(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
10 February 2005 (10.02.2005)

PCT

(10) International Publication Number
WO 2005/013571 A1

(51) International Patent Classification[7]: H04L 12/58, G06F 17/60

(21) International Application Number: PCT/US2004/023946

(22) International Filing Date: 23 July 2004 (23.07.2004)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
10/627,466 25 July 2003 (25.07.2003) US

(71) Applicant *(for all designated States except US)*: **ATTENEX CORPORATION** [US/US]; 925 Fourth Avenue, Ste 1700, Seattle, WA 98104-1125 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **KAWAI, Kenji** [US/US]; 7309 21st Avenue NE, Seattle, WA 98115 (US). **MCDONALD, David, T.** [US/US]; 6056 44th Avenue NE, Seattle, WA 98115 (US).

(74) Agent: **INOUYE, Patrick, Joseph, Sus**; 810 Third Avenue, Ste 258, Seattle, WA 98104 (US).

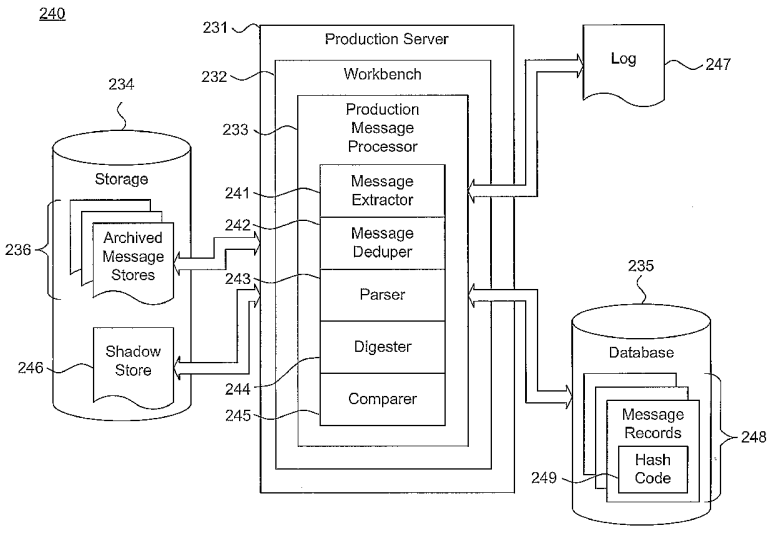
(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) **Title:** EVALUATING A STRUCTURED MESSAGE STORE FOR MESSAGE REDUNDANCY



(57) **Abstract:** A system (230) and method (310) for evaluating a structured message store (236) for message redundancy is presented. A header (75) and a message body (74) from each of a plurality of messages (70) maintained in a structured message store (236) are evaluated. A substantially unique hash code (249) is calculated over at least part of the header (75) and over the message body (74) of each message (70). The messages (70) are grouped by the hash codes (261) and identifying one such message (70) as a unique message (265) within each group.

WO 2005/013571 A1

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

EVALUATING A STRUCTURED MESSAGE STORE FOR MESSAGE REDUNDANCY

TECHNICAL FIELD

The present invention relates in general to stored message categorization and, in particular, to a system and method for evaluating a structured message store for message redundancy.

BACKGROUND ART

Presently, electronic messaging constitutes a major form of interpersonal communications, complimentary to, and, in some respects, replacing, conventional voice-based communications. Electronic messaging includes traditional electronic mail (e-mail) and has grown to encompass scheduling, tasking, contact and project management, and an increasing number of automated workgroup activities. Electronic messaging also includes the exchange of electronic documents and multimedia content, often included as attachments. And, unlike voice mail, electronic messaging can easily be communicated to an audience ranging from a single user, a workgroup, a corporation, or even the world at large, through pre-defined message address lists.

The basic electronic messaging architecture includes a message exchange server communicating with a plurality of individual subscribers or clients. The message exchange server acts as an electronic message custodian, which maintains, receives and distributes electronic messages from the clients using one or more message databases. Individual electronic messaging information is kept in message stores, referred to as folders or archives, identified by user account within the message databases. Generally, by policy, a corporation will archive the message databases as historical data storing during routine backup procedures.

The information contained in archived electronic messages can provide a potentially useful chronology of historically significant events. For instance, message conversation threads present a running dialogue which can chronicle the decision making processes undertaken by individuals during the execution of their corporate responsibilities. As well, individual message store archives can corroborate the receipt and acknowledgment of certain corporate communications both locally and in distributed locations. And the archived electronic message databases create useful audit trails for tracing information flow.

Consequently, fact seekers are increasingly turning to archived electronic message stores to locate crucial information and to gain insight into individual motivations and behaviors. In

particular, electronic message stores are now almost routinely produced during the discovery phase of litigation to obtain evidence and materials useful to the litigants and the court. Discovery involves document review during which all relevant materials are read and analyzed. The document review process is time consuming and expensive, as each document must ultimately be manually read. Pre-analyzing documents to remove duplicative information can save significant time and expense by paring down the review field, particularly when dealing with the large number of individual messages stored in each of the archived electronic messages stores for a community of users.

Typically, electronic messages maintained in archived electronic message stores are physically stored as data objects containing text or other content. Many of these objects are duplicates, at least in part, of other objects in the message store for the same user or for other users. For example, electronic messages are often duplicated through inclusion in a reply or forwarded message, or as an attachment. A chain of such recursively-included messages constitutes a conversation "thread." In addition, broadcasting, multitasking and bulk electronic message "mailings" cause message duplication across any number of individual electronic messaging accounts.

Although the goal of document pre-analysis is to pare down the size of the review field, the simplistic removal of wholly exact duplicate messages provides only a partial solution. On average, exactly duplicated messages constitute a small proportion of duplicated material. A much larger proportion of duplicated electronic messages are part of conversation threads that contain embedded information generated through a reply, forwarding, or attachment. The message containing the longest conversation thread is often the most pertinent message since each of the earlier messages is carried forward within the message itself. The messages comprising a conversation thread are "near" exact duplicate messages, which can also be of interest in showing temporal and substantive relationships, as well as revealing potentially duplicated information.

In the prior art, electronic messaging applications provide limited tools for processing electronic messages. Electronic messaging clients, such as the Outlook product, licensed by Microsoft Corporation, Redmond, Washington, or the cc:mail product, licensed by Lotus Corporation, Cambridge, Massachusetts, provide rudimentary facilities for sorting and grouping stored messages based on literal data occurring in each message, such as sender, recipient, subject, send date and so forth. Attachments are generally treated as separate objects and are not factored into sorting and grouping operations. However, these facilities are limited to processing only those messages stored in a single user account and are unable to handle multiple electronic

message stores maintained by different message custodians. In addition, the systems only provide partial sorting and grouping capabilities and do not provide for culling out message with duplicate attachments.

Therefore, there is a need for an approach to processing electronic messages maintained in multiple message stores for document pre-analysis. Preferably, such an approach would identify messages duplicative both in literal content, as well as with respect to attachments, independent of source, and would "grade" the electronic messages into categories that include unique, exact duplicate, and near duplicate messages, as well as determine conversation thread length.

There is a further need for an approach to identifying unique messages and related duplicate and near duplicate messages maintained in multiple message stores. Preferably, such an approach would include an ability to separate unique messages and to later reaggregate selected unique messages with their related duplicate and near duplicate messages as necessary.

There is a further need for an approach to processing electronic messages generated by Messaging Application Programming Interface (MAPI)-compliant applications.

DISCLOSURE OF INVENTION

The present invention provides a system and method for generating a shadow store storing messages selected from an aggregate collection of message stores. The shadow store can be used in a document review process. The shadow store is created by extracting selected information about messages from each of the individual message stores into a master array. The master array is processed to identify message topics, which occur only once in the individual message stores and to then identify the related messages as unique. The remaining non-unique messages are processed topic by topic in a topic array from which duplicate, near duplicate and unique messages are identified. In addition, thread counts are tallied. A log file indicating the nature and location of each message and the relationship of each message to other messages is generated. Substantially unique messages are copied into the shadow store for use in other processes, such as a document review process. Optionally, selected duplicate and near duplicate messages are also copied into the shadow store or any other store containing the related unique message.

The present invention also provides a system and method for identifying and categorizing messages extracted from archived message stores. Each individual message is extracted from an archived message store. A sequence of alphanumeric characters representing the content, referred to here as a hash code, is formed from at least part of the header of each extracted message plus the message body, exclusive of any attachments. In addition, a sequence of

alphanumeric characters representing the content, also referred to here as a hash code, is formed from at least part of each attachment. The hash codes are preferably calculated using a one-way function, such as the MD5 digesting algorithm, to generate a substantially unique alphanumeric value, including a purely numeric or alphabetic value, associated with the content. Preferably, the hash code is generated with a fixed length, independent of content length, as a sequence of alphanumeric characters representing the content, referred to here as a digest. The individual fields of the extracted messages are stored as metadata into message records maintained in a structured database along with the hash codes. The hash codes for each extracted message are retrieved from the database and sorted into groups of matching hash codes. The matching groups are analyzed by comparing the content and the hash codes for each message and any associated attachments to identify unique messages, exact duplicate messages, and near duplicate messages. A hash code appearing in a group having only one message corresponds to a unique message. A hash code appearing in a group having two or more messages corresponds to a set of exact duplicate messages with either no attachments or with identical attachments. The remaining non-duplicate messages belonging to a conversation thread are compared, along with any associated attachments, to identify any further unique messages or near duplicate messages. Optionally, the exact duplicate messages and near duplicate messages can be stored in a shadow store for data integrity and auditing purposes.

An embodiment is a system and method for evaluating a structured message store for message redundancy. A header and a message body are extracted from each of a plurality of messages maintained in a structured message store. A substantially unique hash code is calculated over at least part of the header and over the message body of each message. The messages are grouped by the hash codes. One such message is identified as a unique message within each group. In a further embodiment, the messages are grouped by conversation thread. The message body for each message within each conversation thread group is compared. At least one such message within each conversation thread group is identified as a unique message.

A further embodiment is a system and method for culling duplicative messages maintained in a structured message store. A plurality of messages maintained in a structured message store are retrieved. Each message includes a header and a message body. A substantially unique hash code is calculated over at least part of the header and over the message body. The hash codes are compared for each message within each group. Each message having an hash code matching the hash code for at least one other message within the group is culled. One such non-culled message is retained as a unique message. In a further embodiment, each such non-culled message is retained as a potential unique message. The potential unique

messages are grouped by conversation thread. The message body for each potential unique message within each conversation thread group is compared. Each potential unique message having a message body contained within at least one other message within each group is culled and one such non-culled message is retained as a unique message.

Still other embodiments of the present invention will become readily apparent to those skilled in the art from the following detailed description, wherein is described embodiments of the invention by way of illustrating the best mode contemplated for carrying out the invention. As will be realized, the invention is capable of other and different embodiments and its several details are capable of modifications in various obvious respects, all without departing from the spirit and the scope of the present invention. Accordingly, the drawings and detailed description are to be regarded as illustrative in nature and not as restrictive.

DESCRIPTION OF DRAWINGS

FIGURE 1 is a functional block diagram showing a distributed computing environment, including a system for efficiently processing messages stored in multiple message stores, in accordance with the present invention.

FIGURE 2 is a block diagram showing the system for efficiently processing messages of FIGURE 1.

FIGURE 3 is a data flow diagram showing the electronic message processing followed by the system of FIGURE 2.

FIGURE 4 is a block diagram showing the software modules of the system of FIGURE 2.

FIGURE 5 shows, by way of example, an annotated electronic message.

FIGURE 6 is a flow diagram showing a method for efficiently processing messages stored in multiple message stores, in accordance with the present invention.

FIGURE 7 is a flow diagram showing the routine for creating a shadow store for use in the method of FIGURE 6.

FIGURE 8 is a flow diagram showing the routine for processing messages for use in the method of FIGURE 6.

