I.J. Modern Education and Computer Science, **2020, 2, 36-52**
Published Online April 2020 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2020.02.05

Inter-Process Communication (IPC) in Distributed Environments: An Investigation and Performance Analysis of Some Middleware Technologies

Hamed Dinari
Department of Computer Engineering (CE), Iran University of Science and Technology (IUST),
Tehran Province, Tehran, Iran,
Email: dinari.hamed@yahoo.com

Received: 29 January 2020; Accepted: 09 March 2020; Published: 08 April 2020

***Abstract*—**Nowadays with improvement in computer science, distributed systems have attracted remarkable attention and increasingly become an indispensable factor in our life. Massive-scale data processing, weather forecasting, industrial control systems, medical science, multi-tire architectures in enterprise applications, and aerospace to name but a few are the cases in point that distributed systems play a notable role. Inter-Process Communication or in a short form, IPC is specified as the heart of all distributed systems, therefore they are not formed without IPC. Numerous methods concerning IPC have been proposed so far that are utilized in diverse circumstances. According to the physical location of communication processes in applications, IPC could be established among either multiple processes on the same computer or several computers across a network. From the communication pattern's perspective, these IPCs can be classified into two broad groups namely, shared memory and message passing. Although, it is not true to say when processes are performed on the same computer definitely employ shared memory to communicate if processes are executed on the different systems they inevitably communicate through message passing. By way of illustration, pipes use message passing patterns to make a connection between various processes but all of the processes are carried out on the same system. The aim of this research is to depict a categorization of the some IPC methods, give a brief description of them, and assess their performance in terms of transferring rate by sending multiple files in different sizes between a client and server. As we expected, socket as the basic IPC, since it does not perform extra operations on the input data to be sent had a desirable performance compared to others. Although, to achieve some of the capabilities, like eliminating platform dependencies and asynchronous communication, it needs to add additional layers that make poor performance.

***Index Terms*—**Inter-Process Communication (IPC), Remote Procedure Call (RPC), Distributed Systems, Web Services, Remote Method Invocation (RMI).

I. INTRODUCTION

Today computer systems are evolving. Since 1945 which has been considered as emerging of computers' ear until approximately 1985, computers had been large and expensive so that a minicomputer cost at least thousands of dollars. As a result, most organizations had a few computers and because these computers were not connected together they worked independently. But since mid-80 two advancements in microprocessors and communications technologies changed the world completely. These progressions eventually led to the development of modern technologies in which instead of using one powerful processor, multiple normal or poor processors were connected together [**1**]. From an architectural perspective, these multiprocessor computers are essentially divided into two categories:

1. Tight Coupling: in this model, there is a primary memory (address space) which is shared among all processes.
2. Loosely Coupling: in this model, processors do not use share memory and each processor has its own local address space. Furthermore, they are connected together through various communication lines with low or high bandwidth.

Normally, the tight coupling systems are recognized as "parallel processing systems" as well as loosely coupling systems that are identified as "distributed systems". As the title of the paper indicates, this research tends to proceed with the second group. There are lots of definitions for distributed systems but, none of them in agreement with any of the other. One of those definitions as follows: A distributed system is a collection of independent computers that appears to its users as a single coherent system [**1**]. Actually, from the user's point of view, who works with a distributed system, this system resembles a single computer". Figure 1 shows an architectural view of distributed systems.

Distributed systems are used in a wide spectrum of areas, for example, multi-tier architectures in enterprise applications [**2**] (a two-tier and a three-tier architecture are presented in Figure 2 and Figure 3 respectively), weather forecasting, computer and wireless sensors networks, banking and airline reservation systems, scientific computing (cluster, grid, cloud), and real-time process control.

The leading purposes of distributed systems include Transparency, Openness, Reliability, Performance, and Scalability. Because in this paper our main target is Inter-Process Communication and also we would not like to reinvent the wheel, you can meet [**1**,**3**] to find more valuable information regarding distributed systems.

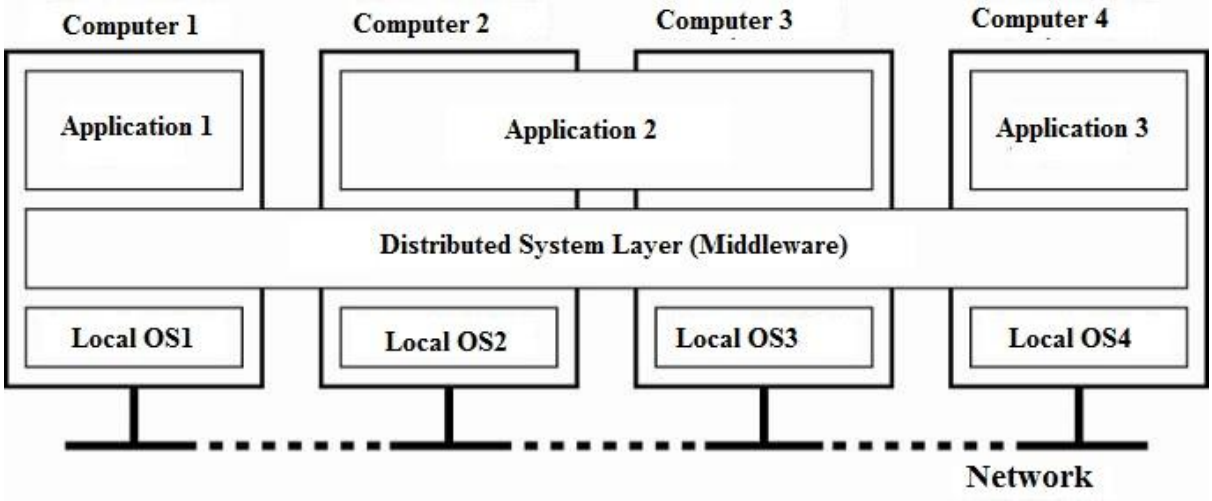

Fig.1. Distributed system architecture [**1**]

The communications between processes are specified as the heart of all distributed systems so that without a profound understanding about the role of them and how they would make, our knowledge about distributed systems is defective. To reach these ends we would attempt to cover corresponding facts about them. Usually, these communications can be classified into two common groups including shared memory, and message passing. In the following sections, we would express them so that you would be able to obtain more knowledge and apply them in your work more effectively. Because communications in distributed systems establish through a network and there are vast concepts that depend on one another, if you desire to grasp more about network's layers and also precise information pertaining to either Transmission Control Protocol /Internet Protocol (TCP/IP) or Open System Interconnection (OSI) models, you can refer to [**4**,**5**].

In this paper, as its title implies, we do not intend to introduce a novel IPC method, but we would like to report a series of IPCs and demonstrate how and why these methods have been developed. For example, what methods or technologies can remove platform dependencies or make possible asynchronous communication. In the results section, you would observe how some of these technologies like Web Services encounter a sharp drop in performance to achieve these goals. There are other methods not mentioned in this study, including signals, which is one of the commonly used in Operating Systems, because this article is focused on high-level IPCs. As we would emphasize throughout this research, these methods are not superior to each other, but in various situations and depending on different requirements you can choose the appropriate method. In some circumstances, some of these methods may be used in combination to achieve your goals. Although, to be master all of these techniques you need experience, practice, and time. So, another purpose of this paper is to give an overview of several methods in a nutshell and to help the reader understand the concept of IPC in less time unlike most of the papers that have focused on two or three methods. One of the limitations of this article is that it does not describe the methods in detail. Because the details of each of these methods can be as much as a book, in addition, everyone who reads this article may not be highly skilled to grasp the concept of IPC, so another target of this paper is to express the concepts in a simple manner. Some of these methods, including CORBA, are not used today, but because our goal is to describe the evolution of the spectrum of some IPCs, CORBA also sits in this range, so, we have to address it.

Because the principal purpose of this study is focused on message passing communications in distributed environments we would like to express a brief history concerning how some communication methods have emerged. In operating systems processes have their own address space to do not affect each other, the main drawback of such mechanisms appears when they tend to exchange data so that they have to copy the data which is a quite time-consuming and tedious operation especially for large-scale data. Shared memory as the first and foremost Inter-Process Communication (IPC) method was recommended to deal with this problem and processes could communicate more easily.

In the early, processes communicated through shared memory, then various methods have been offered in

order to solve different issues depending on diverse conditions. Throughout the history of computing, all network's operations have been performed by operating systems. The Unix operating system was the first provider of network facilities. Personal computing was being fulfilled slowly, Microsoft and Apple software did not support network protocols until the mid-1990s. Although Novell and Banyan companies were popular in this scope, they also supported only network capabilities at the operating system level.

In essence, the concept of networking in the world of computers for implementation of telecommunications was not much discussed until the development of the World Wide Web (WWW).

The Network operating systems provided capabilities in which an application could be shared among multiple users simultaneously. These one-layer systems were not scalable enough to be expanded. The advent of computer networks and improvement in technologies have led to the advancement of systems, consequently, the Object Request Broker (ORB) concept was raised. For instance, Microsoft's MTS and Common Object Request Broker (CORBA) were developed. These interfaces decoupled both layers: User Interface (UI), and Business Logic (BL).

