A QoS Aware Self Adaptive General Scheme to Solve GME Problem

Jai Singh
Asst. Professor, CS&IT Deptt.
KEC, Ghaziabad, U.P. India

Maitreyee Dutta
Professor, CSE Deptt.
NITTTR, Chandigarh, India

Abhishek Swaroop
Professor, SCSE
Galgotias University Greater Noida(U.P.)

ABSTRACT
The Group mutual exclusion problem (GME) is a resource allocation problem which allows concurrency along with mutual exclusion. The concept of GME can be applied to various fields having varying quality of service (QoS) requirements. The present paper presents a self adaptive general scheme to solve GME problem using token-based approach. The striking feature of the scheme is that it considers QoS requirements, checks the QoS requirements time to time and adapts its parameters if there are deviations from the expected behavior. The dynamic analysis of the scheme has also been presented in the present exposition.

Keywords
Resource, adaptive, mutual exclusion, Quality of Service.

1. INTRODUCTION
The group mutual exclusion (GME) problem proposed by Joung [1] deals with two issues of mutual exclusion and concurrency which are opposite of each other. GME is an attempt to increase the efficiency of resource utilization by allowing concurrent accesses to CS, whenever possible. The concept of GME is being used in a variety of areas [2-7] each of which having different performance requirements and QoS requirements. In GME, the processes requesting the same group may be in CS simultaneously. Hence, the concurrency of a GME protocol can be increased if the priority is given to a group g_1 over other groups, while selecting the group to be selected for the next session, provided that the number of processes requesting for group g_1 are highest among the requesting groups. The concurrency can further be increased, if we allow a process, requesting the group currently being accessed to join the currently open session.

However, whenever we try to increase the concurrency the system has to suffer from reduced fairness and may lead to starvation. The relative weight required to be given to the concurrency and fairness may be varied from application to application and depending upon QoS requirements. The QoS requirements are important for any application since these are recorded as an agreement between customer and system designer. Any violation from QoS may lead to customer dissatisfaction, hence, must be taken seriously. Swaroop-Singh [8] proposed a token-based general scheme to solve the GME problem and named it as GS-GME. In the present exposition, we modified the general scheme proposed by Swaroop-Singh [8] to consider the QoS requirements also while selecting scheme parameters. Moreover, the scheme proposed in our is self adaptive in the sense that whenever it sees violations from QoS requirements it tries to adjusts itself so that QoS violations may be reduced or removed.

The rest of the paper is organized as follows. Section 2 discusses related work, the self adaptive scheme is described in section 3, and the dynamic analysis of the scheme is presented in section 4. Finally, section 5 concludes the paper.

2. RELATED WORK
The general scheme is a broad frame work which can lead to different algorithms depending upon the selection of some parameters. As far as the resource allocation problem is concerned, Sanders [9] a generalized algorithm based upon information structure for the permission-based mutual exclusion algorithms. The generalized algorithm may lead to different known and new algorithms. For token- and tree-based mutual exclusion algorithms, Helary-Mostefaoui-Raynal [10] proposed a generic algorithm based upon a very general information structure. This general structure covers, as particular cases, several existing algorithms and gives space for the design of new algorithms for various topologies. Manabe *et al.* [11] presented a general scheme for a quorum-based *h*-out of-*k* mutual exclusion algorithm that relies on a collection of quorums called *k*-arbiter.

As far as Group Mutual Exclusion is concerned, the first general scheme was proposed by Swaroop-Singh [8]. The scheme [8] uses a token-based approach and the captain-follower approach. The group selection policy and entry policy decides the next group to be used in next session when the current session finishes and there are pending requests in token queue. The token-based approach is selected because all requests are available at the token holder node and priorities to the groups can be assigned depending upon the group selection policy. Whenever, a new request arrives for the group being used in the current session, the entry policy decides whether the requesting process will be allowed to join the current session or not. The two token-based GME algorithms namely Mittal-Mohan algorithm [12] and Swaroop-Singh [8] DRS_GME can be considered as a special case of GS-GME.

