be delivered by a group of processes, while not by the rest of the system.

1.2 Project Objectives and Methodologies

The aim of this analysis was to *evaluate* and *compare* the performance of reliable and probabilistic broadcast protocols, highlighting under which conditions they are *usable* and which one is *preferable* with respect to the others under specific environmental settings.
Specifically, the protocols chosen for the analysis are the *Regular Reliable Broadcast* (in both its *Lazy* and *Eager* versions) and the *Eager Probabilistic Broadcast* (relying on the *Gossip* mechanism). The implementation of these algorithms was taken from the book of the course, ([1]).
Considering a system of processes arranged in a full graph by means of point-to-point channels, the performance evaluation was carried out relying on both *simulation* and *analytical* modeling, to be able, in the end, to validate the results against the "real" (simulated) system behavior.
In particular, I developed a framework in Python to simulate the system behavior, which allow to specify several parameters for the environment (e.g. kind of broadcast algorithm, number of processes, arrival rate of the requests, possibility of crash failures, ...) and to compute interesting statistics about the run ([2]).
Eventually, a brief *dependability* evaluation of the system was carried out to have a more complete view of the system.

2 Contribution

2.1 Assumptions

Before deepening into the evaluation details, it is worth underlying the *assumptions* that were considered throughout the project.

- The system is composed by a set of nodes which are arranged in a *fully connected* graph and linked by *perfect point-to-point channels*.
- The delivery of the message happens upon its first receiving by the node.
- Communication channels are always available and *never crash*.
- The memory of the processes is assumed to be *bounded* (this was simulated relying on a Python `deque` of length 200).
- The inter-arrival time of the broadcast requests and the service time to process them for the channels and the processes are assumed to be *independent* and to act as random variables according to an *exponential distribution* ($X \sim Exp(\lambda)$, in figure 2) defined, respectively, by the arrival rate λ, the channel bandwidth μ_C and the service rate of the process μ_P.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

- A *crash failure* model is assumed for the processes, that fail following an exponential distribution according to a *failure rate* equal to $1/MTTF$, where the *Mean Time To Fail* is $MTTF = 30\ s$.
- The detection of a node's failure is assumed to happen within 10 s by all the others in the system.

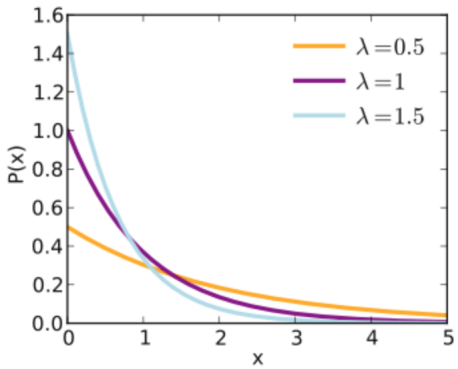

Figure 2: The Exponential Distribution.

2.2 Analyzed Scenarios

Under this system of assumptions, the evaluation was performed by running the broadcast algorithms varying the setting of the environment.
In particular, the system's parameters that where fixed to be able to compare the results among the protocols and the methodologies were:

- the duration of a run, always spanning over $d = 120\ s$;
- the *arrival rate* of the broadcast request, assumed to be equal to $\lambda = 0.3$ *requests/s* and meaning that each process receives the broadcast request of a message statistically every 3.3 s (as inter-arrival time, following the exponential distribution of rate λ);
- the *service rate* of the single node, which is the rate at which it is able to process a request by forwarding the message to every other member of the system, is equal to $\mu_P = 800$ *messages/s*;
- the *channel bandwidth* (service rate) of the communication channel is $\mu_C = 10$ *messages/s* ($\mu_C = 20$ in the *Eager Reliable Broadcast* runs with 30 nodes and no failures, to keep the system stable and avoid to operate at the nominal capacity, since the response time at maximum throughput is too high to be acceptable);
- for the *Eager Probabilistic Broadcast*, we also have the *fan-out* (i.e. the number of processes to which forward the message during the gossip propagation of the message) and the number of *rounds*, that are assumed to be both 2 in a system of 6 nodes and 3 with 30 nodes.

