Solving Critical Section problem in Distributed system by Entangled Quantum bits

Mohammad Rastegari
and
Amir Masoud Rahmani

Department of Computer Engineering,
University of Science and Research, Tehran
February,2008

Rastegari.mohammad@yahoo.com
Rahmani74@yahoo.com

Abstract

In this paper we will propose a new approach to solve critical section problem in distributed system by taking advantage of a magical property of Quantum bits which known as entanglement. by this property the state of many quantum bit can be understood by measuring only one quantum bit. We tried to present a distributed algorithm with as minimum as possible message passing through the network. The current methods that solve this problem have tow main drawbacks one is ***bottleneck and single point of error*** *another is* ***many message per enter and exit into the critical section****. Our algorithm is a distributed algorithm so it dose not have the first drawback. We showed Our proposed algorithm has constant number of message passing with high probability so the algorithm dose not have second problem too.*

1. Introduction

A distributed system is a collection of autonomous computers connected via a communication network. There is no common memory and processes communicate through message passing. One of the most important purposes of the distributed systems is to provide an efficient and convenient environment for sharing of resources. They also provide computational speedup and better reliability. A thorough discussion on various concepts and design issues of distributed systems can be found in Refs. [1,2,3].

In a system consisting of a number of processes, each process has a segment of code, called a critical section, in which the process may be changing common variables, updating a table, writing a file and so on [15]. A distributed system may have thousands of data items and scores of databases residing in many widely dispersed sites. Moreover, many users can have simultaneous access to this data. Therefore, it is required that processes at different sites cooperate with one another and have some sort of coordination among them. In many applications, there are critical operations that should not be performed by different processes concurrently. It may sometimes be required that a typical resource is used by only one process at a time. Or, if a data item is replicated at various sites, we expect that only one of the sites updates it at a time. This gives rise to the problem of mutual exclusion, which requires that only one of the contending processes be allowed, at a time, to enter its critical section (CS). Thus, mutual exclusion is crucial for the design of distributed systems.[13]

An important application of distributed systems, which uses mutual exclusion and needs special mention, is in the field of replicated databases. Replication is the maintenance of on-line copies of data and other resources [2]. A replicated database is a distributed database in which some data items are stored redundantly at multiple sites [4]. In replicated databases, a data item or a file is replicated at a number of sites with independent failure modes. This results in increased availability, better fault-tolerance capability and improved response time. However, maintaining the consistency of the data is a major issue. For example, owing to node or link failures, the network may partition into isolated groups of nodes. In such a case, it is not

desirable that users at isolated groups update the database independently. If a node is to perform updates, it must ensure that no other node (in any group, whatsoever) is doing this activity. Thus, mutual exclusion is needed for updating files in replicated databases. Protocols that control access to replicated data and ensure data consistency in case of network partitioning are called replica control protocols. A transaction (an access request to data) is the basic unit of user computation in a database. It executes in three steps [16]: it reads a portion of the database into a local workspace; then it performs some local computation on it; and finally, it writes some values into the database. Read and write operations performed by a transaction are of interest to us. All replica control protocols require that mutual exclusion must be guaranteed between two write operations and a read and write operation. We have no intention of going into the finer details of replicated databases and related issues. A reference to replicated databases and replica control protocols has been made, simply to emphasize the role of mutual exclusion in this important application of distributed systems. A discussion of the issues involved in maintaining the consistency of replicated database systems, the basic techniques for managing replicated data and their relative merits can be found in Refs. [5,6,7].

Another important application of mutual exclusion is in the field of distributed shared memory. Distributed shared memory (DSM) is an abstraction used for sharing data between computers that do not share physical memory. Processes access distributed shared memory through reads and updates. Any process may access the variables stored in the DSM in the system. However, if more than one process is allowed to update these variables, inconsistencies may arise. Therefore, it is required that only one process is permitted to execute CS (a fragment of code whose simultaneous execution by more than a process may cause some sort of inconsistency). Thus, applications based on DSM may require mutual exclusion. A thorough discussion on distributed shared memory and related topics can be found in Refs. [1,2]. Other areas where mutual exclusion is desirable include atomic commitment.