However, in GS-GME, the group selection policy and entry policy are selected at the time of start of the application. The user may not be aware about the optimum priority levels for concurrency and fairness in the beginning which will satisfy the QoS requirements for that particular application. This may lead to frequent QoS violations. QoS are important because the violations of QoS belong to the service level agreement (SLA) signed between the user and the service provider. SLA is a legal document and hence, binding on the service provider as well as user. Any serious deviation from SLA may lead to severe consequences, hence SLA breech is not considered good. This motivates us to develop a self adaptive general scheme to solve GME problem in message passing system which considers QoS requirements and adapts itself during execution.

Initialization:
Set Initial priority levels α and β; Set QoS1 criteria, QoS2 criteria, QoS3 criteria

Set Adap_interval and Adapt_factor

P_i request for a group *g*:

Invoke *MEA_requestCS* (the request message includes *g* also)

P_i receives token:
state_GME$_i$= CA;*curr_g=g*

follower=id's of all processes requesting for *g* and whose requests are in *token_queue*

Send ALLOW (*i*,*g*) to all processes in *follower*.

Modify *token_queue* and other data structures (except list of followers) related to MEA corresponding to the processes in *follower* as if they have completed execution of CS

/*--- Hold CS----------*/; /*--- exit from CS----------*/; *state_GME*$_i$= CP

If (*follower=Ø*)

state_GME$_i$= I

If (*token_queue≠Ø*)

If (adapt_interval expired)

If (QoS1 violated) change=change+adapt_fact

If (QoS2 violated) change=change+adapt_fact

If (QoS3 violated) change=change-2*adap_fact

α = α+change; β= β-change;

Reinitiate adaptation interval

Invoke group selection policy; Invoke *MEA_release_token*

P_i receives REQUEST for *g* from P_j:
If (*state_GME*$_i$= CA or CP) Invoke entry policy

If (*answer=y*)

follower=follower ∪ *j;* Send ALLOW (*i*,*g)* to P_j

Else Invoke *MEA_rec_request*

Else Invoke *MEA_rec_request*

$P_i$receives ALLOW (*j*,*g*): *captain*$_i$=*j*; *state_GME*$_i$= F;

--executing in CS--; exit from CS

Send COMPLETE (*i*) to *captain*$_i$; *state_GME*$_i$= I;*captain*$_i$=NULL

P_i receives COMPLETE(*j*):
Remove *j* from *follower*

If (*follower=Ø*)

If (*state_GME*$_i$= CP)

state_GME$_i$= I

If (*token_queue≠Ø*)

If (adapt_interval expired)

If (QoS1 violated) change=change+adapt_fact

If (QoS2 violated) change=change+adapt_fact

If (QoS3 violated) change=change-2*adap_fact

α = α+change; β= β-change; Reinitiate adaptation interval

Invoke group selection policy; Invoke *MEA_release_token*

Fig. 1 Pseudo code for self adaptive scheme

3. THE SELF ADAPTIVE GENERAL SCHEME FOR GME

The pseudo code of the scheme is described in fig.1. However, a brief description of the scheme is presented below.

Similar to GS-GME, the scheme consists of following four elements.

1. Group selection policy: When a session terminates and new session has to be started, the pending requests are analyzed and the groups are prioritized. The group with the highest priority is selected for the next session. Priority of a group k can be selected using $\mathbf{Priority_k = \beta * (L - Y_k + 1) + \alpha * n_k}$. Here n_k is the number of processes waiting for group k in the queue, L is the length of the queue and Y_k is the position of the first process in the queue requesting group k. α and β are the relative weights assigned to concurrency and fairness respectively.

2. Entry Policy: When a request is received by the captain for the current session by some other process the entry policy decides whether that request will be satisfied or not. A very restrictive policy (fairness and reduced concurrency) may be not to allow any process to enter session once the session has started. On the other hand, a lenient policy may be to allow any process requesting the current session (may lead to starvation).

