Advanced Systems Lab (Fall'15) – First Milestone

Name: *Your name*

Legi number: *Your legi number*

Grading

Section	Points
1.1	
1.2	
1.3	
2.1	
2.2	
2.3	
3.1	
3.2	
3.3	
3.4	
3.5	
3.6	
Total	

1 System Description

1.1 Database

1.1.1 Schema and Indexes

The database schema was designed accordingly to the systems requirements. In the description of the systems three main components can be clearly identified, the users, queues and messages. Taking those elements as base components of the system, the application works with three main tables in the messaging database, each one corresponds to the main components of the system $SeeFigure1$.
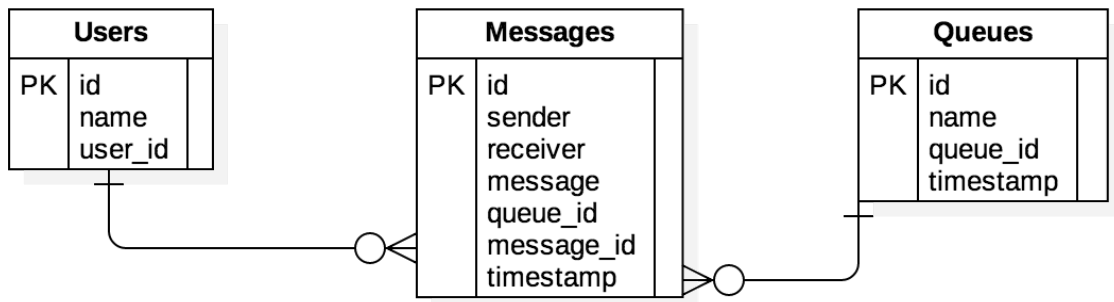

Figure 1: Entity-Relation-Diagram of the database.

The table user contains three columns. The id column is the auto increment value assign from the database for each new row in the table. The column name contains the name of the users, this value is marked as unique, because each user must, have and a unique user name. Finally the userid column, is a second id that binds a user, this one is assigned by the middleware each time a new user connects to the system.

The table queue contains four columns. As in the previous table it has the id auto increment column. The column queuename, has the name of the queues in the database also marked as unique. The column queueid, also shares an id generated by the middleware, this value is assigned by the middleware when a users requests to create a new queue. Finally the column timestamp as the name implies, is the timestamp of queue when this one is created.

The table messages, is the biggest but not necessarily complex table. Each column corresponds with each component of the message object. In the requirements is specified that each message must have a sender, receiver the queue it belongs to, timestamp, the message content and the message id. Therefore, the table has a column for each of these parameters.

The indexes of the database are the following:
In the table users, there is a single index of the users, because that is the attribute use in the stores procedure to a find a particular user.
In the table queues, as in the previous table a single index is also to optimize the search over the id of the object queue.
Finally in the table messages, three indexes were created, one to search over the message id to find a particular message, an index to search over the receiver of a message, when a user request a message entitled to him a search over this value must be done. Lastly and index to search

over the queid of which that message belong, an example of the need of this index is when the user request to delete a queue and all the messages stored in that queue must be found in order to eliminate them.

1.1.2 Stored Procedures

The database messaging contains a total of 14 stored procedures, some created as testing tool and other as part of the functionality of the messaging system. A list of each store procedure that is used as part of the messaging system, and a corresponding description of its function is presented next.
delete_messages_from_queue(queue_id): This functions is executed when user requested to delete a queue. Therefore each message that belongs to that queue is eliminated. This function expects the queue_id value. The value is obtained by the get_queue_id(queue_name) function.

delete_queue(queue_name): This function deletes a queue. However this cannot be done if there exist a message in the messages table that has on its queueid column the value of the id of the queue that is about to be deleted. This protection function is enabled an initial relation established on the database.

delete_user(user_name): This function is on charge of deleting a user from the table users. Nevertheless, the database also has a protection mechanism regarding the messages on the database and the user id of the user that is about to be deleted. This constrained, stops the execution if there exists a message for this user.

get_message(receiver_name): This store functions retrieves a message for the corresponding receiver. If the message doesnt exists it returns a empty message, with empty values on each column.

get_message_from(receiver_n,sender_n): This functions gets a message from the database where the value of the sender and the receiver matches the ones passes as arguments to the function. If the message doesnt exists it returns a empty message, with empty values on each column.

