Uniform Reliable Broadcast Protocol

Dineshkumar RAJAGOPAL (ENSIMAG, Grenoble INP)
dineshkumar.rajagopal@ensimag.grenoble-inp.fr
Distributed System (M2R-MoSIG 2014-205)
18/12/2014

Introduction:

In this practical work we studied typical Distributed algorithm called **Uniform Reliable Broadcast**. Reliably deliver the message to all the correct process from the sending correct process with the condition of dynamic environment (i.e. failure in the processes, etc). We have to find an optimal algorithm for Uniform Reliable Broadcast (i.e. minimum number of messages sent to deliver it to all the processes), because it will be reused in different Distributed applications. For the practical work we assumed with some condition and implemented the distributed protocol in Java and measured the performance in ENSIMAG laboratory computers. From here we abbreviate Uniform Reliable Broadcast as URB. The report of practical work is organized as follows **1) URB specifications and assumptions** cover the URB problem specification and assumptions for which we developed algorithm. **2) Algorithm of URB protocol** covers algorithm for the Uniform Reliable Broadcast, proof and analysis 3**) Implementation of URB protocol** covers the implementation of specified algorithm in Java with Threads and TCP communication **4) Performance evaluation of URB protocol** explains the evaluation results from the implemented protocol in the environment of ENSIMAG lab computers, finally **5) Conclusion** will be summarized the overall work and understanding.

1) URB specifications and assumptions:

The Uniform Reliable Broadcast must satisfy the properties of Validity, Integrity and Agreement. Perfect Failure Detector (PFD) is needed for the URB to work finely.

1.1) URB specification:

1.1.1) Validity:

If a correct process sends a message “m”, then it eventually delivers it only once.

1.1.2) Integrity:

A process “P” delivers a message “m” only once and if “m” was sent by a process.

1.1.3) Agreement:

If a process deliver message “m”, then all the correct processes eventually delivers message “m” .

1.2) Assumptions for implementation:

- Fixed known set of processes – 4 processes in the set.
- Crash faults – If the process failed, then we have to detect and remove the crashed process from the set of processes.
- Reliable links – TCP Socket is used for communication.
- Perfect failure detector – If we find the process is failed then it is failed without any assumption.

2) Algorithm of URB protocol:

For broadcasting message and failure detecting we will use our own set of message format, that is shown in the Figure-1 and Table -1 explains different message types.

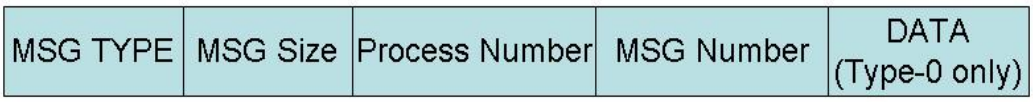

Figure – 1. Message Frame Format

MSG Type	Purpose
0	Data
1	ACK
2	ALIVE
3	Alive Success
4	Alive Failure

Table - 1. MSG Types

2.1) Pseudo-code in C syntax:

Global Variables:

