P-1209

HTTP SCHEDULER FOR SCALABLE WEB SERVERS

PROJECT REPORT

Submitted in partial fulfillment of the
requirement for the award of the degree of
Bachelor of Engineering in Computer Science and Engineering
Of **Bharathiar University**, Coimbatore.

Submitted by

Ramanasundaram. R
0027K0195

Vanitha. R
0027K0208

Ramkumar. V
0027K1119

Under the guidance of

Ms. P. Sudha, B.E, MISTE, MCSI.
Lecturer, IT.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY,
COIMBATORE – 641006.

MARCH - 2004.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY
(Affiliated to Bharathiar University, Coimbatore)

CERTIFICATE

This is to certify that the project entitled

HTTP SCHEDULER FOR SCALABLE WEB SERVERS

is done by

Ramanasundaram. R	**Vanitha. R**	**Ramkumar. V**
0027K0195	**0027K0208**	**0027K1119**

and submitted in partial fulfillment of the requirement for the award of the degree of **Bachelor of Engineering in Computer Science and Engineering** Of **Bharathiar University**, Coimbatore, during the academic year 2003-2004.

[illegible]

Professor & Head of the department
(Dr.S.THANGASAMY)

[illegible]

Guide
(Ms. P. SUDHA)

Certified that the candidates were examined by us in the project work viva -voce examination held on 23.3.2004.

[illegible] 23/3/'04
Internal Examiner

[illegible] 23/3/04
External Examiner

DECLARATION

We, **Ramanasundaram.R, Vanitha.R, Ramkumar.V**, here by declare that the project entitled **"HTTP scheduler for scalable web servers",** submitted to Kumaraguru College of Technology, Coimbatore. (Affiliated to Bharathiar University) is a record of original work done by us under the supervision and guidance of Ms. P.Sudha B.E., Lecturer, Department of Information Technology.

NAME	**REGISTRATION NUMBER**	**SIGNATURE**
Ramanasundaram.R	0027K0195	[illegible]
Vanitha.R	0027K0208	[illegible]
Ramkumar.V	0027K1119	[illegible]

Countersigned by Staff- in- charge ,

[illegible]

Ms. P.Sudha B.E.

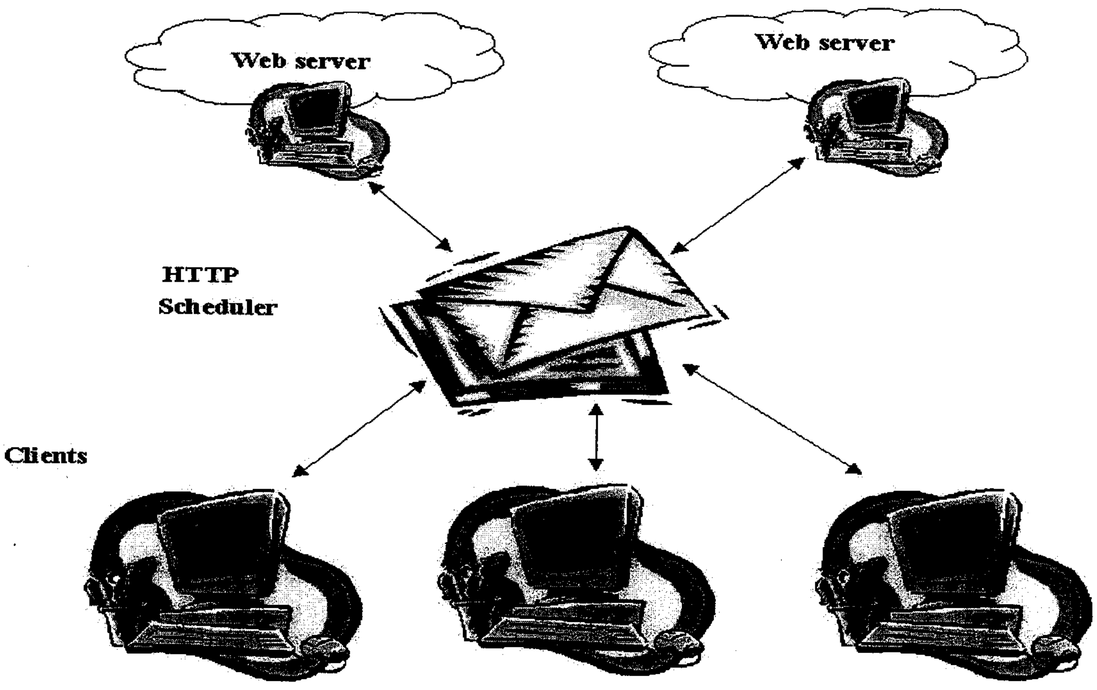
Lecturer,

Department of Information Technology,

Kumaraguru College of Technology.

Place : Coimbatore

Date : 19. 3. 04



HTTP SCHEDULER FOR SCALABLE WEB SERVERS

DEDICATED TO OUR
BELOVED PARENTS
AND FRIENDS

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

The exhilaration achieved upon the successful completion of any task should be definitely shared with the people behind the venture. This project is an amalgam of study and experience of many people without whose help this project would not have taken shape.

At the onset, we take this opportunity to thank the management of our college for having provided us excellent facilities to work with. We express our deep gratitude to our Principal **Dr.K.K.Padmanabhan B.Sc(Engg), M.Tech., Ph.D.**, for ushering us in the path of triumph.

We are always thankful to our beloved Professor and the Head of the Department, **Dr.S.Thangasamy, B.E.(Hons), Ph.D**, whose consistent support and enthusiastic involvement helped us a great deal.

We are greatly indebted to our beloved guide **Ms.P.Sudha,B.E.**, Lecturer, Department of Information Technology, for her excellent guidance and timely support during the course of this project. As a token of our esteem and gratitude, we honor her, for her assistance towards this cause.

