PERSISTENT SOFTWARE ATTRIBUTES AND ARCHITECTURE FOR DISTRIBUTED APPLICATION

Ravi Uyyala, RaghuRam Battini,Kundan Kumar Mishra

Abstract: Software quality attributes are a collection of closely related behaviors. These attributes can increase software application or system's value and gives product differentiation when added properly. Persistent software attributes similar to quality-of-service guarantees: they both require that specified functional properties remain unchanged during operational use. The main difference between them is that quality-of-services requirements focus on immediate operational properties, such as processing capacity and latency, where as persistent software attributes focus on the resilience of long-term software engineering properties such as scalability, reliability, adaptability, and maintainability. Software architects will select off-the-shelf components on the basis of the specific persistent properties they can contribute to large systems, not just on functionality. To reduce the system development and maintenance cost, it is requested to make the system modular using off-the-shelf components. The growth of the Internet has led to explosions in volumes of e-mail traffic and in number of the users of e-mail service. At the same time, large-scale e-mail service providers have appeared. We used the internal structure of a typical Internet mail system for single server to our email system. We designed and implemented email system and achieved all the design objectives like scalability, availability, and extensibility.

Keywords: Common Object Request Broker Architecture (CORBA), Domain Name System (DNS), Internet Message Access Protocol (IMAP), Mail Delivery Agent (MDA), Mail Retrieval Agent(MRA), Mail Transfer Agent(MTA), Mail User Agent(MUA).

1 .Introduction

Software quality attributes are a collection of closely related behaviors. These attributes can increase software application or system's value and gives product differentiation when added properly. Examples of software quality attributes are maintainability, reliability, usability, efficiency, adaptability, availability, security, portability, scalability, safety, fault-tolerance, testability, reusability and sustainability [5]. These attributes have no value as a stand-alone item. These software quality attributes suffer from the direct measurement. Software quality measurement can be subjective and not absolute. Without serious definition and consideration of the quality attributes during the initial project initiation phases, any hope that the software product will ultimately satisfy its end users' expectations is small. In a few decades, software has expanded from its homey slots within isolated computers to vast distributed systems. We should keep reasonable cost when adding software quality attributes to software product. In this new software engineering, people will select off-the-shelf components on the basis of the specific persistent properties they can contribute to large systems, not just on functionality.

The new software engineering will more complex, but it will also lower costs and increase capabilities the way that good house construction does: by placing the right properties at the right real-space locations. Ironically, bringing awareness of real space issues into software design will also bring software engineering closer to its roots in older forms of engineering, ones that haven't had the luxury of neglecting a concept as basic as where components are located in space [6]. Making software truly adaptable is one of software design's trickiest challenges, requiring both unnatural skill for estimating how software will change over time and an artistic ability to express change needs in terms that later users can easily understand.

Constructing persistent software attributes will necessarily be more automated, complex, dynamic and geographically distributed than constructing simple software quality attributes. Everyone is writing software, it's time for software engineering to take up the challenges of architectural leadership. Persistent software attributes similar to quality-of-service guarantees: they both require that specified functional properties remain unchanged during operational use. The main difference between them is that quality-of-services requirements focus on immediate operational properties, such as processing capacity and latency, where as persistent software attributes focus on the resilience of long-term software engineering properties such as scalability, reliability, adaptability, maintainability and portability, etc[1].

Architecture of distributed application is depends on type of application. It defines the agreements through which they interact with external components and with each other. Basically there are two architectures: two-tier (client/server) and three-tier, also commonly called as n-tier. Client-server describes the relationship between two computer programs in which one program, the client program, makes a service request to another, the server program. Standard networked functions such as email exchange, web access and database access, are based on the client-server model. In software engineering, n-tier architecture (often referred to as multi-tier architecture) is a client-server architecture in which the presentation, the application processing, and the data management are logically separate processes. For example, an application that uses middleware to service data requests between a user and a database employs multi-tier architecture. The most widespread use of “n-tier architecture” refers to three-tier architecture.Persistent is the ability to have information from an application instance existing for later instances of the application (or even other applications) to use [2]. Scalability is the ability of computer application or product (either hardware or software) to continue to function well when it (or its context) is changed in size or volume in order to meet a user’s need.

