FAULT TOLERANCE IN DISTRIBUTED SYSTEMS USING DYNAMIC VOTE REASSIGNMENT

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology

In

Computer Science and Engineering

By

BIKASH KUMAR GUPTA

SANGRAM JYOTI BAL

Under the Guidance of

PROF. P.M.KHILLAR

Department of Computer Science and Engineering

National Institute of Technology

Rourkela

National Institute of Technology

Rourkela

CERTIFICATE

This is to certify that the thesis entitled **"Fault Tolerance in Distributed Systems Using Dynamic Vote Reassignment"** Submitted by **Sangram Jyoti Bal, Roll No : 10606037** and **Bikash Kumar Gupta, Roll No: 10606040** in the partial fulfillment of the requirement for the degree of **Bachelor of Technology** in **Computer Science Engineering**, **National Institute of Technology, Rourkela** , is being carried out under my supervision.

To the best of my knowledge the matter embodied in the thesis has not been submitted to any other university/institute for the award of any degree or diploma.

Prof. P.M.Khillar

Date: Department of Computer Science and Engineering
National Institute of Technology
Rourkela-769008

ACKNOWLEDGEMENT

We avail this opportunity to extend our hearty indebtedness to our guide **Prof. P.M.Khillar**, Computer Science Engineering Department, for their valuable guidance, constant encouragement and kind help at different stages for the execution of this dissertation work.

We also express our sincere gratitude to **Prof. B.Majhi**, Head of the Department, Computer Science Engineering, for providing valuable departmental facilities.

Submitted by:

Bikash Kumar Gupta
Roll No: 10606040
Computer Science and Engineering
National Institute of Technology
Rourkela

Sangram Jyoti Bal
Roll No: 10606037
Computer Science and Engineering
National Institute of Technology
Rourkela

CONTENTS

ABSTRACT

There are several fault tolerant protocols for managing replicated files in the event of network partitioning due to site or communication link failures. Previously there has been no software simulation of the voting protocols apart from just stochastic modeling. In this paper, we simulate and analyze the throughput of message transfer during the communication. We use various network topologies to compare the parameters such as throughput, no of packets received and sent during voting process .We have analyzed the effects of various packet properties. The analysis provides evidence for the conjecture that the grouping scheme is the optimal algorithm in the context of the voting protocols. We also compare the proposed genetic approach for voting assignment with random algorithm proposed by Akhil Kumar. This comparison shows that genetic voting assignment gives better availability than random algorithm.

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1. Introduction

1.1 Overview

Computing systems consist of a variety of hardware and software components that are bound to fail eventually [8]. In many systems, such component failures can lead to unanticipated, potentially disruptive failure behavior and to service unavailability. Some systems are designed to be ***fault-tolerant****:* they either exhibit a well defined failure behavior when components fail or mask component failures to users that is, continue to provide their specified standard service despite the occurrence of component failures [9]. To many users temporary errant system failure behavior or service unavailability is acceptable. There is, however, a growing number of user communities for whom the cost of unpredictable, potentially hazardous failures or system service unavailability can be very significant .Examples include the on-line transaction processing, process control, and computer-based communications user communities. To minimize losses due to unpredictable failure behavior or service unavailability, these users rely on fault tolerant systems. With the ever increasing dependence placed on computing services, the number of users who will demand fault-tolerance is likely to increase.
The task of designing and understanding fault-tolerant distributed system architectures is notoriously difficult: one has to stay in control of not only the standard system activities when all components are well, but also of the complex situations which can occur when some components fail. The difficulty of this task can be exacerbated by the lack of clear structuring concepts and the use of a confusing terminology. Presently, it is quite common to see different people use

different names for the same concept or use the same term for different concepts. For example, what one person calls a failure, a second person calls a fault, and a third person might call an error. Even the term "fault-tolerant" itself is used ambiguously to designate such distinct system properties as "the system has a well-defined failure behavior" and "the system masks component failures. [8]"

When a system is designed to mask failures, it continues to perform its specified function in the event of a failure [9]. A system designed for well defined behavior may or may not perform the specified function in the event of a failure; however, it can facilitate actions suitable for recovery.

One key approach used to tolerate failures is redundancy [8]. In this approach, a system may employ a multiple number of processes, multiple numbers of hardware components, multiple numbers of copies of data, etc with independent failure modes.

1.2 Thesis Objective

- To simulate the message transfer throughput in various topologies using various packet properties.
- Comparison of the proposed genetic algorithm with the random voting algorithm proposed by Akhil Kumar [7].
- To calculate availabilities under various nodes probabilities using the proposed genetic algorithm.

1.3 Thesis Organization

This thesis is divided into 7 chapters. Each chapter focuses on a specific topic in the field of fault tolerance.

Chapter 1 gives an overview of the overall fault tolerance issues. It introduces the concept of fault tolerance and the various types fault tolerance mechanisms.

Chapter 2 deals with the literature review of the related works and illustrate various protocols used for fault tolerance. It discusses the various schemes used for the vote assignment including the dynamic vote assignment policies. It covers the group voting mechanism for effective message passing.

Chapter 3 deals with the simulation work carried out in respect of the thesis objective. It shows the various parameter performances during the simulation period. It also shows the various scenarios created for the simulation of the different topologies.

Chapter 4 gives the pseudo code for the proposed genetic voting approach .It depicts the various methods used in the voting process.

Chapter 5 shows the various experimental results that resulted from the simulation of the network topologies. It gives the overview of the performance parameters in the message transfer overhead. The comparison between the proposed algorithm and random algorithm gives the clear picture of the performance of the two algorithms.

Chapter 6 concludes showing conclusion drawn from the various simulation and experimental results

Chapter 2

LITERATURE REVIEW

Static Voting

Majority Based Dynamic Voting

Dynamic Vote Reassignment

Group Based Voting

2. Literature Review

2.1 Static voting

This static voting is proposed by Gifford [2].

Replicating data at many sites is the common approach in the fault tolerance in distributed systems. Data can still be obtained from the other copies if the original fails. Commit protocols [10, 11, 12] can be employed to update multiple copies of data .While the non-blocking protocol [11, 12] of the commit protocol family can tolerate single site failure, it is not resilient to multiple site failures, communication failures and network partitioning. In [10] commit protocols, when a site is unreachable, the coordinator sends messages repeatedly and eventually may decide to abort the transaction, thereby denying access to data. However, it is desirable that the sites continue to operate even when other sites have crashed [11, 12], or at least one partition should continue to operate after the system has been partitioned. Another well known technique used to manage replicated data is the voting mechanism [2]. With the voting mechanism [13, 14], each replica is assigned some number of votes and majority of votes must be collected from a process before it can access a replica. The voting mechanism [13] is more fault tolerant than a commit protocol in that it allows access to data under the network partitions, site failures and message loses without comprising the integrity of the data.

Algorithm

1. Site i issues a Lock_Request to its local lock manager.
2. When the lock request is granted, site i sends a Vote_Request message to all the sites.
3. When a site j receives a Vote_Request message, it issues a Lock_Request to its local lock manager. If the local request is granted, then it returns the version number of the replica (VN_j) and the number of the votes assigned to the replica (V_j) to site i.
4. Site i decide whether it has the quorum or not, based on the replicas received within a timeout period as follows (P denotes the set of sites which have replied).
    a. If the request issued was a read,

$$V_r=\sum V_k$$

    b. If the request issued was a write,

$$V_w=\sum V_k$$

    c. Where the set of sites Q is determined as follows:
        i. $M=\max\{VN_j : j€P\}$
        ii. $Q=\{j€P : VN_j=M\}$

5. If the site i is not successful in obtaining the quorum, then it issues a Release_Lock to the local lock manager as well as to all the sites in P from whom it has received votes.

