Decentralized Electronic Mail

Sivan Bercovici, Yaniv Frishman
Department of Computer Science
Technion - Israel Institute of Technology

Idit Keidar, Ayellet Tal
Department of Electrical Engineering
Technion - Israel Institute of Technology

***Abstract*— E-mail is one of the most popular Internet applications. Unfortunately, the server-centric architecture of today's commercial solutions inherently limits availability, efficiency, and scalability. The single point of failure as well as the increasing processing and storage stress on the server drives the 35 year old infrastructure to the limits of its abilities. This paper proposes decentralized electronic mail (DEM), a novel e-mail architecture that overcomes existing systems' shortcomings. Following the mobile-object paradigm, DEM offers a decentralized approach, which breaks the dependency between a mail user and a single service provider, while relying entirely on participants' resources.**

I. Introduction

E-mail is one of the most common and well known computerized services available today. Home users, corporations, and universities, all expect around the clock availability of the service, with almost immediate delivery of mail items. Nowadays, e-mail architectures are governed by a server-centric design, which implies a handful of weaknesses. These include a single point of failure, storage and processing stress, bottlenecks, and inefficiency. All these lead to continued deterioration in the quality of service as more users join. To address these challenges, current commercial solutions (e.g., Hotmail, Yahoo) focus mainly on the use of clustered servers, implying a high operational and maintenance cost.

Conversely to the centric architectures, the trend of decentralized self-organizing services that utilize user resources is rising. The universal broadband connectivity achieved during the last decade, as well as high performance personal computers with increasing local storage sizes, yield a good ground for many decentralized services. Based on *peer-to-peer* (P2P) technology, with applications such as Gnutella [1] and FreeNet [2], users have been able to harness their home computer resources to the benefit of the global computer home users community. We wish to adopt the properties of such self-organizing, highly-available, decentralized networks in an e-mail service.

In this paper, we present DEM, a novel e-mail architecture based on the mobile-object paradigm. In this paradigm, objects are able to migrate to remote-hosts along with their state and behavior, while conserving the correct execution of the program. As depicted in Figure 1, DEM's concept revolves around a decentralized approach in which the users hold their own mailboxes, as well as the mailboxes of other users that are temporarily offline. The sharing of computation time and storage space opens a wide range of opportunities. User mailboxes are mobile-objects that can travel so as to stay on the live network. This way, mailboxes are able to continue their operation despite users joining and leaving the network. Lightweight mobile servers (depicted in figure 1 as gray disks) keep track of user mailbox movement, thus enabling e-mail senders to locate destination mailboxes. Mail items travel directly from the sender's to the receiver's mailbox. Attachments are also mobile-objects, separate from the mail items, allowing multiple mail-items to point to a single attachment object. Each of the persistent system's components is replicated across a number of hosts, allowing fault-tolerance.

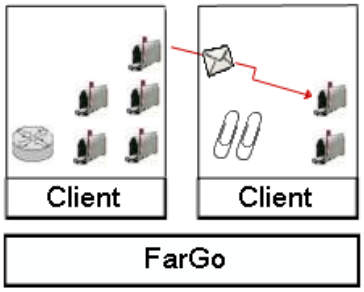


Fig. 1. DEM architecture overview.

We describe the implemented mobile-object based e-mail system. Our implementation is realized using Fargo [3], a java-based lightweight middleware and programming environment that provides services that enable object mobility. To the best of our knowledge, the DEM architecture is the first attempt to create an e-mail system based on the mobile-object paradigm. The work focuses on the development of a new solution to the e-mail systems' challenges, harnessing the power of mobile-objects. We show that through the use of this decentralized approach, the required properties of high-availability, efficiency, and scalability are achieved.

The rest of the paper is organized as follows: Section II discusses previous solutions, and identifies their shortcomings. Section III defines the requirements we have set out for our service. Section IV presents our architecture. Section V discusses our implementation. An analysis of the quality of our solution and a comparison with other solutions is given in Section VI. Section VII concludes the paper and presents possible future directions.

II. Previous solutions

In this section, we list major problems in current e-mail architectures. We also review previous work that proposed decentralized e-mail architectures based on P2P.

Most modern e-mail systems are based on a centralized approach in which users communicate with a central server in order to retrieve their personal mail. In such an approach, the

user depends strongly on the availability and normal function of the server that provides the e-mail service. This centralized system design inherently suffers from key problems in terms of scalability, performance, server stress, and fault-tolerance.