2. Literature Survey

The software attributes are a collection of closely related behaviors that by themselves have little or no value to the end users but that can greatly increase a software application or system’s value when added. Examples of software attributes include:

Maintainability: In software engineering, the ease with which a software product can be modified in order to correct defects, meet new requirements, make future maintenance easier, or cope with a changed environment; these activities are known as software maintenance.

Reliability: The ability of a system or component to perform its required functions under stated conditions for a specified period of time.

Usability: The degree to which an object, device, software application, etc. is easy to use with no specific training.

Adaptability: the ability to change (or be changed) to fit changed circumstances.

Availability: Ability of a component or service to perform its required function at a stated instant or over a stated period of time.

Security: Security is the degree of protection against danger, loss, and criminals.

Portability: Portability is the software codebase feature to be able to reuse the existing code instead of creating new code when moving software from an environment to another.

Scalability: The degree to which a computer application or component can be expanded in size, volume, or number of users served and continue to function properly.

Testability: The capability of the software product to enable modified software to be tested.

Fault-tolerance: The ability of a system or component to continue normal operation despite the presence of hardware or software faults

Reusability: reusability is the likelihood a segment of source code that can be used again to add new functionalities with slight or no modification.

Sustainability: Meeting the needs of the present without compromising the ability of future generations to meet their own needs.

3. Electronic Mail Systems

Electronic mail is one of the most important of the Internet services. As a very large, fast growing, Internet Service Provider, Electronic Mail System requires a robust and powerful email architecture that will support rapid expansion.

MTA: Mail Transfer Agent is a program responsible for receiving, routing, and delivering e-mail messages. MTAs receive e-mail messages and recipient addresses from local users and remote hosts, perform alias creation and forwarding functions, and deliver the messages to their destinations.

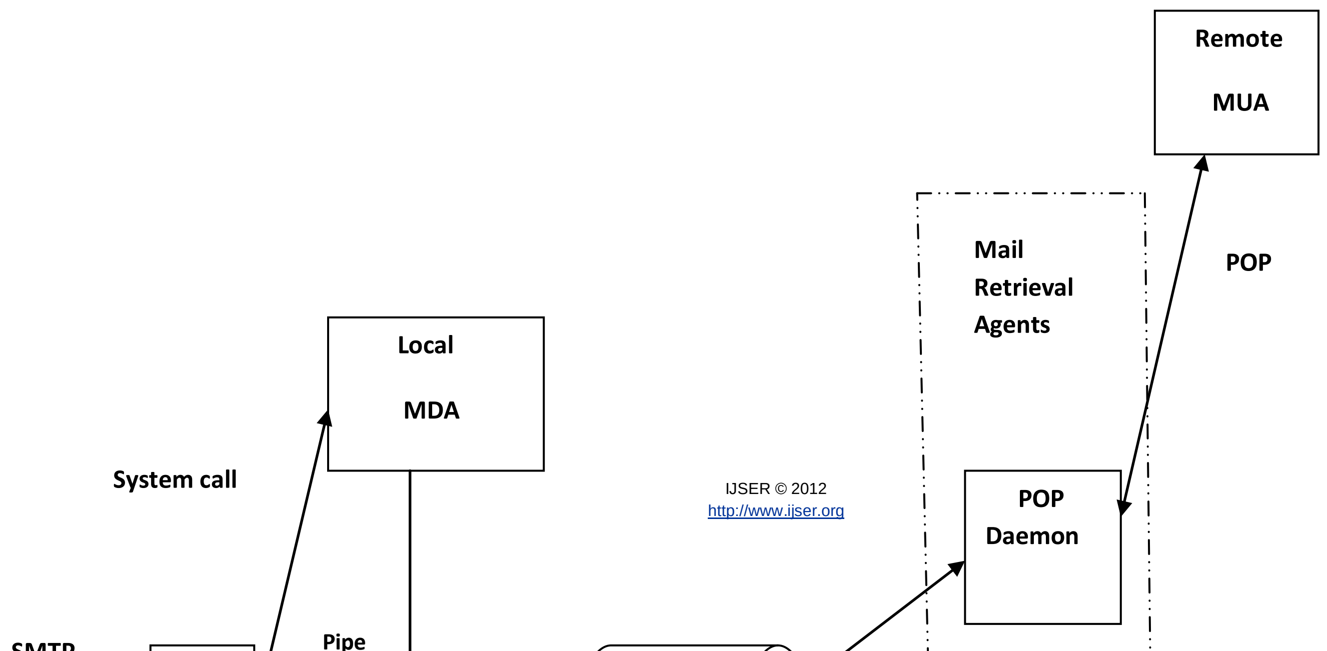
MDA: Mail Delivery Agent is a program that performs the final delivery into the mail box. Some examples of local MDAs in UNIX systems are '*procmail*', '*/bin/mail*', and '*mail.local*'.

