Program – 10

AIM: To implement 2-Phase Commit client-server.

Introduction and Theory

In a local database system, for committing a transaction, the transaction manager has to only convey the decision to commit to the recovery manager. However, in a distributed system, the transaction manager should convey the decision to commit to all the servers in the various sites where the transaction is being executed and uniformly enforce the decision. When processing is complete at each site, it reaches the partially committed transaction state and waits for all other transactions to reach their partially committed states. When it receives the message that all the sites are ready to commit, it starts to commit. In a distributed system, either all sites commit or none of them does.

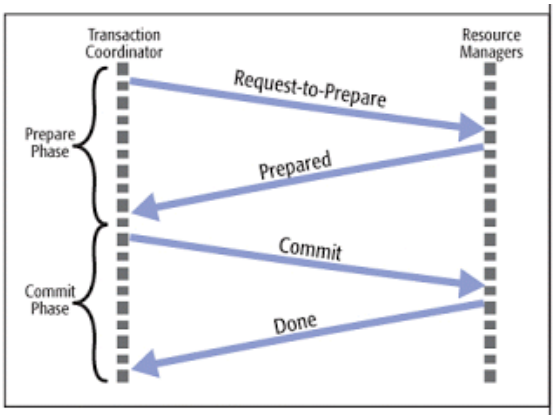

Figure 1 • The two-phase commit protocol

Distributed two-phase commit reduces the vulnerability of one-phase commit protocols. The steps performed in the two phases are as follows −

Phase 1: Prepare Phase

- After each slave has locally completed its transaction, it sends a “DONE” message to the controlling site. When the controlling site has received “DONE” message from all slaves, it sends a “Prepare” message to the slaves.
- The slaves vote on whether they still want to commit or not. If a slave wants to commit, it sends a “Ready” message.
- A slave that does not want to commit sends a “Not Ready” message. This may happen when the slave has conflicting concurrent transactions or there is a timeout.

Phase 2: Commit/Abort Phase

- After the controlling site has received “Ready” message from all the slaves −
  - The controlling site sends a “Global Commit” message to the slaves.
  - The slaves apply the transaction and send a “Commit ACK” message to the controlling site.
  - When the controlling site receives “Commit ACK” message from all the slaves, it considers the transaction as committed.
- After the controlling site has received the first “Not Ready” message from any slave −
  - The controlling site sends a “Global Abort” message to the slaves.
  - The slaves abort the transaction and send a “Abort ACK” message to the controlling site.
  - When the controlling site receives “Abort ACK” message from all the slaves, it considers the transaction as aborted.

Program – 10

Code

Server

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>
#include <string.h>
#define MSG_CONFIRM 0


#define TRUE 1
#define FALSE 0
#define ML 1024
#define MPROC 32

typedef struct wireless_node
{
    int priority;
    int parent;
} wireless_node;

wireless_node w;

int max(int a, int b)
{
    return a >= b? a:b;
}

int connect_to_port(int connect_to)
{
    int sock_id;
    int opt = 1;
    struct sockaddr_in server;
    if ((sock_id = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    {
        perror("unable to create a socket");
        exit(EXIT_FAILURE);
    }
    setsockopt(sock_id, SOL_SOCKET, SO_REUSEADDR, (const void
*)&opt, sizeof(int));
    memset(&server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(connect_to);

    if (bind(sock_id, (const struct sockaddr *)&server,
sizeof(server)) < 0)
    {
        perror("unable to bind to port");
```

Program – 10

```
54          exit(EXIT_FAILURE);
55      }
56      return sock_id;
57  }
58
59  void send_to_id(int to, int from, char message[ML])
60  {
61      struct sockaddr_in cl;
62      memset(&cl, 0, sizeof(cl));
63
64      cl.sin_family = AF_INET;
65      cl.sin_addr.s_addr = INADDR_ANY;
66      cl.sin_port = htons(to);
67
68      sendto(
69          from, \
70          (const char *)message, \
71          strlen(message), \
72          MSG_CONFIRM, \
73          (const struct sockaddr *)&cl, \
74          sizeof(cl));
75  }
76
77  void begin_commit(int id, int *procs, int num_procs)
78  {
79      int itr;
80      char message[ML];
81      sprintf(message, "%s", "SCMT");
82      for (itr = 0; itr < num_procs; itr++)
83      {
84              printf("Sending begin commit to: %d\n", procs[itr]);
85              send_to_id(procs[itr], id, message);
86      }
87  }
88
89  void announce_action(int self, int *procs, int num_procs, char
90  msg[ML])
91  {
92      int itr;
93
94      for (itr = 0; itr < num_procs; itr++)
95      {
96          send_to_id(procs[itr], self, msg);
97      }
98  }
99
100
101 int main(int argc, char* argv[])
102 {
103     int self = atoi(argv[1]);
104     int n_procs = atoi(argv[2]);
105     int procs[MPROC];
106     int sender, okcnt = 0, nocnt = 0, dncnt = 0;
107     int sock_id, coord_id;
108     int itr, len, n, start, ix;
109
110     char buffer[ML], flag[ML], p_id[ML], msg[256];
```

Program – 10

