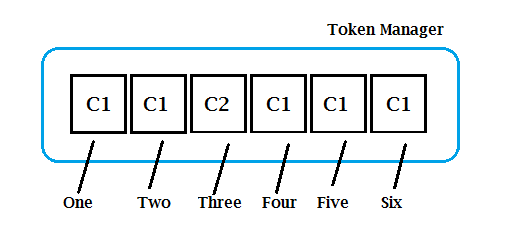
**Discussions and future work:**

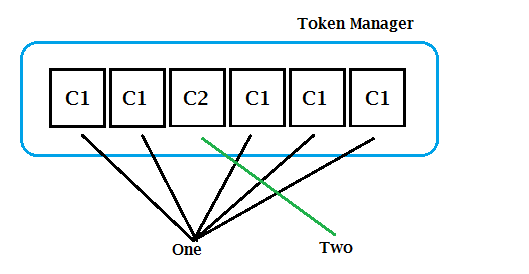
The central server token based algorithm could be improved by implementing few modifications in implementation level, which would in turn improve the speed at which the tokens are processed and served to the clients.

The current configuration of token access by the clients are given in the Figure 7.1 below. Here, the server process each request separately and grants access to them based on the order of the request received. The advantage of this implementation is that, the properties of mutual exclusion are persevered (safety, liveliness and ordering) but the drawback is that it can take a longer time to process each request and the lag becomes noticeable when there is a heavy load. In the Figure 7.1, the terms one, two, three etc. are the order in which the server processes each token request and grants permission to modification of word.



**Figure 7.1: Current implementation**

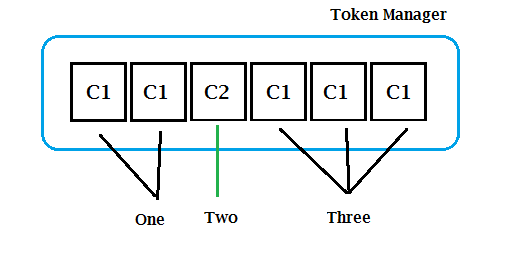
This basic algorithm could be modified in two ways to overcome this drawback. The first method is by grouping all the requests received by a client and then granting access to it, letting the client to make all the changes it requires, before moving on to next client in queue in the token manager. If C1 and C2 are two clients, and the order in which the requests come to token manager are: “***C1, C1, C2, C1, C1, C1***”, then the server shall group all C1’s requests together and grant it access first, before letting C2 make any change as described in the Figure 7.2.



**Figure 7.2: Method One**

This will significantly reduce the load on the server by cutting the number of requests to be processed to dramatically low count, hence increasing the updating speed. While this may be a good option to increase the speed, the drawback is, this does not comply with the basic property of mutual exclusion, which is ordering. It is very essential to preserve ordering of the requests in the queue to make sure every client gets a fair share of processing from the server.

To overcome this drawback, a slight modification can be done in the grouping stage of the messages, thus making the algorithm comply with mutual exclusion properties, while increasing the update speed and reducing the processing load on server. The requests from each client are still processed together as previous algorithm, but only until a new client’s request is found in the queue, as given in the Figure 7.3. If C1 and C2 are two clients, and C1 makes two requests, while third request is of C2, and the rest of the requests are from C1, the server process the first two requests from C1 as a single request, grants it permission to make changes. Once the token is released from C1, the server allocates C2 a token and once it is released, the server then treats the remaining requests from C1 as a single request and grants it permission to make changes as a whole. This considerably reduces the overall processing time, hence making the game faster.



**Figure 7.3: Method 2**

Another area for improvement is the token request-grant method from the central server. In current implementation, as given in Figure 7.4, the central server manages all the token request, release and token grant, which puts significant load on the server and a certain loss of time during heavy load. As an alternative, another optimized version of centralized token algorithm using a central coordinator [1] could be used, where the token requests are sent to the server, which then sends a notification with the details of the node requesting access (N2), to the node which currently has the token (N1). The current node then releases the token to the new requested node (N2) directly, instead of going through the server, hence reducing the time and network usage significantly, as given in Figure 7.5.

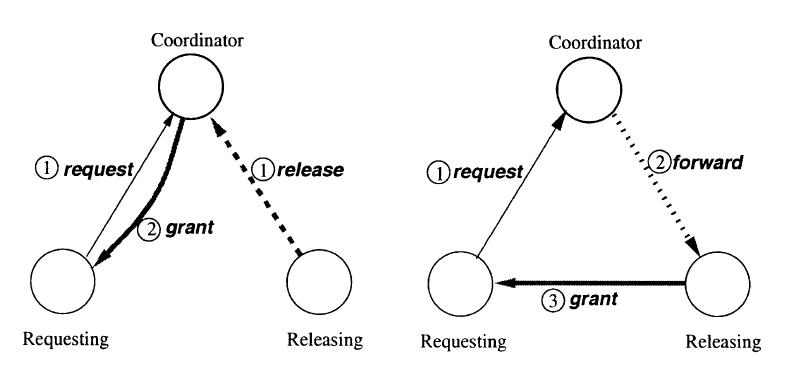


Figure: 7.4: Current Implementation Figure 7.5: Alternative Implementation