UDC 621.31:621.316:519.68
Original scientific paper
Received: 04.07.2011.

Location independent distributed model for on-line load flow monitoring for multi–area power systems

Kannan Nithiyananthan[(1)] and **Velimuthu Ramachandran**[(2)]
[(1)]Department of Electrical and Electronics Engineering, Birla Institute of Technology and Science, Pilani Dubai Campus, International Academic City, Dubai, Post Box No 345055, UNITED ARAB EMIRATES, e-mail: nithi@bitsdubai.com
[(2)]Department of Computer Science and Engineering, College of Engineering, Guindy, Anna University, Chennai 600 025, INDIA, e-mail: rama@annauniv.edu

SUMMARY

The main objective of this paper is to construct a location transparent distributed environment through which the on-line load flow of multi-area power systems can be monitored and controlled. A single-server/multi-client architecture has been proposed which enables that the neighboring powered system clients can access the remote relay control server at any time, with their respective data. The location transparency is the key feature of Common Object Request Broker Architecture (CORBA). Location transparency of the proposed model is the ability to access and invoke operations on the CORBA server object without needing to know where the power system object resides. Developed distributed model also provides language transparency that facilitates the implementation of the power system logic in any programming language. A CORBA based distributed model has been developed in such a way that for every specific period of time, the remote relay control server obtains the system data simultaneously from the neighboring relays which are the clients registered with it and the server send back the response to the respective clients. The relay control server creates a new thread of control for every client request and hence complete distributed environment has been exploited.

Key words: *distributed computing, load flow monitoring, CORBA, client-server model, multi-area power systems.*

1. INTRODUCTION

The power system load flow solution obtained through conventional client-server architecture is complicated, memory management is difficult, source code is bulky, and exception-handling mechanism is not so easy. In the conventional power system operation and control, it is assumed that the information required for monitoring and controlling of power systems is centrally available and all computations are to be done sequentially at a single location [1]. With respect to sequential computation, the server has to be loaded every time for each client's request and the time taken to deliver the load flow solution is also comparatively high [2, 3].

In this present work, a distributed environment has been set up using RMI [4] to estimate and to monitor load flow solutions for different sub-systems of an integrated power system. Each sub-system has been considered as a power system client and hence multi power system clients - single load flow server model is implemented. A client computer basically does the distributed power system monitoring through an applet for every specific period of time and frequently exchanges data with the server. The server does the load flow computation and then distributes the results. Chronologically the server process should be started first, so that it can take the initiative to set up a connection link. It then starts waiting till it receives a connection request from the client. A client can register itself with the remote object (server object), just by invoking the registration procedure on the server object, when it needs a service from it. The remote object obtains the necessary data from the registered client objects and responds back to them respectively with the results. This total process can be automated by making the server get the input data for every specific period of time. Transaction of data among clients and server takes place several times and so the possibilities of the occurrence of errors may be high. Hence it must be handled properly.

2. FLOW MODELLING

2.1 Location transparent model for load flow monitoring

In this proposed model, the power system client can access the remote load flow server through the CORBA server using Internet Inter ORB Protocol (IIOP) as shown in Figure 1. In this model the power system client is represented as a Java applet and it can be downloaded in the client machine. The power system client applet is designed in such a way that it maintains the previous state until it receives the converged load flow results from the load flow server for a given load flow data.

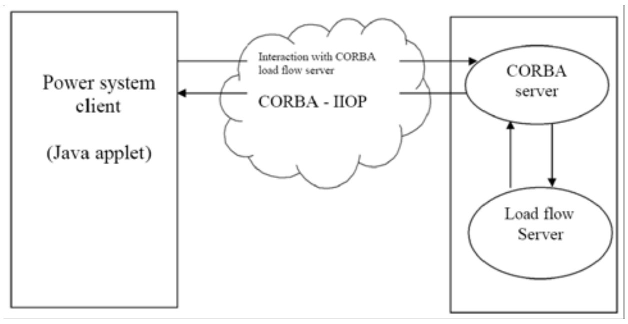


Fig. 1 CORBA based model for load flow monitoring

Figure 1 illustrates the simplest scenario where a power system client interacts with the load flow server through object request brokers (ORB). The power system client and the load flow server are both implemented in JVM. The power system client communicates with the ORB in order to convey a request for an operation invocation to the load flow server, which receives the power system data and then sends the load flow results via the ORB back to the power system client. The interfaces of these components are defined by the CORBA standard and by the application specific IDL.

