AN OVERVIEW OF NON PORTABLE DISTRIBUTED TECHNIQUES.

Sanjay Bansal¹, Sanjeev Sharma¹, Ishita Trivedi², Bharti Bhattred²

¹Acropolis Institute of Technology & Research, Bhopal, India
²School of Information Technology, Rajiv Gandhi Prodyogiki Vishwavidya, Bhopal, India

ARTICLE INFO

Corresponding Author:
Sanjay Bansal
Acropolis Institute of Technology & Research, Bhopal, India.

Key Words: Distributed Computing (DC), Message Passing Interface (MPI)

ABSTRACT

The aim of the present study is to review some non portable distributed computing techniques. The techniques being review are plain MPI, middleware based COM/DCOM and .NET Remoting. The architecture and utilities of each of these techniques are discussed. The issues of interoperability, scalability, fault-tolerance and performance have been addressed. Emphasis is given to factors that affect the performance of each techniques. Finally these methods are compared.

©2012, AJCSIT, All Right Reserved.

INTRODUCTION

Computing power of idle hosts is utilized by distributed computing. There are certain strong reasons that justify using distributed computing in comparison to single powerful computer like mainframes. Distributed systems offer a better price and performance than mainframes. If one machine crashes, the system as a whole can still survive in distributed system. Computing power can be added in small increments in distributed systems. In this way incremental growth can be achieved. There are many techniques of distributed computing.

In general the classification for distributed computing is based on portability viz portable DC and non portable DC. In this paper non portable DC techniques are analyzed. The basis of the choice was that these techniques are most prevailing existing technique. The other basis of choice was that these techniques individually represent their group having a distinguished feature. These techniques are

(1)DC through plain MPI: by plain we imply that the libraries are C based not JAVA based. MPI -1 was defined in 1994 as an DC standard[x].MPI 2 which was followed by MPI 1[Y].MPICH is followed by MPI2.MPICH is implementation of the MPI standard developed by Argonne National Laboratory[Z] MPICH2 is designed to be easily adaptable and efficient and for providing better communication efficiency[q].

(2)Middleware based Techniques COM & DCOM: COM works on single host. Distributed version of COM is DCOM. Both provide a mechanism to support the development of component that can be dynamically activated and may interact with each other [18]. DCOM make use of an IDL (Interface Definition Language), however the DCOM implementation is a more specific IDL called Microsoft IDL (MIDL).However CORBA IDL implementation is not specific. This is a basis CORBA is portable. By publishing DCOM, Microsoft has introduced a new set of call interfaces at the low level called Object Remote Procedure Call (ORPC). ORPC is located on top of the standard Distributed Com. Environment RPC (DCE RPC) environment.

ORPC expands the procedural programming model to accommodate distributed objects [20].

(3).NET Remoting: .NET is the standard platform of windows applications. It has been extended to support parallel computing applications. For example, the parallel extension of .NET 4.0 supports the Task Parallel Library and Parallel LINQ[a].MPI.NET implements a high performance library for the message passing interface. MPI.NET library exploits all of these features to provide a simple, intuitive interface to MPI that fits well with the C# programming model[b].

In the following part of this paper, each of the above mention techniques will be analyzed in the areas of its scalability, performance and load balancing. Finally comparison will make amongst them.

1. NON PORTABLE DISTRIBUTED TECHNIQUES

Portability refers to run in different environments of hardware and software with little or no efforts.

1 Distributed Computing using Plain MPI

The Message Passing Interface (MPI) provides a powerful programming paradigm. It is high performance computing based on the interchange of messages between processes. MPI is the agreed standard way of exchanging messages. MPI was result of much experimental work from roughly 1983-1993. It is available on essentially all parallel computers. MPI is in the form of a subroutine library. There are many issues with MPI. These are mainly active messages, threads, virtual shared memory, process management, parallel input-output, process scheduling, dynamic load balancing, inter-language compatibility, multi-protocol (or multi-vendor) standardization, standards for data conversion strategies, support for debugging and verification, topology awareness, collective communication, derived data types, and parallel I/O [1].

Researchers and practitioners are devoting substantial efforts in improving performance, scalability, better communication protocols, a user level active message...
interface, load balancing, scheduling, latency and bandwidth, communicators, collective communication, portability and fault tolerance.