Int currentProcessNumber ; /* current process number will be given from this, and the process set and process number are {1,2,3,4} */

```
Int * createMessage( int size ) {
/*************************************************************
        Parameters   : Size of the message need to allocate
        Return       : Specified message with correct sequence number
        Function     : Create a message with specific size

        *************************************************************/
        Int message[size];
        Message[3] = ++previousMN;  // previous MSG # will be increased
        If( size > 3) {
                For ( i=4; i< size ; i++ )
                        Message[i] = - 2;
        }
        Return message;
}

Void broadcastMessgae( int *message ,int size) {
/*************************************************************
        Parameters   : message and size of the message is an input
        Return       :
        Function     : Either broadcast the message to next correct process or
                Sent Acknowledgment for the message to next correct process

        *************************************************************/

        nextCorrectProcess = findNextCorrectProcess();

        /* findNextCorrectProcess() will find the next correct process with the
        help of Failure Detector and return the next correct process number */

        If ( (message[2] + 2 ) % 4 + 1 == currentProcessNumber) {
                waitForAck( message[3] ) ;
                Int ackMSG[4]=createMessage(4);
                For ( int i =0 ; i<4 ; i++ )
                        ackMSG[i] = message[i] ;
                ackMSG[0] = 1 ;     // MSG Type is ACK -1
                deliverMessage(message[3]);
                sendMsgToNext(ackMSG,size,nextCorrectProcess);
        /* sendMsgToNext will establish a TCP connection and send the
message correctly to the next correct process */
```

```
        }
        Else {
                SendMsgToNext(message,size,nextCorrectProcess) ;
                waitForAck( message[3] );    /* message number will be sent as
a parameter */
        }


}

Void deliverMessage( int messageNumber) {
/************************************************************
        Parameters   :
        Return       :
        Function     : deliver the message

        ************************************************************/
        If( waitingMSGArray[ messageNumber % 1000 ] ==
messageNumber) {
                println("The message - " + messageNumber + "was
Successfully delivered in the process Number - " + currentProcessNumber);
        }
        Else
                println("The message - " + messageNumber + "was not
delivered in the process Number - " + currentProcessNumber);

}

Void waitForAck( int messageNumber ) {
/************************************************************
        Parameters   : message number as a parameter
        Return       :
        Function     : it will store this value in the waiting message array

        ************************************************************/
        waitingMSGArray[ messageNumber % 1000 ]  = messageNumber;
        /* message number is from 0 to 999 */

}
```

2.2) Proof:

Proving the above algorithm mean it needs to satisfy all the properties in the specification, so we can prove specification by Natural language statement.

Validity:

As per validity only once it should be delivered in the sending processes. This property is satisfied in our algorithm because every processes are receiving messages only once. There is no way to get the duplicate messages so Delivery is not be duplicated in the sending process itself.

Integrity:

Processes are receiving messages only once. There is no way to get the duplicate messages so every processes will receive and deliver the message only once, when a message "m" was sent by a process.

Agreement:

As per agreement, If a process delivers message "m", then the message "m" is received in all the correct process. So all the processes received message and waiting for the Acknowledgment will eventually delivered.

These all these property will be satisfied in our Algorithm while the Perfect Failure Detector is perfect.

2.3) Analysis:

Size of Input	: Number of processes (n)
Primitive operation	: Number of messages sent
Timing analysis T(n)	: 2(n-1) [excluding overhead of Failure Detector]
	: 4(n-1) [including overhead of Failure Detector]

3) Implementation of URB protocol:

Implemented the URB protocol algorithm in Java 1.6.0 (Eclipse Luna IDE) with the Java TCP sockets for communication and server threaded process to connect with a client. In java two socket classes for client (Socket) and server (ServerSocket) , if the process want to communicate with each other than two sockets should be needed in each processes.

Failure detecting also it needs two sockets to check current process is alive or not. For modularity we made three classes. Threaded server in each process. so it will assign each server thread for the client to communicate.

- ClientBroadcast - createMessage, broadcastMessage, sendMsgToNext
- ServerBroadcast - deliverMessage, waitForAck
- FailureDetector - findNextCorrectProcess

All the functions are same as the pseudo-code of the algorithm correspondingly. The message passing between internal and external of the processes complete picture is shown in Figure-2.