Starting from these values, the analysis took into account different system's settings, by playing with the *number of nodes* (to compare its performance in a "*small*" and in a "*large*" peer-to-peer network, respectively assuming 6 and 30 participants) and the possibility for the process to *crash* (observing the system both without interruptions and when processes can stop working).
The four scenarios in which every broadcast protocol was executed were:

1. "*small*" system composed by 6 nodes, assuming no crash failures;
2. "*small*" system composed by 6 nodes, assuming the presence of crash failures;
3. "*large*" system composed by 30 nodes, assuming no crash failures;
4. "*large*" system composed by 30 nodes, assuming the presence of crash failures.

2.3 Simulation

To carry out the simulation of the protocols, I developed "*Broadcast Communication*", a small framework in Python (available at [2]) which is able to run the chosen broadcast protocol among the three selected for this project under the environmental conditions specified by the user at the beginning (an example of execution is reported in figure 3). The tunable parameters are:

- the number N of nodes in the system;
- the broadcast algorithm to simulate;
- the arrival rate λ of the broadcast requests;
- the service rate μ_P of the process;
- the channel bandwidth μ_C;
- the possibility of processes to crash;
- the fan-out F (*Eager Probabilistic Broadcast*);
- the number of rounds R (*Eager Probabilistic Broadcast*).

At the end of the run (showed in figure 3), a report is produced in output, containing the statistics about the processes and the channels. The metrics taken into account for the single process are: the *total number of messages received* and the ones of *unique messages*, the number of *broadcast requests* it has received during the simulation, the *average response time* (as the expected time to deliver a message), the *average throughput* and the *average utilization*.
For each channels, it reports the *average response time*, *throughput* and *utilization*. Eventually, also some aggregated statistics are included, such as the *average total response time* for processing a request and the *mean throughput* and *utilization* for channels and processes.

2.4 Analytical Models

For what concerns the *analytical modeling*, the distinction was made according to the algorithm since, depending on its behavior and on the presence of failures for nodes, the sequence of the channel and the process receives a different kinds (and consequently loads) of requests to elaborate.
The channel and the process were modeled as an *open queuing network*, with the *serial* connection of two *M/M/1* queues (graphically represented in figure 4).
The analysis was done first under the assumption of *no crash failures* in the system.

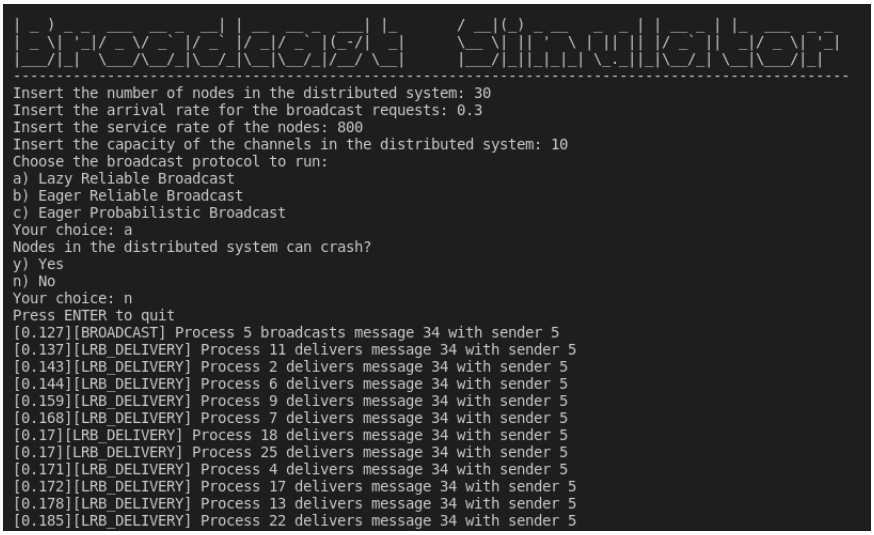

Figure 3: An execution of the framework.

