A Study on Middleware Technologies in Cloud Computing

M. Saranya
Research Scholar
Department of Computer Science
Muthayammal College of Arts & Science, Namakkal, Tamilnadu.

A. Anusha Priya
Associate Professor
Department of Computer Science
Muthayammal College of Arts & Science, Namakkal, Tamilnadu.

Abstract

This paper Middleware connectivity software presents that where all it provides a mechanism for processes to interact with other processes running on multiple networked machines. It Bridges gap between low-level OS communications and programming language. Middleware Application Programming Interfaces provide a more functional set of capabilities than the OS and a network service provide on their own and also hides complexity and heterogeneity of distributed system. Middleware-oriented R&D activities over the past decade have focused on the identification, evolution, and expansion of understanding current middleware services and the need for defining additional middleware layers and capabilities to meet the challenges associated with constructing future network-centric systems. It contains the more different technologies that can be composed, configured, and deployed to create distributed systems rapidly and it working with the cloud computing environment.

Keywords: Middleware Technologies, Cloud Computing Environment, Distributed System

I. INTRODUCTION

Middleware is an important class of technology that is serving to decrease the cycle-time, level of effort, and complexity associated with developing high-quality, flexible, and interoperable distributed systems. When implemented properly, middleware can help to Shield developers of distributed systems from low-level, tedious, and error-prone platform details, such as socket-level network programming. Middleware was invented in an attempt to help simplify the software development of distributed computing systems, and bring those capabilities within the reach of many more developers than the few experts at the time who could master the complexities of these environments. Complex system integration requirements were not being met from the application perspective, where it was too hard and not reusable, or the network or host operating system perspectives, which were necessarily concerned with providing the communication and end system resource management layers, respectively. One also finds business intelligence, content and collaboration tools, as well as portal capabilities that allow connections to customers and partners enabled at the middleware level, middleware and middleware-based architectures. Middleware is systems software that resides between the applications and the underlying operating systems, network protocol stacks, and hardware.

Its primary role is to

1) Functionally bridge the gap between application programs and the lower-level hardware and software infrastructure in order to coordinate how parts of applications are connected and how they interoperate and
2) Enable and simplify the integration of components developed by multiple technology suppliers.

II. USAGE OF MIDDLEWARE

The desktop appliance can be computer or devise like computer e.g. a terminal, personal computer, workstation, word processor etc. The utility is an enterprise wide network of information services which includes applications, databases on LAN and WAN. Servers on LAN support files and file based applications, such as e-mail, bulletin board, document preparation and printing.

Middleware deals with providing environments for mounting systems that can be distributed effectively over a variety of topologies, computing devises and communication network. It aims to provide developers of networked applications with the required platform and tools to

- Formalize and coordinate how parts of applications are composed and how they interoperate.
- Monitor, enable and validate the configuration of resources to ensure appropriate application service quality, in case of failure also.
- Network communication

III. MIDDLEWARE ARCHITECTURE

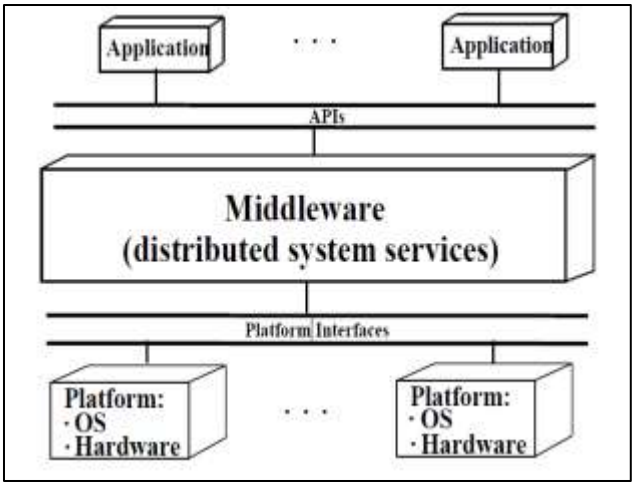

Fig. 1: Middleware Architecture

A. *Types of Middleware*

1) DOC Middleware:

Some of the most victorious of these technologies have centered on distributed object computing (DOC) middleware. DOC is an advanced, full-grown, and field-tested middleware paradigm that supports flexible and adaptive behavior. DOC middleware architectures are composed of relatively autonomous software objects that can be distributed or collocated throughout a wide range of networks and interconnect. Clients call upon operations on target objects to perform interactions and invoke functionality needed to attain application goals.