6. If site i is successful in obtaining the quorum, then it checks whether its cop of the file is current. A copy is current if its version number is equal to M. If the copy is not current, a current copy is obtained from a site that has a current copy. Once a current copy is available locally, site I performs the next step.

7. If the request is a read, site i reads the current copy available locally. If the request is a write, site i updates the local copy. Once all the accesses to the copy are performed, site i updates VN_i, and sends all the updates and VN_i to all the sites in Q. Note that a write operation updates only current copies. Site i then issues a Release_Lock request to its local lock manager as well as to all the sites in P.

8. All the sites receiving the updates perform the updates on their local copy and on receiving a Release_Lock request, release the locks

2.2 Majority Based Dynamic Voting

This protocol is proposed by Jajodia and Mutchler [5].

Version number: The version number of a replica at a site i is an integer that counts the number of successful updates to the replica at i. VN_i is initially set at zero and is incremented b one at every successful updates to the replica at i. VN_i is

initially set at zero and is incremented by one at every successful update.

Number of Replicas updated: It is an integer that almost always reflects the number of replicas the number of replicas participating in the most recent update RU_i is initially equal to the number of replicas.

Distinguished sites list: The distinguished sites list [5] at a site i is a variable that stores ID's of one or more sites. The contents of DS_i depend on RU_i. When RU_i is even, DS_i identifies the replica that is greater than all the other replicas that participated in the most recent update of the replica at site i. When $RU_{i\ is}$ odd, DS_i is nil except when RU_i =3, in which case DS_i lists the three replicas that participated in the most recent update from which a majority is needed to allow access to data.

Outline of the protocol:

1. Site i issues a Lock_Request to its local lock manager.
2. If the lock is granted, site i sends a Vote_Request message to all the sites.
3. When a site j receives the Vote_Request message, it issues a Lock_Request to its local lock manager. If the lock is granted, site j sends the values of VN_j, RU_j and DS_j to site i.
4. From all the responses, site i decides whether it belongs to the distinguished partition, described shortly.
5. If site i does not belong to the distinguished partition [5], it issues a Release_Lock request to its local lock manager and sends Abort messages to all the other sites that responded. A site, on receiving a Abort message, issues a Release_Lock request to its local lock manager.

6. If site i does not belong to the distinguished partition, it performs the update if its local copy is current. Otherwise, site i obtain a current copy from one of the other sites and then perform the update. Note that along with the replica update, VN_i , RU_i and DS_i . It then issues a Release_Lock request to the local lock manager.
7. When a site j receives a commit message, it updates its replica, updates the variables VN_j, RU_j and DS_j and issues a Release_Lock request to the local lock manager.

2.3 Dynamic Vote reassignment protocols

The actual idea was proposed by Gifford [2] but was discussed in detail by Barbara, Garcia-Molina, and Spauster [16].

Barbara et al. [4,15] categorized the Dynamic vote reassignment into two types:

Group consensus

The sites in the active group agree upon the new vote assignment using either a distributed algorithm or bi selecting a coordinator to perform the task. Since the outside the majority group didn't receive any votes.

Because this method relies on active group participation, the current system topology will be known before deciding the vote assignments [4, 15, 16]. by using that information

Autonomous Reassignment

Each node makes its own decision about changing its votes and picking a new vote value, without regarding the rest of the nodes [15,16]. Before the change is made final, though, the node must collect a majority of votes.

The Protocols

The protocols for autonomous vote reassignment [4, 9, 15, 16] are what guarantee mutual exclusion. Once a node picks a new vote value, a vote changing protocol is invoked to install the change. The vote changing protocol uses the vote collecting protocol to ensure that enough votes have been collected to validate the change. In addition, the vote collecting protocol is used for all other operations requiring majority approval.

Protocol P1. Vote increasing. The initiator (node i)

1. Send the new vote value along with [15,16] V_i and N_i to the rest of the nodes with which node i can communicate.
2. Wait for a majority of acknowledgments to arrive (whether or not a majority of votes has been received by node i is determined by following protocol P2 [16]), and then install the change in the local voting vector, that is update $V_i[i]$ and increase the version number $N_i[i]$ by 1.

Protocol P2. **Vote Collecting**

Assume node i is collecting votes to decide upon an event. In this case, each voting

node j will send i two vectors, the voting vector V_j and a version vector N_j. Another vector v_i is maintained where $v_i[j]$ indicates the votes of j as determined b site i upon the collection of votes. An entry $N_i[j]$ represents the version number for the value $V_i[j]$ at site i. Node i decide upon the votes of node k (~6) using the following rules:

(a) If i receives V_j and N_j, then $v_i[j] = V_j[j]$. Also, change $V_i[j]$ to $V_j[j]$ and $N_i[j]$ to $N_j[j]$ if either of the following two conditions applies:

$V_j[j] > V_i[j]$ or $V_j[j] < V_i[j]$ and $N_j[j] > N_i[j]$.

The first condition is simply that of Scenario One. $V_j[j] > V_i[j]$ indicates that j has increased its votes since i last determined $V_i[j]$. The version number is irrelevant in this case, since it provides no additional information.

In the second case, $V_j[j] < V_i[j]$ indicates that either j has decreased its votes or an increase at k has not yet been approved or has been timed out. If, however, $N_j[j] > N_i[j]$, then $V_j[j]$ reflects a later decrease of votes at k or a failed vote increase attempt, and this new information should be recorded.

(b) If i does not receive V_j, then $v_i[j] = V_k[j]$ for k such that $N_k[j] = \max \{N_p[j] : p€G\}$, where G is the set of all sites from which site i has received replies. That is, k assume the newest value among the voting group for the vote value of node j. In addition, i modify its entry $V_i[j]$ to equal $V_k[j]$ and $N_i[j]$ to equal $N_k[j]$.

Protocol P3. Vote Decreasing The same as P1, except that:

The initiator sends V_i and N_i along with its vote decrease. Upon successfully

collecting a majority of votes, the initiator increases $N_i[i]$ by one and sets the $V_i[i]$ to the new value.

Policies

These policies were proposed by Barbara et al. [16]

The Overthrow Technique

Vote increasing under the overthrow technique [16] is straightforward. Consider a system in which node x has gone down, while the rest of the nodes are still up. (This can be considered as a partition of the system into two groups, with x in one group and the rest of the nodes in the other.) Let v, be the number of votes that node x has. Let TOT be the total number of votes in the system and MAJ the majority of votes. Assuming TOT is odd, MAJ = (TOT + 1)/2 [16]. If node a is the node supplanting X, the new number of votes for a, v: will have to be such that it covers the voting power that a had before (v,), plus the voting power of x, plus the increase in the total number of votes. If a increases its votes by $2v_x$, the total number of votes will be TOT ’ = TOT + v_x, and MAJ ’ = MAJ + v_x. It can be shown that all the majority groups that used x can be formed using a instead:

The Alliance Technique

There are many variations of the alliance technique [16]. We describe three here. In general, we want to give each node a fraction of the voting power of a node that has been excluded from the majority group. As in the overthrow technique, we want to be sure to give out at least 2u, votes in the majority group, enough to counteract those votes that node x holds plus the number of votes node x could

have contributed if it were in the active group. Of course, we can always assign a surplus of votes to each node. One possibility is to assign 2v votes to every member of the active group; or we can assign v_x votes to each member of the active group, and assign 2u, votes when there is just one node left. Another possibility is to spread 2v votes out. Say N = number of nodes in the majority group. Then, give each node in the active group ⌐2v/N ¬ votes (henceforth referred to simply as 2v/N). If need be, N can be estimated by the nodes. This may not be as good as possible in terms of resilience to failures [16], but is certainly not dangerous. No matter what the strategy, we have to be careful when there are only two nodes left in the majority group. In that situation, it is senseless to give each node the same number of votes, since if they lose communication with each other, their extra votes will only cancel each other out and no group may have a majority. Instead, it is better to pick one node and give it 2u, votes. We can use a priority mechanism to handle this case

2.4 Group Based Voting

This voting mechanism is proposed by Agarwal and Jalote [6].