In a non-clustered e-mail system, a bottleneck is evident at the single-server's side. An increase in the number of users increases both the storage and communication stress on the single server. On top of these, nowadays, more features are becoming available on the server side, enhancing the mailing experience (e.g., anti-virus and anti-spam facilities), thus further increasing the load. Processing, communication, and storage stress may cause fatal deterioration of service availability, up to the point where the server is unable to provide the service at all.

Attachments in e-mail cause storage stress on the server. In basic e-mail architectures, servers create duplicates of attachments for each mail recipient. While a partial solution, implemented in some commercial architectures, suggests sharing an attachment by users of each server (a single attachment copy per server), one can't avoid the replication of the attachments, when these are sent to multiple distant servers.

The single point of failure is another inherent problem with current e-mail system design. A failure in an e-mail server denies all mailing services from all the users who use that computer to manage their mailboxes. Moreover, partial network disconnection or delays can cause online clients not to be able to reach the functioning server.

Several clustered server solutions have been proposed [4], [5] and some implemented (e.g., Hotmail), yet these solutions are both expensive and hard to maintain. Dedicated hardware as well as dedicated maintenance staff is needed in order to keep the service operational.

Not all e-mail systems' drawbacks are due to its centralized design. Several shortcomings are caused by legacy formats and protocols that are still supported and used. For example, the use of the standard Base 64 for in-mail binary-attachment encoding inflicts a 33% increase in mail size [6].

Finally, current e-mail systems are insensitive to the end-users' computer resources and access rates. A user with narrow-band connection may wish to review part of the mail items, or even a part of a mail item.

Several attempts were made to construct a serverless e-mail system that relay on peer-to-peer substrate to provide the persistent mail storage. In [7], inboxes and messages are replicated across multiple peers. Using the inbox's and messages' unique identifiers and through the use of the underlying *Distributed Hash-Table* (abbreviated DHT), one can locate the hosting users' peers. Each user has a corresponding inbox that holds the keys of unread mail. When a user wishes to read mail, the inbox is retrieved from a storing peer, and a search for each new item is processed using the lookup substrate. When a user wishes to send a message, s/he first stores it on a set of peers, and then locates the destination inbox using the lookup service. A header, specifying the message identifier (i.e., the key), is appended to that inbox. POST [8] describes a similar architecture built on top of Pastry [9]. OceanStore [10] is a distributed utility that provides continuous access to persistent information. The work suggested building an e-mail system as an application on top of the distributed persistent file-system layer. [11] examined the implementation of a mail reader on top of Bayou, a weakly consistent replicated database designed to support distributed applications.

III. Requirements

We identify the following requirements from our service:

1) *Availability*: E-mail users would like their mailbox to be accessible regardless of their connection point. From any Internet-connected computer, one should always be able to read her/his new mail and perform other basic mailbox operations, such as mail composition, deletion, archiving, and search.
2) *Efficiency*: The service should be efficient in terms of communication, space, and performance.
3) *Scalability*: As the number of users of a mail service grows, scaling the system should require adding as little dedicated hardware as possible (if any addition at all). The overhead inflicted on users due to the growth should be kept to a minimum.
4) *Personalized mailbox*: The service should support personalized mail features. These may include client-side e-mail application elements such as an address book, folders, white-list, and general applications' preferences.
5) *End user sensitivity*: Users of e-mail systems are heterogeneous in the sense of the CPU power and connection bandwidth of the machines they use. The system should be sensitive to each user's computation and communication abilities, and suit the individual constraints.
6) *Low cost*: The system should require as little dedicated hardware as possible.
7) *Minimum maintenance*: It is desirable that the maintenance of such a system be kept to a minimum, so as to inflict low operational cost and ease management and scalability. A self-maintaining system is preferred.