Distribution middleware enable clients to program distributed applications much like stand-alone applications, i.e., by invoking operations on target objects without hard-coding dependencies on their location, programming language, OS platform, communication protocols and interconnects, and hardware.

2) Benefits of DOC Middleware

Middleware originated because the problems relating to integration and construction by composing parts were not being met by either

1) Applications, which at best were customized for a single use,
2) Networks, which were necessarily concerned with providing the communication layer, or
3) Host operating systems, which were focused primarily on a single, self-contained unit of resources.

B. *Remote Procedure Calls*

RPC is a client/server mechanism that allows the program to be distributed across multiple platforms.it reduce the complexity of a system that spans multiple operating systems and network protocols by hiding OS and network interface details from the programmer. RPC's are usually implemented by proprietary products, proprietary development tools that create client server stubs. When the software in question is written using object-oriented principles, RPC may be referred to as remote invocation or remote method invocation. Client makes calls to procedures running on remote systems, which can be asynchronous or synchronous. E.g. DCE RPC.

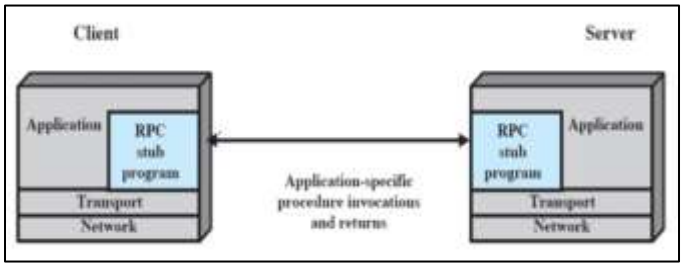

Fig. 2: RPC Middleware Architecture

1) Properties of RPC

- Language-level pattern of function call easy to understand for programmer.
- Synchronous request/reply interaction natural from a programming language point-of-view matches replies to requests built in matching of requests and replies.

– Distribution transparency (in the no-failure case) hides the complexity of a distributed system.
– Various reliability guarantees deals with some distributed systems aspects of failure.

2) *Failure Modes of RPC*

Invocation supported by RPC in the light of network and/or server congestion, client network and/or server failure.

3) *Disadvantage*

Synchronous request/reply interaction tight coupling between client and server. Client may block for a long time if server loaded lead to multi-threaded programming at client slow/failed clients may delay servers when replying multi-threading essential at servers. It invokes functions on servers as opposed to methods on objects.

– Distribution Transparency
– Not possible to mask all problems
– RPC paradigm is not object-oriented

C. *Database Middleware*

Database middleware allows direct access to data structures and provides interaction directly with databases. There are database gateways and a variety of connectivity options. Extract, Transform, and Load (ETL) packages are included in this category. E.g. CRAVE is a web-accessible JAVA application that accesses an underlying MySQL database of ontology via a JAVA persistent middleware layer.

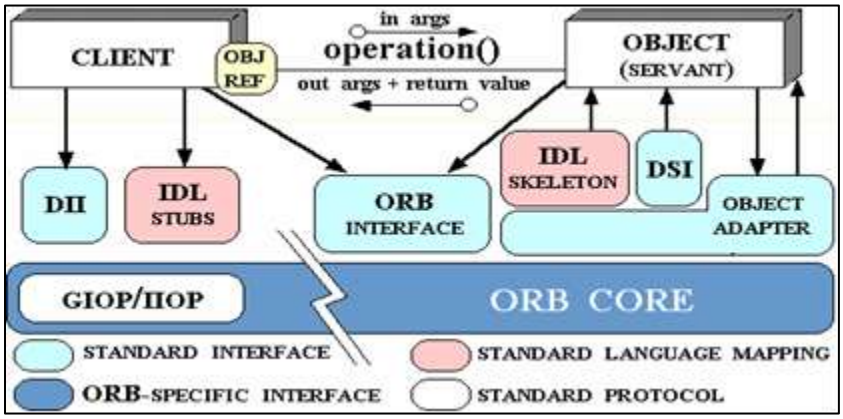

Fig. 3: CORBA Components

CORBA Interface Definition Language (IDL)Stubs (proxies) and skeletons created by IDL compiler. Dynamic remote method invocation. Interface Repository Querying existing remote interfaces. Implementation Repository activating remote objects on demand.