2.4.1 Lazy Reliable Broadcast

In the *Lazy algorithm*, processes relay a message only if the sender fails by crash, which cannot happen under this setting.
To quantify the total response time R for a broadcast request to be processed, we have to sum the response times of the channel and the one of the process.
In particular, they receive a request following the exponential distribution of rate λ, thus we can compute R as follows:

$$R_{tot} = R_C + R_P = \frac{1}{\mu_C - \lambda} + \frac{1}{\mu_P - \lambda}$$

Since the system is stable, the number of completions is equal to the number of arrivals and the throughput is $X_C = X_P = \lambda$.
The utilization (%) of the single component of the network can be found by applying the *Utilization Law* (S is the service time, needed to process a request): $U_C = S_C * X_C = \frac{1}{\mu_C} * X_C$ and $U_P = S_P * X_P = \frac{1}{\mu_P} * X_P$.

2.4.2 Eager Reliable Broadcast

In the *Eager* algorithm, processes always retransmit the message upon its first delivery, even if there are no crashes: the single process will receive every message from all the

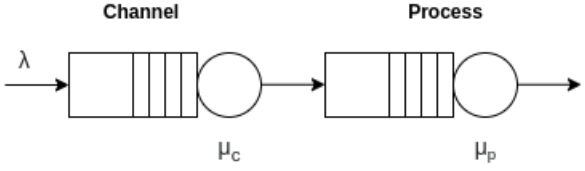

Figure 4: The open queuing network.

N members of the system ($\lambda' = N * \lambda$).
The average total response time R can be computed as follows:

$$R_{tot} = R_C + R_P = \frac{1}{\mu_C - N * \lambda} + \frac{1}{\mu_P - N * \lambda}$$

Since the system is stable, the number of completions is equal to the number of arrivals and the throughput is $X_C = X_P = N * \lambda$. The utilization of the single component of the network is given by the *Utilization Law*:
$U_C = S_C * X_C = \frac{1}{\mu_C} * X_C$ and $U_P = S_P * X_P = \frac{1}{\mu_P} * X_P$.

2.4.3 Eager Probabilistic Broadcast

In this case, the *gossip* procedure selects the processes to which send the message according to the `pick_target` function in every round: since we are assuming 2 or 3 rounds for the information dissemination (depending on the number of participants of the system), the process can receive the message to propagate at most 1 or 2 times respectively, and from at most the number of processes that have to gossip the message in that specific round (if it gets selected as target from all of them). For example, the latter number for a system of 30 nodes, assuming F=3 and R=3, is $n_1 = 1$ if $round = 1$ (only one process has the message to propagate), $n_2 = 3$ if $round = 2$ (4 processes discover the message but the one that propagated it in the previous round is now prevented to do the same) and $n_3 = 3^2 = 9$ if $round = 3$ (such that, after this round, at most 3^3 distinct processes may own that message, covering statistically the whole system).
Starting from these considerations, the arrival rate was considered to vary (since the retransmission load changes) depending on the round and on the number of processes owning a particular message to relay: it was quantified as $\lambda_{r=i} = p_{target} * \lambda * n_i$, where $p_{target} = \frac{F}{N-1}$ is the probability to be chosen as the target in the gossip procedure (quantified as the fan-out over all the possible target nodes in the system, except the one relaying the message) and n_i is the number of processes which can send this message to the one under analysis in that round $r = i$.
Also the service rate and the channel capacity need to be adapted to the arrival rates and the rounds, so I assumed: $\mu_{C,1} = 10$ msgs/s, $\mu_{C,2} = 12$ msgs/s, $\mu_{C,3} = 15$ msgs/s and $\mu_{P,1} = 800$ msgs/s, $\mu_{P,2} = 850$ msgs/s, $\mu_{P,3} = 900$ msgs/s.
This is a *multiple class open network* and the computation of the response time R was carried out following the formulas for the utilization U, the residence time R' and the response time R, exposed in the book *Capacity Planning* ([3]).