2.2 CORBA data flow model

This Figure 2 shows a more concrete view of how the ORB performs the task of conveying an invocation from power system client to the load flow server. The IDL compiler generates a number of Java classes known as stub classes for the client and skeleton classes for the load flow server. The role of the stub class is to provide code for proxy object on which power system clients invoke load flow method. The proxy object method implementations on power system client side invoke operations on the load flow servant, which may be located remotely. If the servant is at a remote location the proxy marshals the power system data and transmits the invocation request. It takes the name of the operation and the types and the values of its arguments from language - dependent data structures and places them into a linear representation suitable for transmitting across a network [5].

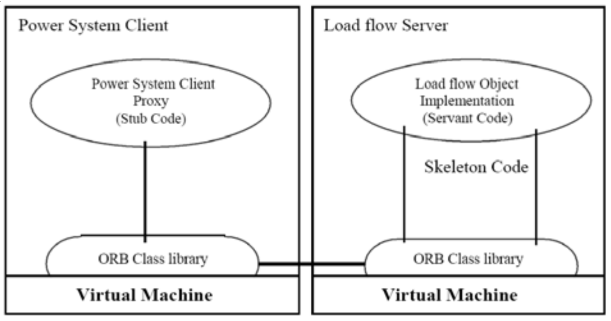


Fig. 2 CORBA based data flow model

The resulting marshaled form of the request is sent to the load flow servant using the particular ORB's infrastructure. In this proposed work, the infrastructure involves a network transport mechanism and additional mechanism to locate the load flow servant and perhaps to activate the CORBA server programs that hosts the servant. The server side skeleton code provides the glue between a load flow server object implementation, a CORBA server, and the ORB, in particular the object adapter. The CORBA specification leaves many of the interfaces between the ORB core, object adapter, and load flow program partially or totally unspecified. For this reason different ORBs have different mechanisms to activate load flow server and for use by object adapters to inform the ORB that their objects are ready to receive invocation requests. After receiving a request, the ORB consults the object adaptor to find the load flow server that is going to execute the operation. The skeleton of the load flow server class implements the mechanism by which invocation requests coming into a load flow server can be unmarshaled and directed to the load flow method of a servant. The steps involved in the development of CORBA based distributed load flow monitoring application are detailed as follows:

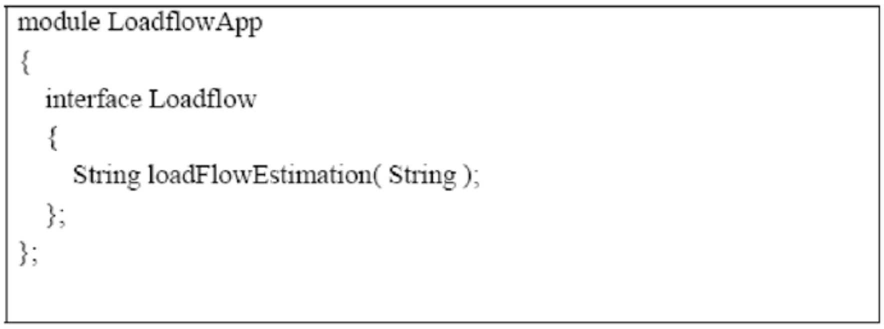
- Write IDL that describes the LoadFlowInterface to the load flow server object that will be implemented. Compile the LoadFlow IDL file.
- This produces the stub and skeleton code that provides location transparency. That is it will cooperate with the ORB library to convert an object reference into a network connection to a remote load flow server and then marshal the power system data as arguments to an operation on the object reference, convey them to the load

flow method in the server object, execute the method and return the load flow results. Compile the .java files, including the stubs and skeletons.

- Identify the IDL compiler generated interfaces and the classes that need to be used in order to invoke or implement load flow monitoring in a distributed environment method.
- The ORB has to be initialized and it has to inform about the load flow server remote objects created. Compile all the generated code and run the distributed load flow monitoring application.