```
111
112     struct sockaddr_in from;
113
114     for(itr = 0; itr < n_procs; itr += 1)
115         procs[itr] = atoi(argv[3 + itr]);
116     printf("Creating node at %d\n", self);
117     sock_id = connect_to_port(self);
118     begin_commit(sock_id, procs, n_procs);
119     while(TRUE)
120     {
121         sleep(2);
122         memset(&from, 0, sizeof(from));
123         // printf("Tring read\n");
124         n = recvfrom(sock_id, (char *)buffer, ML, MSG_WAITALL,
125 (struct sockaddr *)&from, &len);
126         buffer[n] = '\0';
127         printf("Recieved: %s\n", buffer);
128
129         if (strcmp(buffer, "CMOK") == 0)
130         {
131             okcnt += 1;
132         }
133         else if (strcmp(buffer, "CMNO") == 0)
134         {
135             nocnt += 1;
136         }
137         if ((nocnt + okcnt) == n_procs)
138         {
139             printf("Recieved replies from all clients\n");
140             if (okcnt == n_procs)
141             {
142                 printf("Announcing complete commit\n");
143                 announce_action(sock_id, procs, n_procs, "CDON");
144             }
145             else
146             {
147                 printf("Announcing abort commit\n");
148                 announce_action(sock_id, procs, n_procs, "CABT");
149             }
150         }
151         if (strcmp(buffer, "DONE") == 0)
152         {
153             dncnt += 1;
154             printf("clients confirmed commit\n");
155             if (dncnt == n_procs)
156             {
157                 printf("All process announced commit action\n");
158                 exit(EXIT_SUCCESS);
159             }
160         }
161         // printf("Waiting\n");
162     }
163     return 0;
164 }
```

Program – 10

Client

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>
#include <string.h>
#define MSG_CONFIRM 0


#define TRUE 1
#define FALSE 0
#define ML 1024
#define MPROC 32

typedef struct wireless_node
{
    int priority;
    int parent;
} wireless_node;

wireless_node w;

int max(int a, int b)
{
    return a >= b? a:b;
}

int connect_to_port(int connect_to)
{
    int sock_id;
    int opt = 1;
    struct sockaddr_in server;
    if ((sock_id = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    {
        perror("unable to create a socket");
        exit(EXIT_FAILURE);
    }
    setsockopt(sock_id, SOL_SOCKET, SO_REUSEADDR, (const void
*)&opt, sizeof(int));
    memset(&server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(connect_to);

    if (bind(sock_id, (const struct sockaddr *)&server,
sizeof(server)) < 0)
    {
        perror("unable to bind to port");
        exit(EXIT_FAILURE);
    }
```

Program – 10

```
56      return sock_id;
57  }
58
59  void send_to_id(int to, int from, char message[ML])
60  {
61      struct sockaddr_in cl;
62      memset(&cl, 0, sizeof(cl));
63
64      cl.sin_family = AF_INET;
65      cl.sin_addr.s_addr = INADDR_ANY;
66      cl.sin_port = htons(to);
67
68      sendto(
69          from, \
70          (const char *)message, \
71          strlen(message), \
72          MSG_CONFIRM, \
73          (const struct sockaddr *)&cl, \
74          sizeof(cl));
75  }
76
77  void begin_commit(int id, int *procs, int num_procs)
78  {
79      int itr;
80      char message[ML];
81      sprintf(message, "%s", "SCMT");
82      for (itr = 0; itr < num_procs; itr++)
83      {
84              printf("Sending begin commit to: %d\n", procs[itr]);
85              send_to_id(procs[itr], id, message);
86      }
87  }
88
89  void announce_action(int self, int *procs, int num_procs, char
90  msg[ML])
91  {
92      int itr;
93
94      for (itr = 0; itr < num_procs; itr++)
95      {
96          send_to_id(procs[itr], self, msg);
97      }
98  }
99
100
101 int main(int argc, char* argv[])
102 {
103     int self = atoi(argv[1]);
104     int server = atoi(argv[2]);
105     char *action = argv[3];
106     int sender, okcnt = 0, nocnt = 0, dncnt = 0;
107     int sock_id, coord_id;
108     int itr, len, n, start, ix;
109     char buffer[ML], flag[ML], p_id[ML], msg[256];
110     struct sockaddr_in from;
111     printf("Creating node at %d\n", self);
112     sock_id = connect_to_port(self);
```