System involving multiple processes are often most easily programmed using critical section. When a process has to read or update certain shred data structures, it first enters into a critical section to achieve mutual exclusion and ensure that no other process will use the shared data structures at the same time.

In single processor system, critical section are protected using semaphore, monitor, and similar constructs. How critical section and mutual exclusion can be implemented in distributed system? There are several algorithm that wants to solve critical section problem in distributed system. We will briefly explain current method and comparison between them in section 2 (for more detail see [12]), we will see that all of these method has some drawbacks. Tow main drawbacks is *bottleneck and single point of error* and *many message per enter and exit to the critical section.*

1.2. Quantum computation

Quantum computing is a new approach to computation that has the possibility to revolutionize the field of computer science. The late Nobel Prize winning physicist Richard Feynman, who was interested in using a computer to simulate quantum systems, first investigated using quantum systems to do computation in 1982[11] . He realized that the classical storage requirements for quantum systems grow exponentially in the number of particles. So while simulating twenty quantum particles only requires storing a million values, doubling this to a forty particle simulation would require a trillion values. Interesting simulations, say using a hundred or thousand particles, would not be possible, even using every computer on the planet. Thus he suggested making computers that utilized quantum particles as a computational resource that could simulate general quantum systems in order to do large simulations, and the idea of using quantum mechanical effects to do computation was born. The exponential storage capacity, coupled with some spooky effects like quantum entanglement, has led researchers to probe deeper into the computing power of quantum systems. Quantum computing has blossomed over the past 20 years, demonstrating the ability to solve some problems exponentially faster than any current computer could ever do. The most famous algorithm, the integer-factoring algorithm of Peter Shor [17], would allow the most popular encryption methods in use today to be cracked easily, if large enough quantum computers can be constructed. Thus the race is on to develop the theory and hardware that would enable quantum computing to become as widespread as PCs are today. Classical computers, which include all current mainstream computers, work on discrete pieces of information, and manipulate them according to rules laid out by John Von Neumann in the 1940's. In honor of his groundbreaking work, current computers are said to run on a "Von Neumann architecture", which is modeled on an abstraction of discrete pieces of information. However, in recent years, scientists have changed from this abstraction of computing, to realizing that since a computer must ultimately be a physical device, the rules governing computation should be derived from physical law. Quantum mechanics is one of the most fundamental

physical theories, and thus was a good choice to study what computational tasks could be physically achieved. This study led to the profound discovery that quantum mechanics allows much more powerful machines than the Von Neumann abstraction.

A new concept of quantum computation has been developed in recent years. The basic unit of a quantum computer is a quantum mechanical two-level system (qubit) that can be in coherent superpositions of the logical values 0 and 1, as opposed to classical bits that represent either the values 0 or 1. Moreover, qubits can possess mutually tied properties while separated in space, so-called quantum entanglement. The implementation of computations is carried out by unitary transformations, which consist of the individual quantum logic gates. The utilization of superposition and entanglement leads to a high degree of parallelism, which makes the speed of certain types of computation exponentially faster than the classical counterpart.

1.3. Key idea

We saw an entity in quantum computers world known as Qbit, Qbit has several magic properties, one of them is entanglement, by this property given tow entangled Qbit if you see to the state(0 or 1) of one of them, you can understand what is another Qbit's state, even they be so far from each other geographically. and also if you have more than tow entangled Qbit, by measuring(seeing) only one of them you can understand what are other Qbit's state. That was the key idea which we used.

In a distributed system contained N computers the computers may be so far from each other geographically, giving entangled Qbit to each computer, by measuring only the Qbit in one of them we can become aware of the state of all other computers without passing any message to other computers. So a computer before enter to critical section can be aware that whether any other computer is in critical section or not. we will shortly explain fundamental of quantum computing and qbit in section 3(for more detail see [14]), then we will explain our method in section 4, at the end we brought a conclusion on our discussion.