On the other side, the HyperText Transport Protocol (HTTP) was released in 1990. Although several other protocols, such as Gopher had already been developed, the major characteristics that distinguished the HTTP protocol from other ones were its extensibility against web languages like Hypertext Markup Language (HTML) and also the ultra-flexibility in the transmission layer of the TCP/IP protocol. Therefore, the HTTP made it possible to transfer data in various formats without any particular conditions. In the span of the next ten years, low-level protocols were supported by the network operating systems and by the Simple Mail Transfer Protocol (SMTP) and the File Transfer Protocol (FTP) protocols on the Internet, it became easier to transfer files over a network.

Similarly, the Remote Procedure Call (RPC) concept was offered in which a procedure (subroutine) could be performed in a different address space by a computer program. When RPC was released it made an opportunity to expand programs, although this concept was completely dependent on software platforms. For example, RPC on the Unix Operating System by Common Object Request Broker Architecture (CORBA) and on the Microsoft Operating Systems by Distributed Common Object Model (DCOM) were implemented We will describe them in the next sections.

In the development environments data layer (Database), process layer (Core), and application layer (User Interface) were decoupled and installed on multiple connected machines. As a result, application programs became more reliable against extensibility. For many years, Microsoft and Sun corporations competed regarding the RPC challenges. CORBA vs. DCOM (CORBA was developed and released by the Enterprise Management Group (EMG)), and these competitions continued until Sun Corporation developed RMI for Java.

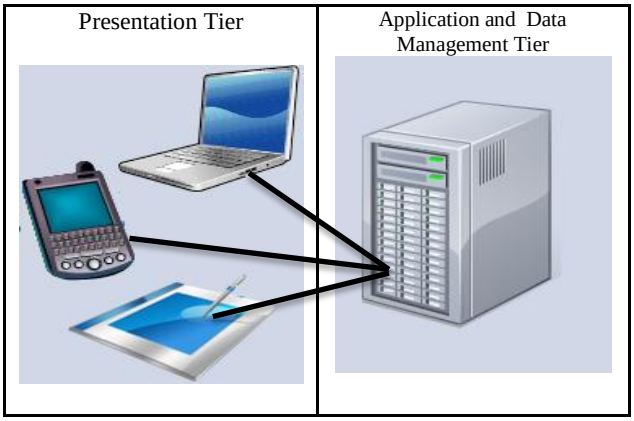

Fig.2. Two-tier architecture

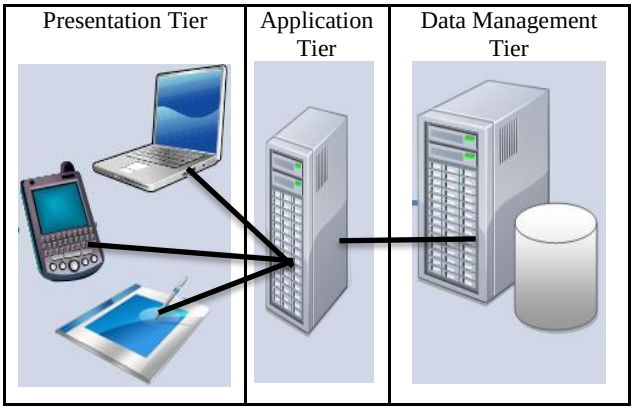

Fig.3. Three-tier architecture

The RPCs like RMI, CORBA, and DCOM have a sophisticated implementation, but it is one of the major drawbacks of them. These middleware technologies to transfer data relied on specific standards of their corporations, which disallowed them to communicate. For instance, if a corporation used DCOM, it was not permitted to communicate with a corporation that utilized CORBA. The following states some of their main disadvantages:

1. These three RPCs have their own format to transfer data as well as they depend on the Operating System so that they could not share information.
2. The methods and mechanisms that they use to exchange facts are different completely. By way of illustration, DCOM adopts ActiveX Data Objects/Remote Data Service (ADO/RDS) to transfer data whereas RMI applies Java DataBase Connectivity (JDBC). Hence, they could not communicate. Researchers endeavored to tackle these troubles, then the Web Services (WS) were suggested that not only covered the pitfalls regarding platform dependencies, but they are elegant solutions to the issues that have ever been reported.

After the mentioned problems were solved, since some computer systems were not always available to process requests or because of the high load of systems, they tended to postpone several requests or services to another time, systems had to communicate asynchronously. To do so, the data that exchanged among systems must be persisted and processed at the proper time. To manage this challenge the Message-Queue-Based communication methods or Message-Oriented Middlewares (MOM) were founded. In these technologies, all data related to communication is preserved in a message provider such as IBMMQ, ActiveMQ, RabbitMQ, and etc., then processed in a suitable time according to particular policies. Equally, other methods and mechanisms have been developed to overcome a special challenge, for example, stream-oriented communication methods were created to carry out replication, transferring audio and video files, and load balancing.

The remainder of the paper is organized as follows. Section two highlights some relevant papers as a literature review that has been conducted so far. Section three indicates a general classification of several IPC methods and also goes through each. Section four is dedicated to the performance evaluation of some middleware technologies. Finally, the conclusion, future works, and summary of multiple IPC are represented by section five.

II. Literature Review

Until now, Inter-Process Communication has been discussed as one of the commonly investigated topics in distributed systems and numerous research papers have been conducted to address the IPC from diverse perspectives. These research studies discussed some IPCs like Java-RMI, web services, CORBA, COM/DCOM, pipe-based, shared-memory, and socket, then analyzed and compared them from different viewpoints such as performance, programming languages, and their architecture. We will represent several of them briefly.

In [**6**] the different IPC mechanisms are analyzed by comparing various sizes of data by a program simulating the messages across the network. All the source code for IPC performance evaluation was written in UNIX. The performance factors such as memory, transfer rate, buffer sizes, data transfer methods, and code complexity are examined and evaluated for all mechanisms. A comparison of different mechanisms shows that the streaming socket performs well.

The authors in [**7**] presented an argumentative comparison of both technologies showing where they relate and where they diverge. They have also stated solutions and challenges for interoperation between both technologies. An oversimplified view is to consider Web services as middleware for middleware that would locate on top of CORBA and relegate CORBA as a lower-level implementation platform. As an illustration from the telephony networks, CORBA sometimes sits on top of SOAP-like applications. In [**8**] authors considered Java-RMI, CORBA and Web Services from different viewpoints. Although improvements in implementations of SOAP communications have significantly reduced the performance failings, while a Web Service solution will still be slower, consume more memory, more network bandwidth, and more CPU cycles than an alternative solution, the differences are less marked in realistic applications. In [**9**] authors explore the diverse mechanisms of several IPCs like CORBA, socket, RPC, and REST alongside their advantages and disadvantages.

In [**10**] authors provided an architectural analysis of the existing distributed object-oriented technologies like CORBA, Java RMI, and COM/DCOM. They pointed out theses IPCs from various perspectives including architectural differences, programming differences (e.g., issues like server object locators, object inheritance), and application differences.

Authors in [**11**] considered various IPCs like sockets, pipes, and shared memory, then assessed their performance. They found which the transmission time of the pipeline was basically unchanged regardless of the amount of data transferred. It was time-consuming to establish a pipeline. But once it was established, the data transmission time was basically the same regardless of the amount of data. However, the data transmission time was increased with the increase of the number of bytes transferred by shared memory and sockets. Therefore, the pipeline was the best method when a large amount of data needed to be transmitted. When the amount of data transferred was less shared memory had obvious advantages in transferring data at very fast speed.

In [**12**] authors have provided a detailed comparison of web services and distributed objects. They tried to compare the design and implementation of a small file server application implemented using RMI and web services. They discovered that using the most straightforward implementation in both technologies, web services outperform RMI when accessing multiple/deeply nested files, especially over high latency channels. However, the default web services interfaces are improper to use, so they develop a technique for wrapping the web service to make it as easy to use as the distributed object implementation.

The authors in [**13**] represent an experimental evaluation of the latency performance of several implementations of Simple Object Access Protocol (SOAP) operating over HTTP, and compares these results with the performance of JavaRMI, CORBA, HTTP, and with the TCP setup time. The main objective of their work to identify the sources of inefficiency in the current implementations of SOAP and discuss changes that can improve their performance.

In [**14**] authors have studied and evaluated three widely-used inter-process communication devices-pipes, sockets and shared memory. They have identified the various factors that could affect their performance such as message size, hardware caches, and process scheduling, and constructed experiments to reliably measure the latency and transfer rate of each device.in [**15**] authors attempted to make a clear investigation between Web Services and Distributed Objects as well as

described some misconceptions that everyone might face.

M.D. Hanes and his co-workers focused on the proper use of the technologies like RMI, CORBA, and web service to implement new Signal and Image Processing (SIP) applications, or developing other applications by these technologies because of an emerging trend in the SIP community are the advent of middleware and middleware can be readily exerted for distributed computing applications by the SIP community [**16**].