We also thank our project coordinator **Mrs.S.Chandrakala M.E.**,Senior Lecturer, Departement of Computer Science and Engineering and our beloved class advisor **Mrs.M.S.Hema B.E.**, Lecturer, Department of Computer Science and Engineering, for their invaluable assistance.

We feel proud to pay our respectful thanks to our Parents for their enthusiasm and encouragement and also we thank our friends who have associated themselves to bring out this project successfully.

SYNOPSIS

SYNOPSIS

In Web, requests from the clients may be delayed or failed when requests made are more than the server capability. So there is a need to upgrade the server configuration or else we can scale the web server. In scalable web servers there is need to schedule the HTTP requests among the servers available.

Our project, a system side application, intends to perform the redirection of the HTTP requests to the servers effectively, such that the response is immediate. Also it does the performance evaluation of various algorithms in processing the HTTP requests.

The main advantage of the project is reduction in time consumption and memory usage in processing the HTTP client requests.

Our project features

- Dynamically we can add and remove servers without any performance degradation.
- We are using threads per client concept instead of process per client, so that the memory and time consumption in processing the requests is minimized.
- Preloaded threads are used here, so that time consumed (in creating threads) is minimized.

Scheduler and client applications are implemented in Linux. The server can be used in any platform, more preferably Linux because of its high security.

INDEX

INDEX

INTRODUCTION

INTRODUCTION

1.1 Existing System and Limitations

Whenever there is HTTP load increase the web server must be upgraded or scaled. Upgradation is a very costly and time consuming option. So the only other option is scaling(load distribution). The current system uses only a particular method in distributing the HTTP requests over the available servers.

- ◆ It is efficient only in particular circumstances and in most other cases the response is very slow.
- ◆ There is no option for dynamically adding the servers .

1.2 Proposed System and Advantages

Our intention is to develop a HTTP scheduler for web servers that can be scaled. It acts as a facilitator between the clients and the servers. It has the responsibility to redirect the clients HTTP request to the server and also send the server's response back to the client. In order to be efficient in all circumstances we will be using four different methods for redirecting the request.

- ◆ Based upon the circumstances we can use any one method effectively.
- ◆ Dynamic addition/deletion of servers.
- ◆ Reduction in time consumption and memory usage.
- ◆ Provision of user friendly interface for configuring scheduler.
- ◆ We provide a performance evaluation, to judge the effective algorithm in the given circumstance.

SYSTEM REQUIREMENT AND ANALYSIS

SYSTEM REQUIREMENT AND ANALYSIS

System study is an activity that encompasses most of the tasks that we have collectively called computer system engineering. System study is conducted with the following objectives.

- Identify the needs.
- Evaluate the system concept for feasibility.
- Perform economic and technical analysis.
- Allocate function to hardware, software, people and other system elements.
- Create a system definition that forms a foundation for all subsequent engineering works.

2.1 Product definition

Scalable web servers can be built using a network of workstations where server capacity can be extended by adding new work stations as the workload increases. Our project is a comparison of different methods to do load balancing of HTTP traffic for scalable web servers.

We present a classification framework for the different load balancing methods and compare their performance. So the main catch is a clustered server may also provide better reliability than a upgraded server by use of appropriate load sharing algorithms that can facilitate fault resilience with graceful degradation of performance as machines leave the cluster due to failure or preventive maintenance. A clustered server also makes it possible to add new machines without interrupting service.

The HTTP protocol is stateless, such that a TCP connection must be setup for every object on a web page. Thus each request can be rooted independently. This HTTP property can be used to achieve load sharing in a cluster by directing requests in that one logical server to different physical server with identical content.

Since web objects are seldom referenced , replication is a valid strategy only for the most popular objects . Another alternative is load sharing through file or object placement on different machines.

2.2 Project Plan

In the analysis phase, the use of protocols like TCP, differentiation of modules, user interface and certain other aspects are identified and ambiguity are eliminated.

From the results of analysis phase, the design phase is commenced where the complete system is depicted in the form of a even flow diagram.

Finally, during the implementation phase, coding is done for each module and they are integrated to function smoothly.

Then testing is done to confirm the required criteria and validity of the product is confirmed.

SOFTWARE REQUIREMENT SPECIFICATION

SOFTWARE REQUIREMENT SPECIFICATION

3.1 Purpose

In scalable web servers , the HTTP requests from the clients has to be distributed in a efficient manner. Otherwise it results in delay or even failure of the request. So we intend to redirect the clients requests to the servers amicably.

3.2 Scope

We are using four different methods in the redirection of the client requests. So that our project can be configured dynamically to the various circumstances. We also presented a performance of each methods in various circumstances.

3.3 Product Overview and Summary

Our product provides a reliable, robust and efficient means of load distribution of HTTP client requests.Our product basically deals with the usage of four different algorithms for minimizing the time consumption and memory usage. The product is developed keeping in mind that it is user friendly and easy to use.

At first, the scheduler module is invoked, the servers are configured and then the algorithm is chosen and now the preloaded threads are started and the scheduler starts listening at the port address 8080. The client program is executed with and it sends the requested URL to the scheduler.

Then the scheduler receives the URL and routes the request to the server based on the algorithms chosen. The server responds with the requested page and send it to the scheduler which in-turn send it to the client.

The time taken between the request and response of the page is calculated dynamically and stored in result module. The result analysis page stored in "result.c" displays the time requirement for each algorithm. From this we can infer which one is the best algorithm in that circumstance.

3.4 Development and Operating Environment

The development environment gives the minimum hardware and software requirements.