2. Related work

So far several researchers have been worked in topic of Critical Section in distributed systems and proposed several algorithm in this regard. The three famous algorithms are Centralized, Distributed and Token Ring algorithm.

2.1. Centralized Algorithm

In this algorithm, we choose a node as coordinator , every other node which going to enter to critical section must consult with coordinator first, if it has allowed to enter, coordinator respond to it OK and permission is granted. If it has not allowed, the coordinator dose not reply and put this request on its queue and whenever other process leave the critical section, coordinator grant permission to the in turn process.

The centralized approach has shortcoming. The coordinator is a single point of failure, so if it crashes, the entire system may go down. If processes normally block after making a request, they cannot distinguish a dead coordinator from "permission denied' since in both case no message comes back. In addition, in a large system, a single coordinator can be come a performance bottleneck.[12]

2.2. Distributed Algorithm

In this algorithm, each node, which want to enter to the critical section, must take permission from all other nodes. Each process has a time stamp and when tow or more processes want to grant permission simultaneously, which that has smaller timestamp let to bigger one to have permission.

Unfortunately, the single point of failure has been replaced by n points of failure. If any process crashes, it will fail to respond to requests. But there are no bottleneck.[12]

2.3. Token Ring Algorithm

In this algorithm, a completely different approach to achieving mutual exclusion in a distributed system is introduced. Here we have a specific network topology with no inherent ordering of the processes. In software, a logical ring is constructed in which each process is assigned a position in the ring . the ring position may allocated in numerical order of network addresses or some other means. It dose not matter what the ordering is. All that matter is that each process who is next in line after itself.

When the ring is initialized, process 0 is given a token. The token circulate around the ring. It is passed from process k to process k+1, in point-to-point messages. When a process acquires the token from its neighbor, it checks to see if it is attempting to enter a critical region. If so, the process enter the region, dose all the work it needs to, and leaves the region. After it has exited, it passes the token along the ring. It is not permitted to enter a second critical region using the same token.

If a process is handed the token by it's neighbor and is not interested in entering in critical region, it just passes it along. As a consequence, when no processes want to enter any critical regions, the token just circulates at high speed around the ring.

As usual, this algorithm has problems too. If the token is ever lost, it must be regenerated. In fact, detecting that

it is lost is difficult, since the amount of time between successive appearances of the token on the network is unbounded. The fact that the token has not been spotted for an hour dose not mean that it has been lost; somebody may still be using it.
The algorithm also run in trouble if a process crashes, but recovery is easier than in the other cases. Here we leave more detail on discussion to [12].

2.4. Comparison of three algorithm

A brief comparison of the three mutual exclusion algorithms is in Table (2.1). Table (2.1) list the algorithms and three key properties: the number of message required for a process to enter and exit a critical section, the delay before entry can occur (assuming message are passed sequentially over a network), and some problem associated with each algorithm.

Table 2.1 : comparison of three algorithm Centralized, Distributed and Token Ring.

Algorithm	**Message per entry/exite**	**Delay before entry(in message time)**	**Problems**
Centralized	**3**	**2**	Single point of error(Coordinator crash), bottleneck
Distributed	**2(n-1)**	**2(n-1)**	Crash of any process, lots of message passing
Token ring	**1 to** ∞	**0 to n-1**	Lost token, process crash, lots of message passing

Finally, all three algorithms suffer badly in the event of crashes. Special measures and additional complexity must be introduced to avoid having a crash bring down the entire system. It is ironic that the distributed algorithm are even more sensitive to crashes than centralized one. In a fault-tolerant system, none of these would be suitable, but if crashes are very infrequent, they might do.
In [14] introduced others algorithms and exactly comparison between them.