The authors pointed out RMI, RMI Tunneling, and Web Services performance elegantly. They have compared technology alternatives for developing distributed Java applications, which have to communicate through firewall and proxy secured networks. These alternatives can be classified into two groups: (1) Using RMI tunneling techniques, including HTTP-to-port, HTTP-to-CGI and HTTP-to-servlet tunneling; and (2) using Web Services instead of Java RMI. The comparison of RMI tunneling alternatives has shown that the transition to RMI tunneling is related to administrative tasks, including the deployment and configuration of corresponding tunneling components and settings [**17**]. N. Lynch and A. Shvartsman, developed and analyzed algorithms to solve problems of communication and data sharing in highly dynamic distributed environments. The term dynamic here encompasses many types of changes, including changing network topology, processor mobility, changing sets of participating client processes, a wide range of types of processor and network failures, their approach to middleware differs from common practice: although middleware framework such as CORBA supports the construction of distributed systems from components, their specification capability is limited to the formal definition [**18**].

The authors evaluated the performance of RMI, RMI-SSL, web service, and WS-security and considered their features. They have conducted a functional and performance analysis. Moreover, they have assessed both regular (unsecured) as well as secured variants, WS-Security and RMI–SSL. Their investigation covers the following evidence: RMI is suitable for distributed applications, which require synchronous remote method invocations only, make use of stateful objects, object references, distributed garbage collection. Web services, on the other hand, are better suitable for dynamic service binding and communication through firewalls. Differences also exist between secured versions. RMI–SSL offers point-to-point security while WS-Security offers message-level security. To recognize the differences in performance they have done these performance analysis on both Windows and Linux. The measurements have illustrated that RMI was superior to Web services in all scenarios [**19**].

III. Categorization Of The IPC Methods

Normally, massive data applications are distributed. To have a profound understanding of how they function in distributed environments you need some concepts that we will describe. One such notion is *Inter-Process Communication* or in short form *IPC*, which explains ways and how the processes communicate. The IPC is specified as the heart of all distributed systems and because of its outstanding role, lots of research papers have ever been performed to deal with its methods and mechanisms.

There are various kinds of IPC, including shared memory, RMI, web service, pipes, and so on. In a nutshell, almost all of them are constructed based on two major concepts: *shared memory* and *message passing*. In Figure 4 a general classification of some IPC technologies is pictured which we would explain later.

A.1 Shared Memory

Processes are located in different address spaces in order to do not touch each other. The main drawback of this isolation is that if one process requires to pass some data to another, the data must be copied, which can be a relatively costly operation for an immense amount of data.

To manage the problem, *shared memory* is used. As its name implies, through shared memory two or more processes have access to the same memory location and would be able to transfer data.

Shared memory does not handle the concurrency problems to the processes involved. To achieve this goal, it often exploits concurrency control techniques such as semaphore. One of the significant benefits of shared memory is when processes tend to exchange a large amount of data. Figure 5 illustrates a shared memory between several processes.

There are two kinds of shared memory that will be investigated as follows:

A.1.1 Mapped File

In this mapping the region of the virtual memory belongs to the process is mapped to the files. To put in another way, reading or writing to those sections of memory is mapped to reading or writing operations to the file. This approach is recognized as the default mapping type. There are two kinds of memory-mapped files as follows:

A.1.1.1 Persisted

In this type as their names offer, when the last process is terminated, data are saved to the source file on the disk. These memory-mapped files are suitable to manage the large source files.

A.1.1.2 Non-Persisted

In this group, as their names suggest, the data are elusive. Simply speaking, when the last process that working with a file is finished, data are wiped out. These memory-mapped files are convenient when shared memory is used to communicate between processes.

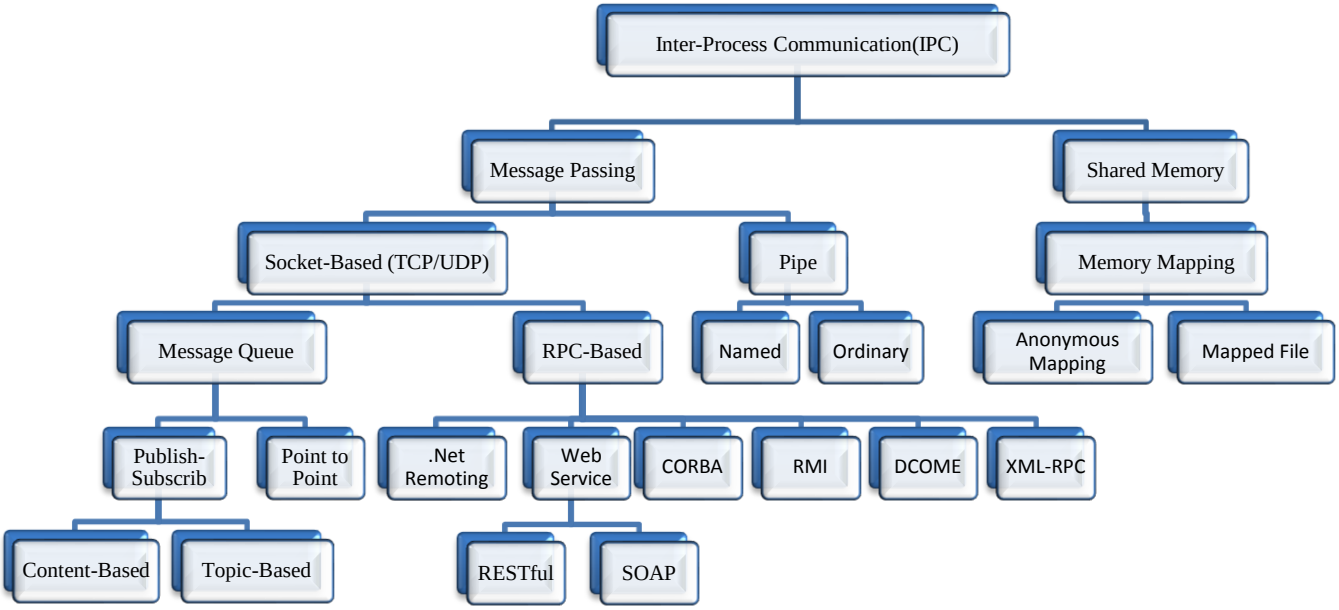

Fig.4. A classification of some IPC technologies

A.1.2 Anonymous Mapping

In this mapping type, the area of the virtual memory owned by a process is mapped. The contents are set to zero. This mapping resembles dynamic memory allocation. The memory in one process mapping may be shared with the mappings pertaining to other processes. This can be performed through two approaches:

- If a segment of a file is mapped by two processes, the same pages of physical memory are shared by them.
- If a child process is built, it inherits the mappings belong to its parent which link to the same pages of the physical memory of that parent. When any modification is done on data in the child process, various pages would be made the child process. When two or more processes share the same pages, each process can detect the changes in the page contents made by other processes depending on the mapping type. There are two mapping types such as private or shared.

Private Mapping: any changes in this mapping are not observed to other processes.

Shared Mapping: when any modifications are done over the content of this mapping, they are discernible by other processes.

B.1 Message Passing

Another notable IPC concept that is routinely investigated is *message passing*. In message passing, processes communicate by passing *messages* just using two operations: *send* and *receive*. The message passing concept relatively seems simple, but it requires multiple design options to be made. There are many methods that employee message passing to communicate. The following examines them.

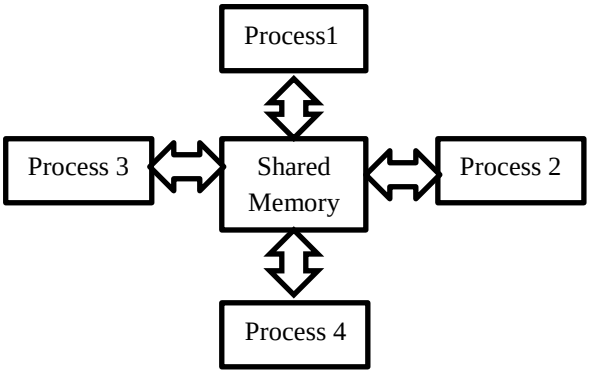

Fig.5. Shared Memory between multiple processes

B.1.1 Pipe

A *pipe* is one of the most straightforward IPC methods, and it can be shared among two or more pertinent or independent processes. A pipe has two endpoints, just as a physical pipe. Normally one process produces data and leaves to one end of the pipe and another process consumes them from the other one. Pipes are divided into two categories namely, ordinary and named which are as follows:

B.1.1.1 Ordinary

Ordinary pipes permit merely one-sided communication. They implement the producer-consumer mechanism, which means one process leaves to the pipe and another one acquires from it. In these pipes, processes use a parent-child relationship to communicate. To be more specific, a process can utilize the pipe which has been constructed by itself or a process that has inherited it. Whenever processes are communicating over a pipe and processes terminated for any reason, the ordinary pipe would be destroyed (figure 6).