$$U_{C,i} = \lambda_{r=i} * S_{C,i} = \lambda_{r=i} * \frac{1}{\mu_{C,i}}$$

$$U_C(\overrightarrow{\lambda}) = \sum_{r=1}^{R} U_{C,r}$$

$$U_{P,i} = \lambda_{r=i} * S_{P,i} = \lambda_{r=i} * \frac{1}{\mu_{P,i}}$$

$$U_P(\overrightarrow{\lambda}) = \sum_{r=1}^{R} U_{P,r}$$

$$R'_{C,i} = \frac{S_{C,i}}{1 - U_C(\overrightarrow{\lambda})} = \frac{\frac{1}{\mu_{C,i}}}{1 - U_C(\overrightarrow{\lambda})}$$

$$R'_{P,i} = \frac{S_{P,i}}{1 - U_P(\overrightarrow{\lambda})} = \frac{\frac{1}{\mu_{P,i}}}{1 - U_P(\overrightarrow{\lambda})}$$

$$R_{r=i} = R'_{C,i} + R'_{P,i}$$

$$R_{avg} = \frac{\sum_{j=1}^{R} R_{r=j}}{R}$$

The throughput, always assuming a stable system, is:

$$X_P = \sum_{j=1}^{R} X_{P,r=j} = \sum_{j=1}^{R} \lambda_{r=j} = \sum_{j=1}^{R} X_{C,r=j} = X_C$$

.

Then, I repeated the evaluation assuming the fact that at most half of the processes in the system can crash, which profoundly complicate the analysis.

2.4.4 Lazy Reliable Broadcast with crash failures

Following this broadcast protocol, the nodes, which can now crash, will broadcast again a message that they receive from a faulty sender and, upon the detection of a failure, they retrasmit all the messages received from it.
In particular, the model consider that the number of messages that the correct nodes have to broadcast again can be bounded under the assumption that the crashes are detected within 10 s by all the correct processes.
Indeed, considering the arrival rate $\lambda = 0.3$ $requests/s$ (average inter-arrival time equal to 3.3 s) and the $MTTF = 30$ s, a node that will crash can send at most $\frac{10}{3.3}$ messages before being detected (that a correct process will have to retrasmit detecting the failure, so $m = 3$ messages), as represented in figure 5.
Hence, also this case is a *multiple class open network* where a single process receives two kinds of requests, with different retrasmission loads:

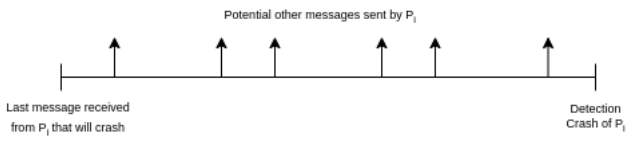

Figure 5: P_i sends messages before the detection of its failure.

- the usual broadcasts requests at $\lambda_1 = 0.3\ requests/s$, by hypothesis;
- the requests for the broadcast of the messages received by the crashed node upon its detection $\lambda_2 = \frac{N}{2} * \frac{1}{MTTF} * m$, where $\frac{N}{2}$ is the number of processes which can fail, $\frac{1}{MTTF}$ is the failure rate and m is the maximum number of messages received from the crashed process in the time interval needed for its failure's detection.