The performance of MPI is low than Remote Procedure Call (RPC). Bandwidth is low as compare to RPC. Roundtrip Latency is more than RPC [2]. Researchers devoted their efforts to improve the performance by minimize the number of operations, minimize the communication time between processors (or nodes), and maximize the hardware utilization of the computer. Performance optimization for an MPI application on different systems can be obtained by using different MPI calls for different systems [3]. Still there is a need for each communication performance factor to be modeled and mapped to the related system hardware and middleware structures.

Researchers also try to improve the performance by enhancing application and load balancing techniques. MPI in a cluster of heterogeneous machines could lead parallel programmers to obtain frustrated results, mainly because of the lack of an even distribution of the workload in the cluster. Researchers suggested many improvements in this area. One of them is mobile agents [4]. This technique is based on acquiring more precise information of machine's workload for scheduling and load balancing. Load balancing algorithms come into two basic categories - static and dynamic. Whereas static load balancing algorithms (SLB) take decisions regarding assignment of tasks to processors based on the average estimated values of process execution times and communication delays at compile time, Dynamic load balancing algorithms (DLB) are adaptive to changing situations and take decisions at runtime [5]. Suitability of any particular techniques is compared by many researchers [6].

Some researchers implemented dynamic load balancing and scheduling with genetic algorithm [7]. Some researchers also suggested load balancing by MPI transport mechanism for internal signaling and synchronization [8]. There is a lot of scope to development of effective techniques/algorithms for the distribution of the processes/load of a parallel program on multiple hosts to achieve goal(s) such as minimizing execution time, minimizing communication delays, maximizing resource utilization and maximizing throughput.

Performance can also be enhanced by improving better the communication requirements and by adding additional communication features in communication protocol. These additional requirements are such as message-oriented nature and provision of multiple streams in an association. This eliminates the head-of-line blocking. Researchers are trying to design and develop improved protocols for communication. SCPT (Stream Control Transmission Protocol) is a recently standardized transport level protocol with such features [9]. These features enable the better support the communication requirements of parallel applications. These features are not present in traditional TCP (Transmission Control Protocol). These features enhance the communication thus performance of a MPI [10]. However there is still need multi-protocol implementation to allow MPI to run everywhere.

Portability is very important for MPI. In order to address these important and desirable properties, researchers and practitioners devoted substantial efforts in the topology functionality of the Message Passing Interface (MPI). The topology functionality of the Message Passing Interface (MPI) provides a Portable, architecture-independent means for adapting application programs to the communication architecture of the target hardware [11]. In MPI a set of processes is represented by a communicator, which also represents a (static) mapping of MPI processes to physical processors. Some researchers try to generalized the communicators in order to achieve allow multiple communication endpoints per process, dynamic creation of endpoints, and the transfer of endpoints between processes [12]. Still there is a strong need to enhance the portability without affecting the performance is very important. There is a lot of scope to research the portability with performance of MPI.

Fault-Tolerance is another important area in which researcher's devoted substantial efforts. Researchers developed many techniques based on check-point, roll back, and replication techniques, user-coordinated recovery, transparency, event handling, as well as evolvability of legacy MPI codes etc. MPI-FT is a fault tolerance version of MPI. MPI/FT is a trade off sufficient MPI fault coverage against acceptable parallel performance. It is based on mission requirements and constraints. MPI codes are evolved to use MPI/RT features. Recovery management is isolated [13]. Still there is a need of improvement. Researchers also developed a FT-MPI. FT-MPI allows the semantics and associated modes of failures to be explicitly controlled by an application via a modified functionality within the standard MPI 1.2 API [14]. However there is overhead or performance cost is introduced that need to optimize as a further scope for research.

2 Distributed Computing using Middleware: COM-DCOM

Middleware is a set of common business-unaware services that enable applications and end users to interact with each other in a distributed system. In essence, middleware is the software that resides above the network and below the business-aware application software. DCOM based middleware must have some important properties such as reusability, modularity, easier, more cost effective to develop and evolve systems using reusable software and flexibility [15]. Object oriented DCOM based middleware are best suited for these important characteristics. Some of the object oriented middleware are Common Object Request Broker Architecture (CORBA), Common Object Model/Distributed Common Object Model (COM/DCOM), and JAVA, Remote Method Invocation (JAVA RMI). These standards encapsulate low level programming tasks. These low level tasks are mainly marshalling and unmarshalling.