3. IMPLEMENTATION

All CORBA objects support an IDL interface; the IDL interface defines an object type. An interface can inherit from one or more other interfaces. The IDL file functionality is the CORBA language-independent analog to a *C++* header file. IDL is mapped into each programming language to provide access to object interfaces from that language. With Java IDL, these interfaces can be translated to Java using the *idltojava* compiler. For each IDL interface, *idltojava* generates a Java interface and the other *.java* files needed, including a client stub and a server skeleton [6]. The IDL interface for the load flow estimation is shown in Figure 3.



```
module LoadflowApp
{
   interface Loadflow
   {
      String loadFlowEstimation( String );
   };
};
```

Fig. 3 IDL interface for the load flow estimation

The IDL interface is complied and this process generates five files in a LoadflowApp sub-directory:

_LoadflowImplBase.java: is an abstract class providing basic CORBA functionality for the load flow server. It implements the *Loadflow.java* interface.

_LoadflowStub.java: is the client stub, providing CORBA functionality for the power system client. *Loadflow.java* interface contains the Java version of IDL interface. It contains the method *loadflowEstimation( ).*

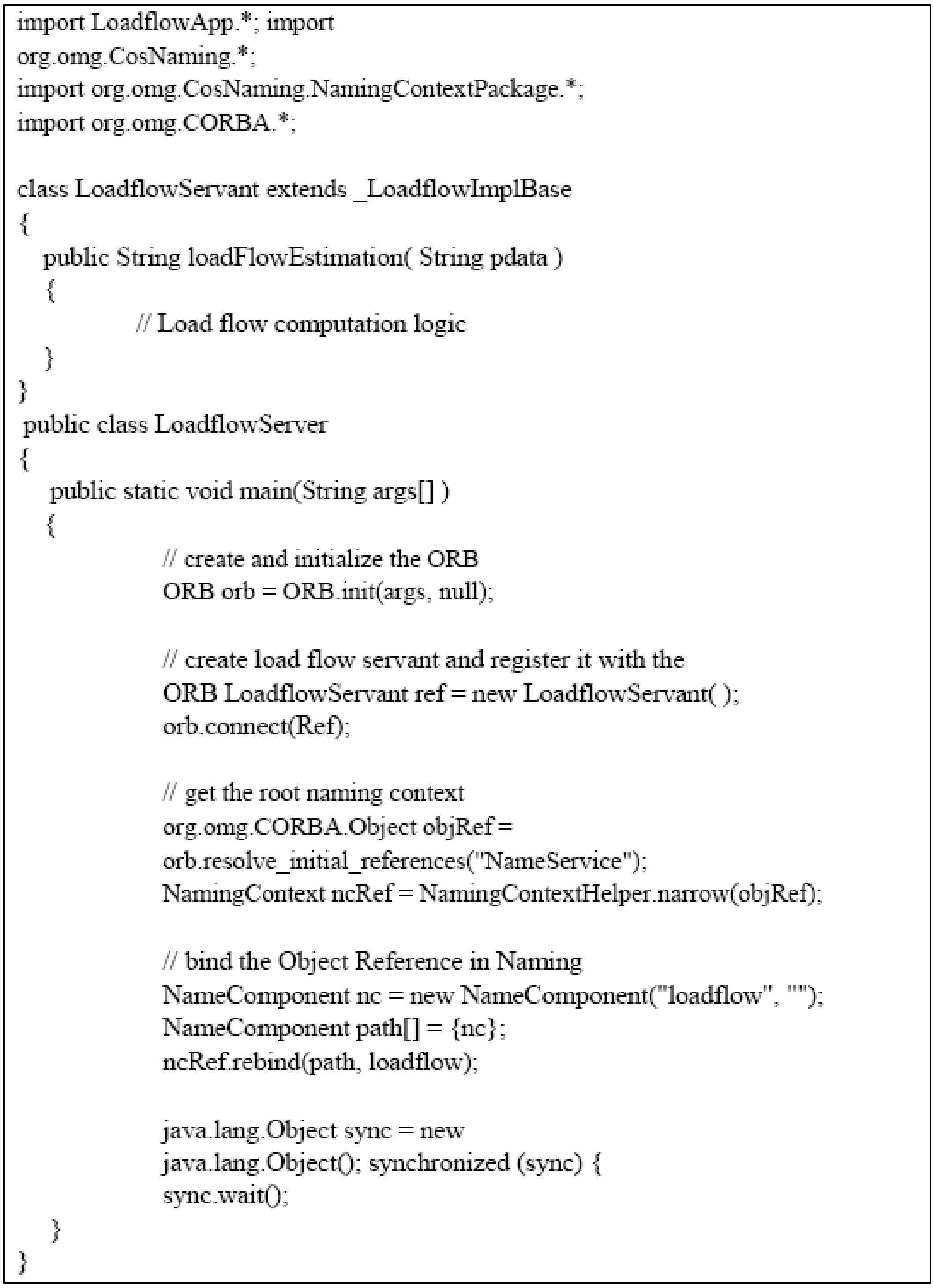
Loadflow.java: is the interface extends *org.omg.CORBA.Object*, providing standard CORBA object functionality as well.