IV. DEM Architecture

This section presents the design of our novel e-mail architecture, which addresses the described challenges. DEM provides the fundamental operations for e-mail clients: mail delivery, automated reception of mail, and mailbox maintenance. The architecture includes administration components, enabling system-wide monitoring and performing run-time application-network layout modifications. DEM is built on top of a distributed application middleware that offers several mobile-object services:

1) *Explicit migration*: The application can explicitly request to relocate specific components to specific hosts at runtime, while conserving correct state and execution.
2) *Implicit migration*: Following developer-defined co-location relationships between application components, the middleware enforces implicit migration (e.g., two components that must reside on the same machine will always migrate together, and to the same destination machine).
3) *Location transparency*: The application interacts with its components through enhanced references that mask a

component's actual host location. These interactions take the form of object method calls through these enhanced references, hiding location information. The middleware updates these references so that pointers remain valid despite source or target migration.

4) *Monitoring*: Object locations and migration events can be monitored by the application, thus enabling application components to react to system events.
5) *Reference construction*: An enhanced reference can be constructed in run-time to enable interaction with remote objects that are searched for and discovered on-the-fly.

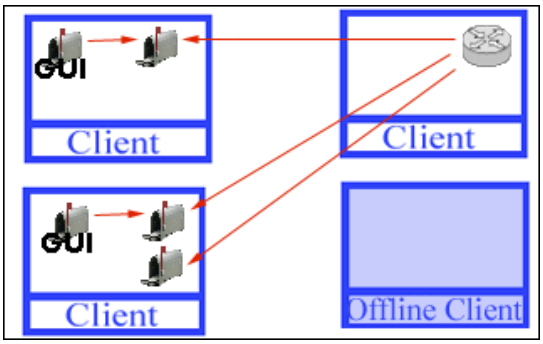
In the DEM architecture, all primary components are mobile. The most basic component is the mobile mailbox. Each user has a private mailbox through which she/he interacts with her/his personal mail items. In order to achieve continuous operation and availability of a mailbox, the mailbox travels among computers, so as to ensure that it remains on the live network (i.e., on a participating computer that is operating and connected to the network).

To help a mailbox stay on the live network, a second mobile component, called the dispatch-unit, is used. The dispatch-unit holds a list of connected hosts. By knowing which computers are on-line, the dispatch-unit is able to act as a match-maker between mobile mailboxes and possible temporary hosts. Every participating computer is considered a legitimate host for mobile mailboxes and attachments. The dispatch-unit also holds references to mailboxes, and references to attachments. Using hash-tables, the dispatch-unit maps unique user identifiers and attachment identifiers to mailbox object enhanced references and attachment object enhanced references, respectively. The location transparency property provided by the middleware keeps the references held in the hash-table valid despite mailbox and attachments migration. A user who wishes to get a valid reference to an object consults the dispatch-unit by providing the object's identifier.

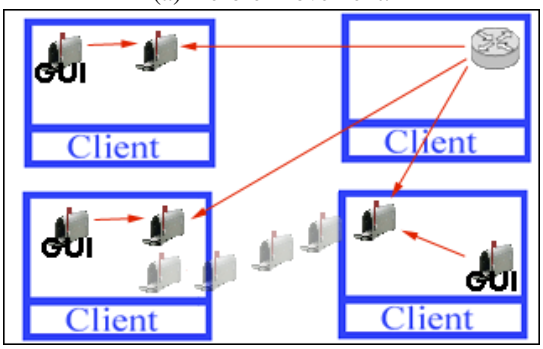
As a mobile component, the dispatch-unit can reside on any of the hosts. A number of dispatch-units co-exist, balancing the number of requests among them and offering a degree of fault-tolerance by replication. From this point on, we describe communication in the system with respect to a single dispatch-unit, which is randomly selected from the set of dispatch-units per request. An external list of backbone dispatch-units is available for first-time requests (i.e., joining the network). To add a new dispatch-unit (other then those in the backbone list), the new dispatch-unit must inform any currently connected dispatch-unit of its existence. The connecting dispatch-unit copies its initial hash-tables from an already connected dispatch-unit.