MRA: Mail Retrieval Agent allows one to read the messages across the Internet. Upon request, an MRA accesses the user's mailbox on a local file system by read/write system calls.

MUA: Mail User Agent. Program that users run to read, reply to, compose and dispose of email. Examples: pine, elm, mutt.

Filter Program: The message can be filtered by mail filtering programs on the way to the mailbox.

This modular MTA-MDA structure allows a typical e-mail system to be constructed as a collection of loosely connected components that are developed independently. Therefore, some e-mail systems have been developed to adopt the MTA-MDA structure on each node for reducing the development cost and reserving the benefits of flexibility and extensibility. On the other hand, others use a different proprietary architecture. Now, we briefly overview some existing email systems on each category.
By decoupling the delivery and retrieval agents from the storage services and user manager in this way, the system can balance mail delivery tasks dynamically; any node can store mail for any user, and no single node is permanently responsible for a user's mail or soft profile information. Another advantage is that the system becomes extremely fault tolerant by always being able to deliver or retrieve mail for a user, even when nodes storing the user's existing mail are unavailable. The final advantage is that the system is able to react to configuration without human intervention. Newly added nodes will automatically receive their share of mail-session and storage-management tasks.The system architecture reveals a key tension that must be addressed in the implementation. Specifically, while a user's mail may be distributed across a large number of machines, doing so complicate both delivery and retrieval. On delivery, each time a user's mail is stored on a node not already containing mail for that user, the user's mail map (a potentially remote data structure) must be updated. On retrieval, aggregate load increases somewhat with the number of nodes storing the retrieving user's mail .Consequently, it is beneficial to limit the spread of a user's mail, widening it primarily to deal with load imbalances and failure. In this way, the system behaves (and performs) like a statically partitioned system when there are no failures and load is well balanced, but like a dynamically partitioned system otherwise



4. Design Approach

The proposed system augments an interface module between an MDA and the remote mailbox in the basic MTA-MDA structure. Figure 2 shows the proposed e-mail server architecture.

MTA receives an e-mail via standardized SMTP (Simple Mail Transfer Protocol). When message arrives, the MTA forks local MDA. As a local MDA, ’mail.local’ program is modified. Its role is to forward the incoming message to the interface module via a UNIX domain socket. Now, the message delivery role is delegated to the interface module that stores the input message into the local file system or transfers it to the interface module of the remote node.

In the proposed system, we use the off-the-shelf components for the MTA and the MDA: ’sendmail’ for the MTA and ’mail.local’ for the MDA. For load balancing of the cluster nodes, the DNS round-robin mechanism is used to determine which cluster node receives a new SMTP request. MUA is a client program used by a user to send or receive emails. The design of the interface module is shown in the figure 3. We define the following basic roles that the interface module should serve:

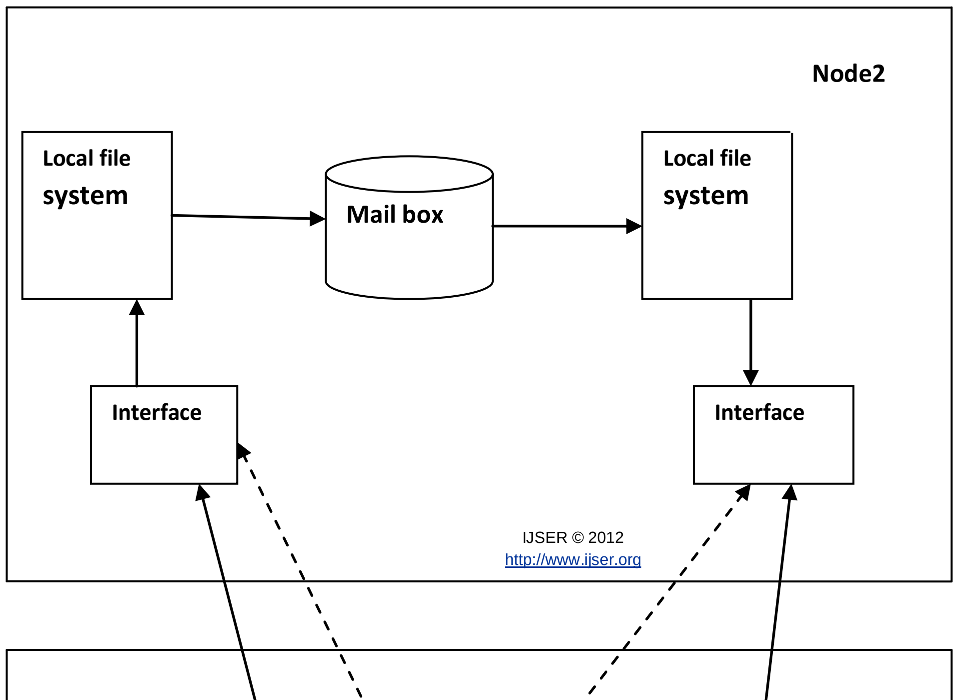
1. Authentication of users
2. User information handling
3. Message delivery to the local mail boxes and
4. Message information handling

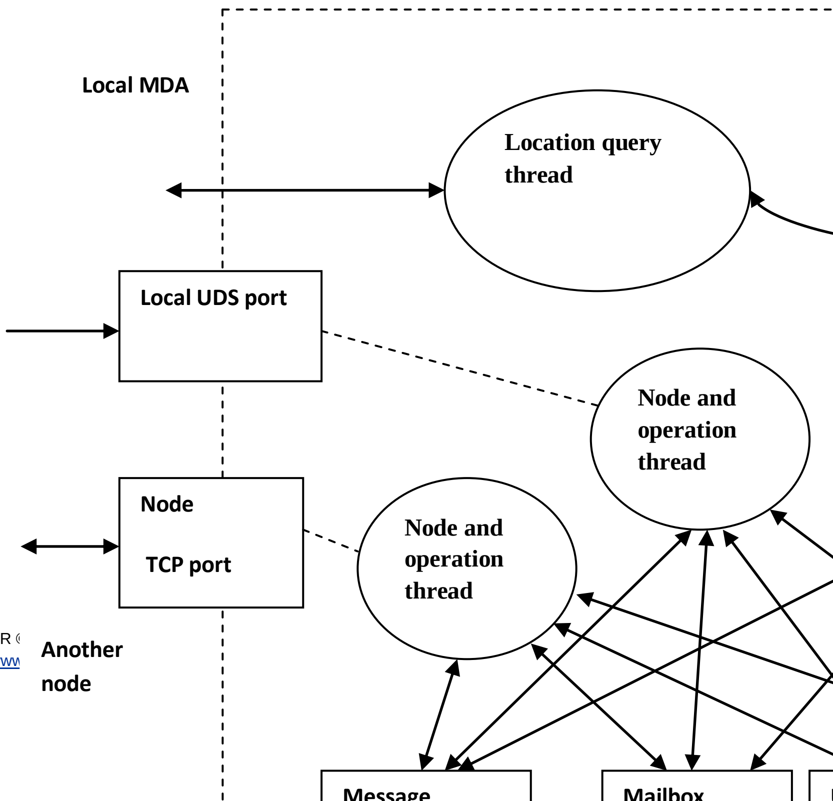
To serve the above roles of interface module, we designed four subsystems in the interface module namely Authentication subsystem, User Information subsystem, Mailbox subsystem, and Message Information subsystem. Each subsystem manages a set of functions. A subsystem is a collection of functions. We expect that these modular subsystems make us easy to replace and improve them. Authentication subsystem: We designed authentication subsystem to manage user authentication information using an SQL Database. User Information subsystem: This subsystem is used to manage additional information of users such as name, address and phone number using an SQL Database. Mailbox subsystem: This subsystem is used to manipulate the mailboxes of users.