This having advantage that the user focuses on the logic of the application rather than on the detail of communication on level. DCOM based middleware has a variety of qualities, such as size, cost, complexity, flexibility, and performance [16].

Object oriented design is based on reusable frameworks and components. These reify successful patterns and software architectures. Distributed object middleware therefore are uses object-oriented techniques to distribute reusable services and applications. The major issues with middleware distributed computing are efficiency, flexibility, and robustly [17]. Many Researchers are devoting substantial efforts in these issues to resolve.

Distributed Component Object Model (DCOM) is an object-based middleware for distributed computing. DCOM is originated from COM by Microsoft. COM forms the basis for many of the operations carried out in the Microsoft operating systems. COM provides a mechanism to "support
the development of components that can be dynamically activated and that can interact with each other” [18]. DCOM is basically an extension of COM. DCOM adds the ability for a process to communicate with components that exist on another physical machine, while also adding a layer of transparency to the distributed system. DCOM make use of an IDL (Interface Definition Language), however the DCOM implementation is a more specific IDL called Microsoft IDL (MIDL). MIDL generates standard layout binary interfaces which are specific to the implementation and methods of DCOM, thus it is still very much a closed distributed system. It has been specifically designed for use with Microsoft operating systems and applications running under these operating systems. Therefore it is not portable to other platforms. However it is by far one of the most widely used systems and this is primarily due to the success and widespread use of Microsoft operating systems in networked environments today [19].

DCOM is a high level network protocol. It takes over the job, from the user, of writing network code for the control of the communication required for the interaction of distributed components over network. DCOM is not a programming language but a specification. Services are built using (and on top of) COM. Thus it uses COM object oriented technology for providing its services. By publishing DCOM, Microsoft has introduced a new set of call interfaces at the low level called Object Remote Procedure Call (ORPC). ORPC is located on top of the standard Distributed Computing Environment RPC (DCE RPC) environment. ORPC expands the procedural programming model to accommodate distributed objects [20].

There are many research areas of COM-DCOM middleware. These areas are mainly dynamic reconfiguration capabilities, efficient user interaction, device and service discovery, ontology, conflict resolution, resource management, distributing, managing, moving large amounts of data using heterogeneous infrastructure, leveraging and adapting existing standards for database and distributed data management, interoperability and connectivity in heterogeneous environments and Connectivity in constrained environments[21].

Scalability is very important for middleware based distributed computing. A scalable COM-DCOM based middleware must be able to support a large number of clients and servers. This is a crucial requirement for internet-scale distributed applications [22]. Researchers and practitioners are trying to design COM-DCOM based middleware for maximum scalability. DCOM allow systems to interoperate at the component level. Thus interoperability is achieved by providing a software layer and protocols. This software layer offer the interoperability needed for components developed in different programming languages to exchange messages. Still, such technologies present scalability issue. Hence, approaches based on web protocols and XML (Extensible Markup Language) have been proposed to allow interoperable distributed systems irrespective the programming language in which they are developed [23].

There is always a requirement to build a large scale distributed system using DOT NET. In literature there is a event-based middleware to build large-scale distributed systems [24]. However there is need of investigating ways of dynamically changing the overlay network topology in response to the distribution of subscriptions, advertisements, and events. Future work will include the provision of more middleware services like composite event detection, persistent events, access control, transactions, and support for mobile event clients with partial connectivity.

Middleware is often used to implement E-business applications such as online ticket reservation, on line banking etc. Performance problems in this application can lead to customer churn and thus loss of revenue. In this regards QOS aspects are service availability, payment security, end to end response time, hardware utilization and network bandwidth consumption, latency, data transfer, parameter marshalling and scalability [25]. E-business application performance is performance of middleware on which application are deployed. Researchers are trying to improve all these factors. Load balancing techniques are used to improve the performance. Researchers are trying to make dynamic and adaptive load balancing strategies for raid improvement in performance [26]. However there are still scope to research in self-adaptive load balancing strategies, which dynamically tune threshold values according to the run-time state of distributed applications they manage.

