What is D1 Queue Manager?

- Managers message and queries for servers and clients
- Communicates with servers and clients over TCP/IP socket networking
- Prioritizes queries and queues them for the databases
What does D1QM do?

- Works as a client to server databases and stagers
- Works as a server to client web query interface
- Accepts, validates, and tracks queries submitted by clients
- Monitors status of query processing
- Processes messages from servers and clients
- Monitors server connection
- Notify clients of query results
- Reconnect to server(s) if the server(s) disconnected
Requirements

1. Input Parameters
   - Get query parameters expressed as a query string
   - Get query ID expressed as a string
   - Check the query parameters for validity
   - Assign a priority number to each valid query

2. Logging Capability
   - Shall log the query to a log file
   - Shall log the following information
     - Error message
     - Server message
     - Client message,
     - Time stamps
Requirements (cont‘d)

3. Message management
   • Read and recognize valid query status message
   • Send a message back to the client to indicate acceptance or rejection
   • Notify the user if an unprocessable query is received
   • Notify users when a query is successfully processed
   • Notify users of timeout when connecting to servers

4. Sending query to servers
   • Send a query to servers in an order of priority
   • Record time a query is sent to each server
Requirements (cont’d)

5. Tracking query process
   • Track each step of the process
   • Record the time a query submitted, validated, and finished
   • Record query results

6. System communication
   • Communicate with servers and clients through TCP/IP socket connection

7. Server reconnection
   • Reconnect to a server after a server disconnection
TO DO

- Implement incomplete message handling
- Expand query priority classification
- Expand more detail query validation
- Implement program shut down
Class Hierarchy

Socket implements a raw socket TCP/IP calls
  ServerSocket implements a server socket connection and communication
  ClientSocket implements a client socket connection and communication
MQMSocket implements multi-servers and multi-clients socket connection and communication
Query defines query information
QueryMap stores query information described by class Query (map type container)
Queue defines query ID and query priority number
PriorityQueue maintains query in a critical order according to its priority number
Selector selects a socket descriptor that is available to be read
ServerData defines server information
ServerMap stores server information, a map type container
SocketException throws an exception
QueryCheck validates a query
Program flow

1. Initialize server socket connect for databases and stagers
2. Accept the new client
3. Initialize server socket
4. Connect for databases and stagers
5. Set up listen socket and open the listener port
6. Check query/message on the listener port
7. Is new connection?
   - yes: Accept the new client
   - no: Is server message?
     - yes: Is a valid query?
       - yes: Assign a priority number
       - no: Notify client: the query is invalid
     - no: Notify client: server is not available
   - no: Is a query?
     - yes: Is server connected?
       - yes: Assign a priority number
       - no: Notify client: server is not available
     - no: Notify client: the query is invalid
   - Update query status

8. Store query ID and query priority in priority queue
9. Store query in query container
Check out top priority query from priority queue

Prepare the query for databases

Check query container prepare message for servers and clients

Is server connected? yes
Send message to server

Is client connected? no
Disconnect to server, clean message to be sent to server

Disconnect to client, clean message to be sent to the client

Is client connected? no
Notify clients of system is down

Send message to clients yes

Reconnect to server if the server is down

Check query/message on the listener port
Configuration files

• Configuration file for each server:
  D1Conf.txt:
  45278
  fafnir.gsfc.nasa.gov

• Configuration file for listener socket
  MainConf.txt:
  4725
Define main variables

1. int listener, sd, fd;                           // socket descriptor
2. int D1sd = -1, D2sd = -1, Stg1sd = -1, Stg2sd = -1; // server socket descriptor
3. std::string status, LogMsg, ErrorMsg;
4. std::string servernam;
5. list<int> *SDList = new list<int>;            // socket descriptor list
6. list<string> *KeyToBeRM = new list<string>;   // to be removed query id list
7. list<TList> *MsgToServer = new list<TList>;   // message to server list
8. list<TList> *MsgToClient = new list<TList>;   // message to client list
9. TQueue *PQueue = new Tqueue;                 // Priority queue container

*TList: c-structure with fields of message ID, message size, message, and socket descriptor
Set up server and listen sockets (code sample)