Message Information subsystem: This subsystem is used to keep additional information of messages such as sender, recipient, date and size. To make interaction among the Mail delivery agent, Mail retrieval agent and subsystems of interface module, we used threads in the interface module namely Location query thread, Authentication thread, Data and operation thread, and Node and operation thread. Authentication thread: This thread receives a user authentication request from POP daemon. It processes user authentication by calling functions in the authentication subsystem. After completion of authentication process, this thread replies to the requesting POP daemon.

Data and operation thread: This thread receives requests from POP daemon. It process the e-mail messages using subsystems of interface moduleNode and operation thread: This thread receives message from the local MDA or from the other nodes. If the recipient's mailbox exists on local node, it stores the message into the recipient's mailbox through function calls in the Mailbox subsystem. at the same time, it stores the summary of the e-mail message in the Message information subsystem. If the mailbox exists on a remote node, it sends message to the remote node. After completing the message saving, the correspondent thread of the remote node acknowledges to the requesting node.

Location query thread: This thread receives the recipient's identity from local mail delivery agent. It obtains the destination node using Authentication subsystem.




Local MDA
Location query thread
Local UDS port
Node and operation thread
Node
TCP port
Node and operation thread
Another node
Message
Mailbox

In the process of remote message delivery, local MDA sends the message to the remote node directly without intervention of the interface module of the local node. To make such decision, however, the MDA should inquire the interface module where the recipient's mailbox is located. Therefore, the interface module of the proposed e-mail cluster system has a thread named Location query thread' as shown in Figure 3. With the information on the recipient's Identity, the Location query thread obtains the destination node using the Authentication subsystem.

If the recipient's mailbox exists on the local node, MDA transfer the message to a 'Node and operation thread' in the interface module. 'Node and operation threads' are the central threads that process the e-mail messages using the subsystems. A 'Node and operation thread' is also receives a message from the local MDA or from the other nodes. We can create as many 'Node and operation threads' as the number of nodes in the cluster. Each 'node and operation thread' is dedicated to each node including the local node. The 'Node and operation thread' dedicated to the local node receives a message by a UNIX domain socket while a 'node and operation thread' assigned to a remote node, receives a message by a TCP socket.

Scalability of this system is expressed by message throughput. A system is highly scalable if the message throughput of the system increases linearly as the cluster size. We define the message throughput of e-mail cluster system as the number of messages that the system can process maximally in a second. We increase number of nodes in cluster to achieve system scalability with increased value of message throughput. We use workload generator generate a certain number of e-mail messages at a constant rate. Availability of this system is expressed by making a local failure cause a local, but not global outage. An available system isolates a local failure from the system operation to avoid global outage. This system allows one to improve the system performance easily by upgrading some components.

5. Implementation Details

The modular structure of the e-mail system allows one to change the system configuration easily by replacing a component with another. We have implemented the proposed system with off-the-shelf software components in Linux platform. We have also implemented e-mail cluster system based on the NFS mechanism. We choose the EarthLink system for performance comparison because it is the only known e-mail cluster to us with an MTA-MDA structure. Christenson et al. proposed a scalable e-mail system using the MTA-MDA structure in EarthLink Network, Inc [5].
System configurations including the EarthLink configuration are summarized in Table 1.

Configuration	MTA	Interface Module
EarthLink	Sendmail	NFS
Present Mail System	Sendmail	Without NFS

Table 1: Experiment configuration

5.1. Experimental Results

We compare the two system configurations of Table 1 in terms of the message throughput, cluster scalability, and availability. Therefore, we created our own experimental environment. The experimental environment consists of a test cluster, workload generators, a DNS server, and a SQL DB server.