A. Connecting to the service

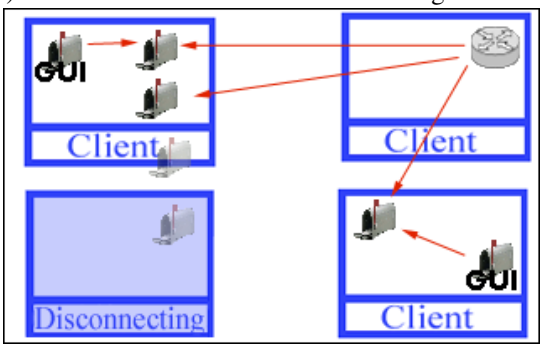
When a user becomes on-line and wishes to retrieve her/his personal mailbox, she/he first searches for a dispatch-unit (depicted in figure 2(a) as a gray disk). Using the middleware's reference construction service, dispatch-unit objects can be searched in different hosts (namely those in the backbone list) to provide the joining user with an entry point to the e-mail service. From that point, the user's component communicates with the dispatch-unit through ordinary method calls (achieved through the use of the enhanced references and the location transparency property provided by the middleware). The dispatch-unit is inquired for a valid reference (shown as a red arrow) to the user's mailbox object, which at that point in time resides on some remote host. Upon resolving the reference, the middleware's explicit object migration service is used in order to move the mailbox from the remote host to the user's own computer. Figure 2 shows a visualization of the movement of a mailbox between computers during the user's connection and disconnection process. In Figure 2(a), one client holds her mailbox, and another client holds both her mailbox, and the mailbox of an offline user. In Figure 2(b), a mailbox moved to the client that just got connected to the service (shown in the lower-right blue frame). The newly connected user can continue the interaction with her/his mailbox at its new location through the use of the same enhanced reference (as it is kept valid by the middleware after the migration ends). As long as that user's computer is operating and on-line, this computer may also serve as a host for other offline users' mailboxes.



(a) Before movement.



(b) A client connected and a mailbox migrated to it.



(c) A mailbox migrates from a disconnecting client.

Fig. 2. Mailbox mobility in the DEM system.

B. Disconnecting from the service

When a user wishes to disconnect from the network, the application transfers all locally operating mailboxes (including her/his own) and attachments to a set of distant hosts. Figure 2(c) illustrates this process. The dispatch-unit provides the match-making service: it uses its list of on-line participants to find suitable operating hosts. By using the middleware's explicit migration service, mailboxes and attachments are transferred from the disconnecting user's computer to their new hosts. The middleware, which provides location transparency, updates the enhanced references in the dispatch-unit hash-tables so that they will reflect the new post-migration locations.

C. Sending mail

The most fundamental operation of an e-mail system user is sending an e-mail. In DEM, a user can send an e-mail to another user's mailbox regardless of the connection status of the destination user (i.e., the destination user may be either on-line or off-line). This is possible thanks to the fact that although a destination user may be off-line, her/his mailbox, being a live mobile-object, continues its operation on some temporary host, and is reachable through the valid references held in dispatch-units. To acquire a valid reference to the destination mailbox, a query with the destination mailbox's identifier is sent to the dispatch-unit. The dispatch-unit uses its hash-table and the given identifier to retrieve a reference to the corresponding mailbox, returning it to the requester. Using the returned reference to directly invoke a store-mail method in the destination mailbox, the new mail item is passed and stored in the destination mailbox object.

D. Mailbox interaction

Mail items reside inside the mailbox object. To enable interaction with the mail items, a valid reference to the mailbox is acquired through the use of the afore mentioned dispatch-unit services. Mailbox operations, such as reviewing and deleting mail items, are possible regardless of the actual location of the mailbox. It is important to note that although during the user connection process the personal mailbox travels to the user's machine, the user can choose to leave the mailbox on the distant machine, performing the operations on the mailbox from distance.

E. Attachments

Our architecture supports floating attachments. When a user wishes to send an attachment as part of a mail item, a mobile-object that encapsulates the attachment is created. A mail item contains a unique identifier to a mobile-attachment rather than the attachment itself. If a mail item with an attachment is sent to multiple recipients, they all contain the same unique identifier of the floating attachment. Attachments, much like mailboxes, travel on the live network using the dispatch-unit's match-making services, and hence they also offer continuous availability. The location transparency property allows the dispatch-unit to offer valid references of mobile-attachments, thus providing clients with the ability to acquire a valid reference to any attachment object.