get_queue_id(queue_name): This function returns the id value of the queue specified in the name parameter. If the queue doesnt exists it returns empty. get_user_id(user_name): This functions returns the user id of the user been requested. If the user doesnt exist it returns empty.

insert_new_queue(name,id,timestamp): This function is executed when a user requests the creation of a new queue. The queue name, the id created by the middleware and the timestamp of creation is provided as parameter to the functions.

insert_new_user(name,id): This functions is executed, when a new users connects to the middleware, in order to add this new user to the database. The name and the id middleware id are provided in the function.

insert_new_message(message,sender,receiver,id,timestamp,queue_id): Tis is the function executed when a users send a new message to the database. The queue id is obtain using the get_queue_id(queue_name) function and the return value depends on the name of the queue where the user want to allocate the message. The rest of the parameters are the default values of a normal new message.

1.1.3 Design decisions

The main goal during the design of the database was to comply with the system requirements of the system description. Furthermore once the requirements were fulfilled, I focused on a small schema in order to keep the size of the database small.
For the improvement of the operations on the database, the postgres functions (Stored procedure) were implemented. Postgres functions written in PL/pgSQL execute the queries like prepared statements; they reused cache query plans, this make and improvement in the execution time, since they decrease the planning overhead for the execution.[1]

In addition of the postgres functions also indexes were created in the database. Indexes improve the performance of the database specially when small amounts of data are expected to be retrieved. Since indexes on postgres are partially or tattly cached accessing data form a cached is always faster then form disk. This improves overall the performance of the database.

1.1.4 Performance characteristics

Even before setting my own experiments to the messaging system. I decided to run some benchmarking tests to the database system. In order to have a first impression about its performance.

1.2 Middleware

1.2.1 Design overview

Overall the Middleware system has three main components, the Middleware server, the Client Handler and The Database Connector Server $See Figure 2$. The Middleware server is main thread on the system. Its always listening to new connections from new clients. Before the running of the main listening connections socket the Middleware server sets up the connection with the database.
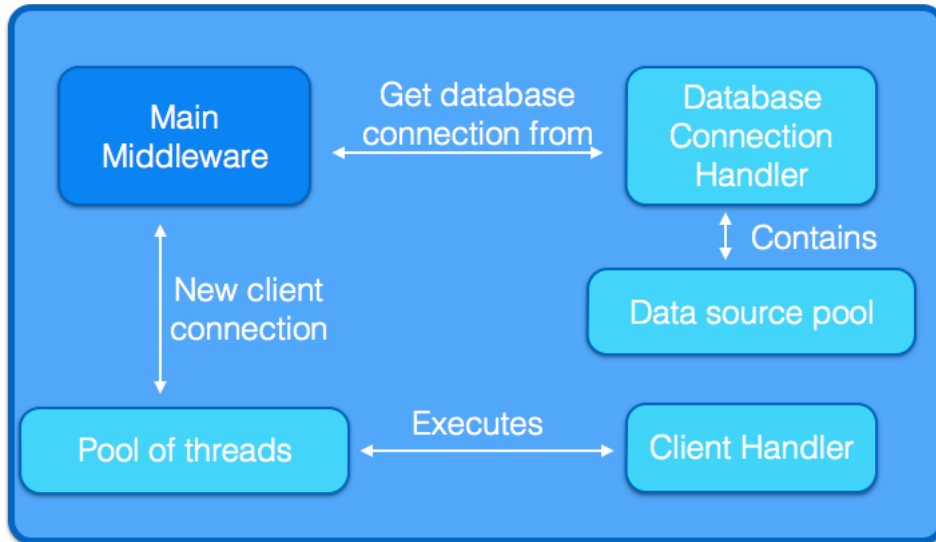

Figure 2: Main middleware component schema.

In order to establish this connection the Middleware needs the following parameter: the address of the database server, the port where the database is listening, the user to connect to the database, the password, the databases name and finally the number of connection to

Operation #	Desciption
99	Initialize connection with client.
0	Read message from the queue.
1	Read message intended for the user.
2	Read message sent by a user.
3	Create a new message.
4	Create a new queue.
5	Send a message to a particular receiver.
6	Delete queue.
7	End connection

handle. The latest parameter defines the behavior of the queuing of new incoming connections and the pool in the connection to the database; this behavior is explained more in detail in the Queuing and connection pool database section.

Whenever the middleware obtains a new connection from a new client, it pairs this connection with a socket and a connection to the database. The Database connector server gives the connection for the incoming user, after getting the connection a new thread from the Executor service in the Middleware is launched. The new thread takes the socket of the incoming connection and the connection to the database. The initial number of connections to handle limits the number of simultaneous threads in the Middleware.