B.1.1.2 Named

Named pipes are more vigorous in comparison to ordinary pipes and they can be bidirectional, unlike the ordinary pipes that are unidirectional. Once a named pipe is built, several independent processes can communicate over it. Named pipes are not demolished even if the communicating processes are terminated. In these pipes, one process can write to one pipe and read from another. This capability allows them to write and read at the same time. They must be explicitly destructed when not required again (Figure 7.)

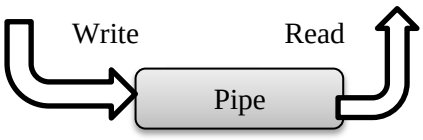

Fig.6. Ordinary pipe

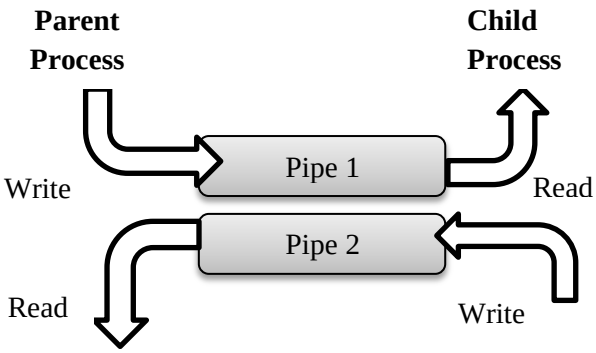

Fig.7. Named pipe

B.1.2 Socket

A socket is one endpoint specified by an IP address and a port number which permits communication between two autonomous processes on the same or different machines. More specifically, it is a manner in which computers talk to one another over a network. By defining a socket, the programmer would announce to the operating systems to provide resources and also required space to establish a connection without going through in TPC/IP details.

More broadly speaking, there are four kinds of sockets including, stream sockets, datagram sockets, raw sockets as well as sequenced packet sockets. The first two are most extensively exploited and the last two are rarely utilized. The aim of our research is to outline the first two.

B.1.2.1 Streaming Sockets

Streaming sockets are identified as "connection-oriented sockets". In such an approach, the delivery process would be guaranteed. More specifically, If the sequence of digits such as "1, 2, 3" are sent through the streaming socket, they would be achieved in the same series "1, 2, 3". These sockets apply TCP protocol to exchange data and also before sending data over a network must make a connection through three steps handshaking, if the delivery process is not fulfilled, an error is sent to the sender. Most services and protocols that are defined in the fourth layer of the network and require authenticity, accuracy as well as maintaining the order of data, use these types of sockets. FTP, HTTP, and SMTP which need a reliable and safe connection are the cases in point.

B.1.2.2 Datagram sockets

These sockets are recognized as "connection-less sockets" and are based on UDP (User Datagram Protocol) protocol. They do not guarantee the delivery process. Unlike the stream sockets which mentioned above, these type of sockets are connectionless. Expressly, before sending data do not require to make a connection. A packet with the detailed information of the destination is produced and send it out. Notwithstanding, datagram sockets are unreliable they still cover a wide spectrum of topics and scopes, including audio and video transmission, and also Domain Name System (DNS). Transferring data with high-speed is one of the most leading merits of them.

RPC-Based methods

In a nutshell, a Remote Procedure Call (RPC) is when an application causes to perform a procedure in a different address space (normally on another computer on a shared network). The following investigates several RPC-based methods.

B.1.3 XML-RPC

XML-RPC is an RPC-based IPC that is applied throughout the Internet. An XML-RPC request is an HTTP-POST request that uses XML schema in its body. It is a procedure that is executed on the server and sends its response as an XML format. It can exert different parameter types like String, Number, Array, and etc.

This IPC was expanded by a group of people in Microsoft corporation in 1998 and a new protocol named SOAP (Simple Object Access Protocol) was developed. In comparison to SOAP, XML-RPC is more simple. Later, we will express SOAP.

XML-RPC uses its methodName characteristic to invoke methods that may contain lowercase or uppercase letters, numbers, commas, and the '/' sign, which is suitable for many purposes, but when we tend to send an object as an argument we would encounter a problem. Normally, in XML-RPC, arrays and structures are

without any name. To grasp more detail refer to [**20**].

B.1.4 CORBA

The CORBA stands for Common Object Request Broker Architecture. It is a standard developed by the Object Management Group(OMG) to simplify the communication of systems without getting involved in hardware platforms, programming languages, as well as operating systems. Normally, this standard is applied in the multi-tier architecture of applications. Although the C++ implementations of CORBA manage In and InOut parameters by nature, C++ developers suffer from various series of challenges with parameters that are related to storage. These issues would appear when object references and varying-length entities such as strings or sequences are passed as arguments [**3**].

The complexity of its structure is one of the dominant drawbacks of CORBA. To expose an interface or API, CORBA utilizes Interface Definition Language (IDL). It provides IDL mappings for a wide range of programming languages, including C, C++, COBOL, JAVA, LISP, PL/1, Pascal, Python, and etcetera. Moreover, in the future, it would be conceivable to afford mappings for other programming languages that require to support this technology. From Java programmers' point of view, unlike RMI it is not a convenient and flexible technology to implement Java-based programs because it does not allow to pass some executable codes as inputs.

When CORBA's clients and services need to communicate, the requests are passed to objects that recognized as Object Request Broker (ORB) to invoke the methods. Additionally, ORBs are interconnected via Internet Inter-Orb Protocol (IIOP) and enable distributed programs to communicate over the Internet regardless of the programming languages. Figure 8 pinpoints CORBA's architecture.

B.1.5 DCOM

DCOM (Distributed Component Object Model) is a distributed extension to COM (Component Object Model) which is a component-based development model for the Windows environment. It is a collection of Microsoft concepts and program interfaces in which client program's objects would be able to demand services and communicate to other computers over a network.

It was Microsoft's response to CORBA. This protocol would be very beneficial while using COM components and also does not require to communicate with non-Microsoft-based systems otherwise, it would not be effective. The COM is used by developers to "create reusable software components". Moreover, it is developed to create a connection between the software components of applications. COM's objects are made using different object oriented-based programming languages such as C++. COM+, DCOM as well as ActiveX controllers that belong to COM's family. For a detailed review of this section refers to [**10**].

B.1.6 Remote Method Invocation (RMI)

RMI is short for Remote Method Invocation. As its name suggests, it provides a mechanism for Java developers to invoke methods of various objects on virtual machines remotely. Diverse RPC-based methods have been offered. Unlike most of them, it is not restricted to primitive data types and would be able to pass or return objects to programs. Precisely, programmers are allowed to transfer code snippets over a network and perform them on remote virtual machines dynamically.

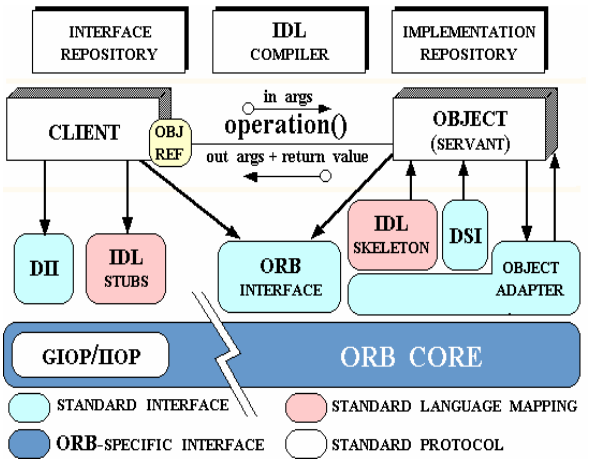

Fig.8. CORBA's Architecture [**21**]

In large scale environments, RMI's clients could access to the new version of Java's services and do not need to distribute programs among clients. This capability could be used both in local networks and web environments. Dividing CPU's loads is one of the chief advantages of RMI. Figure 9 indicates an architectural view of RMI.

Due to the most flexibility and high adaptability of RMI, it has been adopted by most developers to create enterprise applications. Because it is a Java technology, it would not be able to interact with non-Java-based programs like C and C++.

RMI's objects could be accessed in two ways:

- Remote Access by Reference
- Remote Access by Value

In the first case, the object located in a server and when the first client sends a request to make that, it is created as well as any modifications on the object are seen by other clients. The prime advantage of this case is saving the server's resources (fetching the object from secondary memory to the main one and creating it performs only once, furthermore it can be serialized on the secondary memory once again) but the drawback of this approach is increasing the network traffics because various requests are constantly sent out to the server to carry out on the object.

In the second case, whenever a client sends a request, an object is made and placed in the server's memory and since each client can fulfill own modifications on object locally these changes on the object would not be observed by other clients. Simply speaking, these operations are done by the client and consequently, the server's resources are preserved (because of multiple copies of an object). Moreover, transferring information over the network decrease that results in declining in the network's delay.