F. Replication

The personal mailboxes, dispatch-units, and floating attachments are all replicated, improving their availability. Each personal mailbox has a fixed set of replicas scattered randomly over the live network. When a mailbox is acquired by a user, the dispatch-unit returns a reference to one of the mailbox's replicas. Each replica has references to all other replicas of the same mailbox. These references are kept valid by the middleware, regardless of the independent mailbox movements. The mailbox replicas periodically compare their contained mail item lists and history of maintenance operations (such as mail deletion). Each mailbox performs periodic liveness tests on the other replicas by attempting to invoke a method through a replica reference. When a mailbox detects a non-responding replica, it creates a new replica, which in turn is passed to another computer through the use of the dispatch-unit services. In this scheme, there is a positive probability for multiple replicas to detect a faulty mailbox, reacting in multiple creations of new replicas. In order to converge to a fixed number of replicas per mailbox, when a certain threshold is exceeded and this is detected by one of the replicas, synchronized termination of a replica is performed. A self-healing mechanism, similar to the one in the mailbox object, is also used for floating attachments and for dispatch-units.

V. Implementation

In this section, we describe our implementation of the DEM architecture. To make this paper self-contained, we give a brief overview of FarGo [3], which provides all the mobile-component services described in Section IV.

FarGo is a distributed applications development and deployment environment. In FarGo, *complets* are the basic building blocks of an application, defining the minimum unit of relocation - the mobile object. *Cores* are uniquely-identifiable objects that provide the system support needed for the mobilization and interconnection of *complets* across machines. Each *complet* is associated with exactly one *core* at any given time. At runtime, *complets* can move across any running *core* while preserving the correct execution of the application. The FarGo environment is Java-based, which yields easy and short development of architecture-neutral applications. Minor modifications to the regular Java programming model are necessary in order to make use of the mobility features.

Our implementation includes all the fundamental facilities of an e-mail system, with its components built as *complets* using FarGo. On the client end, we implemented the mobile mailbox and a *graphic user interface* (GUI) component. Floating attachments were implemented to enhance the system with the ability to send files or other objects as part of a mail item. On the service providing end, the dispatch-unit was implemented. We implement it as a mobile object in order to allow the greater flexibility, and also to take advantage of optimization opportunities at the network level, such as moving closer to dense users areas. A GUI for the dispatch-unit was implemented to be used as part of the service monitoring facility. As depicted in Figure 3, through the monitor of any specific dispatch-unit, one can study the

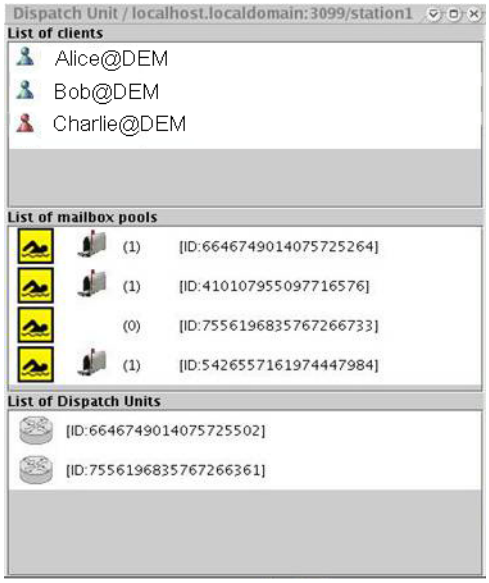


Fig. 3. Dispatch-unit monitoring GUI.

number of known dispatch-units, on-line users, off-line users and mailbox spread.

Log4J [12] is a logging infrastructure that is used as a debugging tool for distributed applications. To enable the system's examination, all components produce event logs managed by Log4J facilities. An integrated administration environment allows monitoring and interaction with any of the system's components. The integrated environment includes an application *complet* browser, a viewer for collective logs from distant components, and a Python [13] scripting console. The scripting console allows an administrator to perform a wide-range of operations on the system, from inquiring a specific component's state, to explicitly ordering the movement of components. The implemented environment supports the executing of Python scripts that interact with the running system. This means that modifications of the system, including creation and addition of new components, can be performed in run-time, without the need for a total system restart.

The replication mechanism for the dispatch-units, mail-boxes, and attachments was implemented. The replicas perform a periodic synchronization, and were also enhanced with a self-regeneration property (as described in section IV-F).

VI. Analysis and Results

In this section we show that the proposed architecture and its implementation correspond to the list of requirements. Availability, efficiency, and scalability are all at the focus of our novel system.