FIGURE 9 is a flow diagram showing the routine for processing the master array for use in the routine of FIGURE 8.

FIGURES 10A-C are flow diagrams showing the routine for processing a topic array for use in the routine of FIGURE 9.

FIGURE 11 is a flow diagram showing the routine for processing a log for use in the routine of FIGURE 8.

FIGURE 12 is a functional block diagram showing a distributed computing environment, including a system for evaluating a structured message store for message redundancy, in accordance with a further embodiment of the present invention.

FIGURE 13 is a block diagram showing the software modules of the production server of FIGURE 12.

FIGURE 14 is a data flow diagram showing the electronic message processing followed by the production server of FIGURE 13.

FIGURE 15 shows, by way of example, a database schema used by the production server of FIGURE 13.

FIGURE 16 is a flow diagram showing a method for evaluating a structured message store for message redundancy, in accordance with a further embodiment of the present invention.

FIGURES 17A-B are flow diagrams showing the routine for extracting messages for use in the method of FIGURE 16.

FIGURES 18A-C are flow diagrams showing the routine for de-duping messages for use in the method of FIGURE 16.

BEST MODE FOR CARRYING OUT THE INVENTION

FIGURE 1 is a functional block diagram showing a distributed computing environment 10, including a system for efficiently processing messages stored in multiple message stores, in accordance with the present invention. The distributed computing environment 10 includes an internetwork 16, including the Internet, and an intranetwork 13. The internetwork 16 and intranetwork 13 are interconnected via a router 17 or similar interconnection device, as is known in the art. Other network topologies, configurations, and components are feasible, as would be recognized by one skilled in the art.

Electronic messages, particularly electronic mail (email), are exchanged between the various systems interconnected via the distributed computing environment 10. Throughout this document, the terms "electronic message" and "message" are used interchangeably with the same intended meaning. In addition, message types encompass electronic mail, voice mail, images, scheduling, tasking, contact management, project management, workgroup activities, multimedia content, and other forms of electronically communicable objects, as would be recognized by one skilled in the art. These systems include a server 11 providing a message exchange service to a plurality of clients 12a, 12b interconnected via the intranetwork 13. The clients 12a, 12b can also subscribe to a remote message exchange service provided by a remote server 14 interconnected via the internetwork 16. Similarly, a remote client 15 can subscribe to

either or both of the message exchange services from the server 11 and the remote server 14 via the internetwork 16.

Each of the systems is coupled to a storage device. The server 11, clients 12a, 12b, and remote client 15 each maintain stored data in a local storage device 18. The remote server 14 maintains stored data in a local storage device (not shown) and can also maintain stored data for remote systems in a remote storage device 19, that is, a storage device situated remotely relative to the server 11, clients 12a, 12b, and remote client 15. The storage devices include conventional hard drives, removable and fixed media, CD ROM and DVD drives, and all other forms of volatile and non-volatile storage devices.

Each of the systems also maintains a message store, either on the local storage device or remote storage device, in which electronic messages are stored or archived. Each message store constitutes an identifiable repository within which electronic messages are kept and can include an integral or separate archive message store for off-line storage. Internally, each message store can contain one or more message folders (not shown) containing groups of related messages, such as an "Inbox" message folder for incoming messages, an "Outbox" message folder for outgoing messages, and the like. For clarity of discussion, individual message folders will be treated alike, although one skilled in the art would recognize that contextually related message folders might be separately processed.

In a workgroup-computing environment, the server 11 collectively maintains the message stores as a workgroup message store (WMS) 22 for each subscribing client 12a, 12b and remote client 15. In a distributed computing environment, each client 12a, 12b and remote client 15 might maintain an individual message store 21 either in lieu of or in addition to a workgroup message store 21. Similarly, the remote server 14 could maintain a workgroup message store 22 for remote clients.

Over time, each of the message stores unavoidably accumulates duplicates, at least in part, of other electronic messages stored in the message store for the same user or for other users. These duplicate and near duplicate electronic messages should be identified and removed during document pre-analysis. Thus, the server 11 includes a message processor 20 for efficiently processing the electronic messages stored in the various message stores 21, 22 as further described below beginning with reference to FIGURE 2. Optionally, an individual client 12a could also include the message processor 20. The actual homing of the message processor 20 is only limited by physical resource availability required to store and process individual message stores 21 and workgroup message stores 22.

The electronic messages are retrieved directly from the individual message stores 21, the workgroup message stores 22, or consolidated from these message stores into a combined message store. For document pre-analysis, the message stores can include both active "on-line" messages and archived "off-line" messages maintained in a local storage device 18 or remote storage device 19.

The individual computer systems including the server 11, clients 12, remote server 14, and remote client 15, are general purpose, programmed digital computing devices consisting of a central processing unit (CPU), random access memory (RAM), non-volatile secondary storage, such as a hard drive, CD ROM or DVD drive, network interfaces, and peripheral devices, including user interfacing means, such as a keyboard and display. Program code, including software programs, and data are loaded into the RAM for execution and processing by the CPU and results are generated for display, output, transmittal, or storage.

FIGURE 2 is a block diagram showing the system for efficiently processing messages of FIGURE 1. The system 30 includes the server 11, storage device 18, and one or more message stores 32. The message stores 32 could include individual message stores 21 and workgroup message stores 22 (shown in FIGURE 1). Alternatively, the system 30 could include a client 12a (not shown) instead of the server 11.

The server 11 includes the messages processor 20 and optionally operates a messaging application 31. The messaging application 31 provides services with respect to electronic message exchange and information storage to individual clients 12a, 12b, remote servers 14, and remote clients 15 (shown in FIGURE 1). On an application side, these services include providing electronic mail, scheduling, tasking, contact and project management, and related automated workgroup activities support. On a system side, these services include message addressing storage and exchange, and interfacing to low-level electronic messaging subsystems. An example of a message exchange server 31 is the Exchange Server product, licensed by Microsoft Corporation, Redmond, Washington. Preferably, the message exchange server 31 incorporates a Messaging Application Programming Interface (MAPI)-compliant architecture, such as described in R. Orfali et al., "Client/Server Survival Guide," Ch. 19, John Wiley & Sons, Inc. (1999 3d ed.), the disclosure of which is incorporated by reference. The messaging application is not a part of the present invention, but is shown to illustrate a suitable environment in which the invention may operate.

The message processor 20 processes the message stores 32 (shown in FIGURE 1) to efficiently pre-analyze the electronic messages, as further described below with reference to FIGURE 3. The message stores 32 are processed to create one or more constructs stored into a

"shadow" store 33. A point-to-point keyed collection 35 stores cross-references between the identifier of the original message store 32 or folder in the original message store and the identifier of the newly created corresponding folder or subfolder in the shadow store 33. During processing, the electronic messages are "graded" into duplicate, near duplicate and unique categories and tagged by longest conversation thread.

The results of message processing are chronicled into a log 34 to identify unique messages 44 and to create a processing audit trail for allowing the source and ultimate disposition of any given message to be readily traced. As well, a cross-reference keyed collection 36 allows unique message identifiers to be submitted and the source location information of those messages that are duplicates or near duplicates of the unique message to be retrieved. The retrieval information allows the optional reaggregation of selected unique messages and the related duplicate and near duplicates messages at a later time, such as by inclusion into the shadow store 33 at the end of the document review process. Optionally, the duplicate and near duplicate messages can be rejoined with their related unique messages for completeness. The log 34 records not only the disposition of each message, but, in the case of duplicate and near duplicate messages, indicates the unique message with which each duplicate and near duplicate message is associated, thereby permitting specific duplicate and near duplicate messages to be located and optionally reaggregated with selected unique messages at a later time. In the described embodiment, the cross-reference keyed collection 36 is maintained as part of the log 34, but is separately identified for purposes of clarity. The unique messages 44 are copied into the shadow store 33 for forwarding to the next stage of document review.

FIGURE 3 is a data flow diagram 40 showing the electronic message processing cycle followed by the system 30 of FIGURE 2. First, the various message stores 41 are opened for access. Metadata consisting of message identification information, including message source location information, and message topics (or subjects), is extracted into a "master" array 42. The master array 42 is a logical collection of the topics and identification information, in the form of metadata, for all of the messages in the various message stores 41. The metadata is manipulated in the various data structures described herein, including the master array 42, topic array 43, and arrays for unique messages 44, near duplicate messages 45, thread lengths 46, and exact duplicate messages 47. However, except as noted otherwise, the messages are described as being directly manipulated during processing, although one skilled in the art would recognize that metadata, messages, or any combination thereof could be used.

The messages in the master array 42 are sorted by topic to identify unique messages and conversation threads, as reflected by ranges of multiple occurrences of the same topic. The

identification information (metadata) for those messages having identical topics is extracted into a topic array 43 as each new topic is encountered within the master array 42.

The topic array 43 functions as a working array within which topically identical messages are processed. The identification information extracted from the master array 42 is used to copy into the topic array further information from messages sharing a common topic, including their plaintext. At any point in processing, the topic array 43 contains only those messages sharing a common topic. These topically identical messages are sorted by plaintext body and analyzed. Exact duplicate messages 47, containing substantially duplicated content, are removed from the topic array 43. The remaining non-exact duplicate messages in the topic array 43 are searched for thread markers indicating recursively-included content and conversation thread lengths 46 are tallied. The messages in the topic array 43 are compared and near duplicate messages 45 are identified. The unique messages 45 are marked for transfer into the shadow store 48.

FIGURE 4 is a block diagram showing the software modules 60 of the system 30 of FIGURE 2. Each module is a computer program, procedure or module written as source code in a conventional programming language, such as the Visual Basic programming language, and is presented for execution by the CPU as object or byte code, as is known in the art. The various implementations of the source code and object and byte codes can be held on a computer-readable storage medium or embodied on a transmission medium in a carrier wave. The message processor 20 operates in accordance with a sequence of process steps, as further described below beginning with reference to FIGURE 6.

The message processor 20 includes four primary modules: exact duplicate message selector 61, thread length selector 62, near duplicate message selector 63, and unique message selector 64. Prior to processing, the message stores 41 are logically consolidated into the master array 42. At each stage of message processing, a log entry is created (or an existing entry modified) in a log 34 to track messages and record message identification information. The exact duplicate message selector 61 identifies and removes those exact duplicate messages 47 containing substantially duplicative content from the topic array 43. The thread length selector 62 tallies the conversation thread lengths 46 and maintains an ordering of thread lengths, preferably from shortest to longest conversation thread length. The near duplicate message selector 63 designates as near duplicate messages 45 those whose content is recursively-included in other messages, such as those messages generated through a reply or forwarding sequence, or as an attachment. The unique message selector 64 designates as unique messages 45 those messages that have been extracted out of the master array 42 as not being topically identical and those messages remaining after the exact duplicate messages 48 and near duplicate messages 46

have been identified. The unique messages 45 are forwarded to the shadow store 48 for use in subsequent document review. The unique, near duplicate, and exact duplicate messages, as well as thread counts, are regularly recorded into the log 34, as the nature of each message is determined. As well, the location information permitting subsequent retrieval of each near duplicate message 45 and exact duplicate message 47 is regularly inserted into the cross-reference keyed collection 36 relating the message to a unique message as the relationship is determined.

FIGURE 5 shows, by way of example, an annotated electronic message 70. Often the message having the longest conversation thread length 47 is the most useful message to review. Each preceding message is recursively included within the message having the longest conversation thread length and therefore these near duplicate messages can be skipped in an efficient review process.

The example message 70 includes two recursively-included messages: an original e-mail message 71 and a reply e-mail message 72. The original e-mail message 71 was sent from a first user, *user1@ aol.com*, to a second user, *user2@ aol.com.* In reply to the original e-mail message 71, the second user, *user2@ aol.com,* generated the reply e-mail message 72, sent back to the first user, *user1@ aol.com.* Finally, the first user, *user1@ aol.com,* forwarded the reply e-mail message 72, which also included the original e-mail message 71, as a forwarded e-mail message 73, to a third user, *user3@ aol.com.*

Each of the e-mail messages 71, 72, 73 respectively includes a message body (recursively-included) 74, 78, 82 and a message header 75, 77, 81. The original e-mail message 71 and the reply e-mail message 72 are recursively-included messages. The original e-mail message 71 is recursively included in both the reply e-mail message 72 and forwarded e-mail message 73 while the reply e-mail message 72 is recursively included only in the forwarded e-mail message 73.

