

Distributed Systems CA1 Report

Hosna Niavarani 810196627

Models

The system consists of three models:

- Broker
- Publisher
- Subscriber

Broker

Specification:

PublishChannel: A buffered channel of messages with a specific buffer size on which publishers publish their messages.

BufferNotFullCond: An instance of “sync.Cond” used by publishers and broker to handle the full buffer problem.

SubscriberIds: A list of subscriber IDs

NextSubscriberId: the next subscriber ID not yet taken

Queuing publisher messages:

The main job of the broker – dequeuing messages inside “PublishChannel”’s buffer and sending them to all clients – is done inside the “Queue” function. This function reads messages sent onto “in_ch”, extracts the string part and sends a post request containing the message to all subscribers.

Listening for RPC’s:

For client subscription and unsubscription and server publishing, the broker listens on a specific host and port and responds to these remote procedure calls by clients and servers. The broker listens for RPC requests on a specific host and port known by all clients and servers.

Subscriber

Subscription:

If a client wishes to subscribe to a broker, it calls the “Subscribe” function using an RPC call which returns a unique subscriber ID. Inside this function subscriber’s ID is added to the “SubscriberIds” and “NextSubscriberId” is incremented by one.

Unsubscription:

If a client wishes to unsubscribe from the server it can do so by calling “Unsubscribe” function using RPC. In this function the ID of subscriber is removed from “SubscriberIds”.

Listening for messages:

Each client after subscribing to server’s messages via broker’s “Subscribe” function receives a unique ID. After that it listens for post messages from broker containing publisher messages on “localhost:PORT” where PORT is the base “CLIENT_LISTEN_PORT” defined as a constant plus the client’s ID.

Publisher

Publishing:

The publisher can publish messages by remotely calling the “Publish” function implemented inside broker using RPC.

If a publisher is sending messages synchronously it blocks until broker dequeues its message from buffer and sends it to the subscribers. Whereas in asynchronous send, publisher waits inside a goroutine for message in a non-blocking fashion.

The “Publish” function blocks until it receives *receive status* from all clients. It does this by initializing a separate channel for each message published and waiting on this channel (in “WaitForStatus”) until the broker sends the receive status on it.

Whether the publisher waits synchronously or asynchronously is determined by how it calls the “Publish” function. If it calls it inside a separate goroutine then it is considered asynchronous and the publisher can proceed with other publishes without waiting for the status of the previous ones to arrive. On the other hand if it calls it inside the main program, it blocks until publishing has finished.

To make sure all status messages are received before publisher exits, an instance of “WaitGroup” called “ongoingPublishes” is passed to all publish goroutines. After each goroutine is done publishing, it decrements it by one and the publisher should wait on “ongoingPublishes” before exiting.

The full buffer problem

In order to handle publisher messages when broker’s buffer is full, we created a condition variable called “BufferNotFullCond” that is signaled by broker every time it dequeues a message from queue.

On the other hand for each message failed to send on channel due to full buffer, publisher waits on this condition and retries sending when the condition evaluates to true.

Usage

Start broker, client and server on three different terminals using:

Terminal1:

```
go run broker.go  
/s BUFFSIZE
```

Terminal2:

```
go run client.go  
/s
```

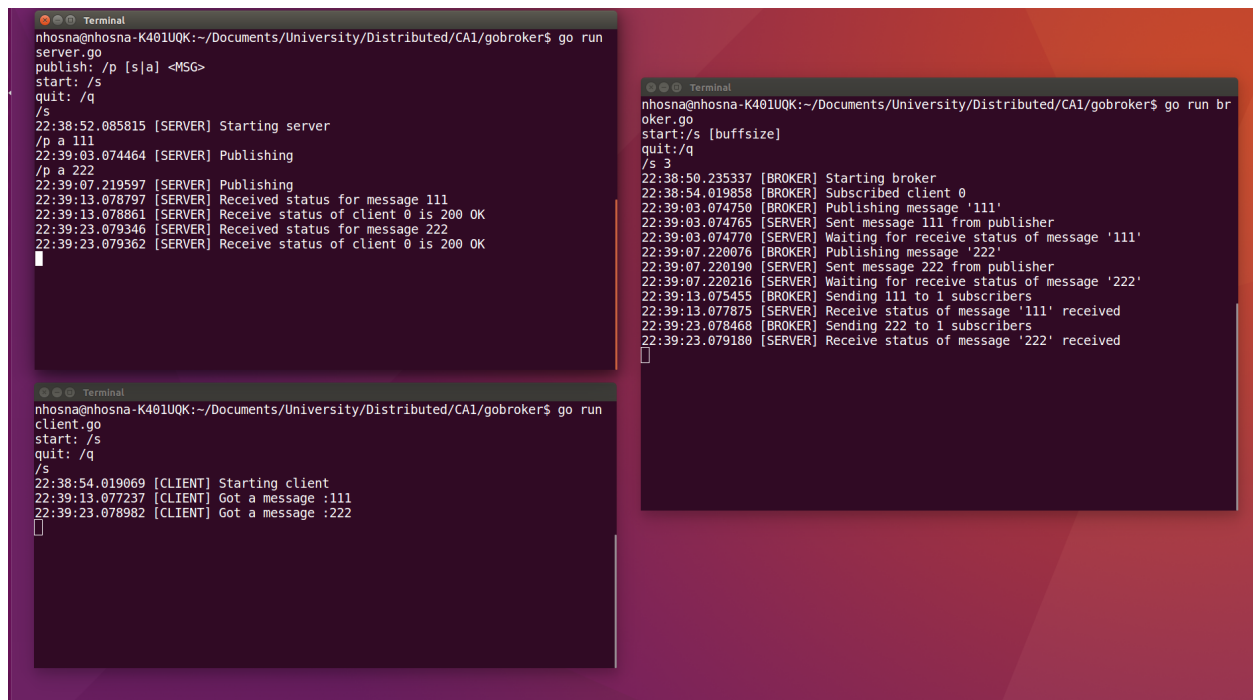
Terminal3:

```
go run server.go  
/s  
/p a MSG or /p s MSG
```

Scenarios

Asynchronous publish

When publishing is asynchronous server does not wait for the result of the previous publish and can have multiple messages sent immediately. In the publisher console after the second publish, it is processed immediately and the status of both messages are received later.



```
Terminal
nhosna@nhosna-K401UQK:~/Documents/University/Distributed/CA1/gobroker$ go run
server.go
publish: /p [s|a] <MSG>
start: /s
quit: /q
/s
22:38:52.085815 [SERVER] Starting server
/p a 111
22:39:03.074464 [SERVER] Publishing
/p a 222
22:39:07.219597 [SERVER] Publishing
22:39:13.078797 [SERVER] Received status for message 111
22:39:13.078861 [SERVER] Receive status of client 0 is 200 OK
22:39:23.079346 [SERVER] Received status for message 222
22:39:23.079362 [SERVER] Receive status of client 0 is 200 OK
█

Terminal
nhosna@nhosna-K401UQK:~/Documents/University/Distributed/CA1/gobroker$ go run br
oker.go
start:/s [buffsize]
quit:/q
/s 3
22:38:50.235337 [BROKER] Starting broker
22:38:54.019858 [BROKER] Subscribed client 0
22:39:03.074750 [BROKER] Publishing message '111'
22:39:03.074765 [SERVER] Sent message 111 from publisher
22:39:03.074770 [SERVER] Waiting for receive status of message '111'
22:39:07.220076 [BROKER] Publishing message '222'
22:39:07.220190 [SERVER] Sent message 222 from publisher
22:39:07.220216 [SERVER] Waiting for receive status of message '222'
22:39:13.075455 [BROKER] Sending 111 to 1 subscribers
22:39:13.077875 [SERVER] Receive status of message '111' received
22:39:23.078468 [BROKER] Sending 222 to 1 subscribers
22:39:23.079180 [SERVER] Receive status of message '222' received
█

Terminal
nhosna@nhosna-K401UQK:~/Documents/University/Distributed/CA1/gobroker$ go run
client.go
start: /s
quit: /q
/s
22:38:54.019069 [CLIENT] Starting client
22:39:13.077237 [CLIENT] Got a message :111
22:39:23.078982 [CLIENT] Got a message :222
█
```