3. Fundamental Basis of Quantum Computer

3.1. Theoretical definition of a Qbit

At first of this section we will introduce a mathematical model for Quantum Bits and then we will see some physical objects which obey this modeling in the real world. [18]

1) **Quantum Superposition***:* In the classical world we have an atomic bit. a small element which has either a 0 or 1 value. Some Quantum experiments[9],[10] showed that in the quantum world we are faced with a probability density, spread all over the world and without a detecting operation it will be impossible to understand whether that value is 0 or 1.We will explain this concept with the following definition.
As a mathematical definition a Qbit is a vector, ***a linear combination of tow fundamental basis state known as |0> and |1>*** , so it can be shown by a vector presentation:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad \textit{where} \quad \alpha^2 + \beta^2 = 1. \qquad (3.1)$$

This linear combination is called a ***Quantum superposition*** of the basis state |0> and |1>.
The only constrain with this definition is the condition: $\alpha^2 + \beta^2 = 1$. This is because α^2 and β^2 are quantum probability density as we will see in Figure(#) show a sample Qbit.
It should be mentioned that the basis vector |0> and |1> are just like all other ordinary vectors and they can be replace by any two linear independent vectors.

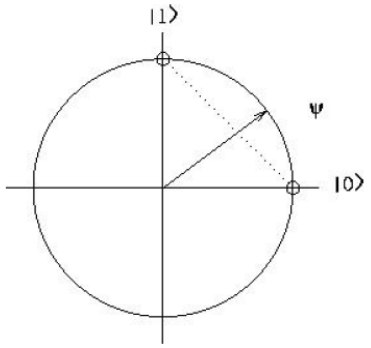

Fig 1: a sample qbit in state: $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$

2) **Quantum Measurement:** Up to now we have introduced a general Qbit. But it is still one thing remaining: ***If a Qbit is in a superposition of of tow basis states how can we measure its state?*** The answer to this question comes from one of the basic principles of the quantum mechanics: *It is impossible to understand a Qubit's state without performing a detection.*
A detector is a mathematical function D(one or more Qbits $\rightarrow$ {|0>,|1>}). When you detect a Qbit using a detector the following scenario happens:
The state of the Qbit collapses to one of the basis states. it collapses to state |0> with probability α^2 and |1> with

probability β^2 . And though we have no other states it is obvious that we must have $\alpha^2 + \beta^2 = 1$.

After that the Qbit acts like any ordinary classic bit and it is impossible to move it back to its quantum superposition state. The operation of detecting a Qbit state is also known as ***Quantum Measurement***.

3) **Some real Qbits:** Also all we have done up to know is only a mathematical modeling but it is more than just some mathematical definitions, as whole of the quantum physics is a mathematical modeling which overcomes many weaknesses of the classical physics in the real world.

There are many situations you can find objects obeying superposition and measurement principles in the quantum world. For example a photon moving toward a half mirror which reflects it with the probability 0.5 and passes with the same probability. before detection the photon is in both of the paths(a superposition of two paths).But if you put a detector in one of the paths it will detect the photon with probability 0.5.

As another example consider an excited atom the valance electrons are in the superposition of both normal and exited states. See Figure(2).And after detecting the electron by measuring its level of energy it collapses to one of these two layers[8].

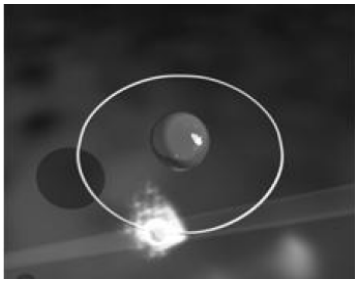

Fig 2: Giving sufficient energy to the electrons will move them to a superposition state

$$\alpha|1\rangle + \beta|0\rangle \qquad (3.3)$$

The answer why the quantum not gate acts linearly and not in some nonlinear fashion is not at all obvious.It turns out that ***this linear behavior is a general property of the Quantum mechanics.***

There is a convenient way of the representing the quantum Not gate in the matrix form, which follows directly from the linearity of quantum Gates. Suppose we define a matrix X to define the Not gate as follow:

$$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad (3.4)$$

3.2. Quantum Gates

Changes occurring to a quantum state can be described using language of *Quantum Computation*. Analogous to the way a classical computer is built from an electrical circuit containing wire and logic gates, a quantum computer is built from a ***Quantum Circuit*** containing wire and elementary ***Quantum Gates*** to manipulate the Quantum Information. In the following subsections we will describe elementary Quantum Gates, their major properties and their combination to form complex Quantum Circuits.