1) *CORBA IDL*

Definition of language-independent remote interfaces Language mappings to C++, Java, Smalltalk. It Translate by IDL compiler. The Type system basic types: long (32 bit), long long (64 bit), short, float, char, boolean, octet. Constructed types are struct, union, sequence, array, enum objects(common super type Object).

2) *CORBA Services*

– Naming ServiceNames
  - remote object references
– Trading ServiceAttributes (properties)
  - remote object references

Implementation of persistent CORBA objects. Transaction Service Making object invocation part of transactions. Event Service and Notification Service In response to applications‘need for asynchronous communication built above synchronous communication with push or pull options .notan integrated programming model with general IDL messages.

3) *Disadvantages of OOM*

– Synchronous request/reply interaction only so CORBA one way semantics added and Asynchronous Method Invocation (AMI) but *implementations* may not be loosely coupled.
– Distributed garbage collection Releasing memory for unused remote objects.
– OOM rather static and heavy-weight Bad for ubiquitous systems and embedded devices

D. *Message-Oriented Middleware (MOM)*

MOM provides asynchronous communication between client and server applications by queuing messages temporarily when one or the other is busy or not connected. The functionality similar to RPC.APIs that extends across assorted platforms and networks are typically provided by the MOM. MOM is software that resides in both portions of client/server architecture and typically ropes asynchronous calls between the client and server applications. Messages stored in message queues Message queues grant temporary

storage when the destination program is busy or not connected. MOM reduces the involvement of application developers with the complexity of the master-slave nature of the client/server mechanism. E.g. Sun's JMS.

1) *Properties of MOM*

Asynchronous interaction, Client and server are only loosely coupled and Messages are queued. It Good for application integration and Support for reliable delivery service. Keep queues in persistent storage. Processing of messages by intermediate message server. Natural for database integration

2) *Queue Managers*

Queue Managers Responsible for queues and transfer messages from input to output queue and also update the routing information in routing tables.

3) *Message Channels*

Message Channels are Reliable connections between queue managers.

4) *Java Message Service (JMS)*

API specification to access MOM implementations and modes of operation *specified*:Point-to-point one-to-one communication using queues..JMS Server implements JMS API.JMS Clients connect to JMS servers. Java objects can be serialized to JMS messages. A JMS interface has been provided for MQ.

5) *Disadvantages of MOM*

- Poor programming abstraction
- Request/reply difficult to achieve

Message formats originally unknown middleware. Queue abstraction only give one communication limits scalability.

E. Web Services

Web Services are well-known web standards for distributed computing Communication.

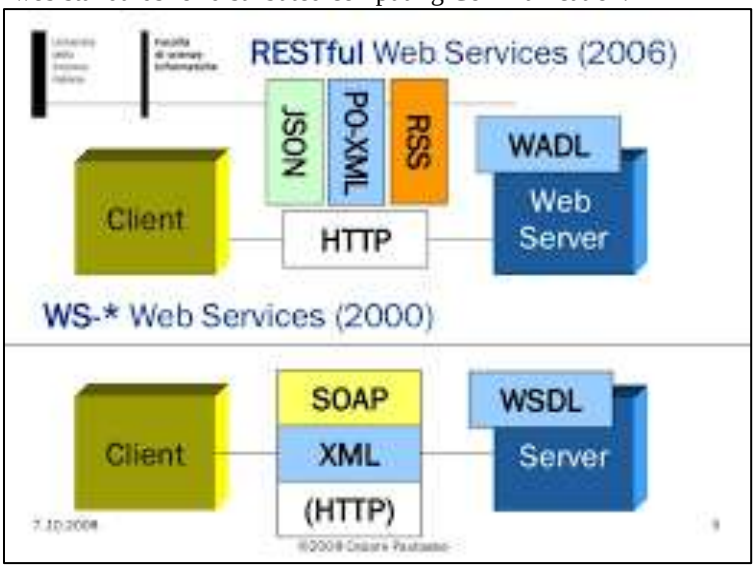

Fig. 4: Web Service Architecture

Message content expressed in XML.Simple Object Access Protocol (SOAP) Lightweight protocol for sync/async communication