Our first requirement from the system is continuous availability. In DEM, the users' mailboxes, attachments, and dispatch-units, all achieve availability by traveling so that they would stay on operating and Internet-connected computers. Replicating the basic system components allows the system to be resilient to any local faults. In our system, if a mailbox, a dispatch-unit, or an attachment fails, any of their replicas can be used. Common e-mail systems use clusters of servers to offer the same solution, yet at the higher-price of additional hardware and maintenance. Researched P2P based solution offers the same fault-tolerant by replication mechanism, but the lack of migration and self-healing properties produces a lower degree of fault-tolerance per the same replication factor. In all previous architectures a user is highly dependable on a fixed set of computers that are responsible for her/his mailbox. If all computers in that fixed set are down, the user can't use the mail service. Conversely, in DEM, the set of computers holding a specific mailbox changes dynamically, thus a specific mailbox is unavailable only if all the hosts holding the replicas fail at the same time-window without regenerating. This yields that DEM has a higher fault-tolerance, in comparison to previous architectures.

Efficiency is the second requirement from DEM. Dispatch-units handle only lookup and match-making requests, while most of the communication is performed between participants, thus utilizing network resource better. In DEM, a mail item travels directly between sender and receiver, which yield $O(1)$ communication cost. In common e-mail systems, mail items pass through a chain of servers until reaching the destination user mail server. In researched P2P solutions, mail delivery costs $O(\log n)$ (where n is the number of participants) due to the underlying DHT substrate. DEM balances the mailbox lookup load by maintaining multiple dispatch-units, and using the efficient hash-table data structure.

Spatial efficiency is treated in the form of attachments optimization. DEM encapsulates attachments as mobile-objects, replacing the in-mail attachment with an in-mail unique attachment identifier. Storage stress due to a single attachment is $att \cdot rep$ (where att is the size of the attachment and rep is the architecture's constant replication factor), regardless to the number of mail recipients. Similar mechanisms in P2P-based architectures yield the same performance, yet the lack of migration and regeneration reduces its degree of fault-tolerance. In commercial solutions, where attachments are in-mail, storage stress is linear to the number of mail recipients.

The third requirement of DEM is scalability. In DEM, most of the communication takes place between a sender and a mailbox holder. Communication with the dispatch-unit is kept to a minimum, allowing the removal of bottlenecks evident in server-centric designs. Concentrated stress is eliminated, as the increasing storage and communication load is balanced evenly among all participates, including the new ones. Relying mostly on users' resources, DEM inherent load balancing offers a scalable system. On the other hand, in the centralized approach, servers are highly stressed in both processing and storage. Popular mail systems, such as Hotmail, distribute the stress through the use of computer farms. Although the stress problem decreases, the cost of acquiring and maintaining this large number of machines is high.

DEM achieves the personalized e-mail experience by using mobile-objects properties. Due to the fact the mobile-mailbox state and behavior are preserved regardless of their host, the personal configuration and additional mailbox plug-ins continue to exist. In centralized solutions, additional features inflicts on system's performance and may require additional hardware. P2P solutions offer only mail-content availability.

DEM offers an end-user sensitive architecture, as a mailbox

can reside on one host while the user's interface on another. Thus, a user can examine any single mail-item at any preferred level with respect to her/his resources. Many commercial mail services are provided through the use of the *Post Office Protocol* (POP) [14], offering the mail service at a mailbox granularity. In this scheme, a user periodically retrievs all available mail-items from the server, deleting the server's copy. At best, current commercial solutions allows services at a mail-item granularity. These solutions are insensitive to the user's computer and network abilities.

DEM relies mostly on users' resources, thus offers a low cost solution. Scaling the system does not imply additional dedicated hardware, as is the case with commercial solutions. The self-healing property of DEM's components yields a self maintaining system, whereas in traditional commercial solutions, a larger e-mail system requires an increasing maintenance cost.

Table I summarizes the comparison among DEM, commercial and researched solutions.