The formulas needed to model this scenario are always the ones reported by the book *Capacity Planning* ([3]).

$$U_{C,i} = \lambda_i * S_{C,i} = \lambda_i * \frac{1}{\mu_{C,i}}$$

$$U_C(\overrightarrow{\lambda}) = \sum_{i=1}^{2} U_{C,i}$$

$$U_{P,i} = \lambda_i * S_{P,i} = \lambda_i * \frac{1}{\mu_{P,i}}$$

$$U_P(\overrightarrow{\lambda}) = \sum_{i=1}^{2} U_{P,i}$$

$$R'_{C,i} = \frac{S_{C,i}}{1 - U_C(\overrightarrow{\lambda})} = \frac{\frac{1}{\mu_{C,i}}}{1 - U_C(\overrightarrow{\lambda})}$$

$$R'_{P,i} = \frac{S_{P,i}}{1 - U_P(\overrightarrow{\lambda})} = \frac{\frac{1}{\mu_{P,i}}}{1 - U_P(\overrightarrow{\lambda})}$$

$$R_i = R'_{C,i} + R'_{P,i}$$

$$R_{avg} = \frac{\sum_{i=1}^{2} R_i}{2}$$

The throughput is $X_P = \sum_{i=1}^{2} X_{P,j} = \sum_{j=1}^{2} \lambda_j = \sum_{j=1}^{2} X_{C,j} = X_C$.

2.4.5 Eager Reliable Broadcast with crash failures

For the *Eager Reliable Broadcast*, there are no relevant changes with respect to the previous analysis since the algorithm foresees the retrasmission of the message upon its first delivery, regardless the presence of crash failures.
The difference is in the fact that, on average, now there are $\frac{N}{2}$ correct processes in the system: the single processes will receive the same message from each of them ($\lambda' = \frac{N}{2} * \lambda$). The average total response time R is computed as follows:

$$R_{tot} = R_C + R_P = \frac{1}{\mu_C - \frac{N}{2} * \lambda} + \frac{1}{\mu_P - \frac{N}{2} * \lambda}$$

Since the system is stable, the number of completions is equal to the number of arrivals and the throughput is $X_C = X_P = \lambda'$.
The utilization of the single component of the network is given by the *Utilization Law*: $U_C = S_C * X_C = \frac{1}{\mu_C} * X_C$ and $U_P = S_P * X_P = \frac{1}{\mu_P} * X_P$.

2.4.6 Eager Probabilistic Broadcast with crash failures

Also in the *Probabilistic Broadcast*, we repeat the same analysis done before, since the algorithm already works for *asynchronous* systems, but assuming that now only $\frac{N}{2}$ are the correct processes remaining in the system.
Thus, the arrival rates of the requests are quantified always depending on the round and on the number of processes owning a particular message to relay but as $\lambda_{r=i} = p_{target} * \lambda * n_i$, where $p_{target} = \frac{F}{\frac{N}{2}-1}$ is the probability to be chosen as target for the gossip and n is the maximum number of processes from which receive the message in that round r.
Since it is still a *multiple class open network*, the formulas are the same as before ([3]).

2.5 Results of the Analysis

For the results, the following notation was used:

- R_{tot} is the total average *response time* for a request (considering a generic channel followed by a generic node), measured in *seconds*;
- X_{node} is the average *throughput* for a node in terms of *processed requests/s*;
- $X_{channel}$ is the average *throughput* for a channel (*sent messages/s*);
- U_{node} is the *utilization* (%) for a node;
- $U_{channel}$ is the *utilization* (%) for a channel;
- N_{crash} is the *number of crashes* happened during the simulation (if expected by the considered scenario).

The results of the *simulation* using the framework are showed in the following tables:

Lazy Reliable Broadcast						
Scenario	R_{tot}	X_{node}	$X_{channel}$	U_{node}	$U_{channel}$	N_{crash}
1	0.0872	0.3335	0.4596	0.0417	4.6955	-
2	0.1265	0.3730	0.2694	0.0466	2.6943	3
3	0.1245	0.2757	0.3956	0.0332	3.9065	-
4	0.1255	0.5319	0.3972	0.0664	3.9720	15

Eager Reliable Broadcast						
Scenario	R_{tot}	X_{node}	$X_{channel}$	U_{node}	$U_{channel}$	N_{crash}
1	0.1437	2.1174	2.8000	0.2647	28.0012	-
2	0.3835	0.5029	0.4943	0.0629	4.9433	3
3	25.4329	6.0985	8.4915	0.7623	42.4574	-
4	2.4347	0.9326	1.0717	0.1166	10.7169	15