Hardware Specification

◆ Processor	Pentium III
◆ RAM	64 MB
◆ Cache	128KB
◆ Hard Disk	10 GB
◆ Floppy Drive	1.44 FDD
◆ Monitor	14" Monitor

Software Specification

◆ Operating System	Red Hat Linux 8.0
◆ Language	Advanced C
◆ Server	IIS or PWS

3.5 Functional Specificaiton

Description of the modules are as follows.

1. Client and time calculation

A socket program is written to create a socket and to establishes the connection with the scheduler's IP address. The request is then send to the scheduler through appropriate system calls. Also the timer is automatically invoked when the request is send and it keep running till the response is received at the client side. Then the timing statistics are stored in the "**result**" file dynamically. By executing the "**result.c**" we can get the time taken for each algorithm.

2. HTTP scheduler

This is the main module which takes care of routing the client requests to the server based on the scheduler configuration. It creates the preloaded threads and waiting for the clents request at port 8080. When it receives the request it finds appropriate server by using the algorithm chosen and then it sends the request to the chosen sever. When it got the response from the server it sends it to the client. Thus it integrates all the algorithms , GUI interface and the server.

3. Algorithm Implementation

Round Robin:

It distributes request to the different web servers in a round robin manner independent of the load on each web server. This scheme is very simple to implement but can, of course overloaded web server if the sequence of the request is non optimal. It transparently return the IP address of the available web servers in a round robin manner.

Random:

It generates the IP address of the available servers randomly using the rand() function as below.

(Number of available servers * rand()) / (RAND_MAX +1.0)

Active Connections:

Scheduler keeps track of the number of active connections to each server and always directs a new connection to the server with least connections. If two or more servers have the same number of active connections the load balancer will choose the server with the lowest server identifier. The consequence is that first server in the order will always be chosen when a new connection arrives in an empty system. If a new connection always arrives in a empty system the same web server will be used.

Priority:

In this policy , each and every is allotted a specific weight. Depending upon the weightage the usage of the server is calculated and the server with most free usage is assigned the request.

Server = Maximum(free usage among servers)

Priority Table:

Suppose we have two servers with weight assigned as 2 and 1.

Server's Usage (/100)	Server with weight :1	Server with weight : 2
Initial	0	0
1st request	0	50
2nd request	100	50
3rd request	100	100

When the usage of all the servers is 100% , then it is reset to 0 and continue as before.

4.GUI module

It is an interface created to scheduler using GTK programming. It consists of three leaves.

- **Scheduler Configuration**

  Here we specify the scheduler IP address, port number and the number of threads to be created..

- **Algorithm Selection**

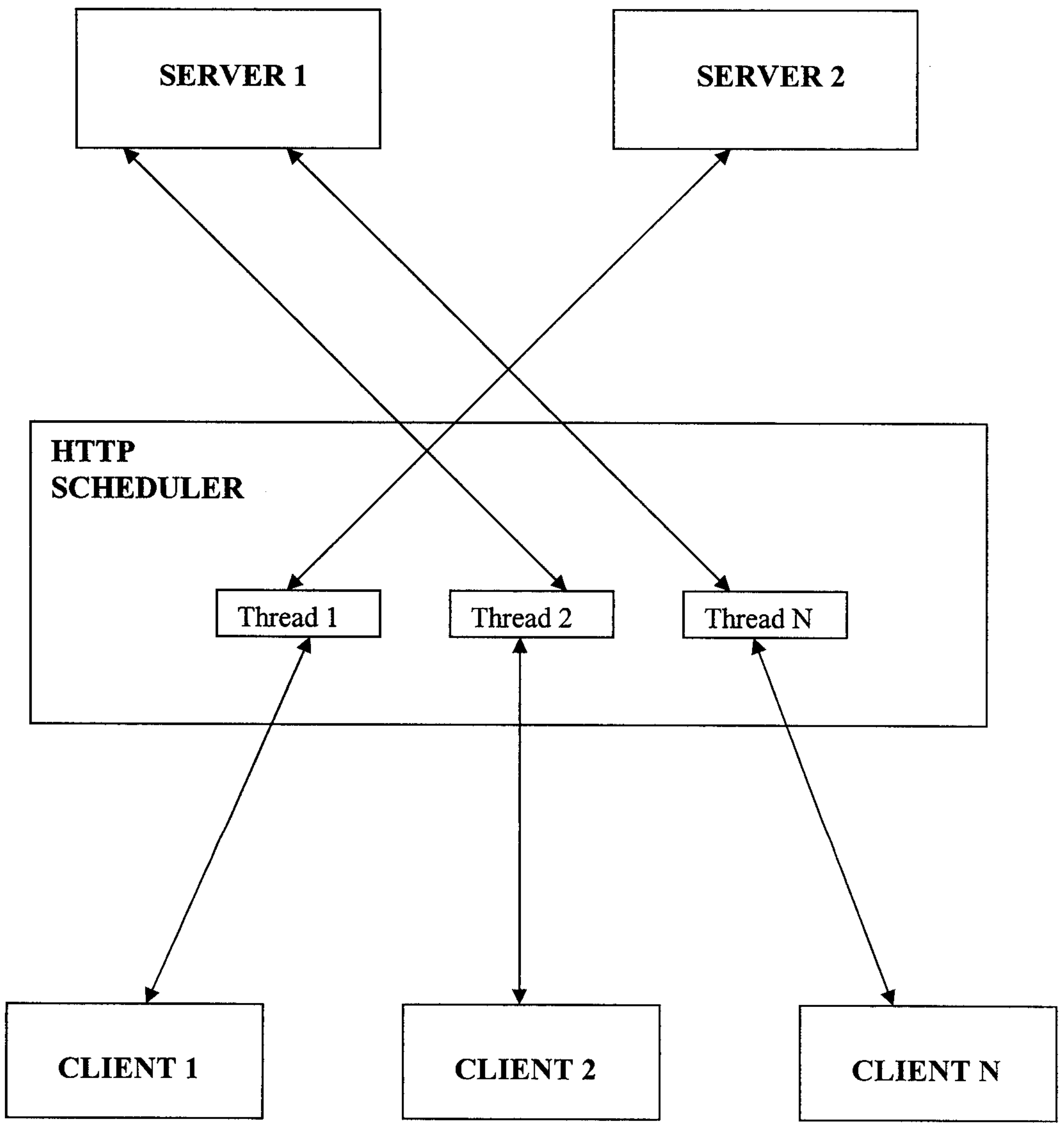
  Here we need to select the algorithm which is represented using radio buttons.