1) **Single Qbit Gates***:* Classical computer circuits consist of wires and logic gates. The wires are used to carry information around the circuits, while the logic gates perform manipulation of the information, converting it from one form to another. Consider for example, classical single bit logic gates. The only non-trivial member of this class is the *Not* gate, whose operation is to change the 0 and 1 states($0 \rightarrow 1$ and $1 \rightarrow 0$) Can an analogous ***Quantum Not Gate*** for Qbits be defined? Imagine that we had some process that took the state |0>to the state |1>, and vice versa. Such a process would obviously be a good candidate for the quantum Not gate. However, specifying the action of the gate on states |0> and |1>does not tell us what happens on the superposition of these states, without further knowledge about the properties of quantum gates. In fact, the quantum ***Not Gate*** acts ***linearly***. It takes the state:

$$\alpha|0\rangle + \beta|1\rangle \qquad (3.2)$$

To the corresponding state:

If the quantum state $\alpha|0\rangle + \beta|1\rangle$ is written in a vector notation as $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$,then the corresponding output of the quantum Not gate is

$$X \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix} \qquad (3.5)$$

So the quantum gates on a single Qbit can be described by two-by-two matrices. Are there any constraints on what matrices can be used as quantum gates?

Recall that the normalization condition requires that $\alpha^2 + \beta^2 = 1$ for a quantum state $\psi = \alpha|0\rangle + \beta|1\rangle$.

This must also be true for the quantum state $\psi' = \alpha' |0\rangle + \beta' |1\rangle$ after the gate has acted. It turn out that the appropriate condition on the matrix representing the gate is that the matrix X representing the single Qbit must be ***Unitary,*** that is $U^{\dagger}U = I$, where $U^{\dagger}$ is the adjoint of U and I is a tow-by-tow identity matrix. ***Amazingly, the unitarily constraint is the only constraint on quantum gates.*** Any unitary matrix specifies a quantum gate. So we have lots of single Qbit gates.

2) **Multiple Qbit gates***:* In the classical world, an important theoretical fact is that any function on bits can be computed from the composition of *NAND* gates alone, which is thus known as *Universal gate*.
The fundamental multi-Qbit quantum logic gate is the ***Controlled-NOT*** or ***CNOT*** gate. This gate has two input Qbits, known as the control Qbit ant the target Qbit, respectively. The circuit representation of the CNOT gate is shown in Figure 5.The top line represents the control Qbit while the bottom line represents the target Qbit. The action of the gate may be described as follows. If the control Qbit is set to 0 then the target Qbit is left alone .If the control Qbit is set to 1 then the target Qbit is flipped. In other words:

$|00>\rightarrow |00>, |01>\rightarrow |01>, |10>\rightarrow |11>, |11>\rightarrow |10>$
Another way to describe the CNOT gate is as a generalization of the classical XOR gate, the action of the gate can be
summarized as:

$$|A, B\rangle \rightarrow |A, A \oplus^{1} B\rangle . \qquad (3.6)$$

Yet another representation of the CNOT is to give a matrix representation.

$$U_{CN} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad (3.7)$$

As for the single Qbit case, the requirement that the output of the gate still hold the probability density condition of Qbits is that U_{CN} is a unitary matrix ,that is $U_{CN}^{\dagger} U_{CN} = I$ Of course there are many interesting quantum gates other than the controlled-NOT.

1-$\oplus$ Is addition module 2.

3.3. Quantum Entanglement (Bell state or EPR pair)

Suppose we have two qubits *x* and *y*, both of them initialized in one of basic states $|\mathbf{0}\rangle$ or $|\mathbf{1}\rangle$. As shown in Fig (3) we pass first qbit from a hadamard gate and then we pass both of them from a controlled-NOT gate

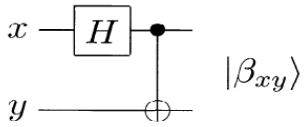

Fig 3: a circuit that make entangled Qbits

After this we are in sate :

$$|\beta_{xy}\rangle \equiv \frac{|0, y\rangle + (-1)^{x}|1, \bar{y}\rangle}{\sqrt{2}} \qquad (3.8)$$

Depends on *x* and *y* we get :

Table 3.1 : All possible entangled state for tow Qbit.