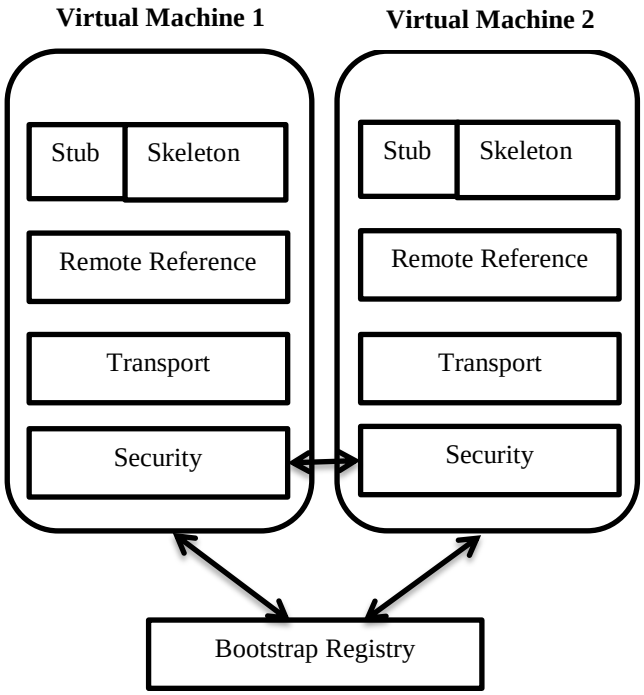

Fig.9. RMI's architecture [**10**]

B.1.7 Web Service

A web service is a client-server application or an application component for communication. It is a technology that allows clients to communicate through invoking their methods regardless of the operating system, programming languages, and hardware platforms and known as an adaptive evolution in distributed computing.

Precisely, it is a middleware that defines a series of operations, protocols, and XML-based standardized messaging to eliminate the hardware and software dependencies and permits programs to exchange data in the most straightforward approach. By way of example, two web services written in .NET and JAVA and installed on the Linux and the Windows platforms can communicate without any issues. Unlike web-based applications that use HTML to exchange data, web services employee XML. What's more, web applications depend on some technologies and platforms like ASP, and PHP but web services can work without any dependencies on other platforms or technologies. Figure 10 indicates a general structure of a web service that we outline how these components work together. There are two major web service components: WSDL, UDDI.

WSDL

WSDL stands for Web Services Description Language. WSDL is an XML-based document contains beneficial information such as the name and parameters of each method as well as how to access it. WSDL is a part of UDDI. It acts as an interface between web service applications.

UDDI

UDDI is an abbreviation for Universal Description, Discovery, and Integration. It is an XML-based framework for describing, discovering and integrating web services. It is a directory of web service interfaces addressed by WSDL and holds worthwhile information about web services. There are three important operations in web services architecture as follows:

1. Publish

To make a service available, its description must be published in such a way that other service requester can find it.

2. Find

In this operation, the service requester extracts the service description directly or by sending a request to the service registrar.

3. Bind

In this operation, the service requester employees the service description to communicate with others.

There are mainly two types of web services: SOAP and RESTful.

SOAP

SOAP is an acronym for Simple Object Access Protocol. It is a platform and language independent as well as simple and extensible. It applies an XML-based protocol to access web services and objects and exchanges messages through the protocol such as HTTP, IIOP, and SMTP over a network.

RESTful

REST shorts for REpresentational State Transfer. It is a software architecture style that compatible with a stateless communications protocol and the most commonly used protocol like HTTP. Some basic HTTP REST requests are: POST, GET, PUT, and DELETE.

It exploits some data types like plain text, HTML, XML, JSON, and etc. Furthermore, it can apply other formats that are machine-readable, although usually, the JSON format is most popular. It supports object-oriented programming paradigms. Normally, people recognize them as RESTful API or RESTful web services which can be utilized interchangeably. It can apply SOAP web services because of compatibility with some protocols like HTTP, and SOAP. Additionally, it uses Uniform Resource Locator (URI) to expose business logic. Unlike SOAP web services, REST requires less bandwidth and resources. Several architectural properties of the REST are stated as follows:

- Performance in component interactions, which can be the leading factor in network efficiency.
- High scalability means supporting the large numbers of components that would be able to interact.
- The modifiability of components is relatively simple (even while the application is performing)
- Visibility of communication between components by service agents
- Portability of components by moving program code alongside the data

Reliability in the resistance to failure at the system level in the presence of failures within components, connectors, or data [**22**].

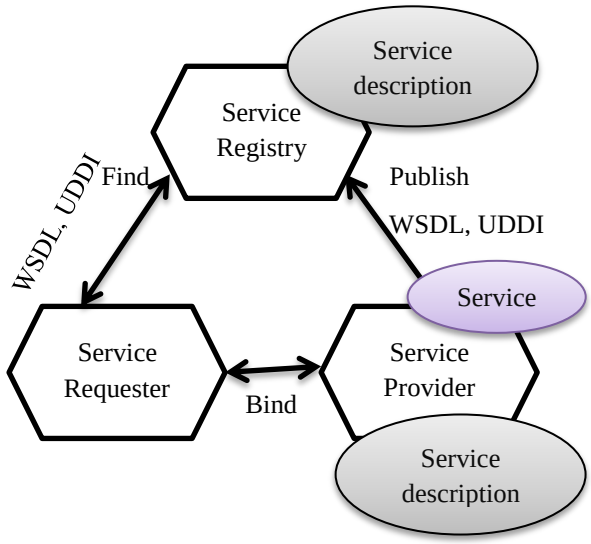

Fig.10. web service structure [**23**]

B.1.8 .NET Remoting

It is a Microsoft Application Programming Interface (API) for IPC that developed in 2002 with the version 1.0 of .NET Framework. Like other RPC-based technologies such as CORBA and RMI, the .NET Remoting is sophisticated. The client and server can communicate by message passing through operating systems and network agents.

This topic is specific to a legacy technology that is retained for backward compatibility with existing applications and is not recommended for new development. Either distributed or multi-tier applications should now be developed using the Windows Communication Foundation (WCF). Because it uses no longer, it will not be argued in detail. If you would like to get more knowledge in this area you can see [**24**].

B.1.9 Message-Queue-Based

Message queue makes it possible in which several applications can communicate asynchronously without blocking them while waiting for the response from each other. For example, consider the sending of an email instead of calling someone, in the first option the person requires to be immediately available to speak on the phone, but in the second case, firstly the e-mails store in middle storage, next lets the recipient manages the messages when available, consequently the message delivery process would be guaranteed and the sender and recipient would not be blocked.

The characteristics and capabilities of messaging systems are comparatively standardized. To be more specific, diverse systems released by various providers may expose the same interface. Java Message Service (JMS) is a case in point which is a platform-neutral and a Java-based interface for messaging systems. Figure 11 depicts the general structure of a message-queue-based system.

Generally, there are two patterns in message-queue-based systems which would be described as follows:

Publish-subscribe:

It is a message-queue-based pattern where message sender named publisher is not responsible for managing and sending the messages directly to specific receivers which called subscribers but, instead messages are classified into different classes without aware of their subscribers, then subscribers receive the messages from the classes that are of their interests without knowledge of the publishers.

In this model, subscribers receive only a bunch of the total published messages. The process of choosing and processing the messages is called filtering. There are two common kinds of filtering: topic-based and content-based.

In a *topic-based* system, messages are put in "topics" or named logical channels. Subscribers will receive messages from the topics they have subscribed to. All subscribers will receive the same messages from the same topics. The publisher must define the classes of messages and determine which subscribers can subscribe.

With the Publish-Subscribe model, the sender never explicitly specifies the receiver, it never even knows if any receiver exists or not. Figure 12 shows a topic-based message-queue-based model.

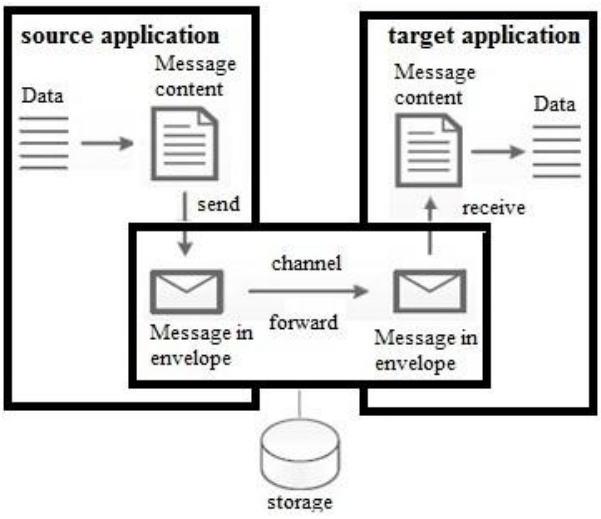

Fig.11. Data exchanging between applicationsthrough message-queue-based [**2**]