In the previous voting algorithm, the site initiating the operation has to communicate with all the nodes incurring high communication costs. In this algorithm [6] the sites are divided into intersecting or overlapping groups. In the absence of failures the site initiating the operation communicates with the sites of its group thus reducing the communication costs. This algorithm suggests a method for constructing such logical groups and show that the message overhead of any operation in a system of N nodes is O (√n), when there are no or few failures in the system.

Logical group formation

Let the number of groups be n. Let the n groups in the system be referred to as

G_i (i=1…n). Each group has the cardinality n-1. This formation ensures that the site has to communicate with its own group members to have a read or write operation in case of no failures. Two numbers are chosen from this group and form combinations. Assume that the number of nodes be n (n-1)/2.

Then one-one mapping [6] is performed from number of nodes to number of combinations generated. If a node is mapped to combination (i, j), then it belongs to group i and j and in no other group. This ensures that the each group has the cardinality n-1 since there is only n-1 combinations in the set 1…n for containing number i.

Consider 15 nodes in a system. These nodes can be grouped as follows. The groups obtained by this grouping are shown below.

Group 1: (1, 2**,** 3, 4, 5)

Group 2: (1, 6, 7, 8, 9)

Group 3**:** (2, 6, 10, 11, 12)

Group 4**:** (3**,** 7, 10, 13, 14)

Group 5: (4, 8, 11, 13, 15)

Group 6**:** (5, 9, 12, 14, 15)

Voting Algorithm

Let us discuss the read quorum condition.

For this the requesting site should get the current version of the replica from the group. Therefore it should have the access to all the groups in the system which can be guaranteed if it can access at least one member in each group.

A set of nodes R satisfies the read quorum if for all i (i= 1...n), for some j, such that

I_{ij} **ϵ** R. That is at least one site from each group participates in read quorum.

Clearly, if R is G_i, then R satisfies the read quorum.

Also, a read quorum can be satisfied if vote from one node from each group can be collected.

A set of nodes R satisfies the write quorum if for some i such that for all j (j = 1...n), I_{ij} **ϵ** R, That is all the sites of particular group participates in the write quorum.

The write availability can be improved if the site initiating the operation can distinguish between the site failures and network partitions.

Suppose that a set of sites R is participating in the operation and a set of sites F is reported to have failed.

Then R and F together satisfy the write quorum if

1. A *write set* is available, that is, for some i such that for all j (j=1…n), I_{ij} **ϵ** (RUF)

2. R satisfies the read quorum condition.

Read Algorithm

1. Send read request to all nodes in G_i and wait for replies.
2. Let R be the set of node replied. If some of the intersecting node of a particular group is not present in R then look for a node in the group of the missing intersecting nodes and send the read request.
3. Read from the node having the current copy.

Write Algorithm

1. Send a write request to all the nodes in the group. Let R be the nodes replied and F be the nodes that failed. Then T=RUF.
2.
    - If the intersecting node is present in T then check if it is missing in R. if it is missing in R, then check for the nodes in the other group of the particular missing intersecting node and send read request and wait for replies.
    - If the intersecting node is not present in T then find the nodes of the other group of the missing intersecting node and send write request to all the nodes of that group and get all replies. If the write set is met then try collecting read quorum.
3. Write to all the operational node of the write set.

Performance Evaluation

Let the no of nodes be T in the system and we have n groups such that T = n (n-1)/2. We have already seen [6] that the cardinality of each group is n-1 .Now from the equation 1 solving the quadratic equation we get

$n = \frac{1 \pm \sqrt{8T+1}}{2}$ from which $n = \frac{1+\sqrt{8T+1}}{2}$ satisfies the equation

since cardinality is n-1 therefore the communication cost is O (n-1) i.e.

$O\left(\frac{\sqrt{8T+1}-1}{2}\right) = O(\sqrt{T})$

Chapter 3

SIMULATION

4. Simulation

QualNet [18] is a network simulation tool that simulates wireless and wired packet mode communication networks. *QualNet Developer* is a discrete event simulator used in the simulation of MANET, WiMAX networks, satellite networks, and sensor networks, among others. QualNet has models for common network protocols that are provided in source form and are organized around the OSI Stack.

Global Simulation Parameters

- Version
- Experiment Name
- Maximum Simulation Time
- Random Number Seed
- Coordinate System
- Terrain Corners
- Terrain Dimensions
- Irregular Terrain
- Node Placement
- Protocol Stack
- Statistics Filtering
- Mobility Options
- Mobility Position Granularity
- Application Setup File

Topology simulation

In the Qualnet Simulation environment we placed nine nodes representing the devices in the fault tolerant network according to the different topologies. We simulated three different topologies that star , ring and group topologies.

In the star we connected the nine nodes through a single hub and tested for different packets such CBR, FTP, CBR receive. The average distance between the nodes is around 1000 meters. We designed the network using the qualnet designer user interface and placed the nodes accordingly.

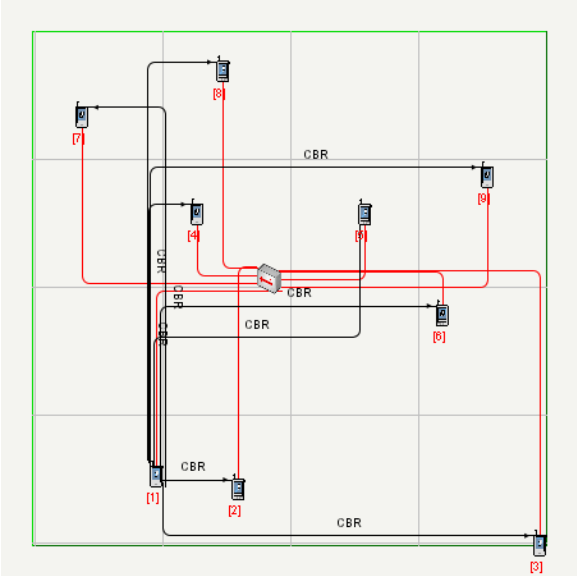
In ring topology we connected each adjacent node with each other. The average distance between the nodes is around 1000 metres.We tested for different packets such CBR, FTP, CBR receive.