Synchronous publish

When publishing is synchronous server waits for the status of the published message before resuming. So a second publish command would be processed only after the status of the first is received.

```
Terminal
nhosna@nhosna-K40IUQK:~/Documents/University/Distributed/CA1/gobroker$ go run
server.go
publish: /p [s[a] <MSG>
start: /s
quit: /q
/s
22:36:14.731582 [SERVER] Starting server
/p s 111
22:36:17.913942 [SERVER] Publishing
/p s 222
22:36:27.918210 [SERVER] Received status for message 111
22:36:27.918245 [SERVER] Receive status of client 0 is 200 OK
22:36:27.918282 [SERVER] Publishing
22:36:37.923717 [SERVER] Received status for message 222
22:36:37.923742 [SERVER] Receive status of client 0 is 200 OK
█

Terminal
nhosna@nhosna-K40IUQK:~/Documents/University/Distributed/CA1/gobroker$ go run
client.go
start: /s
quit: /q
/s
22:35:56.105728 [CLIENT] Starting client
22:36:27.918003 [CLIENT] Got a message :111
22:36:37.920315 [CLIENT] Got a message :222
█
```

```
Terminal
nhosna@nhosna-K40IUQK:~/Documents/University/Distributed/CA1/gobroker$ go run br
oker.go
start:/s [buffsize]
quit:/q
/s 3
22:35:52.009006 [BROKER] Starting broker
22:35:56.106536 [BROKER] Subscribed client 0
22:36:17.914250 [BROKER] Publishing message '111'
22:36:17.914260 [SERVER] Sent message 111 from publisher
22:36:17.914263 [SERVER] Waiting for receive status of message '111'
22:36:27.914597 [BROKER] Sending 111 to 1 subscribers
22:36:27.917283 [SERVER] Receive status of message '111' received
22:36:27.918516 [BROKER] Publishing message '222'
22:36:27.918542 [SERVER] Sent message 222 from publisher
22:36:27.918551 [SERVER] Waiting for receive status of message '222'
22:36:37.919121 [BROKER] Sending 222 to 1 subscribers
22:36:37.923407 [SERVER] Receive status of message '222' received
█
```

Full buffer

The publisher sends 4 messages one after the other but since the broker has some delay when dequeuing buffer contents, the fourth message cannot be sent and has to wait until buffer has at least one space free.