- **Server Configuration**

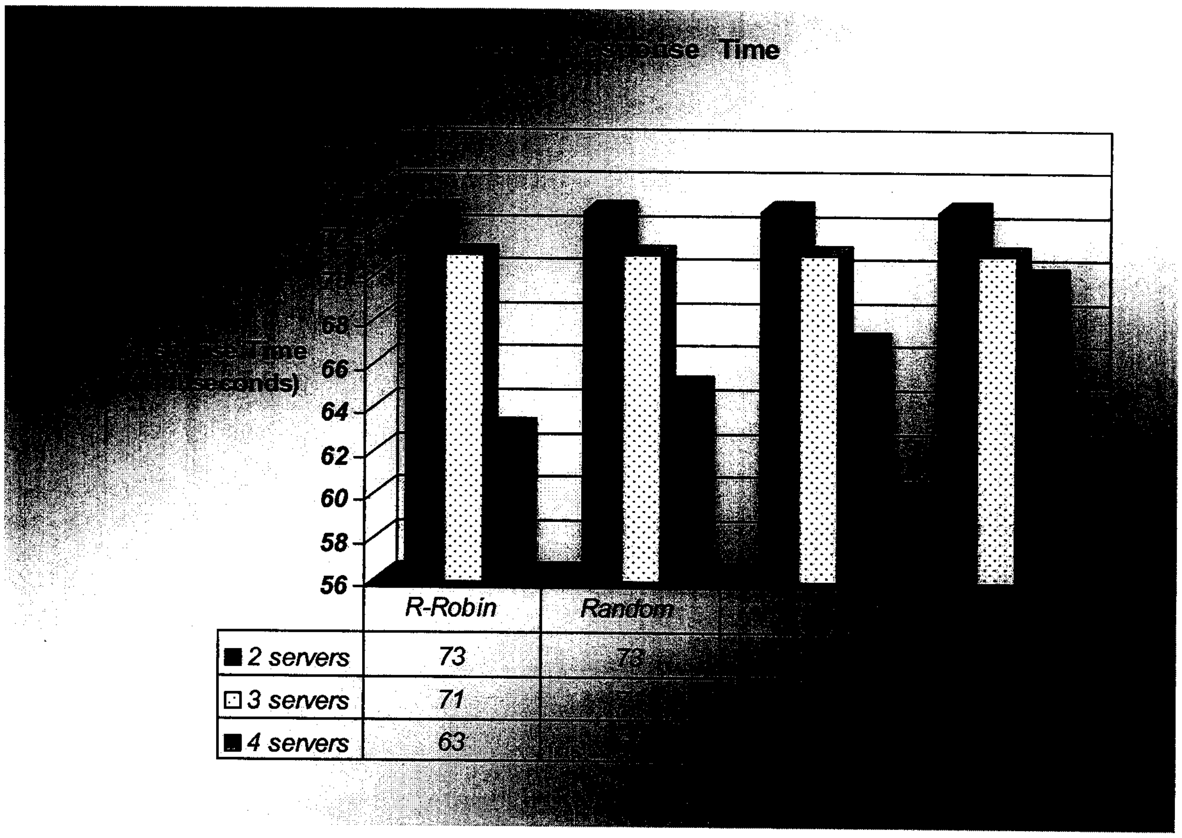
  Here we can dynamically add, delete and update the web servers configuration by specifying its IP address, port number.

3.6 Architectural Diagram

Overview Diagram:



RESULT GRAPH:



	R-Robin	Random	[illegible]	[illegible]
■ 2 servers	73	73	[illegible]	[illegible]
□ 3 servers	71	[illegible]	[illegible]	[illegible]
■ 4 servers	63	[illegible]	[illegible]	[illegible]

SYSTEM DESIGN

SYSTEM DESIGN

The system design is the high level strategy for solving the problem and building a solution. System design includes decisions about the organisation of the system into subsystems, allocation of subsystems to hardware and software components, and major conceptual and policy decisions that form the framework for the detailed design.

Architectural Framework

In retrieving an object from the web, a canonical name must be mapped to an IP address , MAC addresss and an object locator. Each mapping offers an opportunity to redirect the request to the most appropriate machine from a load balancing viewpoint. Therefore, load balancing architechtures are best classified according to where redirection is used: That is at the client , at the server or in the network.

Remapping in the Client

The remapping of addresses at the client side can be classified into two groups: those transparent to the client and those that are not transparent. The later group requires changes to client software. Netscape's extension to the browser client for load balancing of access to Netscape's own server is one example.

However, the smart client approach increases network traffic by frequent polling, and is vulnerable to delays between polling and sending the request.

The DNS system, provides a distributed database for mapping between canonical name and IP address. Each naming domain maintains its own local address and host name information.

A domains name servers are queried by **"resolvers"** in any end system for the mapping between C names within the domain and IP address. These name servers can transparently return the IP address of the available web servers in the list in round robin manner.

However, due to the catching strategies with the configurable time to live TTL used throughout the internet in the DNS , performance is influenced by the spatial and temporal distribution of access. Request from end systems in the same domain will be directed to the same destination since the remote name server will cache the canonical name to IP address mapping. This mapping is reported by the rotating name server at the first request, and is cached by name server in the clients local domain. The result may be skewed load on a server using the rotating name server method if many clients use the same name server.