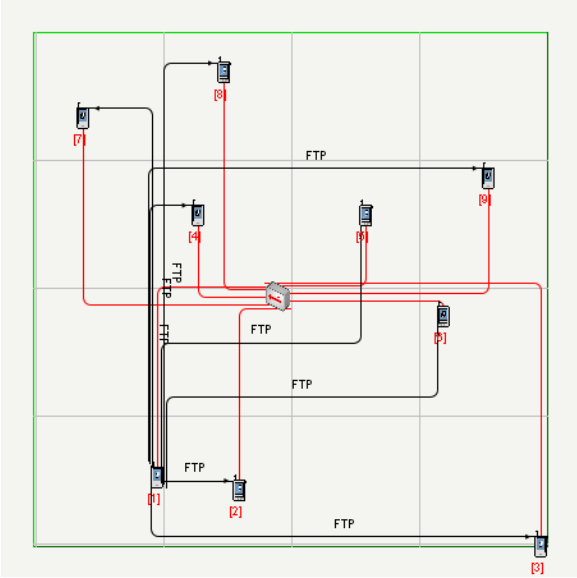
In Group topology we created group by taking three nodes in a group and each group connected to the other through a hub. The average distance between the nodes is around 1000 metres.We tested for different packets such CBR, FTP, CBR receive.

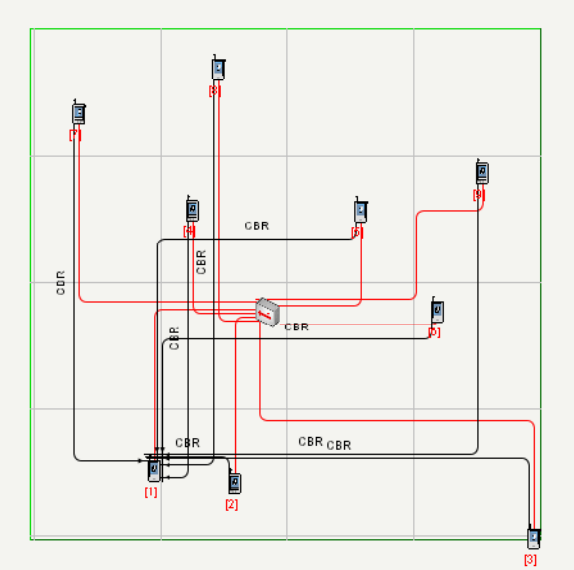
In all of the topologies we simulated for the average throughput rates for varying node density.