In	Out
$\|00\rangle$	$(\|00\rangle + \|11\rangle)/\sqrt{2} \equiv \|\beta_{00}\rangle$
$\|01\rangle$	$(\|01\rangle + \|10\rangle)/\sqrt{2} \equiv \|\beta_{01}\rangle$
$\|10\rangle$	$(\|00\rangle - \|11\rangle)/\sqrt{2} \equiv \|\beta_{10}\rangle$
$\|11\rangle$	$(\|01\rangle - \|10\rangle)/\sqrt{2} \equiv \|\beta_{11}\rangle$

Now consider to the first row in the Table(3.1) the out put indicate that the first and second qbit always have same state. It means that when we do measurement on first qbit if it be zero, the second qbit is zero too and if it be one, the second qbit is one tow. In other word we can understand the state of second qbit by doing measurement on only first qbit.

4. Our Method

Specifying N(N is the number of nodes)-qbit registers to each node in a typical distributed system such that these qbits be entangled in this form:

$$|\psi\rangle = \frac{1}{\sqrt{N}}(|R_1^1, R_2^1, \cdots, R_N^1\rangle + |R_1^2, R_2^2, \cdots, R_N^2\rangle + \cdots |R_1^N, R_2^N, \cdots, R_N^N\rangle) \qquad (4.1)$$

R_i^j indicate the register belong to i'th node which its j'th qbit is 1 .

$$|\psi\rangle = \frac{1}{\sqrt{N}}(|10\cdots0,10\cdots0,\cdots,10\cdots0\rangle + |01\cdots0,01\cdots0,\cdots,01\cdots0\rangle + \cdots + |00\cdots1,00\cdots1,\cdots,00\cdots1\rangle) \quad (4.2)$$

We assign a same unique register-ID to the registers which are entangled with each other. this register-ID is a positive integer from zero. so each node has many registers with different register-ID. It means all registers with same register-ID in different nodes are entangled with each other. ***Current Register*** is a register in each node which must apply for measurement. ***regID*** in each node indicate current register's register-ID.

By measuring the *i*'th qbit of current register in node i, we can understand whether node i has allowed to enter to the critical section or not. In our protocol, if i'th qbit was 1 after measurement, node i are allowed to enter to critical section. but if i'th qbit was 0, we must determine whether node i are not really allowed to enter to the critical section or not. It means , however node i has measured 0, maybe any other node have not been in critical section. In other side because of entanglement we are sure one qbit in current register should be 1 and the position of that qbit in register indicate the node which its i'th qbit in its register is 1. so it seems that the node has permission to enter to critical section. But maybe that node do not be in critical section and also do not want to enter to critical section.

For coming over to this problem , we do measurement on the i'th node's current register, we are sure all other qbit are in one of their basic states, if j'th qbit was 1, we can found out j'th node has the permission to enter to the critical section. Now we send a message to j'th node to find out whether registers by this register-ID have measured by other nodes yet or not, we do this by comparing regID in node i with regID in node j.so if regID in node i was less than regID in node j,we can understand that this register measured befor and if it was bigger than regID in node j, we can understand node j is not in critical section and node i can enter to critical section because the current register in node i either measured before or not, so if it was measured, the regID in node i is less than in node j and if it have not measured by any other node yet, it has biggest regID . if node I's current register has measured, question is: which registers has not measured yet? For finding the answer we change the regID in node i to the regID in node j, in other word we choose another register in node i as current register. And then we try to enter to critical section by this new current register. We repeat this until finding bigger regID than node i's regID. What is it's register-ID?. if have not measured yet, question is: whether node j is in critical section or not. If it was in critical section we block(down) the process in node i and enqueue this process to queue belonging to node j. if it was not in critical section, we change the value of node j's register to i by flipping i'th and j'th qbit in node j's register. then transfer whole of queue that belong to node j except process i to the node i's queue, then put *True* value to the critical-flag variable in node I and then can enter to critical section.