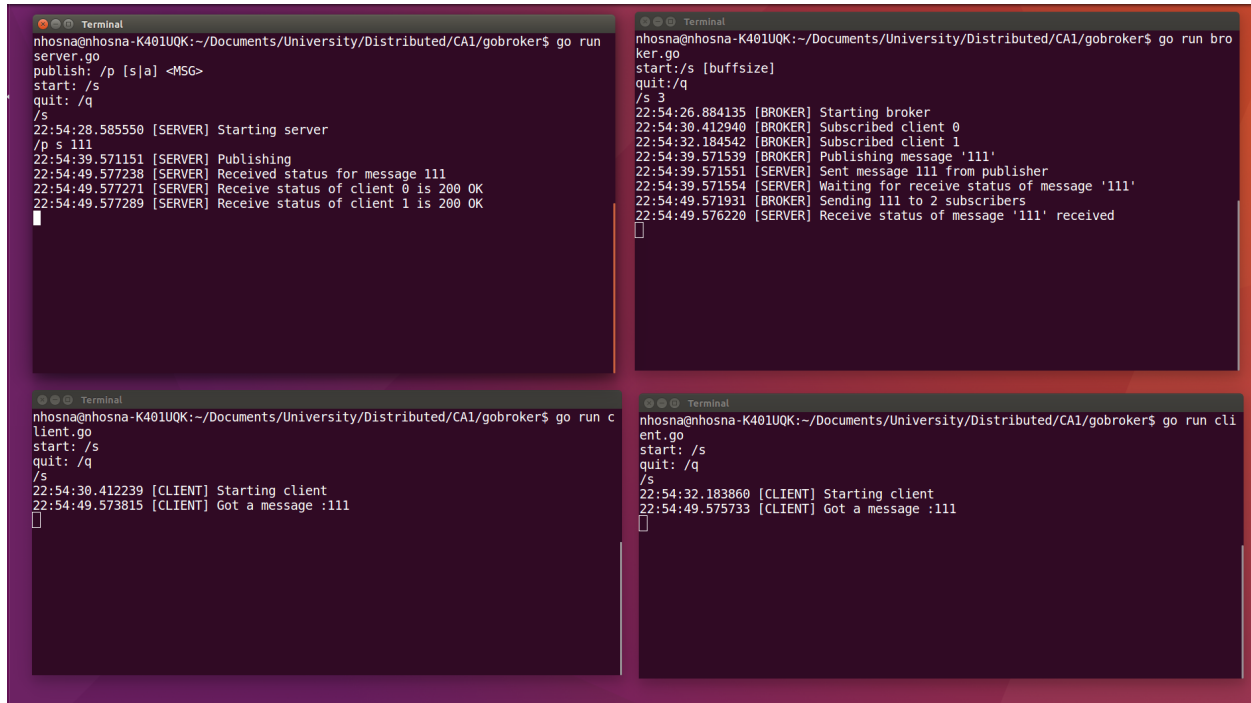
```
Terminal
nhosna@nhosna-K40IUQK:~/Documents/University/Distributed/CA1/gobroker$ go run
server.go
publish: /p [s[a] <MSG>
start: /s
quit: /q
/s
22:32:56.867719 [SERVER] Starting server
/p a 111
22:33:00.394853 [SERVER] Publishing
/p a 222
22:33:04.525115 [SERVER] Publishing
/p a 333
22:33:06.942914 [SERVER] Publishing
/p a 444
22:33:09.435433 [SERVER] Publishing
22:33:10.400486 [SERVER] Received status for message 111
22:33:10.400534 [SERVER] Receive status of client 0 is 200 OK
22:33:20.402211 [SERVER] Received status for message 222
22:33:20.402251 [SERVER] Receive status of client 0 is 200 OK
22:33:30.404540 [SERVER] Received status for message 333
22:33:30.404574 [SERVER] Receive status of client 0 is 200 OK
22:33:40.406745 [SERVER] Received status for message 444
█

Terminal
nhosna@nhosna-K40IUQK:~/Documents/University/Distributed/CA1/gobroker$ go run
client.go
start: /s
quit: /q
/s
22:32:54.930423 [CLIENT] Starting client
22:33:10.398998 [CLIENT] Got a message :111
22:33:20.401464 [CLIENT] Got a message :222
22:33:30.403868 [CLIENT] Got a message :333
22:33:40.406108 [CLIENT] Got a message :444
█
```

```
Terminal
nhosna@nhosna-K40IUQK:~/Documents/University/Distributed/CA1/gobroker$ go run br
oker.go
start:/s [buffsize]
quit:/q
/s 2
22:32:52.910071 [BROKER] Starting broker
22:32:54.931070 [BROKER] Subscribed client 0
22:33:00.395243 [BROKER] Publishing message '111'
22:33:00.395263 [SERVER] Sent message 111 from publisher
22:33:00.395266 [SERVER] Waiting for receive status of message '111'
22:33:04.525528 [BROKER] Publishing message '222'
22:33:04.525556 [SERVER] Sent message 222 from publisher
22:33:04.525569 [SERVER] Waiting for receive status of message '222'
22:33:06.943364 [BROKER] Publishing message '333'
22:33:06.943396 [SERVER] Sent message 333 from publisher
22:33:06.943417 [SERVER] Waiting for receive status of message '333'
22:33:09.436013 [BROKER] Publishing message '444'
22:33:09.436042 [SERVER] Broker buffer is full
22:33:10.396063 [BROKER] Sending 111 to 1 subscribers
22:33:10.396194 [SERVER] Broker buffer not full anymore
22:33:10.396239 [SERVER] Broker buffer is full
22:33:10.399609 [SERVER] Receive status of message '111' received
22:33:20.399870 [BROKER] Sending 222 to 1 subscribers
22:33:20.399919 [SERVER] Broker buffer not full anymore
22:33:20.400048 [SERVER] Sent message 444 from publisher
22:33:20.400071 [SERVER] Waiting for receive status of message '444'
22:33:20.401854 [SERVER] Receive status of message '222' received
22:33:30.402561 [BROKER] Sending 333 to 1 subscribers
22:33:30.404231 [SERVER] Receive status of message '333' received
22:33:40.404070 [BROKER] Sending 444 to 1 subscribers
22:33:40.406475 [SERVER] Receive status of message '444' received
█
```