3.1 Star topology

Figure 1

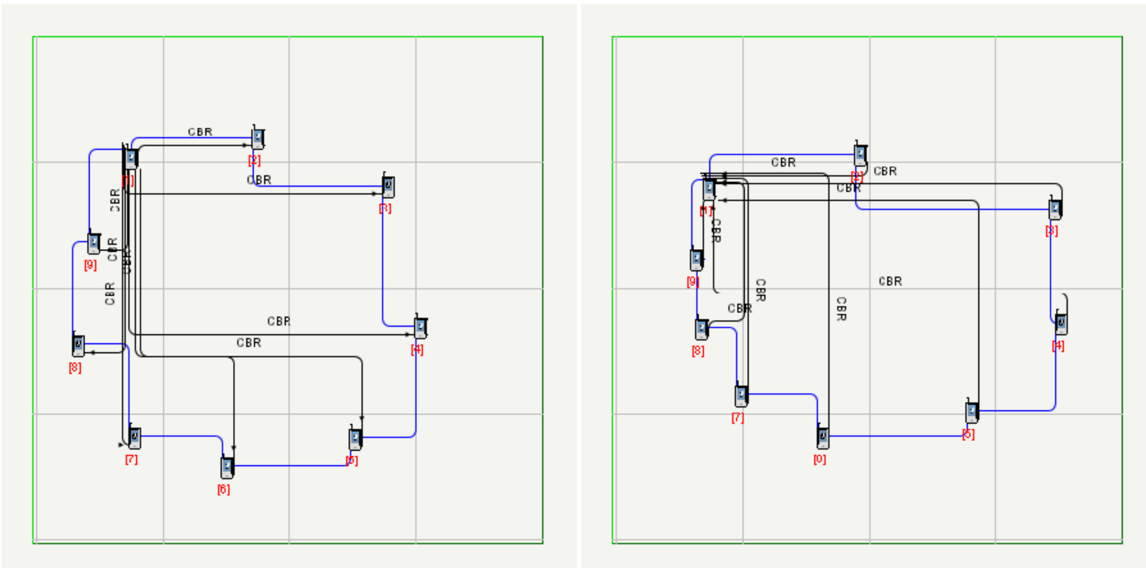






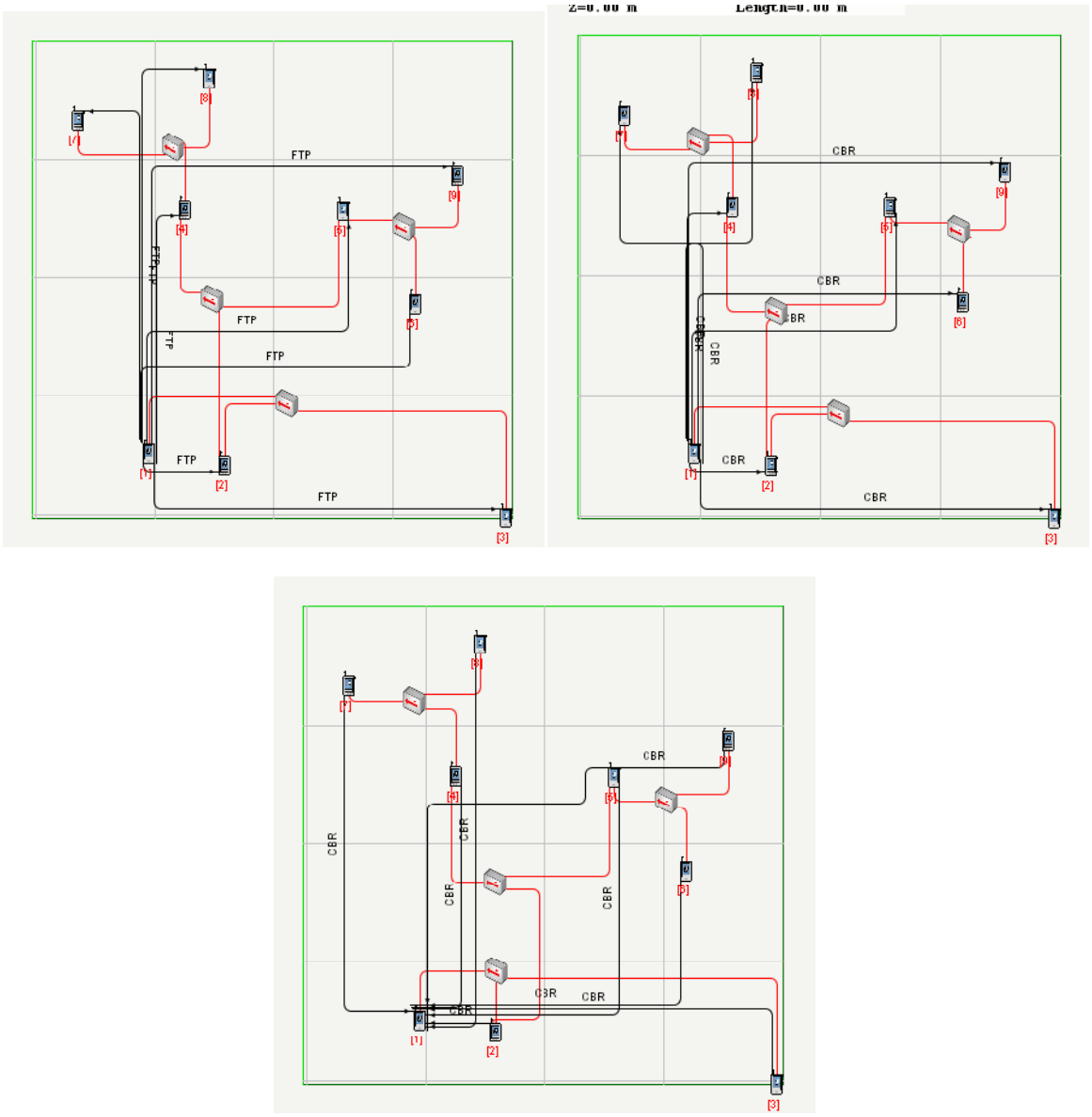
3.2 Ring Topology

Figure 2



3.3 Group Topology

Figure 3



3.4 Observations

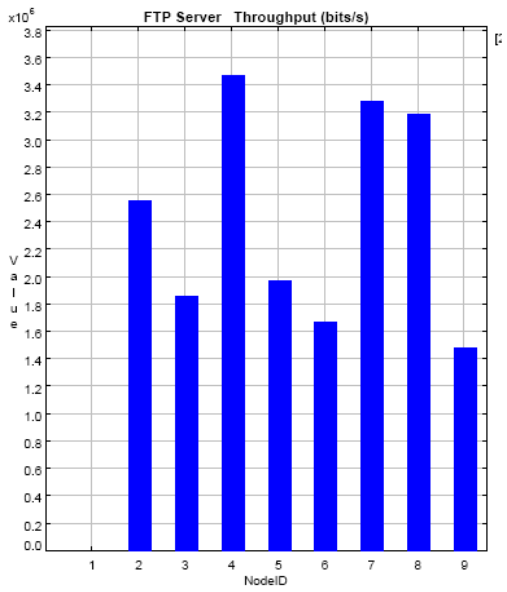
The following graphs are the results of the scenario simulation [18] which shows the various configurations of the network topologies along with the different packet properties.

In the graph 1 we get throughput rate corresponding to the no of nodes in the ring topology for the TCP protocol. In graph 2 it shows the throughput rate corresponding to the no of nodes in the star topology for the TCP protocol. In graph 3 it shows the throughput rate corresponding to the no of nodes in the group topology for the TCP protocol. In graph 4 it shows the throughput rate corresponding to the no of nodes in the ring topology for the CBR (constant bit rate) packets transmission.

In graph 5 it shows the throughput rate corresponding to the no of nodes in the star topology for the CBR (constant bit rate) packets transmission. In graph 6 it shows the throughput rate corresponding to the no of nodes in the Group topology for the CBR (constant bit rate) packets transmission. In graph 7 it shows the throughput rate corresponding to the no of nodes in the Ring topology for the CBR (constant bit rate) receive packets transmission. In graph 8 it shows the throughput rate corresponding to the no of nodes in the star topology for the CBR (constant bit rate) receive packets transmission. In graph 9 it shows the throughput rate corresponding to the no of nodes in the Group topology for the CBR (constant bit rate) receive packets transmission.