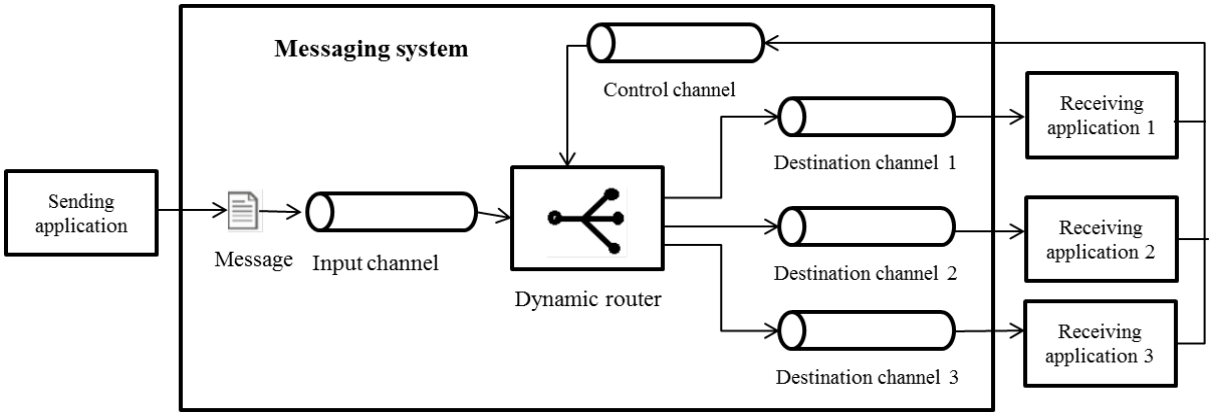

Fig.12. Dynamic routing of messages according to run-time rules, topic-based, and publish-subscribe [**2**]

In a *content-based* system, messages are only delivered to a subscriber that satisfies its constraints on the content of those messages as well as it is responsible for classifying the messages. Some systems are developed in a combination of the two ways which means publishers leave messages to a topic while subscribers may choose one or more topics, distinguish some keywords, and restrictions on the topics' contents (content-based). For example, consider a magazine publishing, each magazine is received by many people. In an airline departure board, all of the passengers can observe airline, departure time, destination, and etc. for each flight and everyone can choose information that is pertinent to himself or herself Take another example, in social networks groups or channels that usually type of communication is asynchronous (some members are offline), then messages persist when the receiver becomes available can get them. The general architecture of the content-based model is pictured in Figure 13.

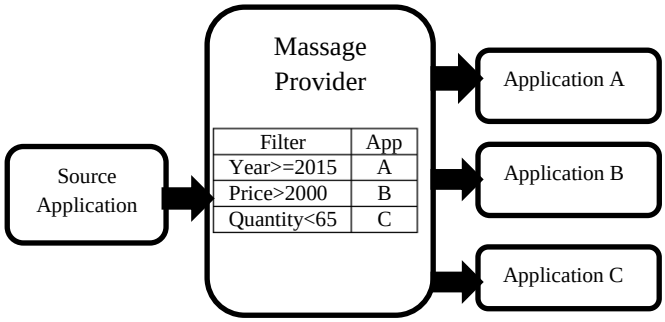

Fig.13. Content-based message-based model [**2**]

Point-to-Point

In this model each node leaves messages into a specific queue, then another one gets from it. To put it in another way, although one node can choose to send a message into multiple queues and only one node can get from each queue at the same time this policy indicates that the sender explicitly specifies the receiver. To give you an idea, consider the following examples:

Postcard: we can send multiple postcards to many people but each one can be received just by one person.

Email: although it might be sent to many people each person gets it from its own queue (Inbox). In this model, if I put a message into a queue, it will be shipped to just one receiver because each application has its own queue. In Figure 14 this model is illustrated.

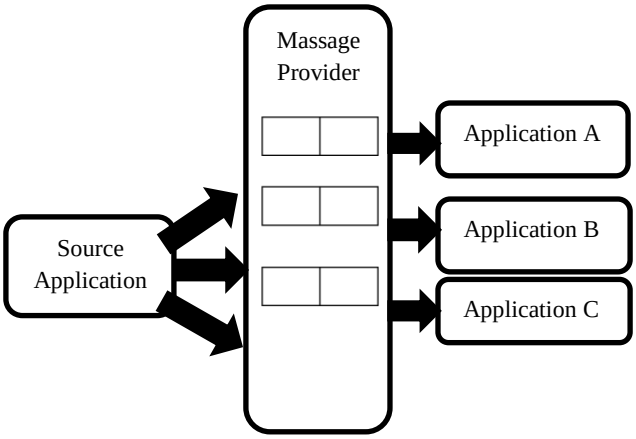

Fig.14. Point to point message-queue-based [**2**]

RPC-based communications that we mentioned above can be categorized into two board groups as follows:

1. Group Communication

RPC-based communication can have one-to-one communication (unicast), one-to-many communication (multicast), and one-to-all communication (broadcast). Multicasting can be implemented using broadcasting. Each system receives a message if the message does not belong to this machine then discard it. Highly available servers (client-server), database replication, multimedia conferencing, online games, cluster management, and to mention just a few are examples of group communication.

2. Stream-oriented communication

Streams can be established between two processes at different machines, or directly between two different devices. combinations are possible as well.

Stream-oriented communication is a form of communication in which time plays a crucial role. For instance audio and video stream. A data stream is nothing but a sequence of data units. There are different

transmission modes in stream-oriented communication that depicted as follows:

Asynchronous transmission mode

The data frames in a stream are sent one after another, but it does not matter when the transferring process is done entirely. As a point of clarification, a file transferred as a data stream.

Synchronous transmission mode

A maximum threshold delay is defined for each unit in a data stream. It is acceptable if one data unit transfers much faster than the delay threshold. For example, if a sensor pass sample temperature through a network and the dissemination time over the network is lower than the time interval between taking samples, it is satisfactory

Isochronous transmission mode

It is essential that data units are transferred on time. In another way, data units must transfer between a maximum and minimum end-to-end delay. By way of illustration, representing audio and video in order to preserve playback quality.

Normally, streams can be classified into two groups, simple or complex as follows:

Simple stream

It involves only a single sequence of data.

Complex stream

It consists of several related simple streams called sub-streams. The relationship between these sub-streams in a complex stream is often time-dependent. For instance, to transmit a movie, the stream made of a single video stream along with two streams to exchange the sound as well as a fourth stream might contain subtitles for deaf or a translation into a different language. For a detailed review see [**1**].

IV. Evaluations Of Some Ipc Methods

1. Experiments description

This section is devoted to performance assessment of some IPC methods such as CORBA, RMI, TCP socket, Web Service (SOAP), Web Service (RESTful), XML-RPC, and Message Provider or Message Queue (MQ).

These methods were implemented by Java programming language (a client and a server for each method) then performed on a system with the following configuration: RAM: 4GB, CPU: Intel(R) 2.4GHz, Operating System (OS): Windows 7-64bit (Version 6.1), Java Development Kit (JDK) version: 1.8.0_112, MQ: Apache ActiveMQ-5.15.5.

To assess the performance of the methods, six audio files with different size including (3, 8, 15.7, 26.6, 60.2, 341) MB were chosen randomly, then these files sent out one by one from the client to the server for each method in order to as the file size increases, we can distinguish the performance of the methods more precisely and easily. These experiments were conducted five times.

2. Experimental Results

Some of the observations are summarized as follows:

As we expected, the socket had a satisfactory performance compared to the others since it is the simplest IPC. The SOAP consumed much memory (for the file with size 341MB, we set Xmx=1500MB at the server side to resolve heap memory issues) and had poor performance besides REST. The CORBA was extremely slow to transfer the files. The RMI was superior to the CORBA, SOAP, and REST.

Because RMI uses stateful objects and object references as well as does not employ extra layers compared to SOAP, in comparison to SOAP and XML-RPC, it had a better performance. Other reasons for the poor performance of web services against RMI can be mentioned as follows: the sizes of the message that transferred over the network, and overhead which requires to processing the messages.

REST had better performance because it has a little overhead on top of HTTP. On the other hand, SOAP has different handlers and parsers to decode the message once each message is received. Consequently, REST was superior to SOAP in terms of transferring rate. Furthermore, REST is just HTTP, It has no overhead. It usually encodes the message in JSON (as opposed to XML in SOAP), consumed little memory, unlike SOAP.

CORBA utilized IIOP and because it is more convenient and more efficient to parse IIOP messages rather than parsing XML, it performs well. SOAP, on the other hand, convert all data to XML, then convert them back, it takes more processing time against IIOP from CORBA. We encountered this issue (parsing the XML caused the poor performance) in XML-RPC as well. SOAP and XML-RPC were terribly time-consuming and memory-consuming compared to CORBA.

When the size of the input file was raised, the amount of memory consumed in the XML-RPC method increased dramatically as well as its speed got slow. To tackle this issue we extend the Java memory heap space and set the Xmx to 2048MB in order to transmit the input file of size 341MB, otherwise, we encountered “memory not enough” or “heap memory shortage” error. A file was shipped by MQ as follows: firstly, the file was sent out from the client and sit in the MQ, afterward, the same file was gotten from MQ and forwarded to the server. It seemed, because the MQ used concurrent threads in a customized manner, in most cases, it had a very close performance with the socket. These results have been elaborated in Figure 15 through Figure 19. For all of the methods, execution time is in milliseconds (ms).