Each successive reply, forwarding or similar operation increases the conversation thread length 47 of the message. Thread lengths 47 are indicated within the messages themselves by some form of delimiter. In the example shown, the inclusion of the original e-mail message 71 in the reply e-mail message 72 is delimited by both a separator 80 and a "RE:" indicator in the subject line 79. Likewise, the inclusion of the reply e-mail message 72 is delimited by a separator 84 and a "FW:" indicator in the subject line 83. The message separators 80, 84 and subject line indicators 79, 83 constitute thread "markers" that can be searched, identified and analyzed by the message processor 20 in determining thread lengths 47 and near duplicate messages 46.

FIGURE 6 is a flow diagram showing a method 100 for efficiently processing messages stored in multiple message stores, in accordance with the present invention. The method 100 operates in two phases: initialization (blocks 101-103) and processing (blocks 104-107).

During initialization, the message stores 41 (shown in FIGURE 3) are opened for access by the message processor 20 (block 101) and the shadow store 48 is created (block 102), as further described below with reference to FIGURE 7. In the described embodiment, the message processor 20 has a finite program capacity presenting an upper bound on the maximum number of electronic messages to be processed during a single run. Consequently, multiple processing passes may be required to process all of the messages stored in the aggregate of the message stores 41.

In the described embodiment, assuming that the aggregate number of messages exceeds the program bounds, the processing is broken down into a series of passes n, during each of which a portion of the aggregate message stores 41 is processed. The number of passes n required to process the source message stores 41 is determined (block 103) by an appropriate equation, such as the following equation:

$$n = \text{ceil}\left\lceil \frac{\text{TotNumMessages}}{\text{ProgMax}} \right\rceil$$

where n equals the total number of iterative passes, *TotNumMessages* is the total number of messages in the aggregate of the message stores 41, and *ProgMax* is the maximum program message processing capacity.

In the described embodiment, the aggregate selection of messages from the message stores 41 is processed by overlapping partition i, preferably labeled by dividing the alphabet into partitions corresponding to the number of passes n. For example, if two passes n are required, the partitions would be "less than M" and "greater than L." Similarly, if 52 passes n were required, the partitions would be "less than Am" and "greater than Al and less than Ba."

During operation, the partitions, if required, are processed in an iterative processing loop (blocks 104-106). During each pass n (block 104) the messages are processed (block 105), as further described below beginning with reference to FIGURE 8. Upon the completion of the processing (block 106), the message stores 41 are closed (block 107). As an optional operation, the exact duplicate messages 47 and the near duplicates messages 45 are reinserted into the shadow store 48 (block 108). The method terminates upon the completion of processing.

FIGURE 7 is a flow diagram showing the routine 120 for creating a shadow store for use in the method 100 of FIGURE 6. The purpose of this routine is to create a holding area, called the shadow store 48 (shown in FIGURE 3) in which unique messages 45 are stored for the next

stage in document review. A message counter is maintained to count the messages in the aggregate of all message stores 41. The message counter is initially set to zero (block 121). Each of the source message stores 41 is then processed in a pair of nested iterative processing loops (blocks 122-128 and 124-129), as follows.

During the outer processing loop (blocks 122-129), a folder corresponding to each source message store 41 is created in the shadow store 48 (block 123). Next, each of the folders in the current selected source message store 41 is iteratively processed in the inner processing loop (blocks 124-128) as follows. First, the message counter is incremented by the number of messages in the folder being examined in the source message store 41 (block 125) and a corresponding folder in the shadow store 48 is created (block 126). An entry is made in a point-to-point keyed collection 35 (block 127) that constitutes a cross-reference between a pointer to the original message store 41 or folder in the original message store and a pointer to the newly created corresponding folder or subfolder in the shadow store 48. When unique messages are later copied into the shadow store 48, this keyed file allows the copying to proceed "point-to-point," rather than requiring that the folders in the shadow store 48 be iteratively searched to find the correct one. Processing of each folder in the current source message store 41 continues (block 128) for each remaining folder in the source message store. Similarly, processing of each of the source message stores themselves 41 continues (block 129) for each remaining source message store 41, after which the routine returns (block 130), providing a count of all the messages in all the source message stores so that the number of passes required can be determined.

FIGURE 8 is a flow diagram showing the routine 140 for processing messages for use in the method 100 of FIGURE 6. The purpose of this routine is to preprocess the messages stored in the message stores 41. Note at each stage of message processing, a log entry is implicitly entered into the log 34 (shown in FIGURE 3) to record the categorization and disposition of each message.

The messages are processed in a processing loop (blocks 141-144). During each iteration (block 141), each message in the selected folder is checked for membership in the current partition *i* of the source message stores 41 (block 142). If the message is in the current partition *i* (block 142), the message is logically transferred into the master array 42 (block 143) by extracting the topic and location information, including message identification information and pointers to the source message store 41, the source message folder, and to the individual message (metadata). Using metadata, rather than copying entire messages, conserves storage and memory

space and facilitates faster processing. Processing continues for each message in the selected folder (block 144).

When all folders have been processed and the metadata for those messages found to be within the partition has been transferred into the master array, message processing begins. The messages are sorted by topic (block 145) and the master array 42 is processed (block 146), as further described below with reference to FIGURE 9. Last, the log 49 is processed (block 147), after which the routine returns.

FIGURE 9 is a flow diagram showing the routine 160 for processing the master array 42 for use in the routine 140 of FIGURE 8. The purpose of this routine is to identify unique messages 44 and to process topically identical messages using the topic array 43. The routine processes the messages to identify unique and topically similar messages using an iterative processing loop (blocks 161-171). During each iteration (block 161), the topic (or subject line) of the each message in the master array 42 is compared to that of the next message in the master array 42 (block 162). If the topics match (block 163), the messages may be from the same conversation thread. If the message is the first message with the current topic to match the following message (block 164), this first message in the potential thread is marked as the beginning of a topic range (block 165) and processing continues with the next message (block 171). Otherwise, if the message is not the first message in the conversation thread (block 164), the message is skipped and processing continues with the next message (block 171).

If the topics do not match (block 163), the preceding topic range is ending and a new topic range is starting. If the current message was not the first message with that topic (block 166), the range of messages with the same topic (which began with the message marked at block 165) is processed (block 168). If the current message is the first message with the matching topic (block 166), the message is extracted as a unique message 45 (block 167) and processing continues with the next message (block 171). If the topic range has ended (block 166), each topically identical message, plus message transmission time, is logically extracted into the topic array 43 (block 168). In the described embodiment, the messages are not physically copied into the topic array 43; rather, each message is logically "transferred" using metadata into the topic array 43 to provide message source location information, which is used to add a copy of the plaintext body of the message into the topic array. The topic array 43 is sorted by plaintext body (block 169) and processed (block 170), as further described below with reference to FIGURES 10A-C. Processing continues with the next message (block 171). The routine returns upon the processing of the last message in the master array 42.

FIGURES 10A-C are flow diagrams showing the routine 180 for processing a topic array for use in the routine 160 of FIGURE 9. The purpose of this routine is to complete the processing of the messages, including identifying duplicate, near duplicate and unique messages, and counting thread lengths. The routine cycles through the topic array 43 (shown in FIGURE 3) in three iterative processing loops (blocks 181-187, 189-194 and 196-203) as follows.

During the first processing loop (blocks 181-187) each message in the topic array 43 is examined. The plaintext body of the current message is compared to the plaintext body of the next message (block 182). If the plaintext bodies match (block 183), an exact duplicate message possibly exists, pending verification. The candidate exact duplicate is verified by comparing the header information 75, 77, 81 (shown in FIGURE 5), the sender of the message (block 184), and the transmission times of each message. If the match is verified (block 185), the first message is marked as an exact duplicate of the second message and the identification information for the first and second messages and their relationship is saved into the log 49 (block 186) and cross-reference keyed collection 36 (shown in FIGURE 2). The processing of each subsequent message in the topic array 43 (block 187) continues for the remaining messages.

Next, the messages marked as exact duplicate messages are removed from the topic array 43 (block 188) and the remaining non-exact duplicate messages in the topic array 43 are processed in the second processing loop (blocks 189-194) as follows. First, each message is searched for thread markers, including separators 80, 84 and subject line indicators 79-83 (shown in FIGURE 5) (block 190). If thread markers are found (block 191), the number of thread marker occurrences *m* is counted and recorded (block 192). Otherwise, the message is recorded as having zero thread markers (block 193). In the described embodiment, the data entries having zero thread markers are included in the sorting operations. These messages have message content, but do not include other messages. Recording zero thread markers allows these "first-in-time" messages to be compared against messages which do have included messages. Processing continues for each of the remaining messages (block 194), until all remaining messages in the topic array 43 have been processed.

The topic array is next sorted in order of increasing thread markers *m* (block 195) and the messages remaining in the topic array 43 are iteratively processed in the third processing loop (block 196-203). During each processing loop (block 196), the first and subsequent messages are selected (blocks 197, 198) and the plaintext body of the messages compared (block 199). In the described embodiment, a text comparison function is utilized to allow large text blocks to be efficiently compared. If the plaintext body of the first selected message is included in the plaintext body of the second selected message (block 200), the first message is marked as a near

duplicate of the second message and identification information on the first and second messages and their relationship is saved into the log 49 and cross-reference keyed collection 36 (shown in FIGURE 2) (block 201). If the plaintext body of the first selected message is not included in the plaintext body of the second selected message and additional messages occur subsequent to the second message in the topic array 43 (block 202), the next message is selected and compared as before (blocks 198-202). Each subsequent message in the topic array is processed (block 203) until all remaining messages have been processed, after which the routine returns.

FIGURE 11 is a flow diagram showing the routine 220 for processing a log for use in the routine 140 of FIGURE 8. The purpose of this routine is to finalize the log 34 for use in the review process. Processing occurs in an iterative processing loop (block 221-226) as follows. Each message in the master array 42 is processed during each loop (block 221). If the selected message is a unique message 45 (block 222), a copy of the message is retrieved from the source folder in the source message store 41 (shown in FIGURE 3) and placed into the corresponding folder in the corresponding message store in the shadow store 48 (block 223) (using the cross-reference keyed collection 36 created at the time of creating the shadow store 34), plus an entry with message source location information and identification information is created in the log 34 (block 224). Otherwise, the message is skipped as a near duplicate message 45 or exact duplicate message 47 (block 225) that is not forwarded into the next phase of the document review process. Processing of each subsequent message in the master array 42 continues (block 226) for all remaining messages, after which the routine returns.

FIGURE 12 is a functional block diagram showing a distributed computing environment 230, including a system for evaluating a structured message store for message redundancy, in accordance with a further embodiment of the present invention. In addition to the message processor 20 executing on the server 11, a production server 231 includes a workbench application 232 for providing a framework for acquiring, logging, culling, and preparing documents for automated review and analysis. The workbench application 232 includes a production message processor (Prod MP) 233 for efficiently processing the electronic messages stored in the individual message stores 21 and the workgroup message stores 22, as further described below beginning with reference to FIGURE 13.

The production server 231 maintains an archived message store (AMS) 236 on a storage device 234 and a database 235. The production server 231 preferably functions as an off-line message processing facility, which receives individual message stores 21 and workgroup message stores 22 for document review processing as the archived message stores 236. The database 235 abstracts the contents of individual messages extracted from the archived message

stores 236 into structured message records as a form of standardized representation for efficient processing and identification of duplicative content, including attachments, as further described below with reference to FIGURE 15.