For exit from critical section in node i, first check its queue if that was not empty, put *False* value to the critical-flag variable in node I then active(up) the process in front of queue, and then exit from critical section. If the queue was empty, increment register-ID then put *False* value to the critical-flag variable in node i.

We brought a pseudo code for this algorithm :

Algorithm 4.1

```
Enter(node(I)){
    Chq=Measurement node(i).Register[regID].qbit(i);
    If (chq= =0){
        j=position of that bit in register which is 1;
        Ans=message(node(i),node(j))
        If (ans = = false){
            Transfer {node(j).queue-prccess i} to
node(i).queue;
            node(i).critical=true;
        }else
        if (ans was a number){
            node(I).regID=ans;
            Enter(node(I))
        } else node(i).down;
    }
   node(i).critical-flag=true;
}
```

Algorithm 4.2

```
Exit(node(i)){
 if (node(i).queue[front]<> empty){
top=node(i).queue[front];
node(i).Register[regID].registervalue= top;
    node(i).critical=false;
   Transfer {node(i).queue-prccess top} to
node(top).queue;
    node(node(i).dequeue]).up;

    }
    else{
     node(i).regID++;
     Node(i).critical-flag=false;
    }
```

```
}
```

Algorithm 4.3

```
iBool Message(node(src),node(trg)){
    If (node(src).regID> node(trg).regID){
       node(trg).regID=node(src).regID;
      node(trg).Rgister[regID].registervalue=src;
     return false;}
   If (node(src).regID<node(trg).regID){
     Retun node(trg).regID;
    }
    trg= node(trg).Register[regID].registervalue
    Node(trg).queue[rear]=src;
    if (node(trg).critical < >
      false)||(node(trg).queue[front] < >
      src)  return true;
    Else{
    node(trg).Rgister[regID].registervalue=src;}
    return false;
 }
```

This algorithm is a distributed algorithm and so dose not has bottleneck and single point of error. In next section we will compute the number of message passing through this algorithm.

4.1. Message passing analysis

In this section we want to compute the number of message passed per enter and exit to the critical section. For exit, as you see in pseudo code, we may only pass one message to active the node in front of queue and one message to transfer queue. So for exit we at most need to pass 2 message.

For enter to critical section, as you see in pseudo code, we need to call the message function once and in this function at most 2 messages will passed. And we need another message to transfer the queue. So in total we need 3 messages to enter. But as you see the Enter function is a recursive function so it will be iterate. The number of iteration depend on how many computer consequently went to their critical section. If $\boldsymbol{M}$ computer consequently went to their critical section and after this the first node in this sequence wants to enter to its critical section, the Enter function iterate $\boldsymbol{M}$ times. So $\boldsymbol{3*M}$ message will passed. Now you suppose our distributed network is a *N-complete graph* and the probability that a node wants to enter to critical section be less than *0.5*. If *M* computer want to enter to their critical section consequently, they must pass a path with length *M* in network. Number of path between tow nodes with length *M* from an special node in a N-complete graph is $(N-M-1)!$. So the probability that M computer enter to their critical section consequently is

$$(N-M-1)!\times(\frac{1}{s})^M,\ s\geq 2\ . \qquad (4.3)$$

and the probability that first node in sequence enter to its critical section is $\frac{1}{N}$. So the total probability is

$$P(M)=\frac{1}{N}\times(N-M-1)!\times(\frac{1}{s})^M,\ s\geq 2\ . \qquad (4.4)$$

And $\lim_{M\to N} P(M)\simeq 0$. And the highest probability is when *M=1*, $P(1)=\frac{N-2}{N}\times(\frac{1}{s})$ and if *N* be a big number $P(1)\simeq\frac{1}{s}$. and so with high probability Enter function called once and so only 3 messages will passed.
The total message per entry and exit will be 5 messages. Delay before entry in terms of message time is only 1 message. As you see in pseudo code before entry just need to transfer the queue to which nodes that wants to enter to critical section. the main problem of this algorithm is *hardness to implement* because of quantum computers has not implemented completely yet but we hope in future it will be construct. Table (4.1) briefly show the properties of our method.