Remapping in the Server

This method uses a modified workserver as a proxy server for the replicated servers. The proxy will not process the HTTP requests, but passes it on to the most suitable machine among the replicated servers. The SWEB project at UCSB is based on HTTP redirection at the server in conjunction with detailed server load estimates. The draw back of this type of solution is that remapping is done at the application level , and the request must transverse the full protocol stack four times before the request can be processed.

Thus, this solution is only interesting when time to process the request is the limiting factor. In most cases the proxy server will be a bottleneck and the solution does not scale.

Remapping in the Network

In this type remapping can be done at two levels. One at the network layer itself or between the network and link layers. The network element performing the remapping is called HTTP scheduler. Apart from this special mapping, operations in the HTTP scheduler are similar to the normal routing and forwarding operations.

In the first alternative each replicated server has a unique IP address. All packets destined for a logical server are inspected , and the destination address is replaced with the address of the replicated server with the lowest load. The remapping network element (HTTP scheduler) must monitor all of the IP traffic and send all incoming IP traffic of a particular HTTP request to the same web server. For this to work the HTTP scheduler must change the IP address of all IP packets in each direction. Also all packets in a HTTP transaction must be mapped to the same server. A TCP connection is identified by

<TCP,IP-src_addr,src_port,IP-dst_addr,dst_port>

The HTTP scheduler has to monitor the TCP connection from setup to tear down. To find out when a connection that is terminated incorrectly can be removed from the connection database, the HTTP scheduler has to mimic the TCP state machine including the TCP timeout values .

In other methods the IP address replacement requires that access to the web cluster be done through the HTTP scheduler, increasing the networking capacity.

In the second alternative , remapping is done between the network and link layers, and all replicated servers have the same IP address as the logical server. With an Asynchronous Transfer Mode(ATM) or Label Switched System (LSS), each server is associated with a set of virtual channels or labels.

With link level redirection , the scheduler must hard core the address resolution protocol table to avoid issue in ARP request to the subnet. An alternative is to use layer 2 multicast to all servers in the cluster and use packet filtering to select the packets processed by individual nodes. The advantage is less overhead in the remapping network element at the expense of higher processing cost due to packet filtering and synchronization between the filtering processes.

SYSTEM TESTING

SYSTEM TESTING

Testing is an activity to verify that a correct system is being built and is performed with the intent of finding faults in the system. Testing is an activity, however not restricted to being performed after the development phase is complete. But this is to be carried out in parallel with all stages of system development, starting with requirement specification. Testing results once gathered and evaluated, provide a qualitative indication of software quality and reliability and serve as a basis for design modification if required.

System Testing is a process of checking whether the development system is working according to the original objectives and requirements. The system should be tested experimentally with test data so as to ensure that the system works according to the required specification. When the system is found working, test it with actual data and check performance.

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. The increasing visibility of software as a system element and the attendant "cost" associated with a software failure are the motivation forces for a well planned, thorough testing.

5.1 Testing Objectives

The testing objectives are summarized in the following three steps. Testing is the process of executing a program with the intent of finding an error. A good test case is one that has high probability of finding an error. A successful test is one that uncovers as–yet-undiscovered errors.

5.2 Testing Principles

All tests should be traceable to customer requirements. Tests should be planned long before testing begins, that is, the test planning can begin as soon as the requirements model is complete. Testing should begin "in the small" and progress towards resting "in large". The focus of testing will shift progressively from programs to individual modules and finally to the entire project. Exhaustive testing is not possible. To be more effective, testing should be one, which has highest probability of finding errors.

The following are the attributes of good tests:

- A good test has a high probability of finding an error.
- A good test is not redundant.
- A good test should be "best of breed"
- A good test should be neither too simple nor too complex.

5.3 Levels of Testing:

The details of the software functionality tests are given below. The testing procedure that has been used is as follow:

- Unit Testing
- Integration Testing
- Validation Testing

- Output Testing

Unit Testing

Unit testing is carried out to verify and uncover errors within the boundary of the smallest unit or a module. In this testing step, each module was found to be working satisfactory as per the expected output of the module. In the package development, each module is tested separately after it has been completed and checked with valid data. Unit testing exercise specific paths in the modules control structure to ensure complete coverage and maximum error detection.

The project is divided into four modules. These three modules are developed separately and verified whether they function properly.

Integration Testing

Integration Testing address the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of higher-order tests are conducted. The main objective in this testing process is to take unit-tested modules and build a program structure that has been dictated by design.

The following are the types of integrated testing:

Top Down Integration:

This method is an incremental approach to the construction of the program structure. Modules are integrated by moving downward the control hierarchy, beginning with the main program module. The module sub-ordinates to the main program module are incorporated to the structure in either a depth-first or breadth-first manner.

Bottom Up Integration:

This method begins the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to given level is always available and the need for stubs is eliminated. The bottom up integration strategy may be implemented with the following steps:

The low level modules are combined in to clusters that perform a specific software sub-function. A driver i.e. the control program for testing is return to coordinate test case input and output. The cluster is tested and drives are removed and clusters are combined moving up ward in the program structure.

The four modules which are developed during unit testing are combined together and they are executed and verified whether the linking of the files and the corresponding modules without any error.

Validation Testing

At the end of integration testing, software is completely assembled as a package, interfacing errors have been uncovered and correction testing begins.

Validation Test Criteria:

Software testing and validation is achieved through series of black box tests that demonstrate conformity with the requirements are achieved, documentation is correct and other requirement are met. Here the four modules which are checked in the integration testing are assembled and programmed in the code is testing for any correction.