TABLE I
COMPARISON AMONG SOLUTIONS

Features	Single server	Clustered server	P2P-based	DEM
No bottlenecks			$\surd$	$\surd$
Mailbox independent of fixed servers				$\surd$
Service failure	Server dependent	Cluster dependent	Fixed set dependent	Dynamic set crash in short window
Communication cost	$O(1)$		$O(\log n)$	$O(1)$
Spatial efficiency	$att \cdot recipients$		$att \cdot rep$	$att \cdot rep$
scaling cost	Stronger server	Larger farm	None	None
Service extension	Requires server upgrade		Not supported	Local changes
end-user sensitivity	At mail item granularity			at mail item content granularity
cost and maintenance	Proportional to the amount of dedicated hardware		No cost. Maintenance not possible	Low cost. self-healing

VII. CONCLUSION AND FUTURE DIRECTIONS

We have presented DEM, a novel decentralized mobile-object approach to the e-mail service. The DEM architecture deals with the principal challenges an e-mail system faces. We have shown that DEM has a higher degree of availability, efficiency and scalability than previous solutions, while maintaining a low operational and maintenance cost. The system also retains the personalization of the service and allows sensitivity to the end-user resources. These features are achieved thanks to the decentralized architecture which eliminates bottlenecks, and the mobility, replications, and self-healing properties.

We have presented the DEM implementation, from which one can learn the feasibility of our solution. The implementation can be used as a ground-base for further research in the direction of a decentralized mobile-object based e-mail system. The system administration and research environment developed as part of our solution offers a convenient experimentation tool, and enables an easy expansion of the current architecture.

Possible future directions include moving from a mobile mailbox granularity to a mobile mail-items granularity, for better utilization of system resources. To this end, we would also like to examine attachments fragmentation. Additional fault-tolerance mechanisms, such as backing-up mailboxes on hosts' disks, should be examined. Research on an adaptive replication factor mechanism may produce a lighter system, improving the overall performance while maintaining the same level of stability. The mailbox content privacy feature should be examined. We believe it can be achieved through the use of known asymmetric cryptographic mechanisms.

Another important direction for future research is to provide with a plan for incremental deployment of DEM. It is possible to integrate DEM networks into the regular worldwide e-mail systems by constructing a mail-delivery interface between current mail systems and DEM. An interface should also be made between common e-mail client applications (e.g., Outlook) and DEM.

REFERENCES

[1] "The Gnutella protocol specification," 2000, available at http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[2] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Workshop on design issues in anonymity and unobservability*, ICSI, Berkeley, CA, USA, July 2000, pp. 311–320.

[3] O. Holder, I. Ben-Shaul, and H. Gazit, "Dynamic layout of distributed applications in FarGo," in *Proceedings of the 1999 International Conference on Software Engineering*. IEEE Computer Society Press / ACM Press, 1999, pp. 163–173.

[4] A. D. Joseph, E. A. Brewer, B. Behren, J. Kubiatowicz, and S. Czerwinski, "NinjaMail: the design of a high-performance clustered, distributed E-mail system," in *Proceedings of the First International Workshop on Scalable Web Services, Toronto, Canada*, Aug. 2000.

[5] B. N. Bershad, H. M. Levy, and Y. Saito, "Manageability, availability and performance in porcupine: A highly scalable, cluster-based mail service," in *Symposium on Operating Systems Principles*, 1999, pp. 1–15.

[6] D. A. Turner and K. W. Ross, "Continuous media e-mail on the Internet: Infrastructure inadequacies and a sender-side solution," *IEEE Network*, vol. 14, no. 4, pp. 30–37, July/Aug 2000.

[7] J. Kangasharju, K. W. Ross, and D. A. Turner, "Secure and resilient peer-to-peer e-mail: Design and implementation," in *proceedings of the Third IEEE International Conference on Peer-to-Peer Computing*, Linkoping, Sweden, Sep 2003.

[8] A. Mislove, A. Post, C. Reis, P. Willmann, P. Druschel, D. S. Wallach, X. Bonnaire, P. Sens, J.-M. Busca, and L. Arantes-Bezerra, "POST: a secure, resilient, cooperative messaging system," in *Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS'03)*, Lihue, HI, May 2003.

[9] A. Rowstron and P. Druschel, "Pastry: scalable, decentraized object location and routing for large-scale peer-to-peer systems," in *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Nov. 2001.

[10] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage," in *Proceedings of ACM ASPLOS*. ACM, November 2000.

[11] D. B. Terry, K. Petersen, M. J. Spreitzer, and M. M. Theimer, "The case for non-transparent replication: Examples from bayou," *IEEE Data Engineering*, pp. 12–20, December 1998.

[12] "Log4J logging services," available at http://logging.apache.org/log4j/.

[13] "Python," available at http://www.python.org/.

[14] M. Rose, "Post office protocol - version 3, extended service offerings; RFC 1082," *Internet Request for Comments*, no. 1082, Dec. 1988.