FIGURE 13 is a block diagram showing the software modules of the production server 231 of FIGURE 12. The workbench application 232 executes on the production server 231, preferably as a stand-alone application for processing messages consolidated from the individual message stores 21 and the workgroup message stores 22 into the consolidated message store 236. The workbench application 232 includes the production message processor 233 for identifying unique messages and culling out duplicate and near duplicate messages.

The production message server 233 includes five primary modules: message extractor 241, message de-duper 242, parser 243, digester 244, and comparer 245. Prior to processing, the production message processor 233 logically assembles the archived message stores 236 by first importing each individual message store 21 and workgroup message store 22 from the physical storage media upon which the message store 21, 22 is maintained. The archived message stores 236 provide a normalized electronic storage structure independent of physical storage media. Consequently, importing each individual message 21 and workgroup message store 22 can include converting the message store from a compressed or archival storage format into a standardized "working" message store format for message access and retrieval. In the described embodiment, the formats used for individual messages and message stores as used in the Outlook family of messaging applications, licensed by Microsoft Corporation, Redmond, Washington, and cc:mail family of messaging applications, licensed by Lotus Corporation, Cambridge, Massachusetts, are supported, and other messaging application formats could likewise be supported, as would be recognized by one skilled in the art. At each stage of message processing, a log entry can be created (or an existing log entry modified) in a log 247 for tracking messages and recording message identification information.

The message extractor 241 retrieves each individual message from the archived message stores 236. The parser 243 parses individual fields from each extracted message and identifies message routing, identification information and literal content within each field. The parsed metadata and message body are then stored in message records 248 maintained in the database 235, as further described below with reference to FIGURE 15. Each message record 248 includes a hash code 249 associated with the message, which is calculated by the digester 244, exclusive of any attachments. Each attachment also includes a separately calculated attachment hash code 249. Each hash code 249 is a sequence of alphanumeric characters representing the content, also referred to as a digest.

The hash codes 249 are calculated using a one-way function to generate a substantially unique alphanumeric value, including a purely numeric or alphabetic value, associated with the message or attachment. The hash codes 249 are calculated over at least part of each message header, plus the complete message body. If the message includes attachments, separate attachment hash codes 249 are calculated over at least part of each attachment. For each message, the hash code 249 can be calculated over at least part of the header, plus the complete message body. In addition, the demarcation between the data constituting a header and the data constituting a message body can vary and other logical grouping of data into headers, message bodies, or other structures or groupings are possible, as would be recognized by one skilled in the art.

In the described embodiment, the MD5 hashing algorithm, which stands for "Message Digest No. 5," is utilized and converts an arbitrary sequence of bytes having any length into a finite 128-bit digest, such as described in D. Gourley and B. Totty, "HTTP, the Definitive Guide," pp. 288-299, O'Reilly and Assocs., Sebastopol, CA (2002), the disclosure of which is incorporated by reference. Other forms of cryptographic check summing, one-way hash functions, and fingerprinting functions are possible, including the Secure Hash Algorithm (SHA), and other related approaches, as would be recognized by one skilled in the art.

Once the message records 248 in the database 235 have been populated with the extracted messages, the message de-duper 242 identifies unique messages, exact duplicate messages, and near duplicate messages, as further described below with reference to FIGURE 18. The messages are grouped by message hash codes 249 and each group of matching hash codes 249 is analyzed by comparing the content and the hash codes 249 for each message and any associated attachments to identify unique messages, exact duplicate messages, and near duplicate messages. A hash code appearing in a group having only one message corresponds to a unique message. A hash code appearing in a group having two or more messages corresponds to a set of exact duplicate messages with either no attachments or with identical attachments. Optionally, the exact duplicate messages and near duplicate messages can be maintained in a shadow store 246 for data integrity and auditing purposes.

FIGURE 14 is a data flow diagram showing the electronic message processing 260 followed by the production server 231 of FIGURE 13. First, the various archived message stores 236 are first opened for access. For each message in each of the archived message stores 236, metadata consisting of message routing, identification information and literal content are extracted. The metadata and message body, exclusive of any attachments, are calculated into a message hash code 261. In tandem, any attachments 262 are calculated into attachment hash

codes 263. The metadata, message body, hash code 261, and hash codes 263 for any attachments are stored into the database 235 as message records 264. Each of the message records 264 is uniquely identified, as further described below with reference to FIGURE 15. Finally, the message records 264 are retrieved from the database 235 and processed to identify unique messages 265, exact duplicate messages 266, and near duplicate messages 267, as further described below with reference to FIGURE 18.

FIGURE 15 shows, by way of example, a database schema 270 used by the production server 231 of FIGURE 13. The message records 248 in the database 235 are preferably structured in a hierarchical organization consisting of tables for individual message files 271, mail properties (MailProperties) 272, compound documents (CompoundDocs) 273, and compound members (CompoundMembers) 274, although other forms of hierarchical and non-hierarchical organization are feasible, as would be recognized by one skilled in the art.

The files table 271 stores one record for each individual message extracted from the archived message stores 236. Each record in the files table 271 shares a one-to-one relationship with an extracted message. Each record is assigned a unique, monotonically increasing identification number (*id*) 275. The files table 271 includes fields for storing the extracted message name 276, type 277, type confirmation 278, path 279, length 280, modified date 281, created date 282, description 283, owner key 284, and Bates tag 286. In addition, the hash code 261 for the extracted message, exclusive of any attachments, is stored in a hash code field 285.

The mail properties table 272 contains the message routing, identification information and literal content associated with each extracted message. Each record in the mail properties table 272 shares a one-to-one relationship with an associated record in the files table 271. Each record in the mail properties table 272 is identified by a file identifier (*FileId*) 287. The mail properties table 272 includes fields for storing message unique ID 288, sent from 289, sent to 290, sent cc 291, sent bcc 292, sent date 293, subject 294, thread subject 295, and message 296. The hash code 261 is calculated by the digester 244 using select fields 302 of each record, which include all of the fields except the file identifier 287 and message unique ID 288 fields, although one skilled in the art would recognize that other combinations and selections of fields could also be used to calculate the hash code 261.

The compound documents table 273 and compound members table 274 share a one-to-many relationship with each other. The records in the compound documents table 273 and compound members table 274 store any attachments associated with a given extracted message stored in a record in the file table 271. Each record in the compound documents table 273 contains a root file identifier (*routeFileId*) 297. The compound documents table 273 includes

fields for storing marked category 299 and the hash code 263 is stored in a hash code field 298. Each record in the compound documents table 273 shares a one-to-many relationship with each attachment associated with an extracted message. Similarly, each record in the compound members 274 is uniquely identified by a file ID (*FileId*) 300 field and a compound document key field 301.

FIGURE 16 is a flow diagram showing a method 310 for evaluating a structured message store for message redundancy, in accordance with a further embodiment of the present invention. The method 310 operates in three phases. During the first phase, the individual message stores 21 and workgroup message stores 22 are obtained and consolidated into the archived message stores 236 (block 311). The individual message stores 21 and workgroup message stores 22 can be in physically disparate storage formats, such as on archival tapes or other forms of on-line or off-line archival media, and could constitute compressed data. Consequently, each of the individual message stores 21 and workgroup message stores 22 are converted into a standardized on-line format for message identity processing.

During the second phase, individual messages are extracted from the archived message stores 236 (block 213), as further described below with reference to FIGURE 17. Briefly, individual messages are extracted from the archived message stores 236, digested into hash codes 261 and 263, and stored as message records 248 in the database 235.

During the third phase, the extracted messages, as stored in message records 248 in the database 235, are "de-dupped," that is, processed to identify unique messages 265, exact duplicate messages 266, and near duplicate messages 267 (block 313). Finally, the routine terminates.

FIGURES 17A-B are flow diagrams showing the routine 320 for extracting messages for use in the method 310 of FIGURE 16. The purpose of this routine is to iteratively process each of the extracted message stores 236 and individual messages to populate the message records 239 stored in the database 235.

The messages in each of the archived message stores 236 are iteratively processed in a pair of nested processing loops (blocks 321-333 and blocks 322-332, respectively). Each of the archived message stores 236 is processed during an iteration of the outer processing loop (block 321). Each message stored in an archived message store 236 is processed during an iteration of the inner processing loop (block 322). Each message is extracted from an archived message store 236 (block 322) and each extracted message is digested into a hash code 261 over at least part of the header, plus the complete message body, exclusive of any attachments (block 324). Each hash code is a sequence of alphanumeric characters representing the content, also referred

to as a digest. The hash codes are calculated using a one-way function to generate a substantially unique alphanumeric value, including a purely numeric or alphabetic value, associated with message or attachment. In the described embodiment, the MD5 hashing algorithm is used to form a fixed-length 128-bit digest of each extracted message and routing information. Next, the metadata for each extracted message is parsed and stored into records in the files table 271 and mail properties table 272 along with the hash code 261 and indexed by a unique identifier 275 (block 325).

If the extracted message contains one or more attachments (block 326), each attachment is iteratively processed (blocks 327-329) as follows. At least part of each attachment is digested by the digester 244 into a hash code 263 (block 328). Each remaining attachment is iteratively processed (block 329). The message hash code 261 and each attachment hash code 263 are concatenated into a compound hash code and are stored as a compound document record in the compound documents table 273 and the compound members table 274 (block 330). Note the message hash code 261 and each attachment hash code 263 could also be logically concatenated and stored separately, as would be recognized by one skilled in the art. Each message in the archived message store 236 is iteratively processed (block 331) and each archived message store 236 is iteratively processed (block 332), after which the routine returns.

FIGURES 18A-C are flow diagrams showing the routine 340 for de-duping messages for use in the method 310 of FIGURE 16. The purpose of this routine is to identify unique messages 265, exact duplicate messages 266, and near duplicate messages 267 ("de-dup") through a process known as "culling."

The messages stored in records in the database 235 are iteratively processed in a processing loop (blocks 341-346). Each message is processed during an iteration of the processing loop (block 341). First, the file record 271 corresponding to each message is retrieved from the database 235 (block 342). If the message is not a compound message, that is, the message does not contain attachments (block 343), the message hash code 261 is obtained (block 344) and processing continues with the next message (block 346). Otherwise, if the message is a compound message (block 343), the compound hash code is obtained (block 345) and processing continues with the next message (block 346).

Next, the messages are grouped by matching hash codes (block 347) and each group of matching hash codes is iteratively processed in a processing loop (blocks 348-351). Any groups with more than one message are processed to identify exact duplicates based on matching hash codes. A randomly selected message in the group is marked as a unique message (block 349) and the remaining messages in the group are culled, that is, marked as exact duplicates messages

(block 350). Other methodologies for selecting the unique message can be used, as would be recognized by one skilled in the art. Processing continues with the next group (block 351).

Next, all non-exact duplicate messages are now iteratively processed for near-duplicates. The messages are grouped by conversation thread (block 352). In the described embodiment, the messages are sorted in descending order of message body length (block 353), although the messages could alternatively be sorted in ascending order, as would be recognized by one skilled in the art. The threads, messages, and "shorter" messages are then iteratively processed in a series of nested processing loops (blocks 354-365, 355-364, and 356-363, respectively). Each thread is processed during an iteration of the outer processing loop (block 354). Each message within the thread is processed during an iteration of an inner processing loop (block 355) and each message within the thread having an equal or shorter length, that is, each shorter message, is processed during an iteration of an innermost processing loop (block 356). The message bodies of the first message and the shorter message are compared (block 357). If the message bodies are not contained within each other (block 358), the shorter message is left marked as a unique message and the processing continues with the next shorter message (block 363).