3. Distributed Computing using .NET

.NET is Microsoft’s platform for XML Web services. Services are designed to connect information, devices, and people in a common, yet customizable way. The common language runtime (CLR) is the host infrastructure middleware foundation upon which Microsoft’s .NET services are built. The Microsoft CLR is similar to Sun’s JVM. It provides an execution environment that manages running code. CLR simplifies software development via automatic memory management mechanisms, cross-language integration, interoperability with existing code and systems, simplified deployment, and a security system [15][27].

.NET Remoting provides a framework that enables interaction between objects over the application domains. The framework ensures many services, including a support for the activation and object lifecycle, as well as communication channels. Communication channel is responsible for the delivery of messages to remote applications and vice versa. Formatters are used for encoding and decoding messages before their transfer over the channel. DOT NET uses two types of encoding binary encoding and XML encoding. Binary encoding is used in the situation where the performance is of a critical nature. XML encoding will be adequate in the situation where the interoperability with other distributed technologies is essential. XML encoding uses Simple Object Access Protocol (SOAP) for the transport of messages from one application domain into another. .NET Remoting is designed with security in mind, so there exists a number of ways in which channel sinks can access the messages and serialized data stream before this stream is transported through the channel [28].

Lifecycle management of remote objects without the support of the inherent framework is often very difficult. Dot NET Remoting provides several activation models to be chosen from. These models belong to the following two categories:

- Client Activated Objects (CAOs)
- Server Activated Objects (SAOs)

Client activated objects are under control of a lifecycle manager based on leases, which ensures that an object is
destroyed when its lease expires. In the case of server activated objects, developers can choose either single call or singleton model. The lifecycle of a singleton object is also controlled by a lease [29].

Distributed Computing using .NET having many research areas mainly Performance, Scalability, Heterogeneity, Scalability, Fault tolerance and failure management, Concurrency, Openness and extensibility, Migration and load balancing, Security etc[30].

In general areas of computing, virtual Environments .Net has proved successful in fostering application development. Unfortunately, these existing virtual environments do not provide the necessary high performance computing abstractions required by e-Scientists. Researcher proposes and demonstrates a new approach to the development of a high performance virtual infrastructure: Motor is a virtual machine developed by integrating a high performance message passing library directly within a virtual infrastructure [31].

MPI is also implemented in .NET. Making effective use of MPI with the CLI requires an interface that reflects the high level object-oriented nature of C# and that also supports its programming idioms. However, for performance reasons, this high level functionality must ultimately be mapped to low-level native MPI libraries. In addition to abstraction penalty concerns, avoiding unwanted overhead in this mapping process is significantly complicated by the safety and portability features of the CLI virtual machine, such as garbage collection and just-in-time compilation. Researcher describe approach to using features of C# and the CLI such as reflection, unsafe code regions, and run-time code generation—to realize an elegant, yet highly efficient, C# interface to MPI. Experimental results demonstrate that there is no appreciable overhead introduced by our approach when compared to the native MS-MPI library [32].

Table 1.
<table>
<thead>
<tr>
<th>Technique</th>
<th>Year of Apperance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI</td>
<td>1994</td>
</tr>
<tr>
<td>DCOM</td>
<td>1996</td>
</tr>
<tr>
<td>DOT NET</td>
<td>2002</td>
</tr>
</tbody>
</table>

Table 2.
<table>
<thead>
<tr>
<th>Technique</th>
<th>Portability</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI</td>
<td>Implementation in c/c++ and fortran having very limited portability</td>
</tr>
<tr>
<td>DCOM</td>
<td>Mainly on Windows</td>
</tr>
<tr>
<td>DOT NET</td>
<td>Mainly on Windows</td>
</tr>
</tbody>
</table>