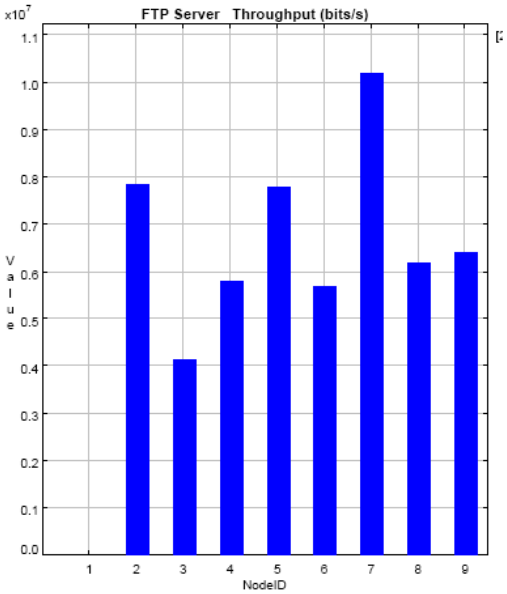
TCP in Ring topology

Graph 1



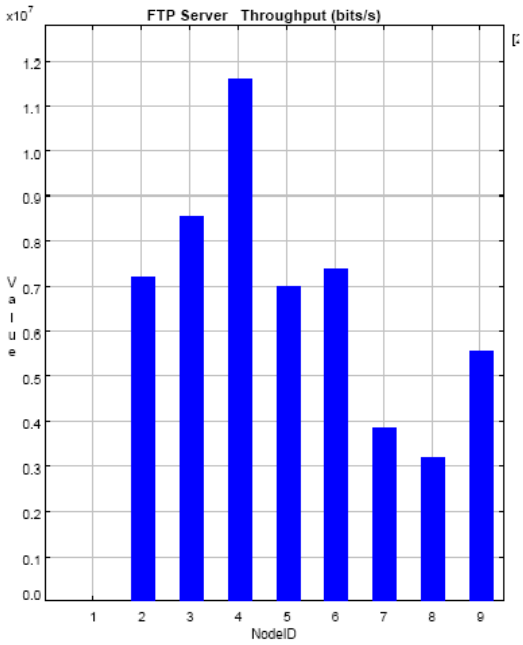
TCP in Star topology

Graph 2



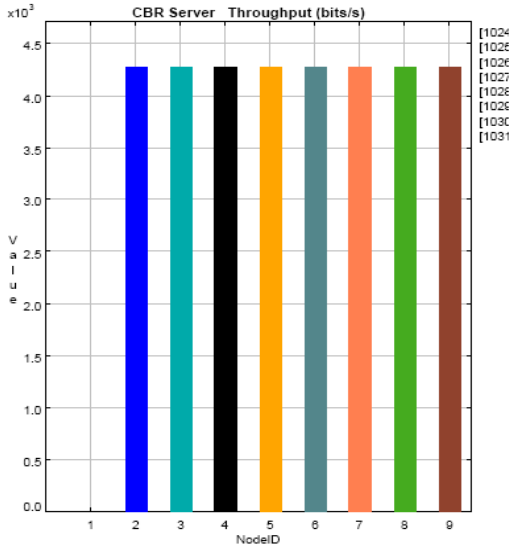
TCP in Group Topology

Graph 3



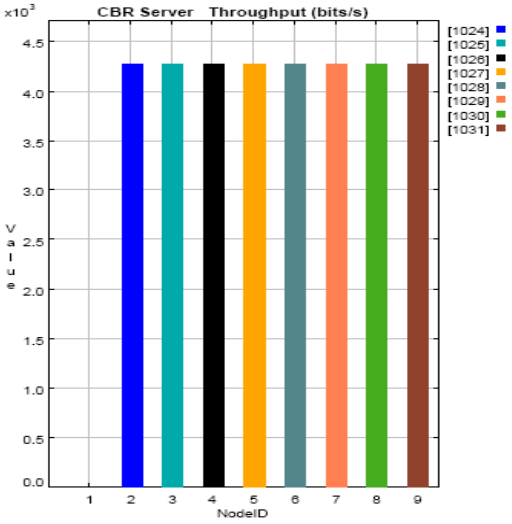
CBR in Ring Topology

Graph 4



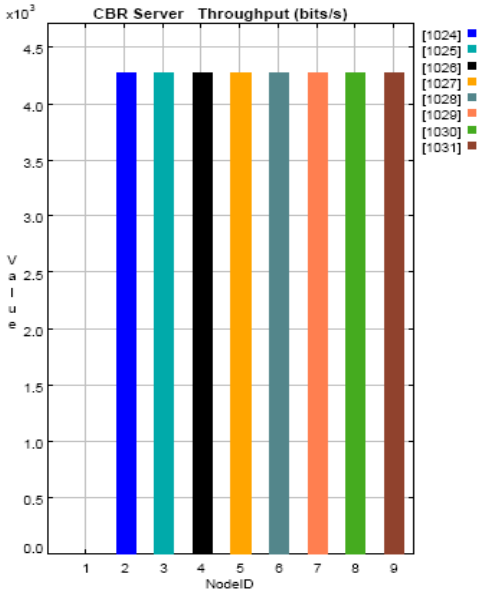
CBR in Star topology

Graph 5



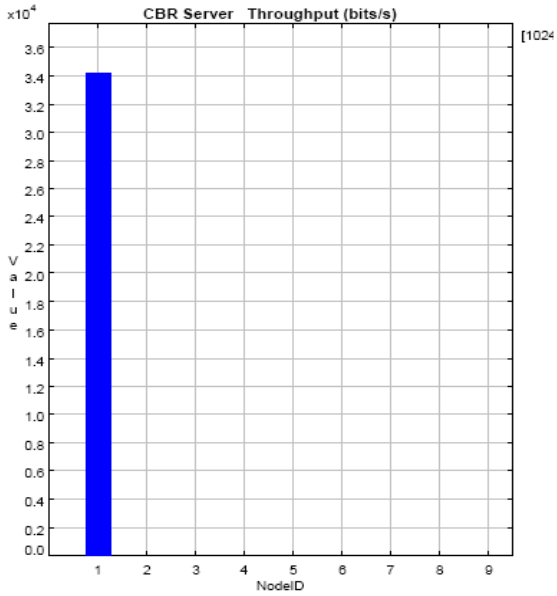
CBR in Group Topology

Graph 6



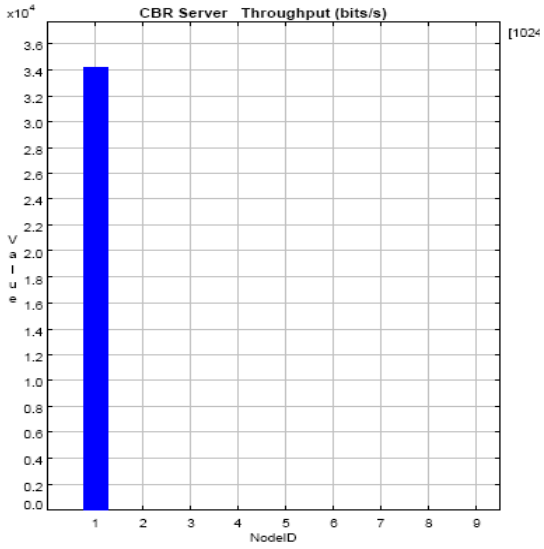
CBR receive in Ring topology

Graph 7



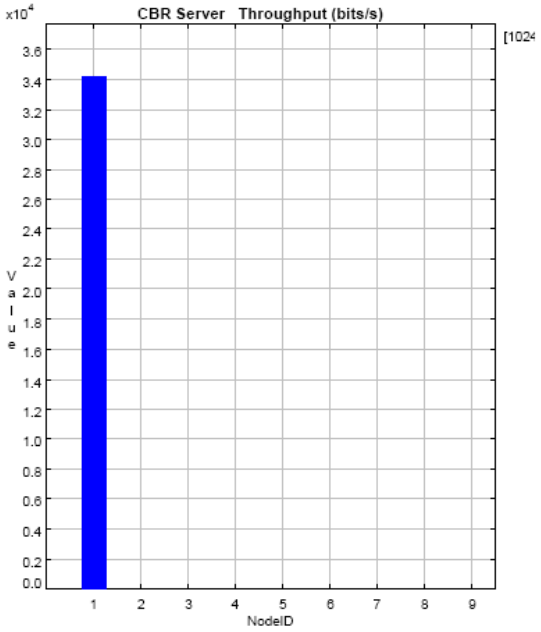
CBR receive in Star topology

Graph 8



CBR receive in group topology

Graph 9



Chapter 4

ALGORITHM

4. Algorithm

4.1 The Genetic Algorithm

Introduction

A genetic algorithm [18] (GA) is a search technique used in computing to find exact or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics. Genetic algorithms are a particular class of evolutionary algorithms (EA) that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover.

This is the key idea in solving combinatorial optimization problems by this technique. Iterative improvement (or greedy) algorithms tend to "dead-end" in locally optimal solutions; however, the genetic algorithm approach makes it possible to come out of such dead-ends and look for still better solutions

A typical genetic algorithm requires:

1. a genetic representation of the solution domain,
2. a fitness function to evaluate the solution domain

Initialization

Initially many individual solutions are randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions.

Selection

During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as this process may be very time-consuming.

Reproduction

For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously. By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents". New parents are selected for each new child, and the process continues until a new population of solutions of appropriate size is generated. Although reproduction methods that are based on the use of two parents are more "biology inspired", some research suggests more than two "parents" are better to be used to reproduce a good quality chromosome. These processes ultimately result in the next generation population of chromosomes that is different from the initial generation. Generally the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected for breeding, along with a small proportion of less fit solutions, for reasons already mentioned above.

Termination

This generational process is repeated until a termination condition has been reached. Common terminating conditions are:

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/money) reached
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Manual inspection
- Combinations of the above

Fitness function

The main objective in the vote assignment problem is to find an assignment of votes that maximizes the availability. We shall assume that both read and write quorums are equally important, and hence each quorum is set equal to a majority of the sum of all votes. The vector whose availability is maximum is selected as the solution of the problem

Chromosome

A chromosome consists of a specific vote assignment or a vector of *n* votes *(*$V_{1,}$ V_2, . . ., V_n) here V_i is the vote assigned to site *i.* A chromosome change is produced by selecting a pair of votes; say V and U, from this vector and performing one of the following two operations:

Mutation

A common method of implementing the mutation operator involves generating a random variable for each vote in a sequence. This random variable tells whether or not a particular vote will be modified. This mutation procedure, based on the biological point mutation, is called single point mutation.

Crossover

A single crossover point on both parents' vote vectors and chromosome is selected. All data beyond that point in either vector is swapped between the two parent vectors. The resulting vectors are the children.