LoadflowHelper.java: class provides auxiliary functionality, notably the *narrow()* method required to cast CORBA object references to their proper types.

LoadflowHolder.java: is a class holds a public instance member of type *Loadflow* interface. It provides operations for out and inout arguments, which CORBA has but which do not map easily to Java's semantics.

The Java IDL Transient Name service is an object server provided with Java IDL. The Name server has been started using *tnameserv* at the command line prompt. This object server conforms to the standard object implementation and invocation techniques. The Name Server stores load flow server object references by name in a tree structure similar to a file directory. A power system client may lookup or resolves object reference by its name.

The load flow server consists of two classes, the load flow servant and the server. The servant, *LoadflowServant*, is the implementation of the *Loadflow* IDL interface; each *Loadflow* instance is implemented by a *LoadflowServant* instance. The servant is a subclass of *_LoadflowImplBase*, which is generated by the *idltojava* compiler. The servant contains *loadflowEstimation( )* method for each IDL operation. Servant methods are just like ordinary Java methods; the extra code to deal with the ORB, with marshaling arguments and results, and so on, is provided by the server and the stubs. The *Loadflow* server class has the *main()* method which creates an ORB instance and creates a servant instance and intimates the ORB about it as shown in Figure 4. Load flow server CORBA object's reference is obtained from a naming context to which a new CORBA server object is registered. The new servant object is registered in the naming context using the name "*loadflow*".