3. Mutual Exclusion Algorithm: This should be token based algorithm considering a logically fully connected distributed system. The algorithm should ensure that all requests eventually reach the token holder node.

4. Captain-Follower approach: The process receiving the token initiates a session as captain. The captain may allow other process (es) to join the session as follower(s). The followers inform its captain on exiting from CS. The captain may terminate the current session only if all the followers and captain have come out of CS.

In GS-GME [8], the entry policy and relative weights assigned to concurrency and fairness (values of α and β) can be selected at the start of the application and may not be changed during the execution of the application. Moreover, each application has its own characteristics and may have different Quality of service (QoS) requirements.

In proposed scheme, initially the user is asked to provide initial priority levels to be assigned for concurrency (α) and for fairness (β). Additionally, the QoS requirements are also provided by the user. The QoS can be embedded in the proposed scheme and their violation is monitored at regular interval. Based upon the feedback received after each monitoring interval, the priority level assigned to concurrency (α) and for fairness (β) are adjusted with the help of an adaptation factor. The rules for changing the priority level are decided at the start of execution of the algorithm. The adaptation factor is also fixed at the beginning of the application and cannot be changed while application is running. The scheme works similar to the scheme proposed in [8] except its adaptive nature and inclusion of QoS requirements monitoring.

The major advantage of this scheme is that even if one does not know the optimum level for priority levels to be given to fairness and concurrency, he may start with arbitrary priority levels and after some time the proposed scheme will automatically adjust to the optimum priority levels for the given QoS requirements and for the given scenario because of its feedback mechanism.

The QoS requirements may vary from time to time and application to application. In this work we considered following three QoS requirements.

I. Average waiting time is less than 8 milliseconds. (QoS1)

II. At least 50% processes have less than 8 milliseconds waiting time. (QoS2)

III. At least 40% processes must head is selected as captain. (QoS3)

QoS1 and QoS2 may be better satisfied if concurrency is given higher priority whereas QoS3 may be better satisfied if fairness is given higher priority. Initially priority levels may not be selected properly since user is not aware about the behavior of the application. Hence, the feedback system is used in our scheme and after certain interval (adaptation interval) it is checked whether any QoS is (are) violated. Based upon this feedback priority levels are adjusted by using certain adaptation rules.

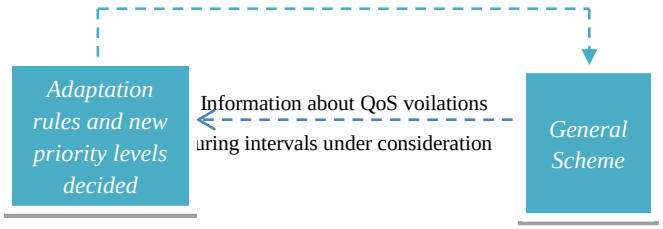

Fig. 2 Adaptation rules and general scheme

Fig. 2 shows how the information about QoS violation is collected at a fixed interval and the adaptation rules use this information to set the relative weights assigned to concurrency and fairness for the next time interval under consideration.

The adaptation rules considered in this work have been given below

Let $\alpha 1$ and $\beta 1$ be the old values of priority levels for concurrency and fairness respectively.

α = old value of priority levels for concurrency.

β = old value of priority levels for fairness.

Change = 0

If QoS1 is violated change = change + adapt_fact.

If QoS2is violated change = change + adapt_fact.

If QoS3 is violated change = change – 2*adapt_fact.

$$\alpha 1 = \alpha + change$$

$$\beta 1 = \beta - change$$

The adaptation rule(s) and adaptation factor may vary from one application to another application. However, for the purpose of dynamic analysis of the proposed scheme, the adaptation rules discussed above have been considered.