5.2. Message throughput

We define the message throughput of an e-mail cluster system as the number of messages that the system can process maximally in a second. The workload generators generate a certain number of e-mail messages in 60 seconds at a constant rate.

And the total elapsed time for the system to process all messages is measured. Then, the number of messages divided by the measured time becomes the average number of messages, which a system processes for a second.

Configuration	1-node	2-nodes
EarthLink	7.2	15.9
E-mail cluster system	8.7	19.1

Table 2: Message throughput of e-mail system configuration (messages/second)

Table 2 . presents the message throughput of two e-mail system configurations we have implemented. For a single node case, cluster E-mail system has a slightly higher throughput than EarthLink system. Result shows that the message throughput of the system is increased as the cluster increases.

6. Conclusion and Future Work

We have presented a design of e-mail cluster system that satisfies requirement of scalability, availability and extensibility. By implementing two different configurations of the system with the MTA-MDA structure, the flexibility and the extensibility of the proposed system are demonstrated.

Experimental results show that all the implemented systems are scalable in the sense the maximum throughput of the system is largely proportional to the number of cluster nodes.

7. References

[1]Cemal Yilmaz, Atif M. Memon, and Adam A. Porter. Arvind S. Krishna, Douglas, Aniruddha Gokhale and Balachandran Natarajan.Preserving distributed systems' critical properties, IEEE software, 2004.

[2] Lu.Wittgenstein, Distributed application architecture, java.sun.com/developer/Books/jdbc.

[3]Feng Zhou, Chao Jin, Yinghui Wu, Weimin Zheng. Cluster Object Storage Platform Designed for Scalable Services, IEEE 2003,

[4]Armando Fox and Steven D. Gribble and Yatin Chawathe and Eric A.Cluster-Based Scalable Network Services, ACM, 1997.

[5] Jeffrey Voas, .Software's secret sauce: The "-ilities", IEEE software, 2004.

[6]Terry Bollinger, Jeffrey Voas, and Maarten Boasson., Persistent Software Attributes. IEEE software 2004.

[7]Yasushi Saito and Brian N. Bershad and Henry M. Levy., Manageability, Availability and Performance in Porcupine: A Highly Scalable, Cluster based Mail Service, ACM, 2000.

[8]Milan Milenkovic, Scott H.Robin, Rob C, Knauerhase, David Barkai, Sharad Grag, Vijay Tewari, Todd A. Anderson, and Mic Bowman Towards Internet distributed computing,2004,.

[9]Henri E.Bal, Jennifer G. Steiner and A.S. Tanenbaum., Programming languages for distributed computing systems. ACM computing surveys, 1989.

[10]PM.Rani, A.V.Srinivas and D.J.Ram., Scalability issues in CORBA, IEEE software engineering for parallel and distributed systems, 2000.

[11]Aung, S.S., Naing T, Khaing MoeSan, Naing, T.T. and NiLar Thein, consistency control systems for remote object replication, IEEE information and telecommunication technologies, 2005.

[12]A.V.Srinivas and D.Janakiram. A model for characterizing the scalability of distributed systems, ACM sigops operating systems review, 2005.

[13]K. Nadiminti, M. Dias de A., and R. Buyya., Distributed systems and recent innovations: challenges and benefits, 2005.

8.Author's profile

Name: Ravi Uyyala

Email:uyyala.ravi@gmail.com.

Qualification: BE(CSE),MTECH(CSE)

Designation: Associate professor:

Institute:Bm college of technology,indore,india

Name:Raghuram Battini

Email:Raghu.battini @gmail.com.

Qualification: BE(CSE),MTECH(CSE)

Designation: Associate professor:

Institute:MVGREngineering College,Vijayanagaram,,india

Name: Kundan Kumar Mishra

Email: kundankumarmishra@gmail.com.

Qualification: BE(IT),MTECH(CSE)

Designation: Associate professor:

Institute:Bm college of technology,indore,india