For example, say *n* is *5*, and the two vote vector is V_1 (2,2,1,1,1) and V_2(2,1,1,3,1). By respectively applying the two operations above to V_1 and V_2, the following states are produced:

Mutation: VC_1= mutated child of V_1

= (2, 1, 1, 2, 1)

VC_2= mutated child of V_2

= (2, 3, 1, 1, 1)

Crossover: crossover child of V_1 and V_2

= (2, 1, 1, 1, 1), (2, 2, 1, 3, 1)

4.2 Pseudo code

```
public double p[]= {0.95,0.90,0.85,0.80,0.75,0.75,0.70,0.70};
public TreeSet<String> popu ;
public   String chromosome ;
public TreeSet<String> popu1;
public double best_avail=0.0;
public String combo;
public String temp_chromosome;

public double avail (String s){
        Double avail = 0;
        int sum=0;
        for (int j=0;j<s.length();j++){
                Sum=sum+ (s.charAt (j)-48);
         }
        sum=sum/2;
        double product=0;
        for (int i0=0;i0<=1;i0++)
        for (int i1=0;i1<=1;i1++)
        for (int i2=0;i2<=1;i2++)
        for (int i3=0;i3<=1;i3++)
        for (int i4=0;i4<=1;i4++)
```

```
        for (int i5=0;i5<=1;i5++)

        for (int i6=0;i6<=1;i6++)

        for (int i7=0;i7<=1;i7++)


                if((i0*s.charAt(0)+i1*s.charAt(1)+i2*s.charAt(2)+i3*s.charAt(3)+i4*s.charAt(4)+i5*
                s.charAt(5)+i6*s.charAt(6)+i7*s.charAt(7))> sum){

                        product = (i0*p[0]+(1-i0)*(1-p[0]))*(i1*p[1]+(1-i1)*(1-p[1]))*(i2*p[2]+(1-
                        i2)*(1-p[2]))*(i3*p[3]+(1-i3)*(1-p[3]))*(i4*p[4]+(1-i4)*(1-
                        p[4]))*(i5*p[5]+(1-i5)*(1-p[5]))*(i6*p[6]+(1-i6)*(1-p[6]))*(i7*p[7]+(1-
                        i7)*(1-p[7]));

                        avail = avail + product ;

                }

        return  avail;

}


public void mutate(){

        Iterator<String> itr = popu.iterator();

        String p1,p;

        popu1.clear ();

        While (itr.hasNext ()){

                p= itr. Next ();

                p1 = p.replace (p.charAt (2),(p.charAt(4)));

                p1 = p1.replace (p.charAt (4),(char) (p.charAt(2)+1));

                popu1.add (p1);

        }

        itr = popu1.iterator ();

        While (itr.hasNext ()){
```

```
                popu.add(itr. Next ());
        }
        popu1.clear ();
 }
 public void crossover ( ){
        Iterator<String> itr = popu.iterator ();
        String p1;
        String p2;
        While(itr.hasNext()){
                p1= itr. Next();
                If(itr.hasNext())
                p2= itr. Next();
                else break;
                String p3 = p1.substring(0, 3)+p2.substring(3);
                String p4 = p2.substring(0, 3)+p1.substring(3);
                popu1.add(p3);
                popu1.add(p4);
        }
        popu.clear();
        itr = popu1.iterator ();
        While (itr.hasNext()){
                popu.add (itr. Next ());
        }
}
public void checkAvail(){
```

```
        Iterator<String> itr = popu.iterator();
        String s;
        while(itr.hasNext()){
                s=itr. Next();
                double new_avail = avail(s);
                if (new_avail>best_avail){
                        best_avail=new_avail;
                         combo=s;
                }
        }
        System.out.println (best_avail);
        System.out.println (combo);
}
```

The array ***p[]*** consists of the site probabilities. The ***popu*** data structure contains the total population of the various voting configuration assignment to sites. The method ***avail ()*** checks the availability of the given configuration and stores best configuration in the ***best_avail*** variable. The method ***mutate ()*** performs the mutation operation for the genetic approach. The method ***crossover ()*** perform the crossover operation for the genetic approach.

Chapter 5

EXPERIMENTAL RESULTS

5. Experimental Results

5.1 Experimental results of Algorithm

We implemented the algorithm by coding it in java and calculating the availability by varying the no. of copies and site reliabilities. We then compared our values with the randomized algorithm [7] and plotted the table as below.

Table 1 Comparison between the Genetic algorithm and the Randomized algorithm for 5 no. of copies

# of copies	**Site Reliabilities**	**genetic**	**random**
5	0.8,0.8,0.8,0.8,0.9	0.99984	0.99855
5	0.8,0.8,0.8,0.9,0.9	0.99992	0.99855
5	0.98,0.94,0.90,0.85,0.80	0.99999	0.99996
5	0.90,0.85,0.80,0.75,0.70	0.99977	0.99857
5	0.74,0.68,0.62,0.58,0.54	0.99389	0.97828
5	0.97,0.90,0.81,0.73,0.65	0.99994	0.99970
5	0.96,0.94,0.90,0.68,0.60	0.99996	0.99985
5	0.97,0.96,0.94,0.93,0.90	0.99999	0.99998

Table 2 Comparison between the Genetic algorithm and the Randomized algorithm for 6 no. of copies

# of copies	**Site Reliabilities**	**genetic**	**random**
6	0.50,0.60,0.60,0.80,0.80,0.70	0.99903	.99070
6	0.95,0.93,0.90,0.85,0.80,0.78	0.99999	.99999
6	0.68,0.67,0.64,0.63,0.62,0.58	0.99775	.99775
6	0.97,0.87,0.79,0.73,0.68,0.60	0.99997	.99997
6	0.98,0.97,0.95,0.92,0.92,0.90	0.99999	.99999

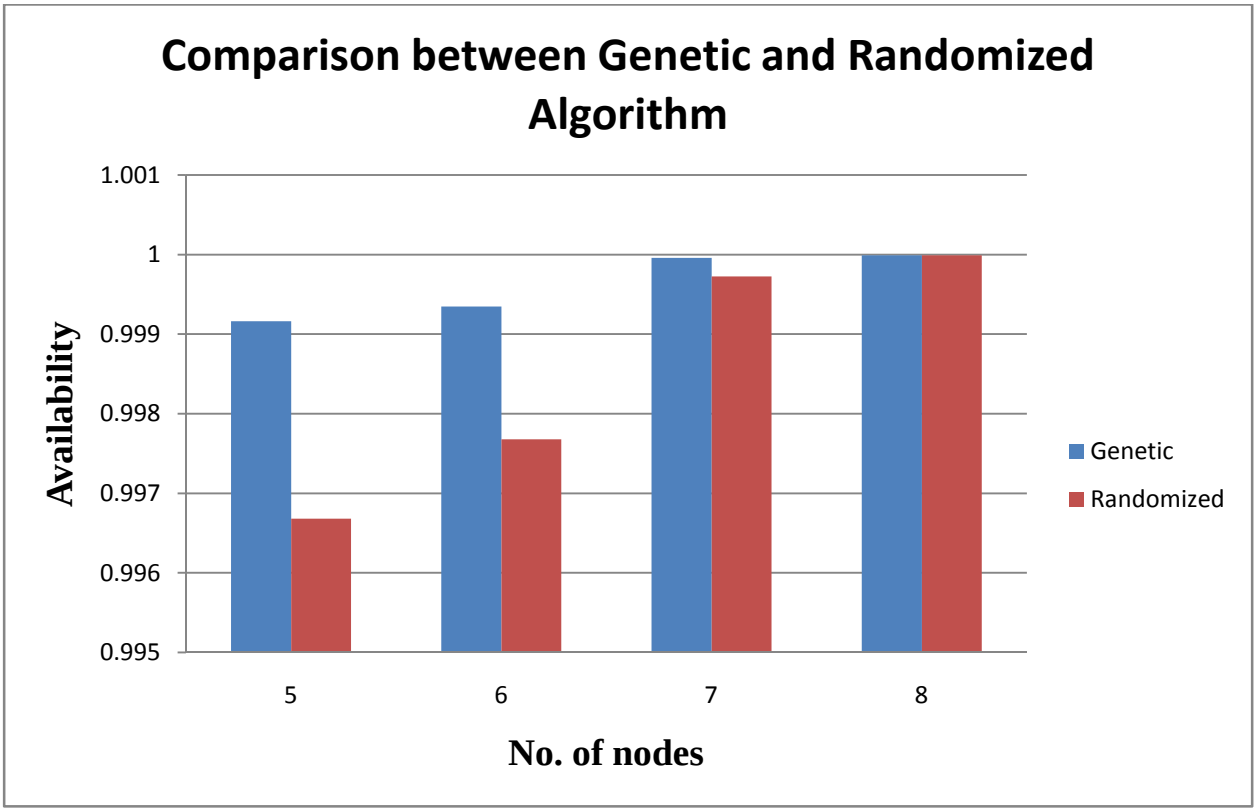
Table 3 Comparison between the Genetic algorithm and the Randomized algorithm for 7 no. of copies