1) *Service Description*

Web Services Description Language (WSDL) Interface description for web services

2) *Service Discovery*

Universal Description Discovery and Integration (UDDI)Directory with web service description in WSDL

3) *Properties of Web Services*

- Language-independent and open standard
- SOAP offers OOM and MOM-style communication:
  - Synchronous request/reply like OOM
  - Asynchronous messaging like MOM

F. Content Centric Middleware

This type of middleware allows you to abstract specific content without worrying how it is obtained. This is done through a simple provide / consume abstraction. It is similar to publish / subscribe middleware, which is another type of this software that is often used as a part of web-based applications.

G. Cloud Middleware:

Datacenters running a cloud environment often enclose a large number of machines that are connected by a high-speed network. Users access sites hosted by the cloud environment through the public Internet. A site is typically accessed through a URL that is translated to a network address through a global directory service, such as DNS. A request to a site is routed through the Internet to a machine inside the datacenter that either processes the request or forwards it.

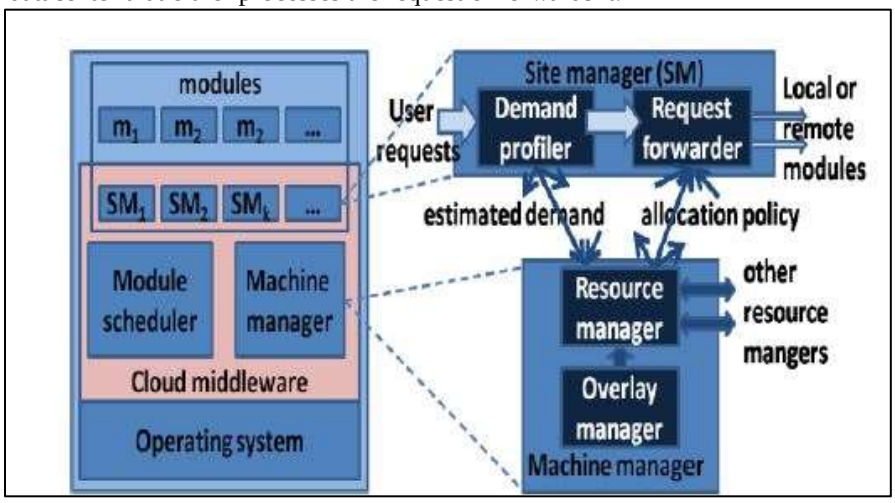

Fig. 5: Cloud Middleware

IV. EXAMPLES OF MIDDLEWARE

The term middleware is used in other contexts as well. Middleware is sometimes used in a similar sense to a software driver, an abstraction layer that hides detail about hardware devices or other software from an application.

1) The Android environment uses the Linux operating system at its core, and also provides an application framework that developers incorporate into their applications.
2) The Android middleware layer also contains the Dalvik virtual machine and its core Java application libraries.
3) Game engine software such as Gamebryo and Render ware are sometimes described as middleware, because they provide many services to simplify game development.
4) In simulation technology, middleware is generally used in the context of the high level architecture (HLA) that applies to many distributed simulations.
5) The QNX operating system offers middleware for providing multimedia services for use in automobiles, aircraft and other environments.
6) Multimedia Home Platform (DVB-MHP) is an open middleware system standard designed by the DVB project for interactive digital television.

V. CHALLENGES AND OPPORTUNITIES

An increasing number of next-generation applications will be residential as distributed "systems of systems," which include many interdependent levels, such as network/bus interconnects local and remote end systems, and multiple layers of common and domain-specific middleware. The advantageous properties of these systems of systems include predictability, controllability, and adaptability of operating characteristics for applications with respect to such features as time, quantity of information, accuracy, confidence, and synchronization. All these issues become highly volatile in systems of systems, due to the dynamic interplay of the many interconnected parts. These parts are often constructed in a similar way from smaller parts. An essential part of what is needed to build the type of systems outlined above is the integration and extension of ideas that have been found traditionally in network management, data management, distributed operating systems, and object-oriented programming languages.