Eager Probabilistic Broadcast								
Scenario	R_{tot}	X_{node}	$X_{channel}$	U_{node}	$U_{channel}$	N_{crash}	F	R
1	0.1833	0.3049	0.2961	0.0381	2.9609	-	2	2
2	0.0762	0.2635	0.2878	0.0307	2.6777	3	2	2
3	0.1672	0.3316	0.2151	0.0415	2.1505	-	3	3
4	0.1849	0.1677	0.0869	0.0209	0.8694	11	3	3

The results of the *analytical* modeling are reported right away:

Lazy Reliable Broadcast						
Scenario	R_{tot}	X_{node}	$X_{channel}$	U_{node}	$U_{channel}$	N_{crash}
1	0.1044	0.3	0.3	0.038	3	-
2	0.0848	0.6	0.6	0.1	5	3
3	0.1044	0.3	0.3	0.038	3	-
4	0.0962	1.8	1.8	0.3	13	15

Eager Reliable Broadcast						
Scenario	R_{tot}	X_{node}	$X_{channel}$	U_{node}	$U_{channel}$	N_{crash}
1	0.1232	1.8	1.8	0.23	18	-
2	0.1112	0.9	0.9	0.1	9	3
3	0.0922	9	9	1.125	45	-
4	0.1913	4.5	4.5	0.6	45	15

Eager Probabilistic Broadcast								
Scenario	R_{tot}	X_{node}	$X_{channel}$	U_{node}	$U_{channel}$	N_{crash}	F	R
1	0.9556	0.3	0.3	0.04	2.7	-	2	2
2	0.1012	0.9	0.9	0.1	8	3	2	2
3	0.0872	0.587	0.587	0.05	2.99	-	3	3
4	0.09	0.8357	0.8357	0.1	6.12	15	3	3

2.6 Dependability Evaluation

The *dependability* is the system's ability to deliver a service which can justifiably be trusted and to avoid service failures which are more severe and frequent than acceptable. As assumed for the performance evaluation, the channels are not going to fail but, for the processes, a crash failure model was assumed, with a $MTBF = 30$ s (thus, a failure rate of $\lambda_{crash} = 0.033$ $failures/s = 118.8$ $failures/hour$, obviously not acceptable for a real system).

The *reliability function R(t)* is the probability that a component functions properly in the time interval of the analysis. Under the assumptions of a constant failure

rate λ_{crash} (independent of time), it corresponds to the one of the process so $R(t) = e^{-\lambda_{crash}*t} = e^{-118.8*t}$ (figure 6). Considering that a client can make a broadcast request to whatever node, the system can be seen as the *parallel* interconnection of this components previously analyzed: the reliability function of the whole system becomes $R(t) = 1 - (1 - e^{-118.8*t})^N$.

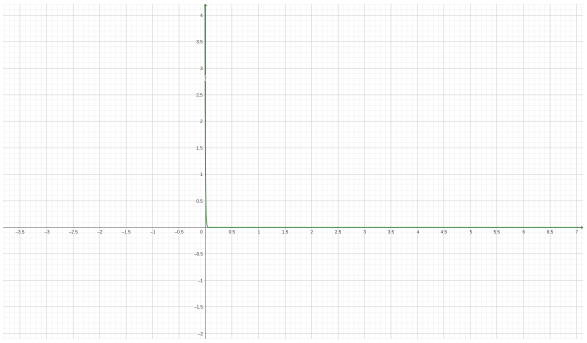

Figure 6: The reliability function R(t) for a single node.