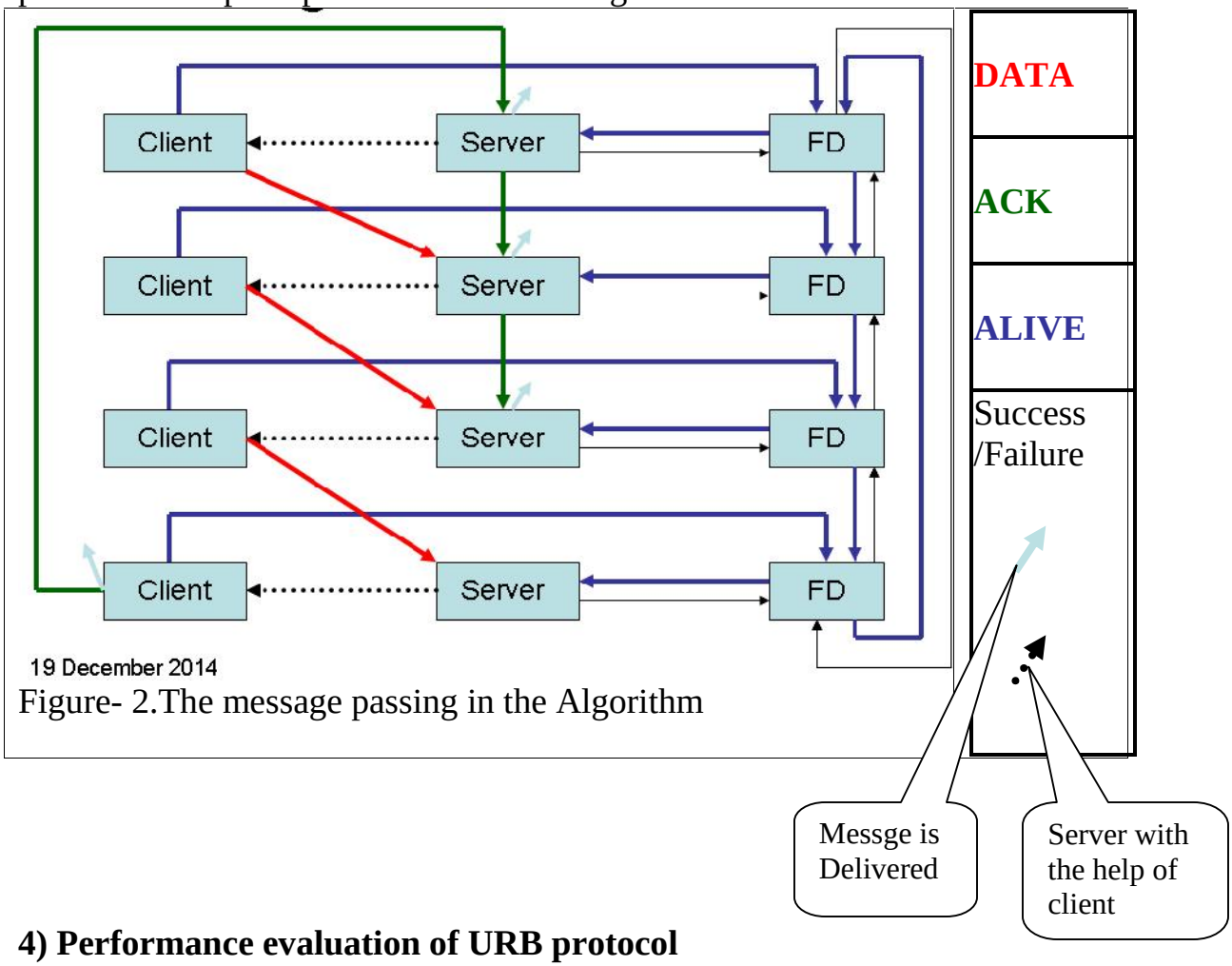

Figure- 2.The message passing in the Algorithm

4) Performance evaluation of URB protocol

For the performance evaluation, I did an experiment1 and experiment2 in the ENSIMAG enipsys pc room to have four computers and found the results in Figure-3 and 4 respectively.

In the experiment-1 one client send hundred messages repeatedly and measured the messages delivered in the ending Acknowledgment process

with that Large size gets delivered very less than the Small size messages and In the mid it is waiting for reading message long time.