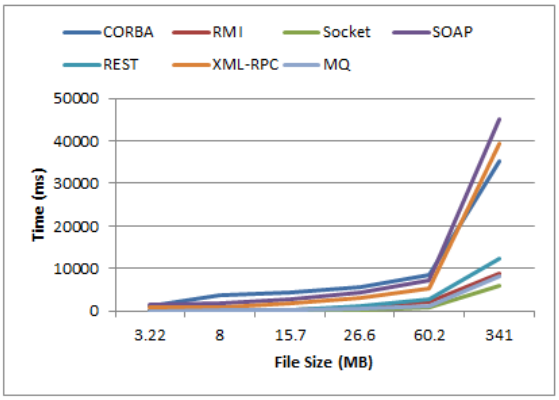

Fig.15. Experimental result 1

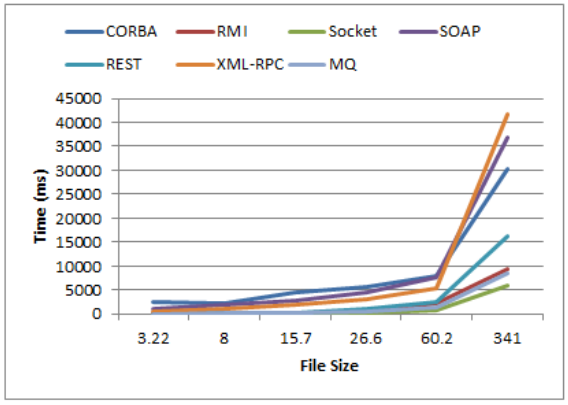

Fig.16. Experimental result 2

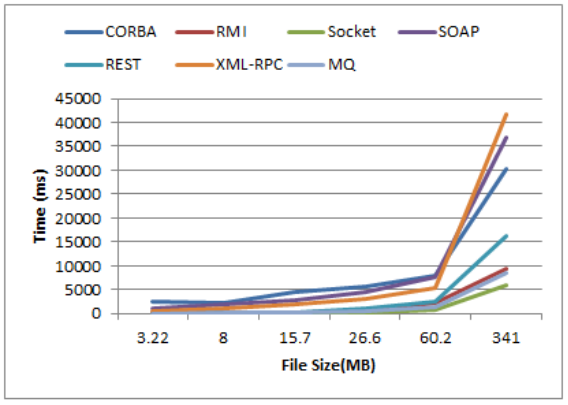

Fig.17. Experiment result 3

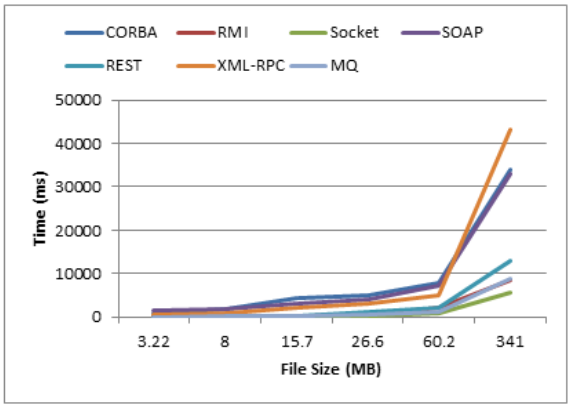

Fig.18. Experimental result 4

V. Conclusion And Future Works

To sum up, in this research firstly the main reasons that distributed systems have emerged, the importance and some of their basic concepts were described. Next, since IPC plays an outstanding role in the distributed systems, a pervasive and precise history of the IPC technologies, how and why they have been appeared, a comprehensive categorization (briefly, shared memory and message passing) alongside a brief explanation of them were stated. Afterward, some IPCs were implemented and compared. As we expected, the socket as a basic IPC method, because it does not perform extra operations on the input data to be sent, was the fastest method. Two methods XML-RPC and Web Service (SOAP) consumed a lot of memory. Express differently, XML-based IPCs had poor performance.

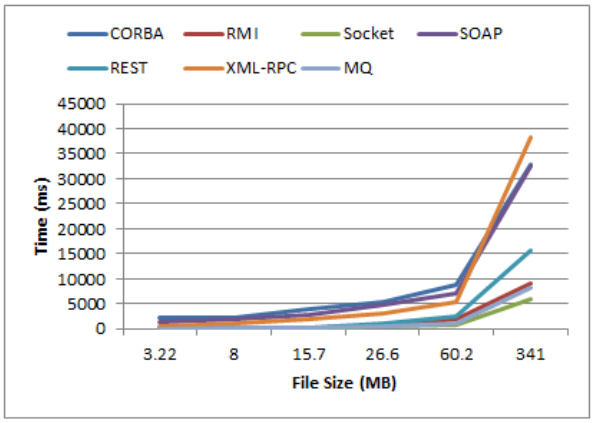

Fig.19. Experimental result 5

Because they perform marshaling and unmarshalling over the messages in order to eliminate some platform dependencies, and this processing requires a lot of time and memory. Since RMI uses object references it had satisfactory performance against XML-based IPCs and CORBA. Message-Queue-Based IPCs like ActiveMQ have made possible asynchronous communication and because they use multiple threads to communicate between processes, had acceptable performance. For future works, it is possible to conduct the benchmarks on different systems and evaluate the network traffics with regard to each method. Furthermore, because some systems utilize signals to notify each other and communicate, on the other hand, each operation has a priority to perform and usually use a specific signal to proceed. In this paper, we have not mentioned them because of their details. We would attempt to study them in the future as well.

In Table 1, Table 2, and Table 3 at the follows we attempt to illustrate a summarization of some IPCs alongside their advantages and disadvantages. Keep in mind that none of these methods are superior to others. As mentioned above, depending on diverse circumstances you should choose the most appropriate one.

Table 1. Advantages and disadvantages of some IPC methods

Methods		Strengths	Weakness
Sockets	TCP	- The ability to recover from any failures (reliability) - It can be added to a network without disturbing for other services - Handling the errors effectively - Independent of platform - Overhead of data is litter. - The sending and receiving of the packets are guaranteed -Simple implementation, client and server can be implemented in two different languages. - It is not required to know the procedural details of the server application.	- It is made for Wide Area Networks (WAN), so it has weak performance across the networks with poor resources. -It performs various operations on the layers concurrently, so it causes low speed over the network.
	UDP	- Because it uses a small packet alongside small header size, it needs less time and memory to process. -Because the packets in UDP have not ACK, and also do not require to keep in memory to be confirmed, their speed is high and memory consumption is low. -It does not require to resend the missing packets, so it is a suitable option for applications that are sensitive to delay like audios and videos. -It has a good transferring rate. - Simple implementation, client and server can be implemented in two different languages. - It is not required to know the procedural details of the server application.	-It is an unreliable protocol and there is no guarantee to make sure all of the packets received or keep in the same order. - Because of error control deficiency, if it detects any errors in received packets, it eliminates them. - It is suffering from congestion control when a large number of users transfer huge amounts of data by UDP, it causes congestion and no one knows how to manage it. - It has not the order convention, and also no ACK message for received data. - Because only the application layer is responsible for error recovery, under those circumstances, applications can inform the user to retransmit the message. - Routers do not send a UDP datagram after a collision and usually remove UDP packets before TCP ones
RMI		- Dividing processors' loads - High transferring rate - An ability to pass executable files. - You can modify or add some classes easily. - RMI is good for simple method calls (like queries) - The marshaling of objects is automatic - It will give you better performance over web services.	- It is Java's technology, so the client and server must be written in Java language. - Called and caller methods must be performing when communicating. -Overhead of marshaling and unmarshaling - Overhead of object serialization. - Reflection in RMI seems to be more expensive than the HTTP protocol.
Web Service		-Unlike RMI, the client and server could be written in two different languages. -It works over HTTP and port 80/443 which are normally not blocked by firewalls and can work behind Network Address Translation (NAT). - It is much easier to debug web services over the wire because the data can be easily captured via sniffing tools. - Web services will probably be more maintainable and flexible for future requirements. - Integrating with other enterprise components like Enterprise Service Bus (ESB), Single Sign-On (SSO), Identity Management, load balancing, security filters, and security certificates is fairly easy.	- It has very low speed - It is not changeable if you write a web service, it does not allow you to change the parameters and methods. You could add new changes as new methods, but if you want to change the methods and input parameters, the customer programs do not work properly.
CORBA		-Supporting a wide range of languages like Java, C, C++, Python, Smalltalk, Ada, COBOL, PL/I, LISP. -It integrates with other technologies easily -Supporting numerous operating systems including, UNIX, Windows, AS/400, Open VMS, Apple's OS X, a diverse range of capabilities such as dealing with transactions, security, Naming, messaging and publish-subscribe services - By employing marshaling in a compact format, making it easier exchanging of the data because other applications utilize similar formats to transfer. - A convenient choice for enterprise applications - High scalability which means the flexibility and server-side architecture of CORBA permit developing the servers that can be scaled to handle a huge number of objects	-It has a complicated structure. -Downloading process via CORBA is time-consuming -It has no standard mapping for Perl language. Although some people might believe that Perl is not used any longer, it still uses in some countries. - It does not play well with firewalls. -It has no standard to manage the life cycle of objects.