To evaluate of the *steady-state availability* of a single component of the system, constituted by the *serial* connection of the channel and the process, the assumption that the node can be *repaired* or *replaced* (each fault occurrence and repair is independent) must be added: the *Mean Time To Repair* is assumed to be $MTTR = 10\ s$.
The steady-state availability of the component is the same as the one of the node (the channel has $A_C = 1$) and it is $A_P = \frac{MTBF}{MTBF+MTTR} = \frac{30}{30+10} = 0.75 = 75\%$.
Considering now the whole system composed by N parallel components, the system's steady-state availability is: $A_{system} = 1 - \prod_{i=1}^{N}(1 - A_{P,i}) = 1 - (1 - A_{P,i})^N$, so $A_{system} \cong 1 = 100\%$ if there are 30 nodes and $A_{system} = 0.9997 = 99.97\%$ if there are 6 nodes.

3 Comments

3.1 Validation

After the analytical modeling and the simulation, the *validation* step is needed to check how *close* the values predicted by the model are with respect to the ones measured during the simulation, to understand if the model reflects the "real" (simulated) behavior of the system.
Indeed, the theoretical analysis may prove that the performance guarantees are respected, while the simulation shows the opposite: for this reason, validation is needed to build *credibility* for the analytical model and the simulation, to make them acceptable for the domain experts that will use it.
There are different techniques that can be applied (expressed in [4]).
The main guidelines followed in this project were:

- run the simulation under *different settings and inputs* and check that the output are *reasonable*;
- *verify* that the estimates of the parameters given by the two methodologies are *close*, to build a certain degree of *confidence* about the correctness of the analysis.

Thus, this step was performed by *comparing* the results given for every metrics in each of the four scenarios taken into account for the three analyzed algorithms.

3.1.1 Lazy Reliable Broadcast

1. Considering a system of 6 nodes and no failures, the metrics' values are quite close, except for the average throughput and utilization of the channel (e.g. the gap between $U_{channel,simulation}$ and $U_{channel,analytical}$ is of 1.7%);
2. Adding the possibility to crash for the processes, also the node's metrics are more distant in the two analysis but the biggest difference is in the average response time R, since it is smaller with respect to the one of the first scenario for the analytical model but for the simulation we get the opposite result.
3. In a system of 30 nodes with no failures, the metrics are estimated almost in agreement by the two techniques.
4. Considering also crash failures for processes, the estimations of the response time and of the throughput are within a small gap while the utilization is profoundly overestimated by the analytical model ($U_{channel,simulation} = 3.972\%$ and $U_{channel,analytical} = 13\%$).

3.1.2 Eager Reliable Broadcast

1. In a system of 6 nodes and no failures, the analytical model slightly underestimates R (of 0.02 s) and the other parameters. The main difference is in the utilization, where $U_{channel,simulation} = 28\%$ and $U_{channel,analytical} = 18\%$.

2. Conversely, adding the crash of the processes, the analytical model overestimates the parameters, quantified as almost the double of the ones returned by the simulation.

3. Considering a system of 30 nodes with no failures, the measures of the average throughput and utilization for the channel and the node are quite close, but the response time experienced in the simulation is much higher.

4. With crash failures for processes, the response time R is similar in the two analysis but the other metrics are all far from each other: especially, they are overestimated by the analytical model.

3.1.3 Eager Probabilistic Broadcast

1. Considering a system of 6 nodes and no failures, the measurements of throughput and utilization are close, but the response time R is underestimated when computed analytically.

2. Adding the possibility to crash for the processes, the results are more distant, with the simulation that reports smaller values for each parameter.

3. In a system of 30 nodes with no failures, the metrics are closer, except for the response time R, which is almost the double for the simulation.

4. Also considering crash failures for processes, the response time is still considered to be double by the simulation with respect to the one returned by the analytical model but all the other metrics are overestimated by the latter modeling.

After the observations done in the validation phase, in real systems the *calibration* is executed, to tune the model accordingly to make it fitting better the simulated system's behavior (it was not performed during this project).

3.2 Choosing the Right Algorithm

Once validated the results, some considerations can be done relying on these outputs. In particular, based on the simulation's outputs, we can observe that the *Lazy Reliable Broadcast* algorithm is the one that performs better in every metrics almost in the majority of the scenarios, both for small and large systems and with or without crash failures for the nodes.