Otherwise, if the message body of the shorter message is contained within the message body of the first message (block 358), the attachment hash codes 263 are compared (block 359) to identify unique messages 265 and near duplicate messages 267, as follows. First, if the message does not include any attachments, the shorter message is culled, that is, marked as a near duplicate of the first message (block 362). If the message includes attachments (block 359), the individual attachment hash codes 263 are compared to identify a matching or subset relationship (block 360). If the attachment hash codes 263 indicate a matching or subset relationship between the first message and the shorter message (block 361), the shorter message is culled, that is, marked as a near duplicate message 267 of the first message (block 362). Otherwise, the shorter message is left marked as a unique message 265. Processing continues with the next shorter message in the thread (block 363). After all shorter messages have been processed (block 363), processing continues with the next message (block 364) and next thread (block 365), respectively. The routine then returns.

While the invention has been particularly shown and described as referenced to the embodiments thereof, those skilled in the art will understand that the foregoing and other changes in form and detail may be made therein without departing from the spirit and scope of the invention.

CLAIMS:

1. A system (230) for evaluating a structured message store (236) for message redundancy, comprising:

a parser (243) extracting a header (75) and a message body (74) from each of a plurality of messages (70) maintained in a structured message store (236);

a digester (244) calculating a substantially unique hash code (249) over at least part of the header (75) and over the message body (74) of each message (70); and

a message deduper (242) grouping the messages (70) by the hash codes (261) and identifying one such message (70) as a unique message (265) within each group.

2. A system (230) according to Claim 1, further comprising:

a comparer (245) grouping the messages (70) by conversation thread (295), comparing the message body (74) for each message (70) within each conversation thread group, and identifying at least one such message (70) within each conversation thread group as a unique message (265).

3. A system (230) according to Claim 2, further comprising:

a sorter (233) sorting the messages (70) in each conversation thread group by length (46), and identifying one such message (70) having a longest length (46) as the unique message (265).

4. A system (230) according to Claim 1, wherein a plurality of the messages (70) each further comprise at least one attachment (262), for each such message (70), further comprising:

an attachment digester (244) calculating a substantially unique hash code (263) over at least part of the at least one attachment (262) for each such message (70);

a comparer (245), comprising:

a concatenator logically concatenating the hash code (261) for the message (70) and the hash code (263) for the at least one attachment (262); and

an attachment deduper grouping the messages (70) by the logically concatenated hash codes (261,263), comparing the logically concatenated hash codes (261,263) for each message (70) within each group, and identifying each message (70) with a matching logically concatenated hash code (261,263) as an exact duplicate message (266).

5. A system (230) according to Claim 4, further comprising:

the comparer (245) identifying each message (70) with one or more hash codes (263) for the at least one attachment (262) comprising a subset of the hash codes (263) for the at least one attachment (262) for at least one other message (70) within each group as a near duplicate message (267).

6. A system (230) according to Claim 5, further comprising:

the comparer (245) identifying each message (70) without such a subset of the hash codes (263) as a unique message (265).

7. A system (230) according to Claim 1, further comprising:

a structured database (235) storing the messages (70) with each message (70) maintained in a separate indexed record (248) in accordance with a database schema (270).

8. A system (230) according to Claim 1, further comprising:

a shadow store (246) storing the non-unique messages (266,267).

9. A system (230) according to Claim 1, further comprising:

a log (247) tracking message processing.

10. A system (230) according to Claim 1, wherein each message header (75) further comprises routing data (302) comprising one or more data fields selected from the group comprising recipient (290), sender (289), copy-to (291), blind-copy-to (292), and subject (294).

11. A system (230) according to Claim 1, wherein each hash code (261,263) is calculated using a one-way function and comprises alphanumeric, numeric, and alphabetic character strings.

12. A system (230) according to Claim 11, wherein the one-way function is selected from the group comprising at least one of the MD5 and secure hashing algorithms.

13. A method (310) for evaluating a structured message store (236) for message redundancy, comprising:

extracting (320) a header (75) and a message body (74) from each of a plurality of messages (70) maintained in a structured message store (236);

calculating (324) a substantially unique hash code (249) over at least part of the header (75) and over the message body (74) of each message (70); and

grouping (347) the messages (70) by the hash codes (261) and identifying one such message (70) as a unique message (265) within each group.

14. A method (310) according to Claim 13, further comprising:

grouping (352) the messages (70) by conversation thread (295);

comparing (357) the message body (74) for each message (70) within each conversation thread group; and

identifying (358) at least one such message (70) within each conversation thread group as a unique message (265).

15. A method (310) according to Claim 14, further comprising:

sorting (353) the messages (70) in each conversation thread group by length (46), and identifying (357) one such message (70) having a longest length (46) as the unique message (265).

16. A method (310) according to Claim 13, wherein a plurality of the messages (70) each further comprise at least one attachment (262), for each such message (70), further comprising:

calculating (328) a substantially unique hash code (263) over at least part of the at least one attachment (262) for each such message (70);

logically concatenating the hash code (261) for the message (70) and the hash code (263) for the at least one attachment (262);

grouping the messages (70) by the logically concatenated hash codes (261,263);

comparing the logically concatenated hash codes (261,263) for each message (70) within each group; and

identifying each message (70) with a matching logically concatenated hash code (261,263) as an exact duplicate message (266).

17. A method (310) according to Claim 16, further comprising:

identifying each message (70) with one or more hash codes (263) for the at least one attachment (262) comprising a subset of the hash codes (263) for the at least one attachment (262) for at least one other message (70) within each group as a near duplicate message (267).

18. A method (310) according to Claim 17, further comprising:

identifying each message (70) without such a subset of the hash codes (263) as a unique message (265).

19. A method (310) according to Claim 13, further comprising:

maintaining a structured database (235) storing the messages (70) with each message (70) maintained in a separate indexed record (248) in accordance with a database schema (270).

20. A method (310) according to Claim 13, further comprising:

maintaining a shadow store (246) storing the non-unique messages (266,267).

21. A method (310) according to Claim 13, further comprising:

maintaining a log (247) tracking message processing.

22. A method (310) according to Claim 13, wherein each message header (75) further comprises routing data (302) comprising one or more data fields selected from the group comprising recipient (290), sender (289), copy-to (291), blind-copy-to (292), and subject (294).

23. A method (310) according to Claim 13, wherein each hash code (261,263) is calculated using a one-way function and comprises alphanumeric, numeric, and alphabetic character strings.

24. A method (310) according to Claim 23, wherein the one-way function is selected from the group comprising at least one of the MD5 and secure hashing algorithms

25. A computer-readable storage medium (234) holding code for performing the method (310) of Claim 13.

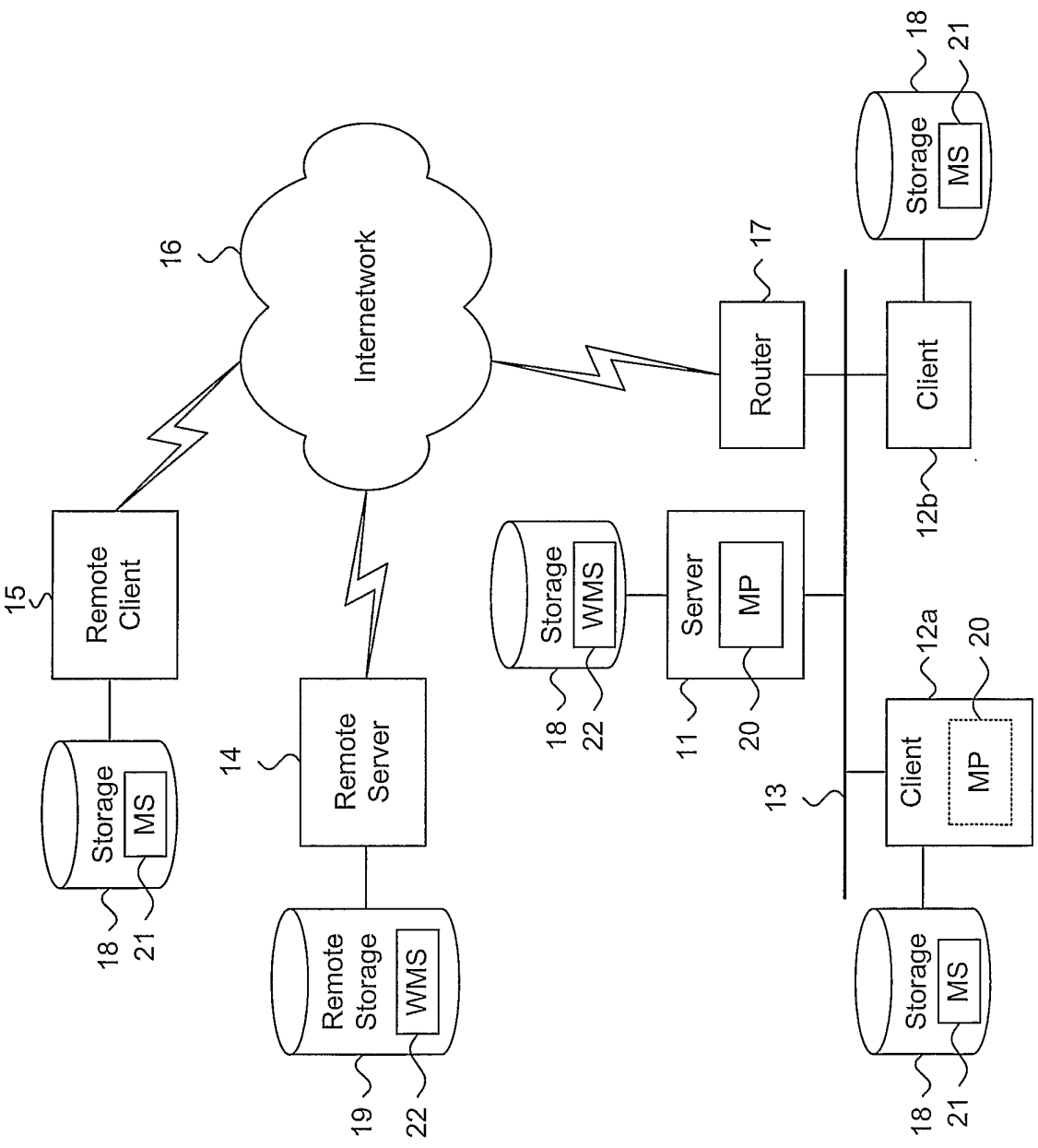
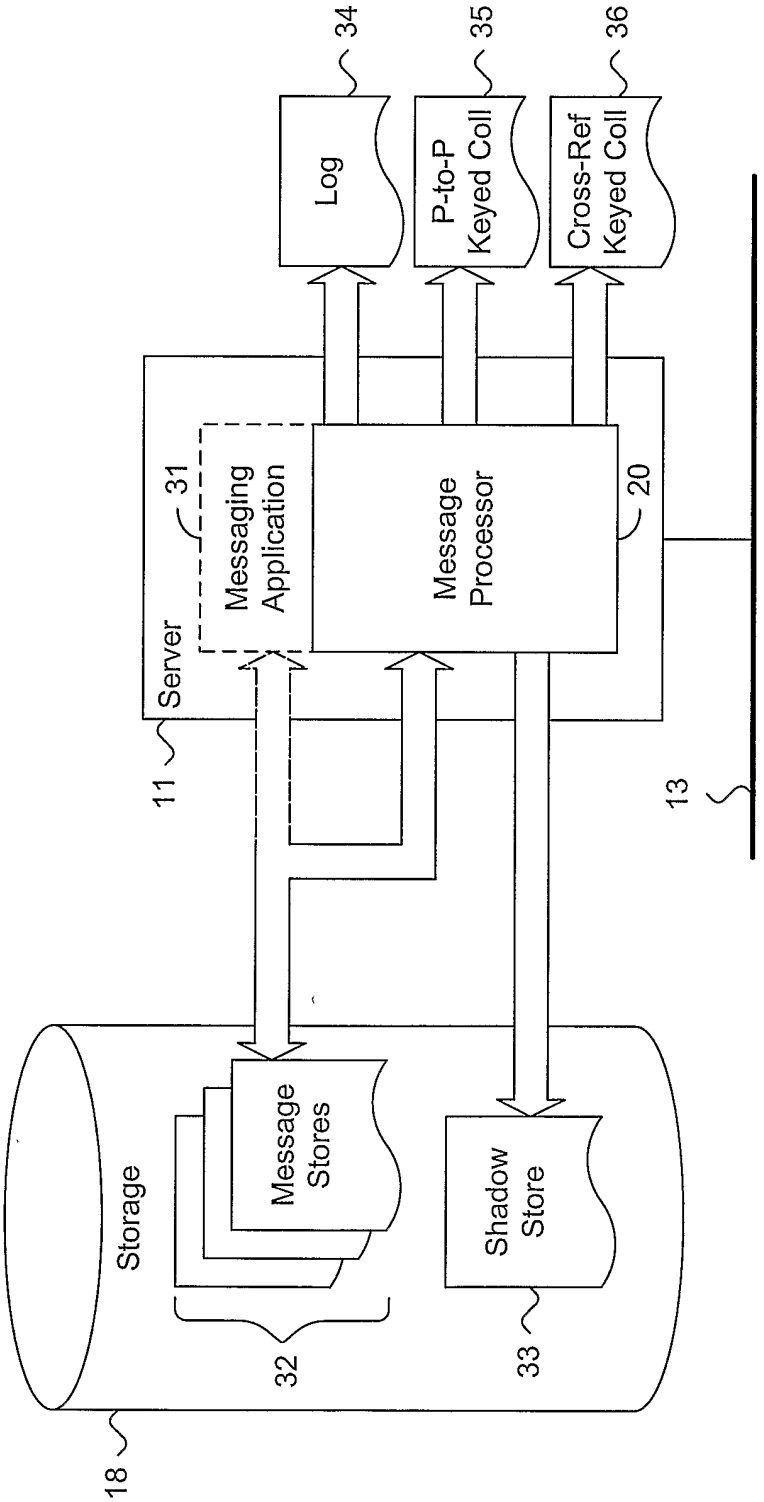