Table 3.
<table>
<thead>
<tr>
<th>Technique</th>
<th>Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI</td>
<td>Identifying processes</td>
</tr>
<tr>
<td></td>
<td>MPI_Comm_size(MPI_COMM_WORLD, &amp;numprocs);</td>
</tr>
<tr>
<td></td>
<td>MPI_Comm_rank(MPI_COMM_WORLD, &amp;myrank);</td>
</tr>
<tr>
<td></td>
<td>Sending message to server</td>
</tr>
<tr>
<td></td>
<td>MPI_Send(message, LEN, MPI_CHAR, n, 99, MPI_COMM_WORLD);</td>
</tr>
<tr>
<td></td>
<td>Sending message to clients</td>
</tr>
<tr>
<td></td>
<td>MPI_Bcast(message, 255, MPI_CHAR, 0, MPI_COMM_WORLD);</td>
</tr>
<tr>
<td>DCOM</td>
<td>Interface definition</td>
</tr>
</tbody>
</table>
|           | import "oaidl.idl";
|           | import "ocidl.idl";
|           | [object,
|           | uuid(4ED9E6AD-AB0F-4F93-B911-515A0EB19609), dual,
|           | helpstring("IDCOMIRCServerImpl Interface"),
|           | pointer_default(unique)
|           | ]
|           | interface IDCOMIRCServerImpl : IDispatch
|           | {
|           | [id(1), helpstring("method send_message")]
|           | HRESULT send_message([string] wchar_t *message);
|           | );
|           | [uuid(43CCE165-2922-4583-B502-D04239152SF7),
|           | version(1.0),
|           | helpstring("DCOMIRCServer 1.0 Type Library")
|           | ]
|           | library DCOMIRCSERVERLib
|           | {
|           | importlib("stdole32.tlb");
|           | importlib("stdole2.tlb");
|           | [uuid(3A3A8AA6-8DBF-4AF1-8E0FCA645D545F0),
|           | helpstring("_IDCOMIRCServerImplEvents Interface")
|           | ]
|           | dispinterface_IDCOMIRCServerImplEvents
|           | {
|           | properties:
|           | methods:
|           | [id(1), helpstring("method message_callback")]
|           | HRESULT message_callback([string] wchar_t *message);
|           | );
|           | [uuid(9A64C80B-C8A9-461D-BA75-7507D352B565),
|           | helpstring("DCOMIRCServerImpl Class")
|           | ]
|           | coclass DCOMIRCServerImpl
|           | {
|           | [default] interface IDCOMIRCServerImpl;
|           | [default, source] dispinterface

2. Comparison

3.1 Year of Appearance

3.2 Portability

3.3 Sample Codes For a simple Communication
Instantiating remote object
server=new DCOMIRCServerLib.DCOMIRCServerImplClass();

Registering callback procedure
server.message_callback+=new DCOMIRCServerLib.
_IDCOMIRCServerImplEvents_message_callbackEventHandler(server_message_callback);

Sending message to server
server.send_message(nickTB.Text+":"+messageTB.Text);

.NET Remoting Interface definition
namespace General
{
    public delegate void messageEventDelegate
    (string message);
    public interface IREMOTIRCServer
    {
    event messageEventDelegate messageEvent;
    void send_message(string message);
    }
}

Instantiating remote object
RemotingConfiguration.Configure
("REMOTIRCClient.exe.config");
RemoteHelper() remoteHelper=new RemoteHelper();
server=(IREMOTIRCServer)
remoteHelper.getObject(
typeof(IREMOTIRCServer));

Registering callback procedure
server.messageEvent+=new messageEventDelegate(
new MessageEvent(new
messageEventDelegate(server_messageEvent)).
server_messageEvent);

Sending message to server
server.send_message(nickTB.Text+":"+messageTB.Text);

CONCLUSION
MPI is a simplest one distributed technique with limited portability if implemented using c/c++ or fortran. In case of Dot Net based distributed computing portability is limited to windows based operation systems. DCOM is also best suited for windows. COM supports different languages like Visual C++, Visual J++, Visual Basic, and COBOL. DCOM is a distributed system for windows. Performance of each technique can be improved by load balancing, enhancing the communication.

REFERENCES


[27].NET framework essentials Thuan L. Thai Hoang Lan O’Reilly 2003.


[32] Douglas Gregor and Andrew Lumsdaine “Design and Implementation of a High-Performance MPI for C# and the Common Language Infrastructure”PPOPP ’08, February 20–23, 2008, Salt Lake City, Utah, USA Copyright c 2008 ACM 978-1-59593-960-9/08/0002... $5.00