4. SIMULATION RESULTS

The scheme proposed in previous section has been simulated using NS2 which is a popular discrete event network simulator. In order to simulate the proposed scheme, we considered a distributed system consisting of 50 nodes, and the numbers of groups considered are 10. The QoS requirement considered as follows

I. Average waiting time is less than 8 milliseconds. (QoS1)

II. At least 50% processes have less than 8 milliseconds waiting time. (QoS2)

III. At least 40% processes must head is selected as captain. (QoS3)

After every 10 sessions the QoS are measured and adaptation is done that means the adaptation interval is fixed to ten sessions. The adaptation is done according to the adaptation rules specified above. The values of adaptation factor considered during simulation are 0, 4, 8, 12, 16, and 20 respectively. The parameters considered are QoS1 violations, QoS2 violations, QoS3 violations and total violations with respect to change in adaptation factor. The priority levels for concurrency (α) and fairness (β) are considered as follows

Table 1: Relative weights to Fairness and Concurrency

Alpha	Beta
100	**0**
80	**20**
60	**40**
40	**60**
20	**80**
0	**100**

The simulation results of the proposed scheme have been presented in fig. 4(a), fig. 4(b), fig. 4(c), fig. 4(d), fig. 4(e), and fig. 4 (f).

Fig. 4(a). Indicates that results when initial priority levels selected are $\alpha = 100$ and $\beta = 0$. On analyzing the results, it can be observed that the QoS3 violations decrease on increasing the adaptation factor form 0 to 8. This is due to the fact that initial value of priority level for fairness is selected very low. As far as, the total violations are considered, the best result appears when the adaptation factor is 8 which is a middle value (not very high and not very low).

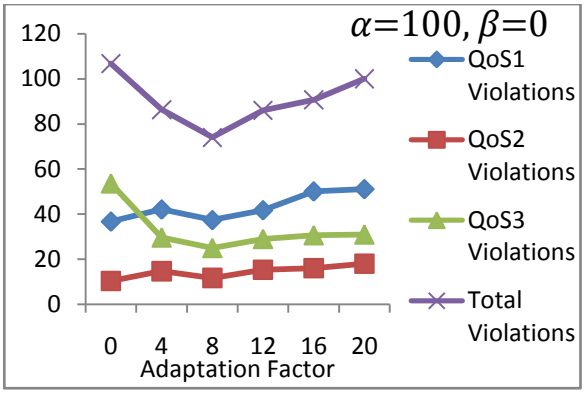

Fig. 4(a)

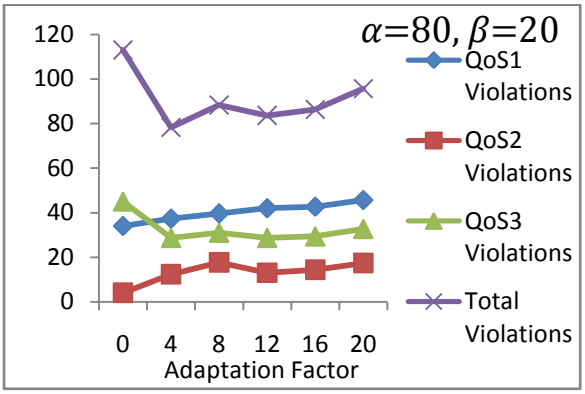

Fig. 4(b)

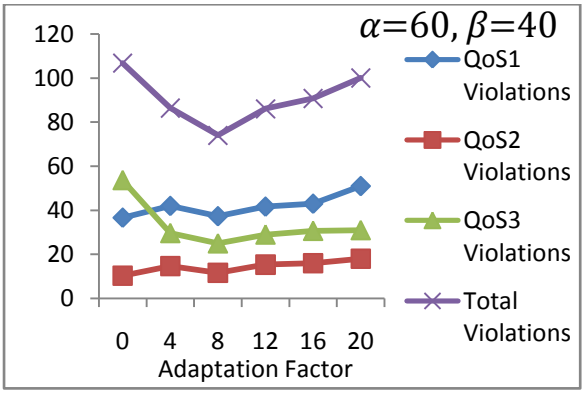

Fig. 4(c)