10
Internetwork
16
Remote Client
15
Storage
MS
18
21
Remote Server
14
Remote Storage
WMS
19
22
Router
17
Storage
WMS
18
22
Server
MP
11
20
13
Client
MP
12a
20
Client
12b
Storage
MS
18
21
Storage
MS
18
21

Fig. 1.

Fig. 2.


30
18
Storage
Message Stores
32
Shadow Store
33
11
Server
31
Messaging Application
Message Processor
20
13
34
Log
35
P-to-P Keyed Coll
36
Cross-Ref Keyed Coll

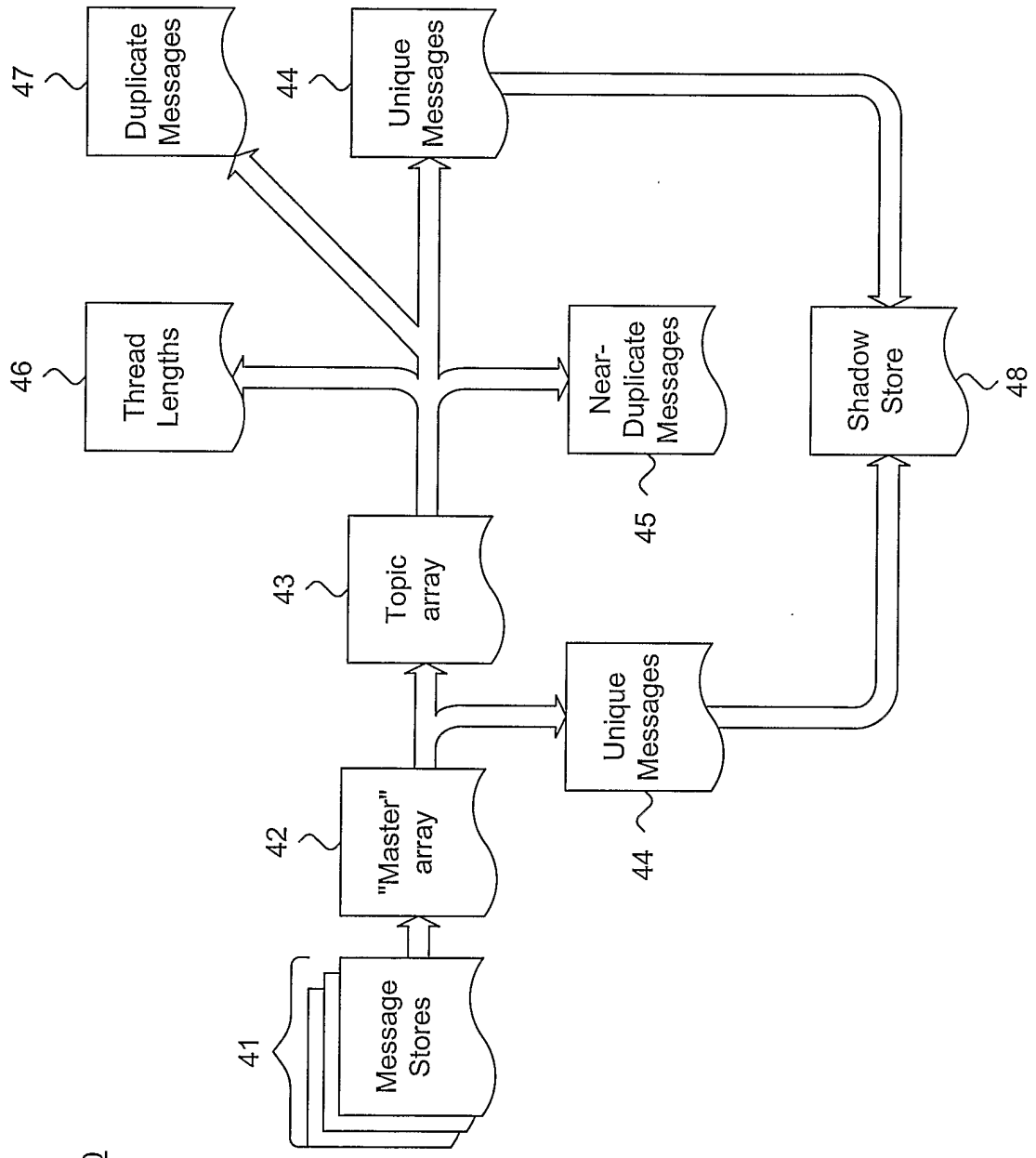

40
41
Message Stores
42
"Master" array
43
Topic array
44
Unique Messages
45
Near-Duplicate Messages
46
Thread Lengths
47
Duplicate Messages
44
Unique Messages
48
Shadow Store

Fig. 3.

Fig. 4.

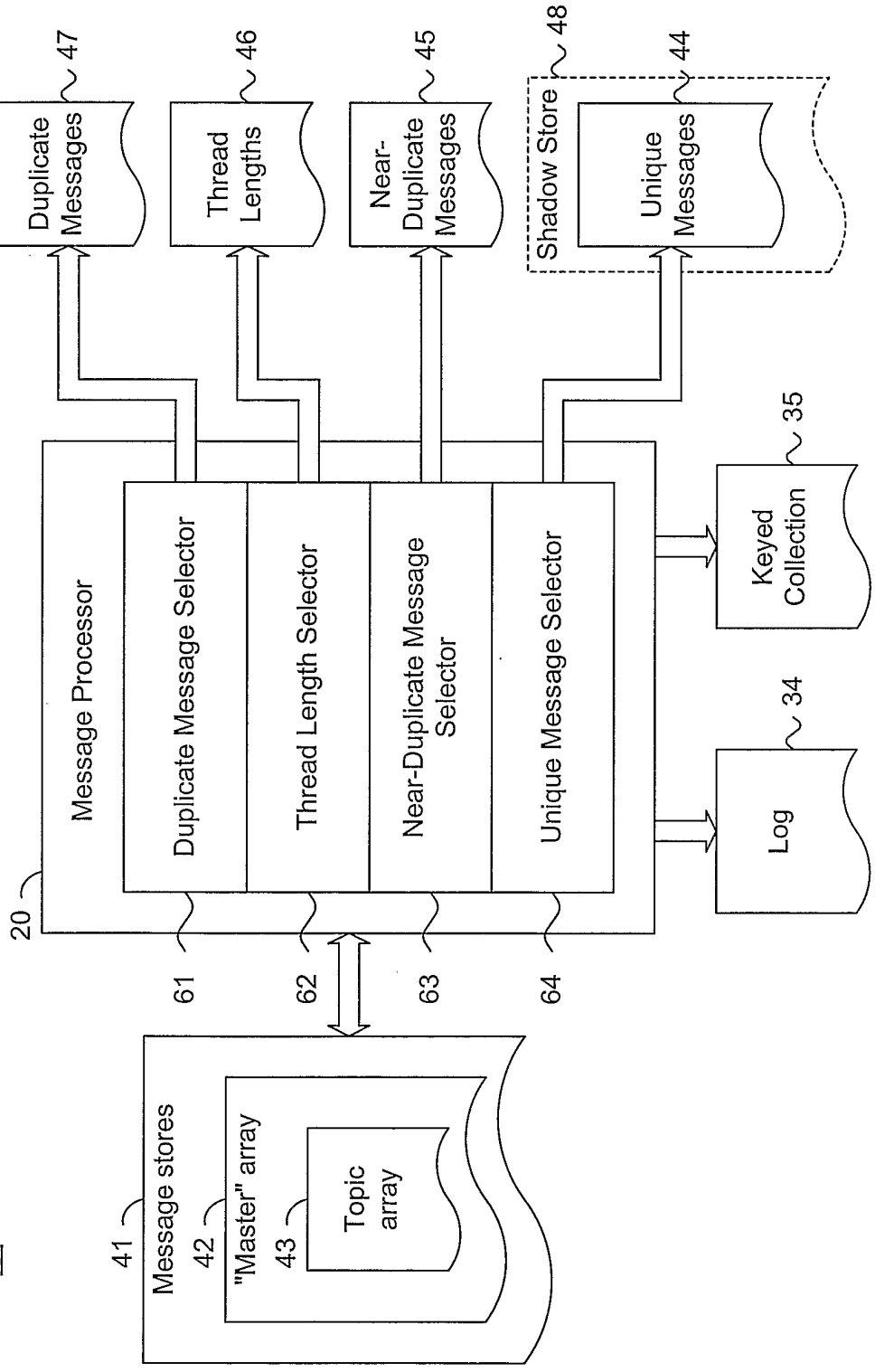

60
Message stores
41
"Master" array
42
Topic array
43
Message Processor
20
Duplicate Message Selector
Thread Length Selector
Near-Duplicate Message Selector
Unique Message Selector
61
62
63
64
Duplicate Messages
47
Thread Lengths
46
Near-Duplicate Messages
45
Shadow Store
48
Unique Messages
44
Keyed Collection
35
Log
34

Fig. 5.

