## **Design Document**

The Server is highly configurable and scalable system capable of handling multiple clients sending continuous data.

**Configurability:** Various system parameters can be configured to suit the load. More details under system design

**Scalability:** The Server is highly scalable with no limit on number of clients and number of messages per client

**Server:**

**System design:**

1. Server is a multi-threaded system with main thread accepting connections from clients and a pool of threads processing client messages.
2. Main thread accepts connections from clients and enqueues them into a queue(clientQ).
3. Worker threads dequeues clients from clientQ and processes messages from the client.
4. The Server system is configurable to suit the client load. A set of system parameters can be provided during start of the system.

**System parameters:**

*num\_of\_threads*: Number of worker threads

*client\_timeout:* Timeout before disconnecting a client if no data is received

*max\_continuous\_msgs:* Number of continuous msgs from a client. The client is put back in queue and a different client is picked from queue.

**Default system parameters:**

If the server is launched without any command line arguments, below default values are used.

*num\_of\_threads*: 5

*client\_timeout: 180s*

*max\_continuous\_msgs:100*

*file\_name:* "file"

*port:* 55555

**Security aspects:**

Any client can establish connection. No authentication done.

To avoid a client hogging the Server: Server disconnects the client if no data is received from client within timeout: *client\_timeout* (configurable).

To give fair chance to all connected clients, a client is put back in queue after processing certain number of messages: *max\_continuous\_msgs*(configurable).

**Data structures:**

*ClientQ:* A queue of Client nodes. Each node contains connection details of a client.

*Message*: Message exchanged between Client and Server.

**Main thread:**

1. Creates pool of worker threads.
2. Creates clientQ to store client socket fds.
3. Opens a tcp socket to accept connections.
4. Waits indefinitely for incoming connections from clients.
5. When a client request is received, accepts the request and enqueues the connection details into *clientQ*.
6. Signals threads after enqueuing a client into *clientsQ.*

**Worker Threads:**

1. Thread dequeues a client from *clientQ* and starts receiving messages from the Client.
2. Writes messages received from client into server file
3. Closes the connection if there is no data from client within *client\_timeout*.
4. Puts back client into clientQ after processing *max\_continuous\_msgs* of messages from the client

**Synchronization:**

1. Access to *clientQ* is protected by a *qMutex*.
2. Writing to file is protected by a *fMutex*.
3. Conditinal variable:*cond* to signal worker threads when a client is enqueued. Avoids busy wait by worker threads.

**Client:**

1. Sets up tcp socket connection with server.
2. Reads input from user.
3. Forms message with msg\_id and msg sends it to server.

**Further improvements:**

1. System counters and stats
2. cli support to configure the Server dynamically and extract stats.
3. Multi-level debug prints