The behavior of the executor service is explained more in details in further sections. The Client Handler is the component that is instantiated by the new thread in the executor service. Each thread of a Client Handler is in charge of a single user, therefore the system has a total number of handlers equal to the numbers of users been handle by the Middleware.

The Database Connection server: Since the Middleware get the parameters to enable the connection the database. It passes those parameters to the Database connection server in order to instantiate a new set of connections in a pool for the database.

The Client Handler: This component of the middleware has all the logic to fulfill every request a client can do to the middleware. It uses the socket given by the Middleware to interact with the client and the database connection from the pool to interact with the database server. As result each client can successfully send messages and these will be stored in the database.

1.2.2 Interfacing with clients

The interface between users the Middleware is the Client Handler. Nevertheless, the Client Handler uses a special object that helps during the interaction with the client. This is the Protocol component; it defines a message object, a queue object, a user object and a protocol number. The value on the protocol number defines the type of operation been requested by the user. The values and their corresponding protocol number are the following:

1.2.3 Queuing and Connection pool to database

In the Middleware there are three queuing components: The Pool of executor in charge of executing a Client Handler per connection, the connection pool in charge of managing the connection to the database, and the queuing message system it self $SeeFigure3$.
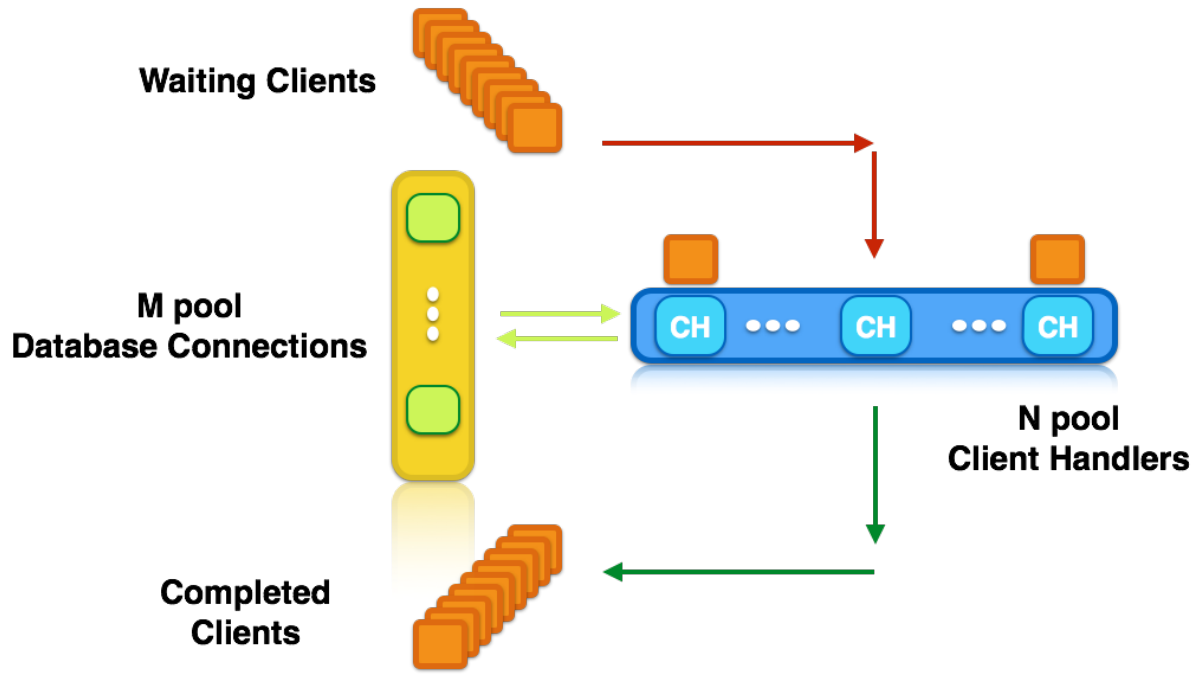
Figure 3: Behavior of the system, M and N is the number of connection in each pool, the may be equal or differetn. The number of connections in the pool may affect the througput in the system.

The Executor service acts as a blocking queue. In this case this is bounden queue since it has a maximum number of thread in this case users that it can handle. As part of the design of the system I wanted to use this technique in order to prevent resource starvation. When a system uses of large queues and a relative small pools then the usage of the CPU and the OS resources, and the overhead due to context-switching decreases. Nevertheless this may also affect the throughput of the system.