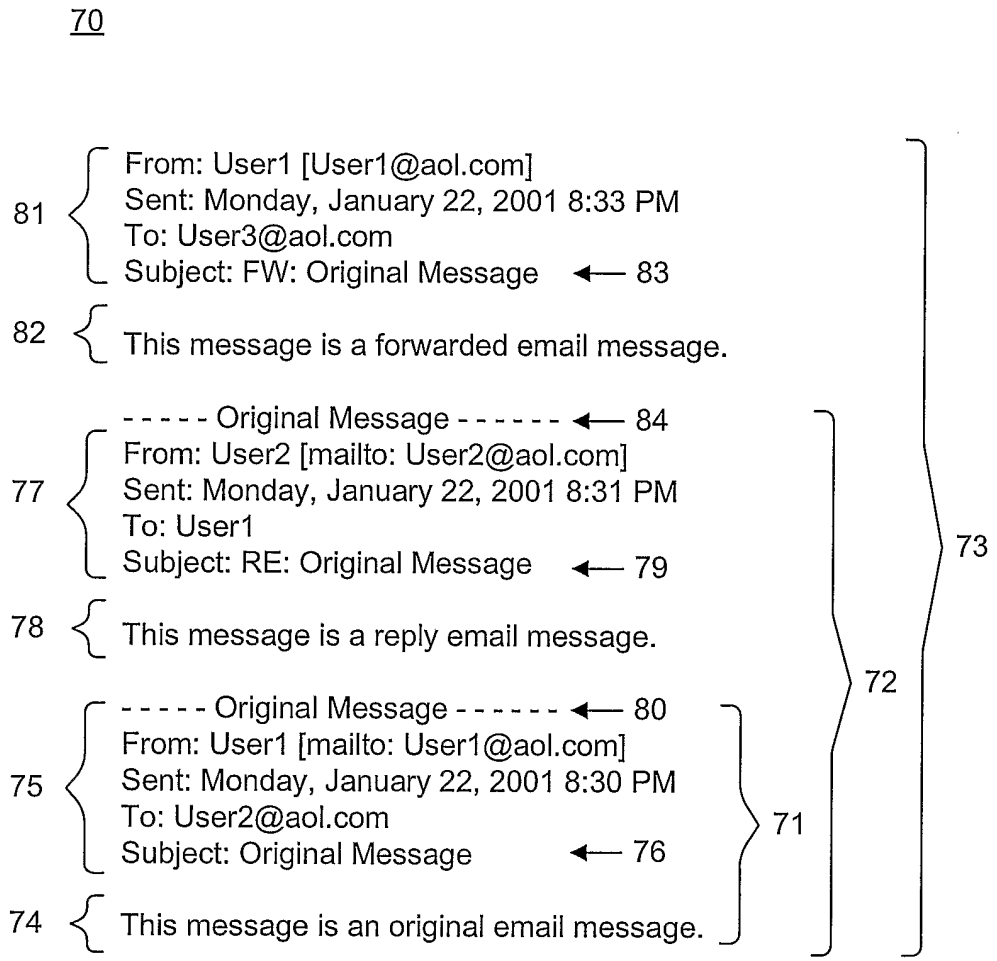

70
81
From: User1 [User1@aol.com]
Sent: Monday, January 22, 2001 8:33 PM
To: User3@aol.com
Subject: FW: Original Message
83
82
This message is a forwarded email message.
- - - - - Original Message - - - - - -
84
77
From: User2 [mailto: User2@aol.com]
Sent: Monday, January 22, 2001 8:31 PM
To: User1
Subject: RE: Original Message
79
78
This message is a reply email message.
- - - - - Original Message - - - - - -
80
75
From: User1 [mailto: User1@aol.com]
Sent: Monday, January 22, 2001 8:30 PM
To: User2@aol.com
Subject: Original Message
76
74
This message is an original email message.
71
72
73

Fig. 6.

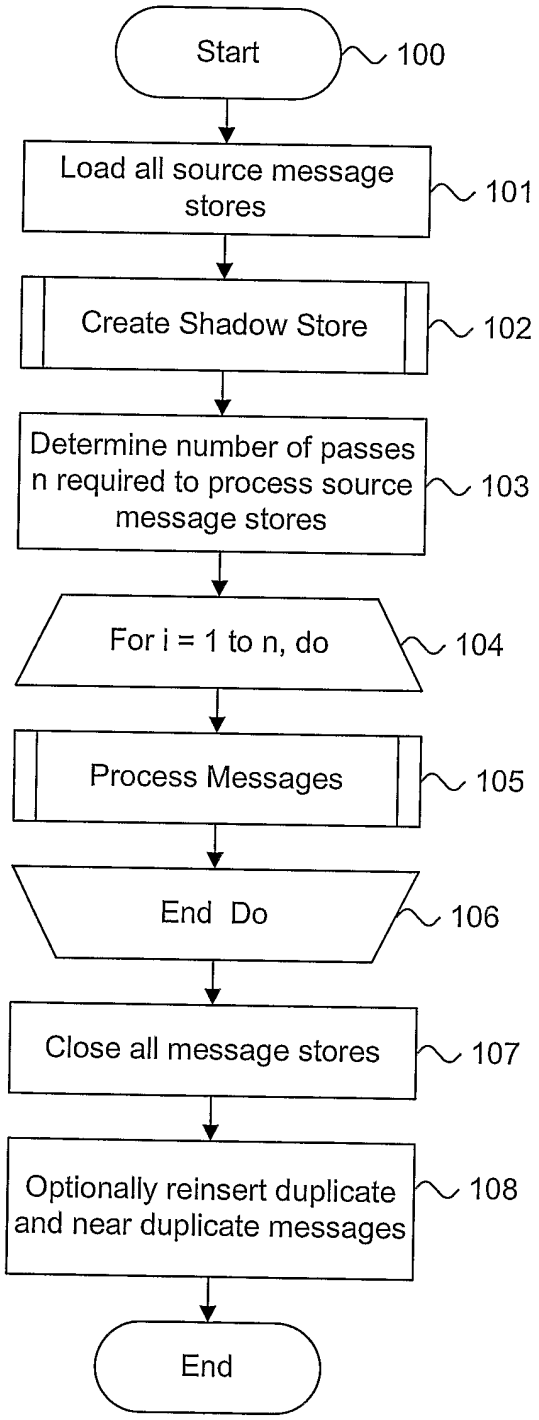

Start
100
Load all source message stores
101
Create Shadow Store
102
Determine number of passes n required to process source message stores
103
For i = 1 to n, do
104
Process Messages
105
End Do
106
Close all message stores
107
Optionally reinsert duplicate and near duplicate messages
108
End

Fig. 7.

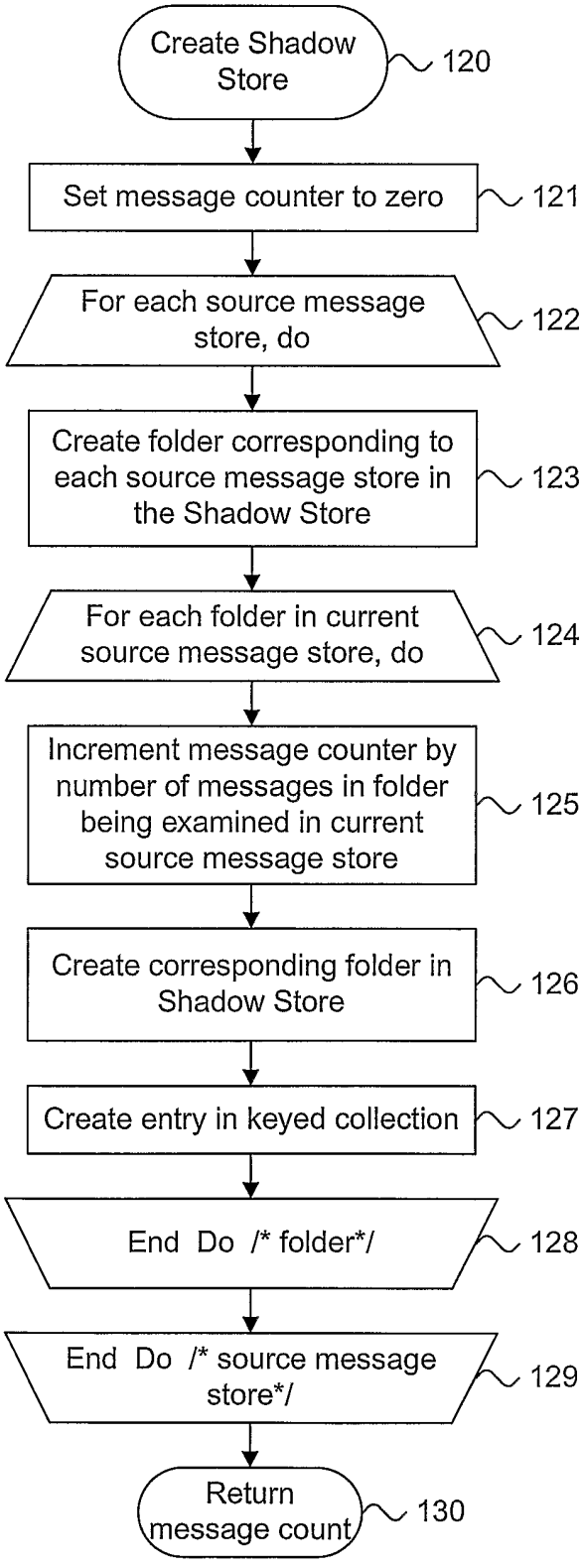

Create Shadow Store
120
Set message counter to zero
121
For each source message store, do
122
Create folder corresponding to each source message store in the Shadow Store
123
For each folder in current source message store, do
124
Increment message counter by number of messages in folder being examined in current source message store
125
Create corresponding folder in Shadow Store
126
Create entry in keyed collection
127
End Do /* folder*/
128
End Do /* source message store*/
129
Return message count
130

Fig. 8.

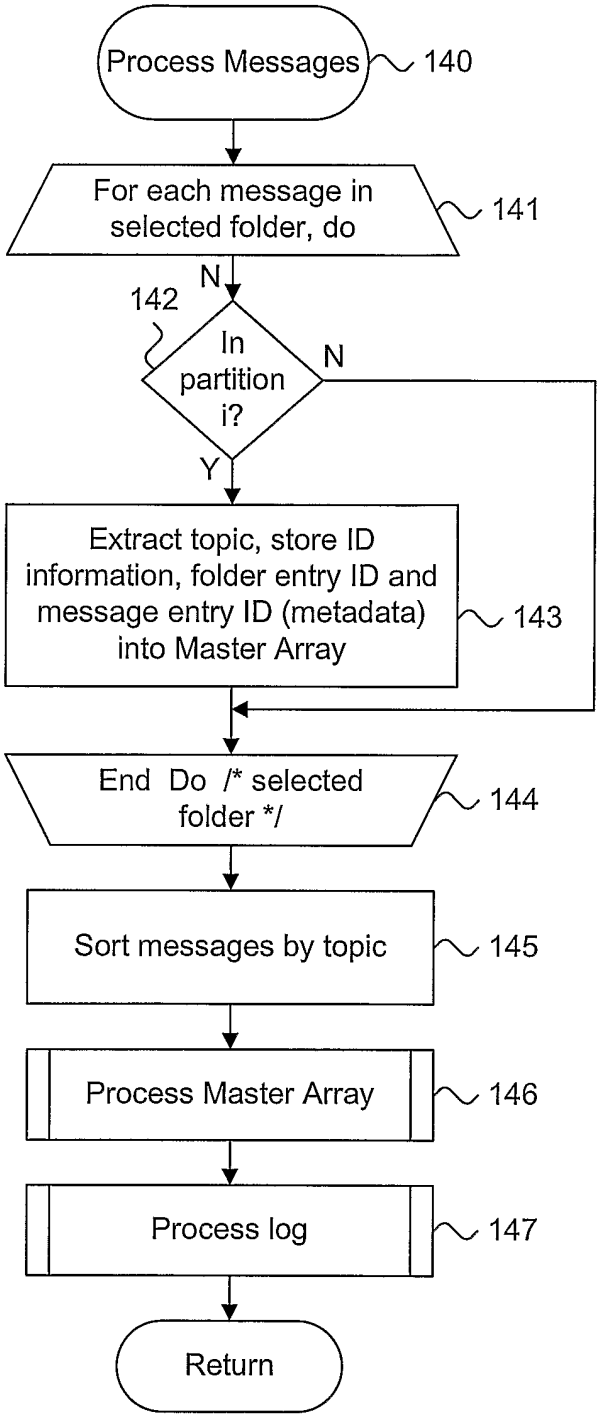

Process Messages
140
For each message in selected folder, do
141
N
142
In partition i?
N
Y
Extract topic, store ID information, folder entry ID and message entry ID (metadata) into Master Array
143
End Do /* selected folder */
144
Sort messages by topic
145
Process Master Array
146
Process log
147
Return

Fig. 9.