# of copies	**Site Reliabilities**	**genetic**	**random**
7	0.97,0.90,0.85,0.70,0.68,0.65,0.60	0.99999	0.99994
7	0.89,0. 86,0.80,0.75,0.70,0.65,0.57	0.99996	0.99966
7	0.89,0.89,0.87,0.70,0.70,0.64,0.57	0.99997	0.99980
7	0.95,0.90,0.85,0.80,0.75,0.70,0.65	0.99999	0.99995
7	0.90,0.90,0.60,0.60,0.60,0.60,0.60	0.99989	0.99928

Table 4 Comparison between the Genetic algorithm and the Randomized algorithm for 8 no. of copies

# of copies	Site Reliabilities	genetic	random
8	0.97,0.90,0.85,0.70,0.68,0.65,0.60,0.60	0.99999	0.99999
8	0.90,0.90,0.85,0.85,0.80,0.80,0.70,0.70	0.99999	0.99999
8	0.95,0.90,0.85,0.80,0.75,0.75,0.70,0.70	0.99999	0.99999

Graph 10



5.2 Experimental Results of Simulation

Based on the simulation of different topology like star, ring, group topology we found the maximum throughput in each of the case and plotted the table.

Table 5 Maximum Throughput

Topology	TCP	CBR	CBR Receive
Star Topology	$1.02*10^7$	$4.25*10^4$	$3.4*10^4$
Ring Topology	$3.5*10^6$	$4.25*10^4$	$3.4*10^4$
Group Topology	$1.15*10^7$	$4.25*10^4$	$3.4*10^4$

Chapter 6

CONCLUSION

6. CONCLUSION

From the table 5 we conclude that in using TCP packets star topology shows the maximum throughput in comparison to ring topology .The maximum throughput in ring topology is $3.5* 10^6$ bits/sec where as the maximum throughput in star topology is $1.02*10^7$ bits/sec . Also we observed that using CBR packets the throughput is almost of equal value in all topologies. The group scheme has the maximum throughput of all the topologies due to its less overhead of packet transferring to its neighborhood. The group topology has $1.15*10^7$ ^{bits}/sec as maximum throughput among the various nodes in operation.

The optimal assignment of votes to sites so as to maximize overall availability is an important issue. From table 1-4 we found that a miniscule 1% increase in availability from 0.98 to 0.99 is quite large in terms of system availability. On the other hand, it can also be viewed as a decrease in the probability of the system being inaccessible from 0.02 to 0.01, reflecting a 50% decrease in down time. Viewed in this manner, the increase in availability from 0.98 to 0.99 is a dramatic improvement. Hence, even small increases in availability are useful. In this paper we described and tested a genetic algorithm for vote assignment. The algorithm runs very fast, and extensive comparisons with a random algorithm show that its performance is excellent. Although testing was restricted to 9 sites, this approach looks very promising even for a larger number of sites.

Because it runs fast, a Genetic vote assignment algorithm like the one described here would make it possible to dynamically change the assignment of votes to sites as the network changes, rather than maintaining a certain fixed assignment.

REFERENCES

[1] Davidson, S., Garcia-Molina, H., and Skeen, D. , "Consistency in partitioned networks.," ACM Computing. Survey. 17, 3 (Sept. 1985), 341-370.

[2] Gifford, D. K., " Weighted voting for replicated data.," In Proceedings of the Seventh Symposium on Operating Systems Principles (Pacific Grove, Calif., Dec. 1979). ACM, New York, 1979, pp.150-162..

[3] Garcia-Molina, H. "Reliability issues for fully replicated distributed databases." IEEE Comput.15,9 (Sept. 1982), 34-42.

[4] Barbara, D., Garcia-Molina, H., and Spauster, A., "Policies for dynamic vote reassignment," *Proc. IEEE Conf. on Distributed Computing*, pp. 37-44, 1986

[5] Jajodia, S., and Mutchler, D. "Integrating static and dynamic voting protocols to enhance file availability." In Proceedings of the Fourth International Conference on Data Engineering (Los Angeles, Feb. 1988). IEEE, New York, 1988, pp. 144-153.

[6] Agarwal, G., and Jalote, P., " An Efficient Protocol for Voting in Distributed Systems," Proceedings of the 12th International Conference on Distributed Computing Systems, June 1992, pp. 640-647

[7] Kumar, A., "A Randomized Voting Algorithm," Proceedings of the 11th International Conference on Distributed Computing Systems, May 1991, pp. 412-419

[8] Christian, F., “Understanding Fault-Tolerant Distributed Systems,” Communications of the ACM, vol. 34, no. 1, Jan. 1989, pp. 93-97

[9] Singhal, M., and Shivaratri, N. G., “Advanced Concepts in Operating Systems,” Tata McGraw-Hill 2001, New York, pp. 330-368

[10] Gray, J. N., “Notes on Data Base Operating Systems,” Operating Systems An Advanced Course, Springer-Verlag, 1979, New York, pp. 393-481

[11] Skeen, D., “Non Blocking Commit Protocols,” Proceedings of the ACM SIGMOD International Conference on Management of Data, 1981, pp. 133-142

[12] Skeen, D., “A Formal Model of Crash Recovery in a Distributed System,” IEEE Transactions on Software Engineering, vol. 9, no.3, May 1983, pp. 219-228

[13] Jajodia, S., and Mutchler, D., “Dynamic voting,” In Proceedings of the ACM SIGMOD International Conference on Management of Data (May 1987). ACM, New York, 1987, pp. 227-238.

[14] Jajodia, S., and Mutchler, D., “Enhancements to the voting algorithm,” Proc. 19th Int’l. Conf. on Very Large Data Bases, pp. 399-406**,** September 1987

[15] Barbara, D., Garcia-Molina, H., and Spauster, A., “Protocols for dynamic vote reassignment,” Proc. 5th ACM Symp. On Principles of Distributed Computing, pp. 195-205, 1986

[16] Barbara, D., Garcia-Molina, H., and Spauster,A., “ Increasing Availability Under Mutual Exclusion Constraints with Dynamic Vote Reassignment.,”ACM Transactions on Computer Systems,vol.7,no. 4, Nov. 1989,pp. 394-426

[17].Garcia Molina and D.Barbara, “How to assign votes in a Distributed System”, Journal of the ACM**,** vo1.32, no. 4**,** pp. 841-860**,** October 1985

[18] http://en.wikipedia.org/