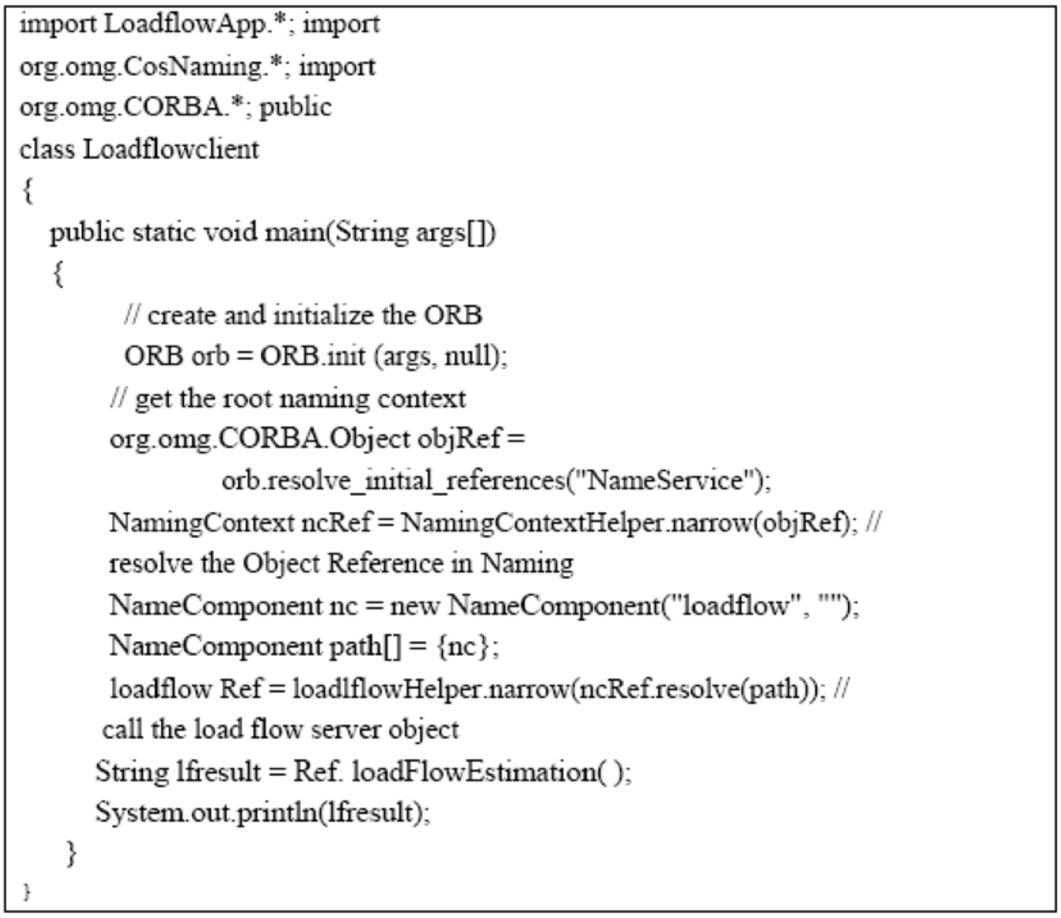


```
import LoadflowApp.*; import
org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

class LoadflowServant extends _LoadflowImplBase
{
   public String loadFlowEstimation( String pdata )
   {
               // Load flow computation logic
   }
}
 public class LoadflowServer
{
    public static void main(String args[] )
   {
                 // create and initialize the ORB
                 ORB orb = ORB.init(args, null);

                 // create load flow servant and register it with the
                 ORB LoadflowServant ref = new LoadflowServant( );
                 orb.connect(Ref);

                 // get the root naming context
                 org.omg.CORBA.Object objRef =
                 orb.resolve_initial_references("NameService");
                 NamingContext ncRef = NamingContextHelper.narrow(objRef);

                 // bind the Object Reference in Naming
                 NameComponent nc = new NameComponent("loadflow", "");
                 NameComponent path[] = {nc};
                 ncRef.rebind(path, loadflow);

                 java.lang.Object sync = new
                 java.lang.Object(); synchronized (sync) {
                 sync.wait();
    }
}
```

Fig. 4 Implementation of the load flow server

Power system client code is linked with *idltojava* generated *.java* files and the ORB library. It will create CORBA objects via the published factory interfaces that the server provides. Since a CORBA object may be shared by many clients around a network, only the object server is in a position to know when the object has become garbage. The client code's only way of issuing method requests on a CORBA object is via the load flow server object's object reference as shown in Figure 5.

The object reference is an opaque structure which identifies a CORBA object's host machine, the port on which the host server is listening for requests, and a pointer to the specific object in the process. Because Java IDL supports only transient objects, this object reference becomes invalid if the load flow server process has stopped and restarted. Power system clients typically obtain object references from the name service. Once an object reference is obtained, the power system client must *narrow* it to the appropriate type and can invoke the *loadFlowEstimation()* method. The load flow results are monitored at regular intervals.