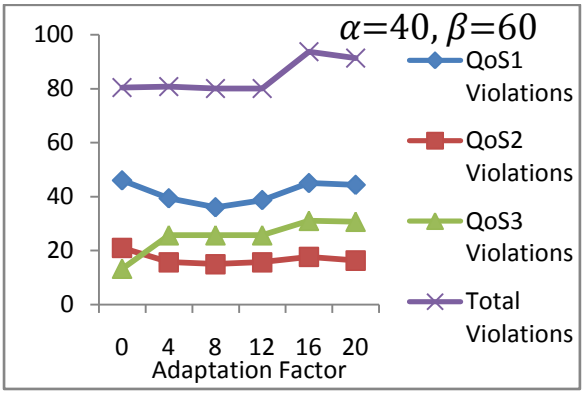

Fig. 4(d)

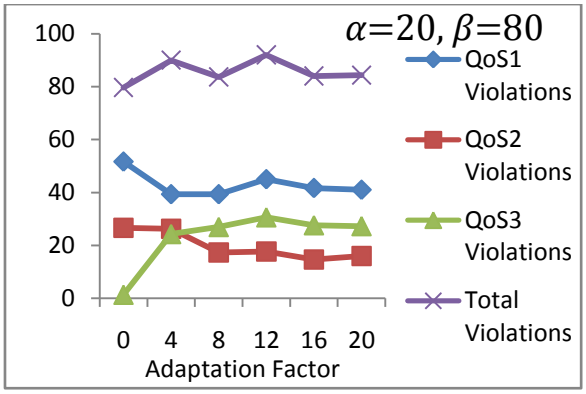

Fig. 4(e)

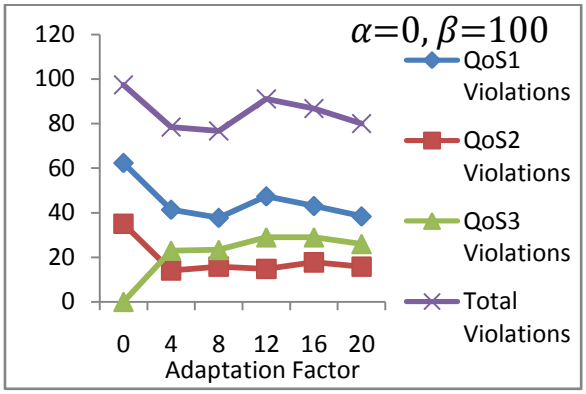

Fig. 4(f)

When $\alpha = 80$ *and* $\beta = 20$, from Fig. 4(b) we found that for adaptation factor 0 to 4 QoS1 violations and QoS2 violations are increasing whereas QoS3violations and total violations are decreasing. The reason for this phenomenon is the fact that initially more weighted is given to concurrency, hence; initially QoS1/QoS2 violations are very low whereas QoS3 violations are frequent. Never the less, when we try to adapt QoS1/QoS2 violations increases from low base and QoS3 violations decreases rapidly. However, for large adaptation factor this initial advantage is gone very soon. In this case, the best results in terms of total violations are achieved with a adaptation factor 4.

When $\alpha = 60$ *and* $\beta = 40$ from fig. 4(c) it is found that QoS3 voilations decreases and QoS1/QoS2 violations increases rapidly when we change adaptation factor from 0 to 4. However, after that QoS3/QoS2/QoS1 violations are stabilized more or less when the adaptation factor is further increased. The optimum level for QoS violations is at adaptation factor 8.

The results When $\alpha = 40$ *and* $\beta = 60$ have been presented in fig. 4(d). In this Fig., as expected, QoS3 violations increases and QoS2/QoS1 violations decreases when we increase the adaptation factor from 0. The reason behind these results is the higher level of priority given to fairness in comparison to concurrency. However, after a certain stage these violations are stabilized. Total violations/QoS1/QoS2 are at lowest level with adaptation factor 8. Hence, adaptation factor 8 may be considered as optimum adaptation factor for this case

The case when= 20 *and* $\beta = 80$, the results have been shown in fig. 4(e). Due to significantly higher level of priority given to fairness, the best results for QoS3 and worst results for QoS1/Qos2 are for the case when no adaptation is done.