Output Testing:

Output testing is series of different test whose primary purpose is to fully exercise the computer based system. Although each test has a different purpose, all the work should be verified so that all system element have properly integrated and perform allocated functions.

Output testing is the stage of implantation, which is aimed at ensuring that the system works accurately and efficiently before live operation commences. The input screens, output documents were checked and required modification made to suite the program specification. Then using rest data prepared, the whole system was tested and found to be a successful one.

Here giving checks the whole system sample inputs i.e. the predefined no of threads in the scheduler and clients. It is checked for the whole function so that the system functions well and the requirements are met.

FUTURE ENHANCEMENTS

FUTURE ENHANCEMENTS

- The TCP/IP protocol stack can be changed such that the server sends it response directly to the client not via the scheduler.
- Extending the application to other operating system such as WINDOWS, MAC, SOLARIS.
- Performance evaluation of each and every algorithm can be shown in graphical form.
- Providing network security using encryption and decryption techniques between client and scheduler, and between scheduler and webserver.
- Dynamically detect and recover the server from overload.

CONCLUSION

CONCLUSION

We have built a robust and effective system by which the four algorithms can be easily evaluated in various circumstances or situations and they can be suitably compared inorder to determine which one is best suited for that situation.

According to our testing round robin algorithm has proved itself to be reliable in most of the circumstances. Notably efficiency, speed, transmitted bytes etc. Also we have created a unique user friendly interface using GTK programming.

We can add, delete or update whatever number of servers we want to determine the end result. Above all the use of Linux platform has provided full security and speed without frequent congestions.

BIBLIOGRAPHY

BIBLIOGRAPHY

- ◆ Richard Stevens, "UNIX Network Programming ", Addison Wesley Longman, 1996, Second Edition.
- ◆ Richard Stevens, "UNIX Advanced Programming",Addison Wesley, 1995, First Edition.
- ◆ Michael K. Johnson and Eric W.Troan, "Linux Application Development",Addison Wesley,2000,First Edition.
- ◆ www.ieee.org

APPENDIX

APPENDIX

9.1 SAMPLE CODE

Client. C:

```
#include <stdio.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <string.h>
#include <sys/timeb.h>
#include "result.h"
#define NOT  5
int socketfd;
struct sockaddr_in serveraddr;
pthread_t  tid[10];
void thread_make(int);
void * thread_main(void *);
int main(int argv, char ** argc)
{
 int i;
 struct result scheduler_result;
 struct timeb  t_start;
 struct timeb  t_end;
 struct timeb t_total;
 FILE * fp_read, * fp_write;
 fp_read = fopen("result.rs","r");
 fread(&scheduler_result, sizeof(scheduler_result),1,fp_read);
 fclose(fp_read);
 ftime(& t_start);
 for(i=1;i<=NOT ;i++) {
   thread_make(i);
 }
```

```
printf("Main:waiting for thread termination");
fflush(stdout);
for(i=1;i<=NOT;i++)
{
    pthread_join(tid[i],NULL);
}
printf("\nMain:All Threads Ended..\n");
ftime(&t_end);
printf("\nStarting time:%ld:%d",t_start.time,t_start.millitm);
printf("\nEnding time:%ld:%d",t_end.time,t_end.millitm);
t_total.time = t_end.time - t_start.time;
if(t_end.millitm < t_start.millitm){
        t_total.time --;
        t_end.millitm += 1000;
}
t_total.millitm = t_end.millitm - t_start.millitm;
printf("\nTotal time:%ld:%d\n\n",t_total.time,t_total.millitm);
printf("\nIn Milli-Sec:%d\n\n",t_total.time*1000 + t_total.millitm);
printf("\nAll:%d\n\n",atoi(argc[1]));
switch ( atoi(argc[1]) )
{
        case 1: scheduler_result.round_robin = t_total.time*1000 +
                  t_total.millitm;
                    printf("Alg:ROund Robin\n\n");
                    break;
        case 2: scheduler_result.weight = t_total.time*1000 +
                  t_total.millitm;
                    printf("Alg:Weight\n\n");
                    break;
      case 3: scheduler_result.connections = t_total.time*1000 +
                  t_total.millitm;
                    printf("Alg:Connection\n\n");
                    break;
      case 4:  scheduler_result.random = t_total.time*1000 +
                  t_total.millitm;
```

```
                    printf("Alg:Random\n\n");
                    break;
                default: printf("\nUn supported algorithm\n");
 }
 fp_write = fopen("result.rs","w");
 fwrite(&scheduler_result,sizeof(scheduler_result),1, fp_write);
 fclose(fp_write);
 fflush(stdout);
}
void thread_make(int n) {
printf("\nCreating thread %d...",n);
pthread_create(&tid[n], NULL, &thread_main, (void *) n );
}
void * thread_main(void * arg)
{
 char buff[1024] = "GET /newwalk.html HTTP/1.0\r\n\r\n" ;
 int n= strlen(buff);
 printf("\nThread %d started...", (int)arg );
 fflush(stdout);
 socketfd = socket(AF_INET, SOCK_STREAM, 0 );
 if(socketfd ==-1 ) {
         printf("\nError creating socket");
         exit(0);
 }
 else
         printf("\nSocket created");
 bzero(& serveraddr, sizeof(serveraddr) );
 serveraddr.sin_family = AF_INET;
 serveraddr.sin_port = htons(8080);
 if( inet_pton(AF_INET, "192.168.1.6", &serveraddr.sin_addr) <=0 )
         printf("IP Addr is wrong");
 else
         printf("IP Addr is right");
if ( (connect(socketfd, (struct sockaddr * )&serveraddr, sizeof(serveraddr) ) ) == -1 )
         printf("\n%d:Connection failed",(int)arg);
printf("\n\n*************************************************
 *****\n);
fflush(stdout);
}
```

RESULT.H:

```
struct result
{
int round_robin;
int weight;
int connections;
int random;
};
```