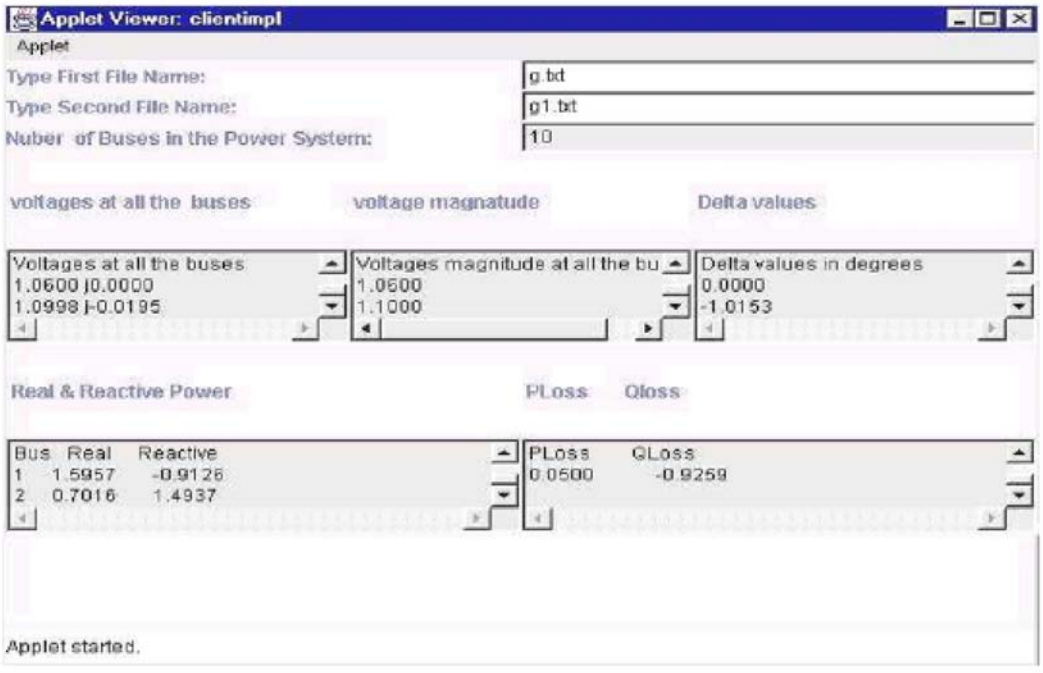


```
import LoadflowApp.*; import
org.omg.CosNaming.*; import
org.omg.CORBA.*; public
class Loadflowclient
{
   public static void main(String args[])
   {
           // create and initialize the ORB
           ORB orb = ORB.init (args, null);
         // get the root naming context
         org.omg.CORBA.Object objRef =
                    orb.resolve_initial_references("NameService");
         NamingContext ncRef = NamingContextHelper.narrow(objRef); //
         resolve the Object Reference in Naming
         NameComponent nc = new NameComponent("loadflow", "");
         NameComponent path[] = {nc};
         loadflow Ref = loadlflowHelper.narrow(ncRef.resolve(path)); //
        call the load flow server object
       String lfresult = Ref. loadFlowEstimation( );
       System.out.println(lfresult);
    }
}
```

Fig. 5 Power system client implementation for location transparency model

4. RESULTS

The above distributed algorithm has been implemented in Windows NT based HP workstations connected in an Ethernet LAN. The results are shown in a client applet as given in Figure 6.



Type First File Name: g.txt
Type Second File Name: g1.txt
Nuber of Buses in the Power System: 10

voltages at all the buses | voltage magnatude | Delta values

```
Voltages at all the buses     Voltages magnitude at all the bu     Delta values in degrees
1.0600 j0.0000                1.0600                               0.0000
1.0998 j-0.0195               1.1000                               -1.0153
```

Real & Reactive Power | PLoss | Qloss

```
Bus  Real    Reactive         PLoss      QLoss
1    1.5957  -0.9126          0.0500     -0.9259
2    0.7016   1.4937
```

Applet started.

Fig. 6 Applet with load flow solution

The applet from Figure 6 shows the load flow solution for a specific 10-bus power system client. When each power system client applet is loaded, it registers with the load flow server, the client sends the request and receives the output. Using this approach, different power system clients can monitor continuous updated load flow solutions at regular time intervals.

The major factor that influences the performance of the proposed models is the round trip time (RTT) that includes the convergence time. The round trip time measures the time needed from the point when the power system client initiates a method invocation to the point when the client receives the results [3]. The round trip time is measured for all the power system clients that invoked the load flow method simultaneously without any delay. The performance analysis of the proposed distributed models has been carried out with respect to load flow monitoring and the variations of round trip time with respect to the number of clients are shown in Figure 7.

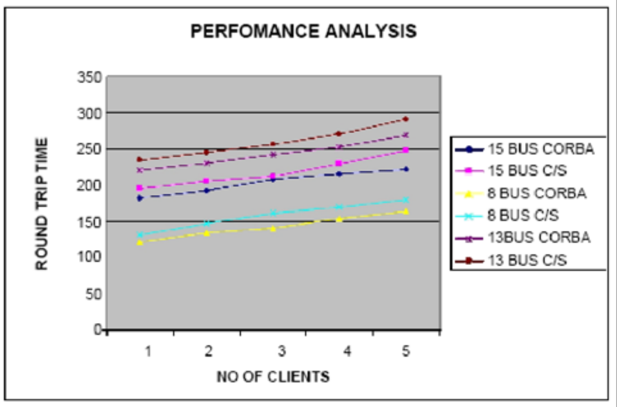


Fig. 7 RTT vs. No of clients (load flow monitoring)