Multiple Clients

The protocol works for more than 1 client as well. All clients will receive the message server has published.



```
nhosna@nhosna-K401UQK:~/Documents/University/Distributed/CA1/gobroker$ go run server.go
publish: /p [s|a] <MSG>
start: /s
quit: /q
/s
22:54:28.585550 [SERVER] Starting server
/p s 111
22:54:39.571151 [SERVER] Publishing
22:54:49.577238 [SERVER] Received status for message 111
22:54:49.577271 [SERVER] Receive status of client 0 is 200 OK
22:54:49.577289 [SERVER] Receive status of client 1 is 200 OK

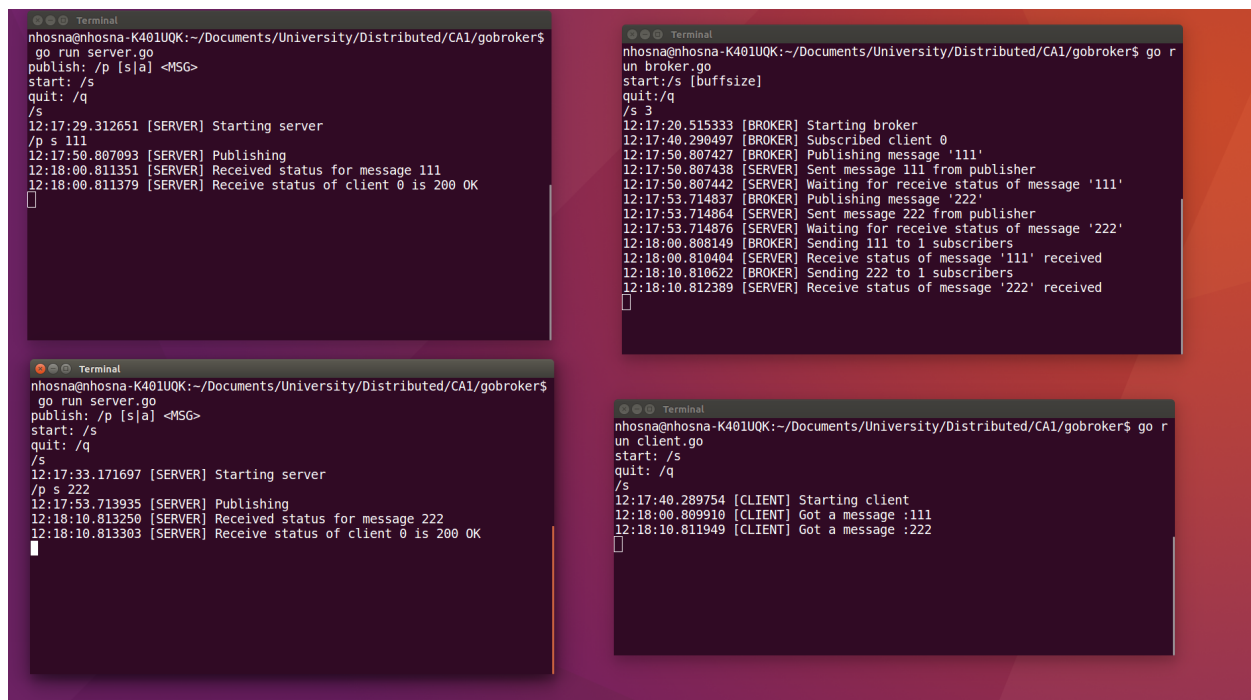
nhosna@nhosna-K401UQK:~/Documents/University/Distributed/CA1/gobroker$ go run broker.go
start:/s [buffsize]
quit:/q
/s 3
22:54:26.884135 [BROKER] Starting broker
22:54:30.412940 [BROKER] Subscribed client 0
22:54:32.184542 [BROKER] Subscribed client 1
22:54:39.571539 [BROKER] Publishing message '111'
22:54:39.571551 [SERVER] Sent message 111 from publisher
22:54:39.571554 [SERVER] Waiting for receive status of message '111'
22:54:49.571931 [BROKER] Sending 111 to 2 subscribers
22:54:49.576220 [SERVER] Receive status of message '111' received

nhosna@nhosna-K401UQK:~/Documents/University/Distributed/CA1/gobroker$ go run client.go
start: /s
quit: /q
/s
22:54:30.412239 [CLIENT] Starting client
22:54:49.573815 [CLIENT] Got a message :111

nhosna@nhosna-K401UQK:~/Documents/University/Distributed/CA1/gobroker$ go run client.go
start: /s
quit: /q
/s
22:54:32.183860 [CLIENT] Starting client
22:54:49.575733 [CLIENT] Got a message :111
```