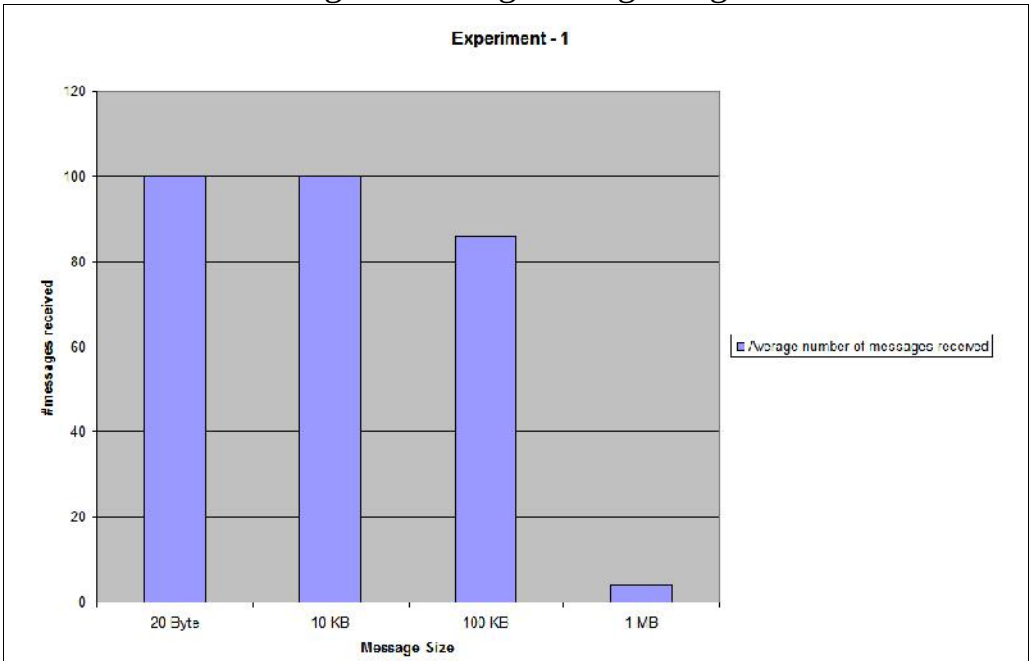

Figure-3. Single process sending with varying message size and throughput

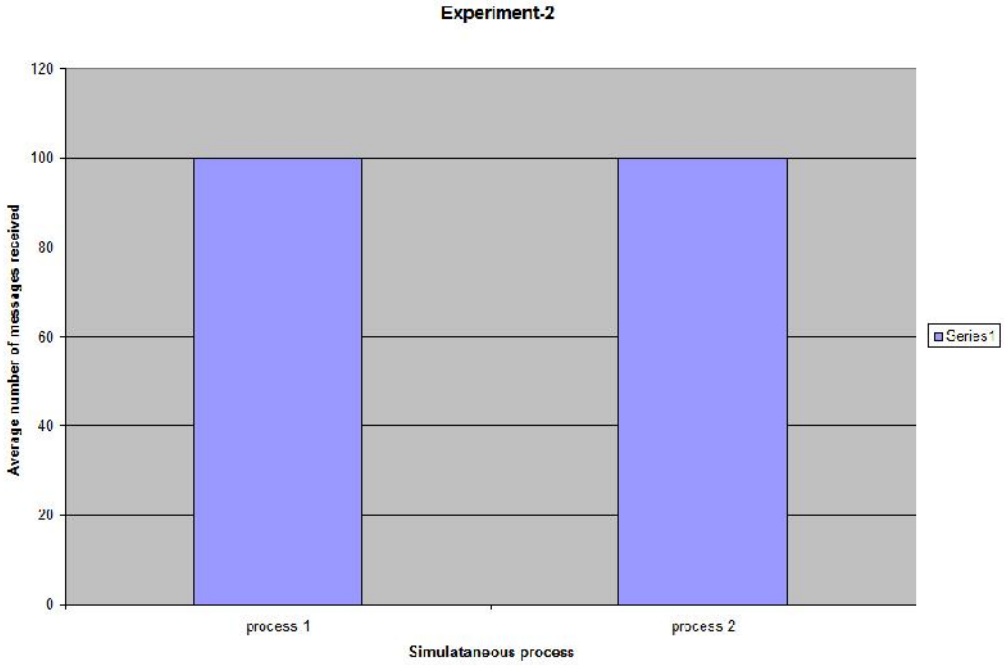

Figure-4. Simultaneously two processes sending 10 kB message and throughput

In the experiment-2 the throughput does not affect by simultaneously sending processes. In the both the results the Buffer size of the TCP is the matter for throughput, in 1MB process it will exhaust the Buffer with the few messages, after the process is waiting for too long to process the earlier messages in the buffer.

5) Conclusion

In this practical work we had an opportunity to explore Distributed algorithm from specification (problem statement) to performance measurement. If we made a mistake in the one step then it will follow in the further steps also, so taking decision in each step should be precious. In the performance analysis I got a strange result that sending big messages taking fewer throughputs than the small messages. As well sending message from simultaneous processes does not affect the throughputs in the protocol.