However, the optimum levels of results are achieved at adaptation factor 8 as far as total violations and QoS1/QoS2 violations are considered.

The last case for simulation is when $\alpha = 0 \; and \; \beta = 100$ in fig. 4(f) is considered as the priority level for concurrency and fairness. As expected QoS3 shows the least violations and QoS1/QoS2 shows most violations for adaptation factor 0. Additionally, QoS3 violations increases rapidly and QoS2/QoS3 violations decreases rapidly up to a certain increase in adaptation factor and stabilizes thereafter. In this case, 8 turn out to be the optimum adaptation factor as far as total QoS violations are considered.

Therefore from the analysis we can conclude that for optimum QoS requirement satisfaction, the adaptation factor should not be very low and should not be very large.

5. CONCLUSIONS AND FUTURE WORK

Group mutual exclusion problem is an important resource allocation problem in distributed systems In the present exposition, we develop an adaptive QoS aware general scheme which adjusts its group selection policy and entry policy based upon the application goals/QoS requirements provided by the user and feedback received during run time. The dynamic analysis of the proposed scheme has also been conducted for three selected QoS and various adaptation factors. The proposed scheme is fully flexible and self-adaptive. It adapts itself according to QoS requirements and feedback received during run time. However, the adaptation rules in this scheme are fixed in the beginning of the application; these rules do not evolve as the application progresses. Hence, use of evolutionary algorithm to adjust the adaptation rules at run time may be a future extension of this work.

6. REFERENCES

[1] Joung, Y.J., Asynchronous group mutual exclusion. Distributed computing, vol. 13, no. 4, pp. 189-206, 2000.

[2] Park, J.H., Gang, S., Kim, K., Group mutual exclusion based secure distributed protocol. In the proceedings of the computer security symposium 2003, pp. 445- 450, 2003.

[3] Mamun, Q.E.K., Nakazato, H., A new token-based algorithm for group mutual exclusion in distributed systems. In the proceedings of the 5th international symposium on parallel and distributed computing, pp. 34-41, 2006.

[4] Advance system format (ASF) specification by Microsoft Corporation, available at URL: http://www.msdn.microsoft.com/en-s/library/bb643323.aspx

[5] Blelloch, G.E., Cheng, P., Gibbons, P.B., Scalable room synchronizations. Theory of computing systems, vol. 36, no. 5, pp. 397-430, 2003.

[6] IEEE 802.11 Working Groups for Wireless LAN, available at URL: http://ieee802.org/11/, 2003

[7] Jiang, J.R., A group mutual exclusion algorithm for ad hoc mobile networks. In the proceedings of the 6th international conference on computer science and information, pp. 266-270, 2002.

[8] Swaroop, A., Singh, A.k., From mutual exclusion to group mutual exclusion: a token based general scheme. In 2nd IEEE International conference on parallel, distributed and grid computing, pp. 645-650, 2012.

[9] Sanders, B.A., The information structure of distributed mutual exclusion algorithms. ACM transactions on computer systems, vol. 5, no. 3, pp. 284-299, 1987.

[10] Helary, J.M., Mostefaoui, A., Raynal, M., A general scheme for token- and tree based distributed mutual exclusion algorithms. IEEE transactions on parallel and distributed systems, vol. 5, no. 11, pp. 1185-1196, 1994.

[11] Manabe, Y., Baladoni, R., Raynal, M., Aoyagi, S., k-arbiter: A safe and general scheme for *h* out of *k* mutual exclusion. Theoretical computer science, vol. 193, no.1, pp. 97-112, 1998.

[12] Mittal, N., Mohan, P.K., A priority-based distributed group mutual exclusion algorithm when group access is non-uniform. Journal of parallel and distributed computing, vol. 67, no. 7, pp. 795-815, 2007.