VI. CONCLUSION AND FUTURE WORK

Middleware has become inseparable part of the almost all applications now days. This is mainly due to widespread use of Information Technology and the emerging trend of distributed computing. Middleware's are user friendly, requires no additional language learning. Although current middleware solves a number of basic problems with distribution and heterogeneity, many challenging research problems remain. In particular, problems of scale, diversity of operating environments, and required level of trust in the sustained and correctly functioning operation. These activities are expected to continue forward well into this decade to address the needs of next-generation distributed applications. In future it can be applicable in an increasing number of real-world

applications, such as e-commerce web sites, consumer electronics, avionics mission computing, hot rolling mills, command and control planning systems, backbone routers, and high-speed network switches.

REFERENCES

[1] Loyall JL, Gossett JM, Gill CD, Schantz RE, Zinky JA, Pal P, Shapiro R, Rodrigues C, Atighetchi M, Karr D. "Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications". Proceedings of the 21st IEEE International Conference on Distributed Computing Systems (ICDCS-21), April 16-19, 2001.
[2] Bernstein P., "Middleware An Architecture for Distributed System Services," CACM, 39:2, March 2, 1993.
[3] Bernstein, P., "Middleware, A Model for Distributed System Service"", Communications of the ACM, 39:2, February1996
[4] A.S.Syed Navaz, C.Prabhadevi & V.Sangeetha"Data Grid Concepts for Data Security in Distributed Computing" January 2013, International Journal of Computer Applications, Vol 61 – No 13, pp 6-11.
[5] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregationin large dynamic networks," ACM Trans. Computer Syst., vol. 23, no. 3, pp. 219–252, 2005.
[6] Schmidt D., Kuhns F., "An Overview of the Real-time CORBA Specification," IEEE Computer Magazine, June,
[7] Object Management Group, "Dynamic Scheduling Real-Time CORBA 2.0 Joint Final Submission," OMG Document orbos/2001-04-01.
[8] F. Wuhib, R. Stadler, and M. Spreitzer, "Gossip-based resource management for cloud environments," in 2010 International Conference on Network and Service Management.
[9] A Anusha Priya, R Gunasundari, Securing Data on the Cloud Server by the User Authentication and Data Security Techniques, 2017, International Journal of Computer Applications, Volume 165, Issue 4.
[10] A.Anusha Priya, Lavanya.C, Enhanced Focus on User Revocation in Secure Dynamic Auditing For Data Storage in Cloud, 2016/8, International Journal of Emerging Technology in Computer Science & Electronics, Volume 23, Issue 4, Pages 51-55.
[11] S.Yasmin, A.Anusha Priya, Decentralized Entrance power with Secret Endorsement of data Stored in Clouds, 2015/8, International Journal of Innovative research in Computer and Communication Engineering, Volume 3, Issue 8, Pages 7279-7284.
[12] V.Yamuna, A.Anusha Priya, Efficient and Secure Data Storage in Cloud Computing RSA and DSE Function, 2015/7, International Journal of Innovative Research in Computer and Communication Engineering, Volume 3, Issue 7, Pages 6758-6763.
[13] M.Balaji, E.Aarthi, K.Kalpana, B.Nivetha, D.Suganya "Adaptable and Reliable Industrial Security System using PIC Controller" 2017/5, Journal International Journal for Innovative Research in Science & Technology, Volume 3, Issue 12, Page 56-60.
[14] A.S.Syed Navaz, V.Sangeetha & C.Prabhadevi, "Entropy Based Anomaly Detection System to Prevent DDoS Attacks in Cloud" January 2013, International Journal of Computer Applications, Vol 62 – No 15, pp 42-47.
[15] A.S.Syed Fiaz, N.Asha, D.Sumathi & A.S.Syed Navaz "Data Visualization: Enhancing Big Data More Adaptable and Valuable" February – 2016, International Journal of Applied Engineering Research, Vol No - 11, Issue No - 4, pp.–2801-2804.
[16] A Anusha Priya, A Mohanapriya, An Effective Scrutiny of Static and Dynamic Load Balancing In Cloud, August 2016, International Journal of Emerging Technology in Computer Science & Electronics, Volume 23 Issue 4, Pages – 27 -30.