Program – 10

```
113     while(TRUE)
114     {
115         sleep(2);
116         memset(&from, 0, sizeof(from));
117         // printf("Tring read\n");
118         n = recvfrom(sock_id, (char *)buffer, ML, MSG_WAITALL,
119 (struct sockaddr *)&from, &len);
120         buffer[n] = '\0';
121         printf("Recieved: %s\n", buffer);
122         if (strcmp(buffer, "SCMT") == 0)
123         {
124             printf("Sending %s to server\n", action);
125             send_to_id(server, sock_id, action);
126         }
127         else if (strcmp(buffer, "CDON") == 0)
128         {
129             printf("Got complete commit, committing to logs\n");
130             send_to_id(server, sock_id, "DONE");
131             exit(EXIT_FAILURE);
132         }
133         else if (strcmp(buffer, "CABT") == 0)
134         {
135             printf("Got abort commit, deleting updates\n");
136             send_to_id(server, sock_id, "DONE");
137             exit(EXIT_FAILURE);
138         }
139         // printf("Waiting\n");
140     }
141     return 0;
142 }
143
```

Program – 10

Results and Outputs:

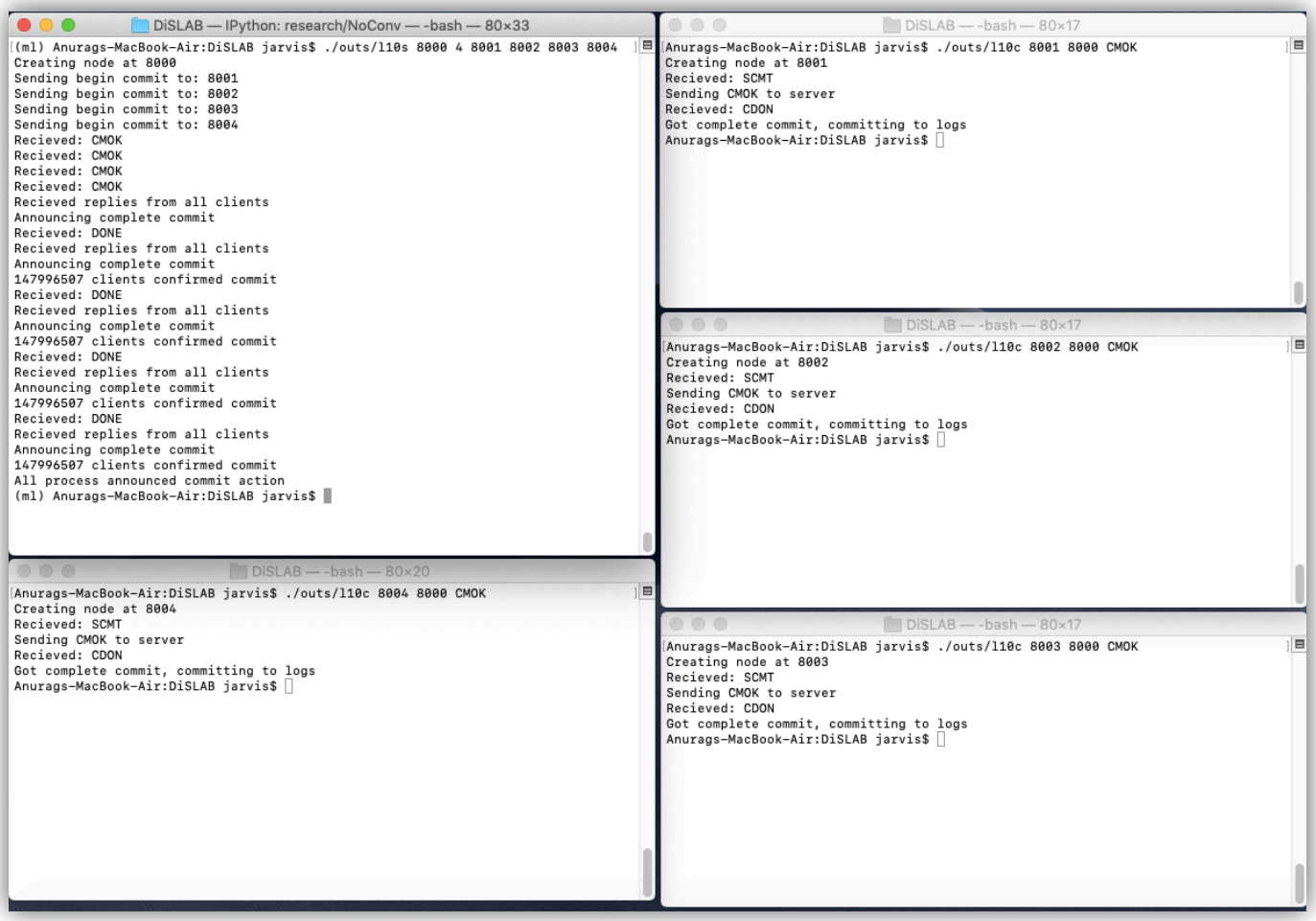

Figure 1 Normal Operation

Program – 10

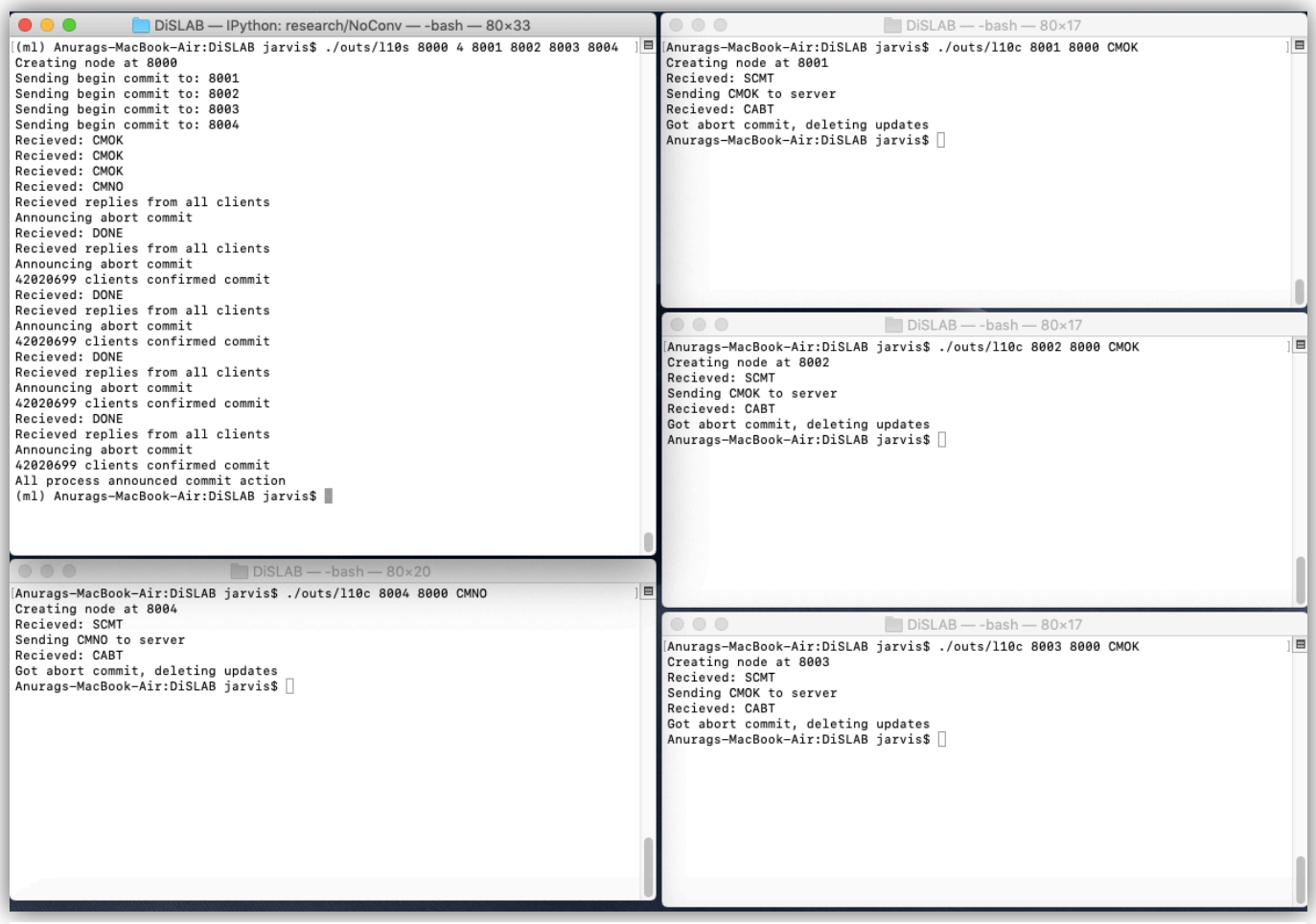

Figure 2 Abort Operation

Findings and Learnings:

1. We successfully implemented 2-phase commit.