The connection pool in charge of the connections has a fixed number of connections ready for the clients. Each connection is assigned to a Client Handler to interact with the request coming from the client. When a Client Handler is done with a client and it calls the closing method for the connection, this one never actually closes, instead it returns to the pool of connection in order to be used by another Client Handler. This improves the performance of the system, since there is no need to setup a new connection for new Client in the system.

1.2.4 Performance characteristics

1.3 Clients

1.3.1 Design and interface

The design of the clients is simple. They accept different running modes, those are defined by the number of parameter they get during the running. The main parameters for running are the following: server address of the middleware, listening port of the middleware, and the client name. Once they start running they could accept the can design which operation to perform by console and fulfill the entire fields in the same manner. Nevertheless for experimenting purposes the clients are initialize with more parameter to avoid the interaction with a real user and automatize the operations that it performs. The extra parameter are the following: a running time that defines how long the clients are going to run, the workload under it will send a new request to the middleware, the operation number that is going to perform, this could be any of

the one defined in section 1.2.2.

Furthermore inside each client there is embedded two message types one with a length of 200 and 2000 characters, this as defined in the system description. Additionally, they count with a list of pre-set user name and queue names from where they can choose randomly during the execution of certain client operations. (e.x. Sending a new message) $See Figure 4$.
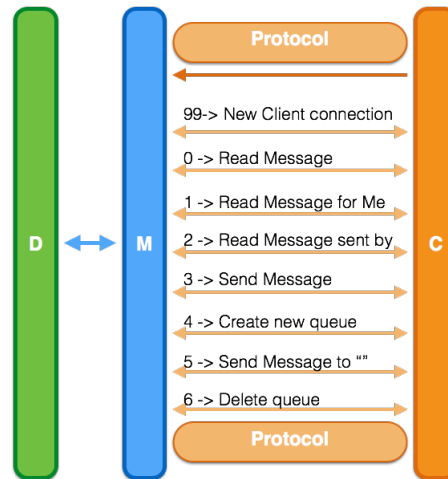

Figure 4: Client, middleware and database interaction procesdure. This diagram show the operations that a client can requesta and the behavior between thr C:Client, M:Middleware and D:Database.

1.3.2 Instrumentation

1.3.3 Workloads and deployment

1.3.4 Sanity checks

2 Experimental Setup

Length: 1-2 pages

Explain the overall design of the complete system and list the configurations (number of middlewares, number of clients, types of machines, communication patterns) corresponding to the main workloads.

Describe the mechanisms for deploying the system for experiments and the way performance numbers are gathered and processed. Make the description so that someone unfamiliar with your system can replicate the steps, and reference the different script files you submit as code in the SVN repository.

2.1 System Configurations

2.2 Configuration and Deployment mechanisms

2.3 Logging and Benchmarking mechanisms

3 Evaluation

Length: up to 10 pages

In this section we expect to see the different experiments you ran to exercise the system, and with each experiment we expect a clear description of the system configuration used, the

hypothesis on behavior and the explanation of the behavior observed (in terms of the different design decisions taken beforehand) – *missing either of these for an experiment might make you lose all points for that given experiment!* Keep in mind that for a good explanation of the results of an experiment you might have to use one or more methods of data analysis presented in the lecture and in the book.

See below for a short description on what each part should contain.

3.1   System Stability

To prove that your system functions correctly and that it is stable include the trace of a 30 minute run, plotting both response time and throughput. Use at least 30 clients (sending and receiving data), 2 middlewares and a non-empty database.

3.2   System Throughput

Measure the maximum throughput of the system (describe the exact configuration and workload, and the reasoning behind choosing these particular ones) and show the average response time for this experiment.

3.3   System Scalability

Explain the different configurations used to explore the scalability of your system, and the outcomes of these experiments in terms of throughput and response times. The main goal of this subsection is to define the ranges in which your system operates best.

3.4   Response Time Variations

Report and analyze how the response times change in the system with different message sizes, different number of clients and different number of middleware nodes.

3.5 2^k Experiment

Conduct a 2^k analysis of your system (aim at exploring non-obvious interactions of parameters). Use the methods learned in this lecture to conduct the detailed analysis.

3.6   Conclusion

To conclude the report summarize the behavior of the system in terms of the design and the representative workloads. Finally, outline in a few points what would you do differently if you could design the system anew.

References

[1] POstgres. Postgresql sql-prepared statements.