DCOM		-By marshaling and elimination of the dependencies, it allows data to be exchanged from one COM object instance to another on a different computer. -It has rich built-in capabilities that make it possible to communicate with other middleware products such as Microsoft Message Queue Server (MSMQ). -It lets to implement components in any language. -Having an ability to perform on any platform such as UNIX, Linux, SUN, and OSX. -Because of the garbage collector, it supports the networks with immense traffic and would be able to remove unnecessary or completed objects on the server.	-DCOM did not succeed to become a standard protocol -It is not compatible with disconnected environments -It is complicated to code in C++. - It does not work well through firewalls.
Message Queue		- Asynchronous communication - Performance improves: it allows asynchronous communication which means the endpoints negotiate with the queue, not each other. The producer can put messages into the queue without waiting to be consumed and the consumer gets messages only when they are available. No part in the systems is blocked or waited for another. As a result, the performance of the system improves. - It has high reliability: since queue makes the data persistent, if some errors occur in some components of the program or goes down for any reason, by splitting various components with a message queue, your program becomes more fault- tolerant because the data in the queue are not lost, other components that are reachable can interact with queue without any problem. - High scalability: When the workloads of your system increases, multiple instances of an application can put their requests to a queue, then you can distribute the workloads among diverse consumers. - It makes simple the decoupling: message queues eliminate dependencies between components and provides an easy way to decouple the components of applications which means each component has its own business function specifically when you utilize micro-services architectures. - Separating the Apps: because message queue and micro-services architecture make easier the decoupling of code and business function of components as well, consequently testing, and also debugging, is more straightforward. -Using micro-services: because micro-services patterns are connected with events, It is easier through message queue to route multiple services as well as notify them when the data change. -Message-driven processing: you can start and stop a specific application by triggering depending on a message received on a queue or processed -Event-driven processing: relying on any event that might occur in the queue you can conduct different actions.	-Operational complexity (every queue must be created, configured, and monitored) that are very tedious actions. -Only the sender is guaranteed that the message would be sent. But, information about the delivery time does not provide to the sender.
XML-RPC		- It is easy to implement. - There are a wide number of libraries that you can use easily. -it is based on XML, which is a widely used language. -It uses the HTTP protocol, which makes it possible to work with a firewall perfectly.	- It is memory-consuming - It is time-consuming - It has poor packet validation.

Table 2. Shared Memory vs. Message Passing

Method	Advantage	Disadvantage
Shared Memory	-Communication is fast because there is no overhead related to system calls. -Memory mapping of a file, improves I/O performance, chiefly on large files. -it can support message passing like pipes.	-It needs concurrency control mechanisms and memory protection which leads to complications in programming because the programmers have to make sure to control all the critical regions effectively. -It does not support the persistence of data which means if the system crash for any reason, the data are lost. - When the number of processors in the machine are ever-increasing, it progressively makes it difficult and expensive to construct shared memory.
Message Passing	It does not require concurrency control mechanisms like semaphores, which results in performance improvement.	Transferring of large files over the busy network are time-consuming.

Table 3. SOAP vs REST

No.	**SOAP**	**REST**
1	SOAP is a protocol.	REST is an architectural style.
2	SOAP stands for Simple Object Access Protocol.	REST shorts for REpresentational State Transfer.
3	SOAP can't apply REST because it is a protocol.	REST can utilize SOAP web services because it is a concept and can use any protocol like HTTP, SOAP.
4	SOAP uses service interface to expose business	REST uses URI to expose business logic.
5	JAX-WS is the Java API for SOAP web services.	JAX-RS is the Java API for REST web services.
6	SOAP defines standards to be strictly followed.	REST does not define too many standards.
7	SOAP needs more bandwidth and resources	REST requires less bandwidth and resources
8	SOAP permits only XML data format.	REST allows different data formats such as Plain text, HTML, XML, JSON, etc.

REFERENCES

[1] Tanenbaum, Andrew S and V.Steen, Maarten, Distributed systems: principles and paradigms. Prentice-Hall, 2007.

[2] M. Fowler, Patterns of enterprise application architecture. Addison-Wesley Longman Publishing Co., Inc., 2002.

[3] Coulouris, George F and Dollimore, Jean and Kindberg, Tim, Distributed systems: concepts and design. pearson education, 2005.

[4] Peterson LL, Davie BS, Computer networks: a systems approach. Elsevier, 2007.

[5] Tanenbaum, Andrew S., and David Wetherall, Computer networks. Harlow, Essex: Pearson, 2014.

[6] D.Ruby, Ms. S.Krishnaveni and Ms., "Comparing and Evaluating the Performance of Inter Process Communication Models in Linux Environment," lnternational Journal of Trend in Research and Development (IJTRD), pp. 51-55, Sep. 2016.

[7] Gokhale, Aniruddha and Kumar, Bharat and Sahuguet, Arnaud, "Reinventing the wheel? CORBA vs. Web services," in Proceedings of international world wide Web conference, 2002.

[8] Gray, Neil AB, "Comparison of Web Services, Java-RMI, and CORBA service implementations," in The Fifth Australasian Workshop on Software and System Architectures, Australasian, 2004, p. 52.

[9] S.Ghodake, A. R. Buchade, "Survey on Interprocess Communication and Managment," International Journal of Innovative Research in Computer and Communication Engineering, vol. 5, no. 2, pp. 1511-1515, Feb. 2017.

[10] Patil, Abhishek and Korde, Rajesh and Sabharwal, Kapil, "Comparison of Middleware Technologies-CORBA, RMI & COM/DCOM," Citeseer.

[11] ZHANG, Xiurong, "The Analysis and Comparison of Inter-Process Communication Performance Between Computer Nodes," Management Science and Engineering, vol. 5, no. 3, 2011.

[12] Cook, William R and Barfield, Janel, "Web service versus distributed objects: A case study of performance and interface design," International Journal of Web Services Research (IJWSR), vol. 4, no. 3, pp. 49-64, 2007.

[13] Davis, Dan and Parashar, Manish P, "Latency performance of SOAP implementations," in Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on, 2002, pp. 407-407.

[14] Venkataraman, Aditya and Jagadeesha, Kishore Kumar, "Evaluation of inter-process communication mechanisms," Architecture, vol. 86, p. 64, 2015.

[15] Vogels, Werner, "Web services are not distributed objects," IEEE Internet computing, vol. 7, no. 6, pp. 59-66, 2003.

[16] Ahalt, Mark D Hanes Stanley C and Krishnamurthy, Ashok K, "A Comparison of Java RMI, CORBA, and Web Services Technologies for Distributed SIP Applications," HPEC, 2002.

[17] Juric, Matjaz B and Kezmah, Bostjan and Hericko, Marjan and Rozman, Ivan and Vezocnik, Ivan, "Java RMI, RMI tunneling and Web services comparison and performance analysis," ACM Sigplan Notices, vol. 39, no. 5, 2004.

[18] Lynch, Nancy and Shvartsman, Alex, "Communication and data sharing for dynamic distributed systems," in Future directions in distributed computing, 2003, pp. 62-67.

[19] Juric, Matjaz B and Rozman, Ivan and et al., "Comparison of performance of Web services, WS-Security, RMI, and RMI-SSL," Journal of Systems and Software, vol. 79, no. 5, pp. 689-700, 2006.

[20] Allman, Mark, "An evaluation of XML-RPC," SIGMETRICS Performance Evaluation Review, vol. 30, pp. 2-11, 2003.

[21] Carl-Fredrik Sørensen, "A Comparison of Distributed Object Technologies".
[22] Roy T Fielding, Richard N Taylor, "Architectural styles and the design of network-based software architectures," vol. 7, Jun. 2000.
[23] Kreger, By Heather, "Web Services Conceptual Architecture (WSCA 1.0)," 2001.
[24] Scott McLean, James Naftel, Kim Williams, Microsoft .NET Remoting. 2002.

Author's Profie

Mr. Hamed Dinari was born in Abdanan, Ilam, located in the west of IRAN. He is an M.Sc. graduate in Computer Engineering (Software) from the Department of Computer Engineering (CE), Iran University of Science and Technology (IUST), Tehran, IRAN. His research interests lie primarily in the area of Database Systems, Data Mining, Graph Mining, Indexing, and Distributed Systems. He is currently working as a Software Engineer and Enterprise Application Developer. In his free time, he would like to listen to music and study psychology and linguistics books.

How to cite this paper: Hamed Dinari, " Inter-Process Communication (IPC) in Distributed Environments: An Investigation and Performance Analysis of Some Middleware Technologies", International Journal of Modern Education and Computer Science(IJMECS), Vol.12, No.2, pp. 36-52, 2020.DOI: 10.5815/ijmecs.2020.02.05