Table 4.1: properties of our proposed algorithm

Algorithm	Message per entry/exite	Delay before entry(in message time)	Problems
Proposed algorithm	**5**	**1**	Hardness to implement

5. Conclusion

We proposed an algorithm for solve critical section problem in distributed system. Our method do not has which problems that is in current algorithms such as bottleneck and single point of error and lots of message per entry and exit. We proved that our algorithm needs to 4 messages for enter to critical section and 1 message for exit. And delay before entry is just 1 message.

The problem of this algorithm is *hardness to implement* because of quantum computers has not implemented completely yet. Nowadays scientifics could construct a quantum computer with ability to run some special algorithm such Shore's algorithm [17]. but we hope in future quantum computer will be used for distributed system and so we hope on that time they can use of our algorithm.

References

[1] J. Bacon, ***Concurrent Systems***, 2nd ed., Addison Wesley, Harlow, England, 1998.

[2] G. Coulouris, J. Dollimore, T. Kindberg, Distributed Systems: ***Concepts and Design***, *2nd ed., AddisonWesley*, Harlow, England, 1994.

[3] S. Mullender, ***Distributed Systems***, 2nd ed., Addison-Wesley, Harlow, England, 1993.

[4] P.A. Bernstein, N. Goodman, ***An algorithm for concurrency control and recovery for replicated database***, ACM Transactions on Database Systems 9 (4) (1984) 596–615.

[5] M. Ahmad, M.H. Ammar, S.Y. Cheung, ***Replicated data management in distributed systems, Readings in Distributed Computing Systems***, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 572–591.

[6] S.B. Davidson, H. Garcia-Molina, D. Skeen, ***Consistency in partitioned networks, Computing Surveys*** 17 (3) (1985) 341– 369.

[7] S.H. Son, ***Synchronization of replicated data in distributed systems, Information Systems*** 12 (20) (1987) 191–202.

[8] A. Ambainis. ***Short course on quantum computing***. http://www.tcs.hut. /Research/Crypto/minicourses/index.shtml #ambainis, 2004.

[9] M. A. Nielsen and I. l. ***Chuang. Quantum Computation and Quantum Information. Cambridge*** University Press, 2000.

[10] J. Stolze and D. Suter. ***Quantum Computing : A Short Course from Theory to Experiment***. Springer, 1989.

[11] R. Feynman, ***Simulating physics with computers***. International Journal of Theoretical Physics *21*, 6&7, pp.467 488.

[12] A.S.Tanenbaum, M.V.Steen, ***Distributed Systems Principles and Paradigms***, *Prentice-Hall 2002.*

[13] P.C. Saxena, J. Rai, ***A survey of permission-based distributed mutual exclusion algorithms***, Computer Standards & Interfaces 25 (2003) 159–181.

[14] Shiwa S. Fu, Nian-Feng Tzeng, Jen-Yao Chung, ***Empirical Evaluation of Mutual Exclusion Algorithms for Distributed Systems***, Journal of Parallel and Distributed Computing 60, 785_806 (2000)

[15] A. Silberschatz, P. Galvin, ***Operating System Concepts,*** 5th ed., Addison-Wesley, Harlow, England, 1998.

[16] P.A. Bernstein, D.W. Shipman, W.S. Wong, Formal ***aspects of serializability in database concurrency control***, IEEE Transactions on Software Engineering SE 5 (3) (1979) 203–216.

[17] P. W. Shor, ***Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer***, Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, Nov. 20–22, 1994, IEEE Computer Society Press, pp. 124–134.

[18] S.Jafarpour ***Introduction to the World of Quantum Computers***. IEEE ICCI 2006: 760-764