ALGORITHMS :

ROUND ROBIN:

```
int roundRobin()
{
NextWorker ++;
if(NextWorker > NofWorkers ) {
            NextWorker = 1;
}
return( NextWorker - 1 );
}
```

RANDOM:

```
int random_algorithm()
{
float number;
int result;
number = (float)NofWorkers;
result = (int) (number * rand() /(RAND_MAX +1.0) );
        return(result);
}
```

PRIORITY:

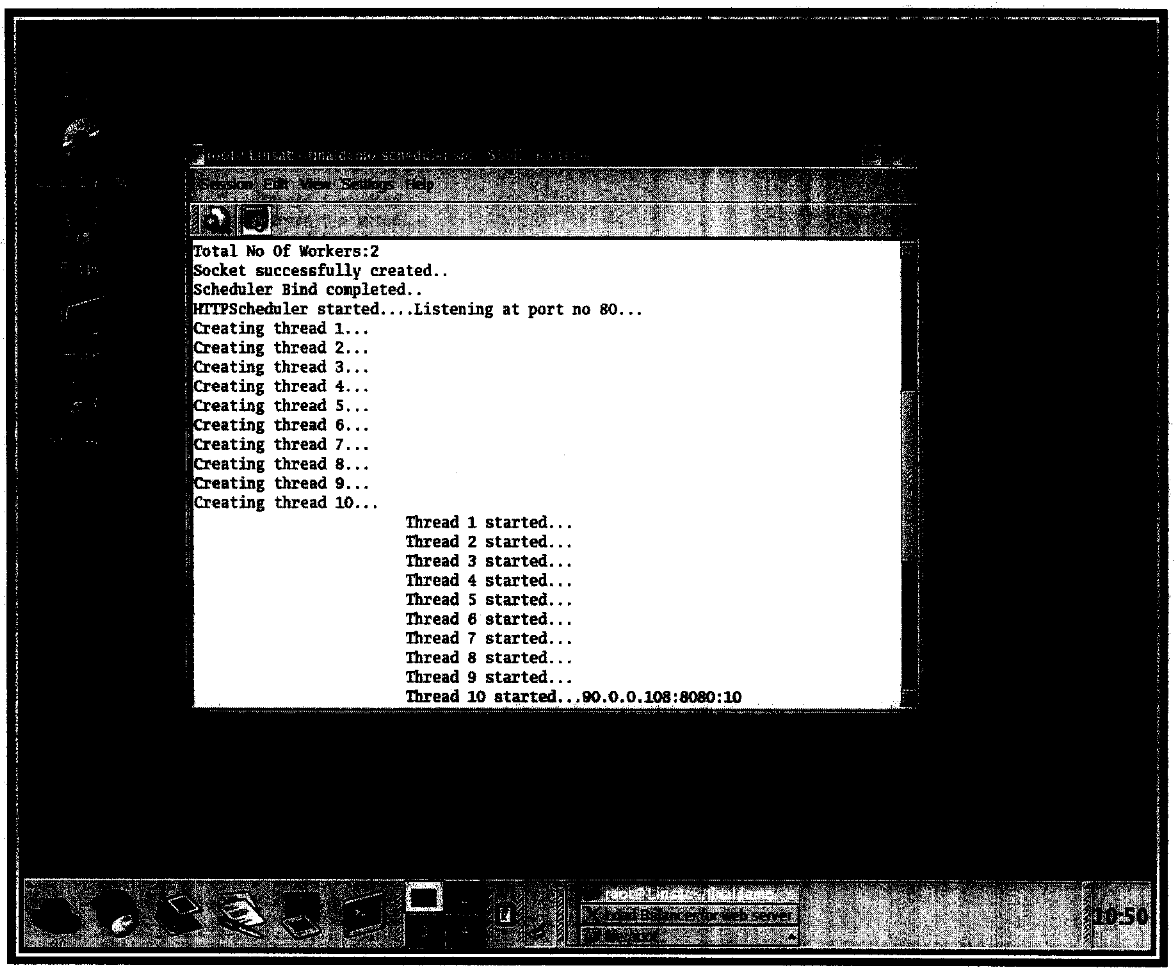
```
struct prirority_table
{
```

```
int  max;
int  served;
float usage;
int  index;
};
int  weight_algorithm( )
{
int i,j,id;
float min;
static  struct prirority_table ptable[10];
struct prirority_table temp;
static int pindex;
if(Priority_reset == 0)
{
            Priority_reset = 1;
            printf("\nEntering INIT\n");
            fflush(stdout);
for(i=0; i < NofWorkers; i++) {
            ptable[i].max = atoi(worker[i].weight);
            ptable[i].served = 0;
            ptable[i].usage = 0.0;
            ptable[i].index = i;
 }
pindex = NofWorkers -1;
for(i=0;i <= pindex ;i ++) {
      for(j= i+1; j<=pindex ;j++) {
            if(ptable[i].max < ptable[j].max) {
                     temp.max = ptable[i].max;
                     temp.served = ptable[i].served;
                     temp.usage = ptable[i].usage;
                     temp.index = ptable[i].index;
                     ptable[i].max = ptable[j].max;
                     ptable[i].served = ptable[j].served;
                     ptable[i].usage = ptable[j].usage;
                     ptable[i].index = ptable[j].index;
```

```
                        ptable[j].max = temp.max;
                        ptable[j].served = temp.served;
                        ptable[j].usage = temp.usage;
                        ptable[j].index = temp.index;
              }
      }
  }
}//if Priority_reset
// Find MIN
min = 100.0;
id = -1;
for(i=0;i <= pindex ;i++) {
  if(min > ptable[i].usage) {
    min = ptable[i].usage;
    id = i;
  }
}
if(id != -1 ) {
          ptable[id].served ++;
          ptable[id].usage = (ptable[id].served *100) /ptable[id].max;
          /***********del*/
 for(i=0;i<=pindex;i++) {
    printf("\nUsage: %f| Priority:%d",ptable[i].usage,ptable[i].max);
    printf("\nServed: %d",ptable[i].served);
 }
 /*****************************/
 return (ptable[id].index);
 }
  else {
          printf("\nReseting....\n");
          fflush(stdout);
 for(i=0; i <= pindex ;i ++) {
          ptable[i].served = 0;
          ptable[i].usage = 0.0;
 }
```