The time taken to invoke the load flow method and to return the results increases linearly as the number of clients gets increased in the proposed model. The graph is plotted between the round trip time calculated and the number of clients registered with the server at a specific interval of time. From the graph shown in Figure 7, it is found that CORBA model performs better than the conventional client-server architecture. Apart from that location and language independence is the key feature of the distributed model implemented.

5. CONCLUSION

An effective distributed model has been developed to monitor the load flow of multiple power systems. It has been tried out in overcoming the overheads associated with sequential power system load flow computation through this model. Although, client-server architecture for load flow solution is very well established, the value of this study lies in that it emphasizes a unique methodology based on CORBA to serve a large number of clients in a distributed power system environment, across various platforms based on communication between virtual machines. A practical implementation of this approach suggested in this paper was assessed based on 6, 9, 10 and 13 bus sample systems. Accordingly the proposed model can be implemented for large power systems network spread over geographically apart.

6. REFERENCES

[1] G. Bandyopandhyay, I. Sengupta and T.N. Saha, Use of client-server model in power system load flow computation, *IE(I) Journal-Electrical*, Vol. 79, pp. 199-203, 1999.

[2] B. Qiu and H.B. Gooi, Web based SCADA display systems (WSDS) for access via Internet, *IEEE Transactions on Power Systems*, Vol. 15, No. 2, pp. 681-686, 2000.

[3] A. Mos and J. Murphy, Performance management in component-oriented systems using a model driven architecture approach, Proc. of the 6th IEEE Int. Enterprise Distributed Object Computing Conference - EDOC, Lausanne, pp. 1656-1667, 2002.

[4] K. Nithiyananthan, V. Ramachandran and S.M. Peeran, RMI based distributed database model for multi-area power system load flow monitoring, *Int. J. for Engineering Intelligent Systems*, Vol. 12, No. 3, pp.185-190, 2004.

[5] A. Buss and L. Jackson, Distributed simulation modelling: A comparison of HLA, CORBA and RMI, Proc. of the 1998 IEEE Winter Simulation Conference, Vol. 1, pp. 819-825, 1998.

[6] A.Wollrath, J. Waldo and R. Riggs, Java centric distributed computing, *IEEE Micro*, Vol. 17, No. 3, pp. 44-53, 1997.

LOKACIJSKI NEOVISAN DISTRIBUIRANI MODEL ZA ON-LINE PRAĆENJE TOKA OPTEREĆENJA ZA VIŠEPOVRŠINSKI ENERGETSKI SUSTAV

SAŽETAK

Glavni cilj ovog rada je napraviti lokacijski transparentan distribuirani okoliš pomoću kojeg se može pratiti i kontrolirati on-line tok opterećenja za višepovršinski energetski sustav. Predlaže se jedan-server/više-klijenata arhitektura, koja omogućava klijentima susjednih energetskih sustava da mogu pristupiti udaljenom serveru kontrole releja u svako doba sa svojim određenim podacima. Lokacijska transparentnost je glavna značajka tzv. Common Object Request Broker Architecture (CORBA). Lokacijska transparentnost predloženog modela je sposobnost pristupu i traženja pomoći CORBA servera bez potrebe da se zna gdje se nalaze objekti energetskog sustava. Razvijeni distribuirani model također osigurava jezičnu transparentnost koja olakšava provedbu logike energetskog sustava u bilo koji programski jezik. Distribuirani model temeljen na CORBA-i je razvijen na takav način da za svaki specifični vremenski period udaljeni server kontrole releja dobiva istovremeno podatke sustava od svih susjednih releja gdje su klijenti registrirani te server vraća odgovor natrag svakom pojedinom klijentu. Server kontrole releja stvara novi korak kontrole za svaki klijentov zahtjev te je tako cjelokupni distribuirani okoliš iskorišten.

Ključne riječi: *distribuirano računanje, praćenje toka opterećenja, CORBA, model klijent-server, višepovršinski energetski sustav.*