3. QMQSocket *QM = new MQMSocket; // Set an object *QM
4. try{
5.     ConfFname = "ListenConfig.txt";
6.     (*QM).SetListener(ConfFname); // create QM server to clients
7.     listener = (*QM).GetListener(); // get the socket descriptor
8. }
9. catch(SocketException& err){
10.    cout << "Exception was caught in creating listener:" << err.description() << endl;
11.    exit(1);
12. }
14. try{ // Set D1 server
15.     ConfFname = "D1Config.txt";
16.     servername = "D1";
17.     if ((*QM).SetServer(ConfFname, servername))
18.         D1sd = (*QM).GetServerSd();
19. }
20. catch(SocketException& err){
21.     (*QM).ReSetServerFlag();
22.     cout << "Exception was caught:" << err.description() << endl;
22. }
Read in message/query (code sample)

1. If (!(*QM).SelectReadFd()) throw SocketException( "Standing by\n");
2. while ((fd =(*QM).GetReadFd())!=-1){  // get the ready socket descriptor
3. try{
5.   if (fd == listener){                  // new connection
6.     sd = (*QM).GetNewClient(); // get the new connection socket descriptor
7.   }
8.   else if (fd == D1sd || fd == D2sd || fd == Stg1sd || fd == Stg2sd){
9.     *InMesg = (*QM).GetServerMesg(fd); // get message from server
10.    ...
11.  }
12. else{                                   // from client
13.   (*InMesg) = (*QM).GetClientMesg(fd); // get message from client
14.    ...
15. }
16. }                                        // end try
17. catch(SocketException& err){
18.   close (fd); // close connection
19. }
20. if (fd == D1sd) D1sd =-1;  // reset descriptor
21. ...
22. }
Prepare query for D1 (code sample)

3. Queue *pq = new Queue;
4. Query *r = new Query;
5. if (!(*PQueue).empty()) {  // if priority queue is not empty
6.   *pq = (*PQueue).top();   // get the top priority query ID
7.   string key = [(*pq).GetQKey();
8.   // Get the corresponding query from query container
9.   *r = (*QueryMap)[key];
10.  // Implement query/mesg to be send to D1
11.   (*OutMesg) = GetOutMesg(3001, (*pq).GetQKey(),(*r).GetQuery(),D1sd);
12.  // store the query/mesg in a message list to be sent to D1
14.   (*PQueue).pop();// remove the top query from priority queue
15. }
Check query container

1. For ( it= (*MapQuery).begin(); it != (*MapQuery).end(); it++ ) {
2.   // check timeout for servers
3.   clock_t Timer = (*it).second.GetTimeToServer(servername);
4.   if( (elapsed_time(Timer,mark_time()) > TimeLimit) && (Timer!= 0.0) ){
5.     string status = "Time_out_for_"+servername;
6.     ...  
7.   }
8.   // check for finished query
9.   if(atoi((S1CStatus.substr(0,4)).c_str()) == WEB_QUERY_RESULTS_RECEIVED &&
    atoi((S1SStatus.substr(0,4)).c_str()) == MERGE_FINISHED &&
    atoi((S2SStatus.substr(0,5)).c_str()) == D2_MERGE_FINISHED &&
    atoi((S2CStatus.substr(0,4)).c_str()) ==
    WEB_D2_QUERY_RESULTS_RECEIVED)
10.    (*KeyToBeRM).insert((*KeyToBeRM).end(),(*it).first);
11.    ..
12.}  
13.  ...
Send message to server (code sample)

1. // Send mesg to server
2. while (!(*MesgToServer).empty()) { // check message to be sent to servers
3.     try{
4.         *OutMesg = (*MesgToServer).back(); // get message from message list
5.         (*QM).SendServerMesg(OutMesg); // send the message to server
6.         (*MesgToServer).pop_back(); // remove the message from the list
7.         ...
8.     }
9.     catch (SocketException& err) {
10.        close((OutMesg).Tsd); // close the connection to the server
11.        ...
12.     }
13. }
Server reconnection (codes sample)

2. if ((*QM).Is_SysDown() == 1 ){ // check if there is system crashes
3.     try{
4.         double ServerTimer = TConnect; // time to reconnect server
5.         (*QM).ReconnectServer(ServerTimer); // reconnect to server
6.     // get the reconnect server name, and set the socket descriptor
7.         if ((*QM).GetServerName() == "D1") D1sd == (*QM).GetServerSd();
8.         ...
9.     } 
10.    catch(SocketException & err){
11.        LogMesg = "Could not reconnect to "+(*QM).GetServerName();
12.        .... // logging
13.    } // logging
14. }