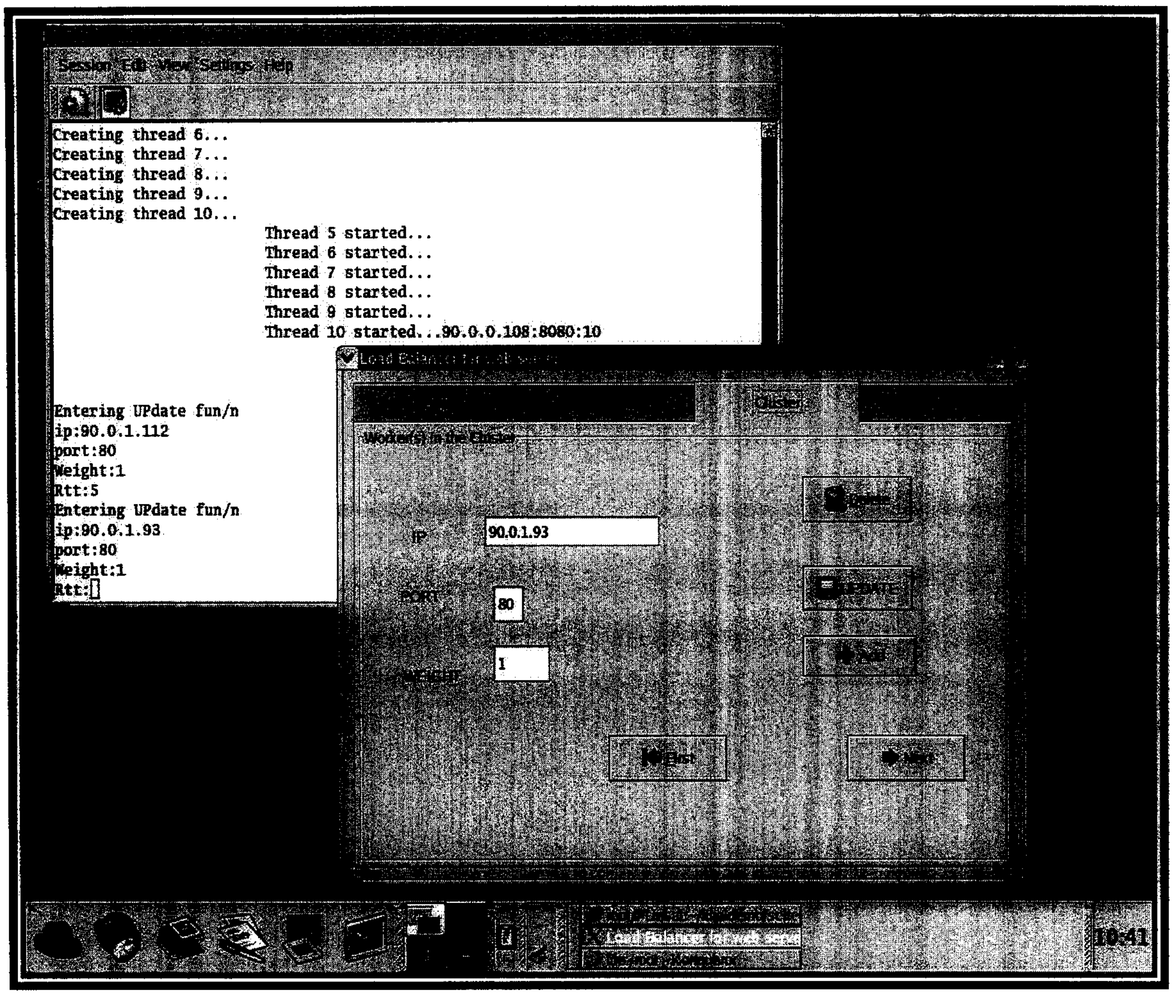
9.2 SAMPLE OUTPUT

THREAD CREATION:



```
Total No Of Workers:2
Socket successfully created..
Scheduler Bind completed..
HTTPScheduler started....Listening at port no 80...
Creating thread 1...
Creating thread 2...
Creating thread 3...
Creating thread 4...
Creating thread 5...
Creating thread 6...
Creating thread 7...
Creating thread 8...
Creating thread 9...
Creating thread 10...
                          Thread 1 started...
                          Thread 2 started...
                          Thread 3 started...
                          Thread 4 started...
                          Thread 5 started...
                          Thread 6 started...
                          Thread 7 started...
                          Thread 8 started...
                          Thread 9 started...
                          Thread 10 started...90.0.0.108:8080:10
```

SERVER CONFIGURATION:



```
Creating thread 6...
Creating thread 7...
Creating thread 8...
Creating thread 9...
Creating thread 10...
                        Thread 5 started...
                        Thread 6 started...
                        Thread 7 started...
                        Thread 8 started...
                        Thread 9 started...
                        Thread 10 started...90.0.0.108:8080:10



Entering UPdate fun/n
ip:90.0.1.112
port:80
Weight:1
Rtt:5
Entering UPdate fun/n
ip:90.0.1.93
port:80
Weight:1
Rtt:
```

Worker(s) in the Cluster

IP: 90.0.1.93

PORT: 80

WEIGHT: 1