Multiple Servers

More than one server can use this protocol to publish messages to all clients.



```
nhosna@nhosna-K401UQK:~/Documents/University/Distributed/CA1/gobroker$ go run server.go
publish: /p [s|a] <MSG>
start: /s
quit: /q
/s
12:17:29.312651 [SERVER] Starting server
/p s 111
12:17:50.807093 [SERVER] Publishing
12:18:00.811351 [SERVER] Received status for message 111
12:18:00.811379 [SERVER] Receive status of client 0 is 200 OK

nhosna@nhosna-K401UQK:~/Documents/University/Distributed/CA1/gobroker$ go run server.go
publish: /p [s|a] <MSG>
start: /s
quit: /q
/s
12:17:33.171697 [SERVER] Starting server
/p s 222
12:17:53.713935 [SERVER] Publishing
12:18:10.813250 [SERVER] Received status for message 222
12:18:10.813303 [SERVER] Receive status of client 0 is 200 OK

nhosna@nhosna-K401UQK:~/Documents/University/Distributed/CA1/gobroker$ go run broker.go
start:/s [buffsize]
quit:/q
/s 3
12:17:20.515333 [BROKER] Starting broker
12:17:40.298497 [BROKER] Subscribed client 0
12:17:50.807427 [BROKER] Publishing message '111'
12:17:50.807438 [SERVER] Sent message 111 from publisher
12:17:50.807442 [SERVER] Waiting for receive status of message '111'
12:17:53.714837 [BROKER] Publishing message '222'
12:17:53.714864 [SERVER] Sent message 222 from publisher
12:17:53.714876 [SERVER] Waiting for receive status of message '222'
12:18:00.808149 [BROKER] Sending 111 to 1 subscribers
12:18:00.810404 [SERVER] Receive status of message '111' received
12:18:10.810622 [BROKER] Sending 222 to 1 subscribers
12:18:10.812389 [SERVER] Receive status of message '222' received

nhosna@nhosna-K401UQK:~/Documents/University/Distributed/CA1/gobroker$ go run client.go
start: /s
quit: /q
/s
12:17:40.289754 [CLIENT] Starting client
12:18:00.809910 [CLIENT] Got a message :111
12:18:10.811949 [CLIENT] Got a message :222
```