This result confirms what we could already intuitively state, since this protocol is minimizing the number of retrasmissions of the same messages.
The only exception is the case of a *small* system (assumed to have 6 nodes in this project) with *crashes* where, conversely, the *Eager Probabilistic Broadcast* behaves better in all the parameters.

3.3 Limitations

The reason of the *discordance* in the results returned by the simulation and by the analytical modeling needs to be searched in the fact that both the methodologies present their own *limitations*, which are quite complex to mitigate.

3.3.1 Limitations in Simulation

The simulation's outputs are mainly influenced by the *intrinsic* limitations of the simulation framework and of Python, the programming language used for its development.
Indeed, despite the multi-threading implementation, the Python's interpreter runs in a *single process* and the framework deals directly with the *sockets*, opening and closing them "by hand" and increasing the *overhead*.
Another influential factor is that each channel is *bidirectional* and includes a queue for each process on its sides but it implements the receiving and the forwarding of the messages in two threads, working in both the directions. This is also why the utilization of the channel is often *underestimated* by the analytical modeling.
Moreover, another limitation is the presence of the processes all *on the same machine*, sharing its resources: this is exactly the opposite of real systems, in which they are distributed over several entities in a network.
For this reason, also the number of processes which can be simulated by the framework *cannot* be too *high*, to avoid invalidating the measurements due to the resource's contention. Besides, the values of the metrics can be affected by a huge variance which can be *mitigated* only running a *much larger* number of experiments.

3.3.2 Limitations in Analytical Modeling

On the other side, analytical modeling is not always easy to perform, becoming even almost *impossible* to carry out for representing complex behaviors and systems.
In the case of this analysis, the main complication was in understanding the arrival rate of the processes when retrasmission was needed due to crashes: indeed, the move from a single class to a multiple class open network was necessary to have a more "realistic" modeling of the nodes' load (still not so accurate).
Talking of load, it is also worth noticing that the analytical models are *not load dependent* as the real systems they aim to represent, whose performances may vary with its fluctuations.

Another aspect to consider is that the analytical results were computed assuming always the *worst case* scenario in terms of messages that a process receives.

3.3.3 Limitations of the Project

There are also other limitations in the way the project was carried out, since no techniques to increase the *validity* and the *credibility* of the results were embedded. Some of them (reported by [4]) are that, as a modeler, I could not benefit from a complete and accurate set of information about the real system (since there isn't one) and a formal (quantitative) *results validation* leveraging on statistical procedures was not performed, in favor of a simpler qualitative analysis.

4 Conclusions

As the complexity of distributed systems increases every year, the importance of broadcast primitives is becoming greater, to support reliable and efficient communications among processes, even in presence of failures.
This project offered me the occasion to carry out a formal *performance evaluation* of broadcast protocols, comparing their behaviors and discussing which is more suitable under different system's scenarios.
Both *simulation* and *analytical modeling* were explored, to be able to (qualitatively) validate the values of the chosen metrics.
In this way, I could experiment by myself the complexity of developing analytical models in real applications and how it can differ in its outputs from the values returned by the simulation.
To further develop this work, several improvements can be put in place: mainly, the implementation of the algorithms should be done on a real distributed system of *physically separated* machines and the *number of experiments* should be much *larger*, to have a more precise quantification of the metrics taken into account for the analysis.

References

[1] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[2] Angelo Casciani. *Broadcast Communication*, 2022. `https://github.com/AngeloC99/Broadcast-Communication`.

[3] Daniel A. Menasce and Virgilio A.F. Almeida. *Capacity Planning for Web Performance: metrics, models, and methods*. Prentice-Hall, Inc., 1998.

[4] Averill M Law, W David Kelton, and W David Kelton. *Simulation modeling and analysis*, volume 3. Mcgraw-hill New York, 2007.

[5] Silvia Bonomi and Giovanni Farina. *Dependable Distributed Systems, Slides of the Course*. Sapienza, University of Rome, 2022.