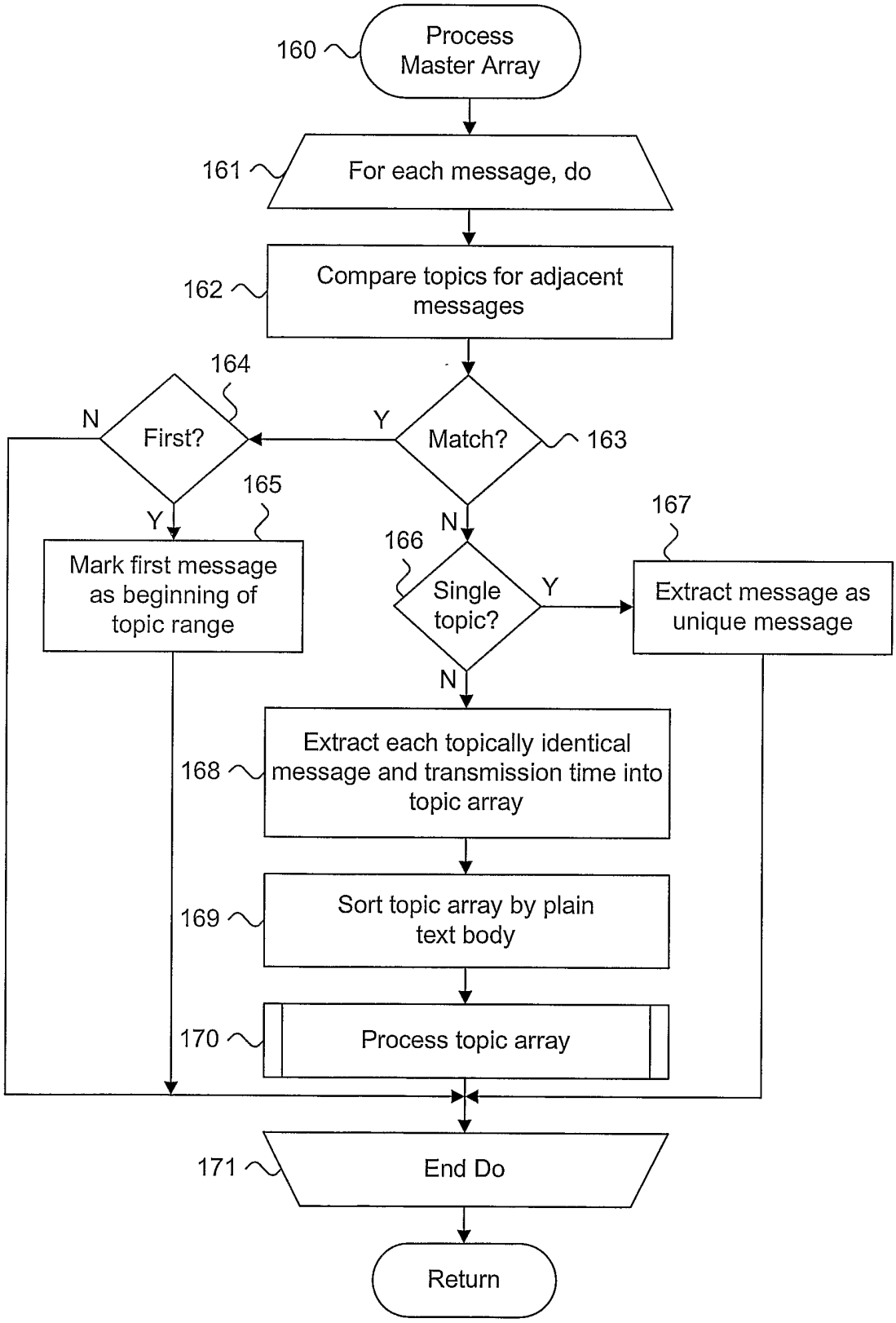

160
Process Master Array
161
For each message, do
162
Compare topics for adjacent messages
164
First?
N
Y
Match?
163
Y
165
Mark first message as beginning of topic range
N
166
Single topic?
Y
167
Extract message as unique message
N
168
Extract each topically identical message and transmission time into topic array
169
Sort topic array by plain text body
170
Process topic array
171
End Do
Return

Fig. 10A.

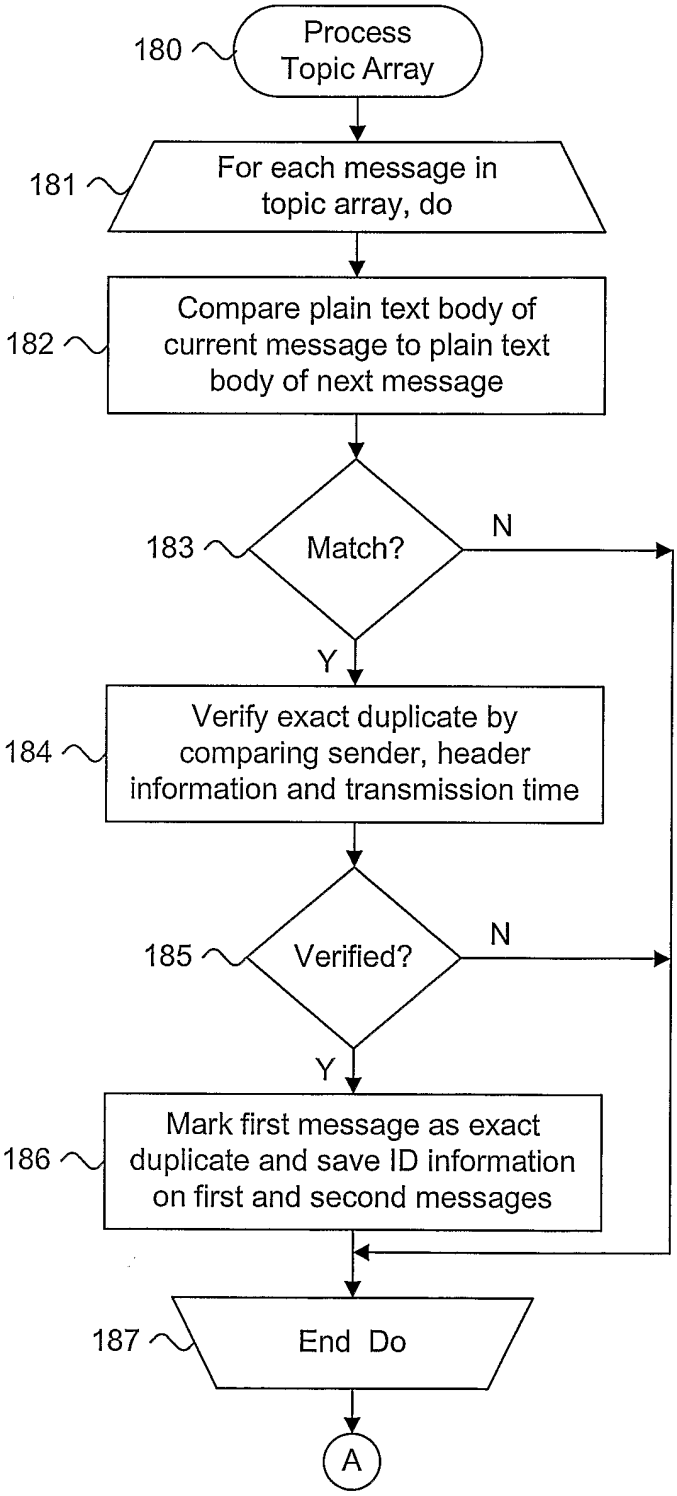

180
Process Topic Array
181
For each message in topic array, do
182
Compare plain text body of current message to plain text body of next message
183
Match?
N
Y
184
Verify exact duplicate by comparing sender, header information and transmission time
185
Verified?
N
Y
186
Mark first message as exact duplicate and save ID information on first and second messages
187
End Do
A

Fig. 10B.

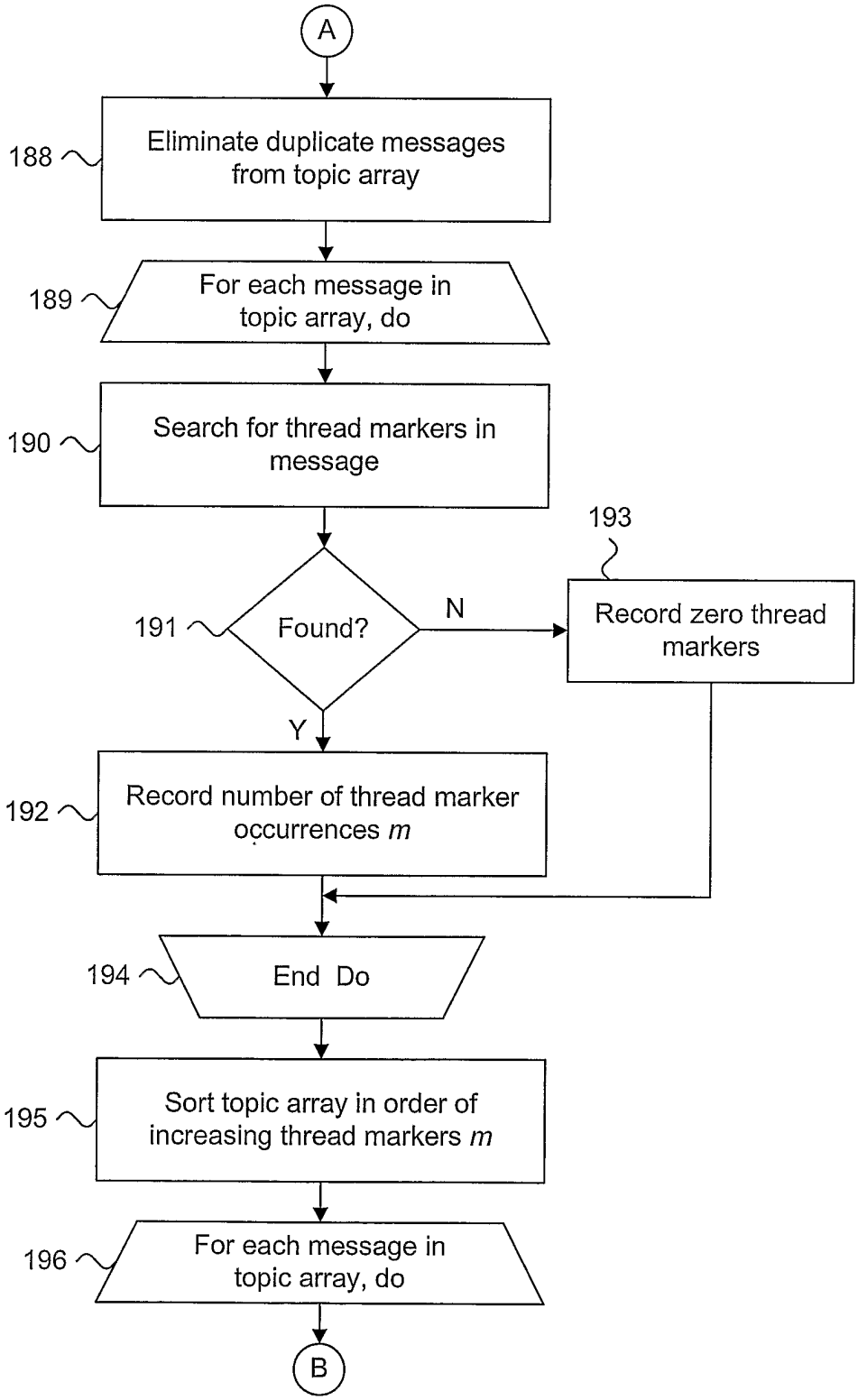

A
188
Eliminate duplicate messages from topic array
189
For each message in topic array, do
190
Search for thread markers in message
193
191
Found?
N
Record zero thread markers
Y
192
Record number of thread marker occurrences m
194
End Do
195
Sort topic array in order of increasing thread markers m
196
For each message in topic array, do
B

Fig. 10C.

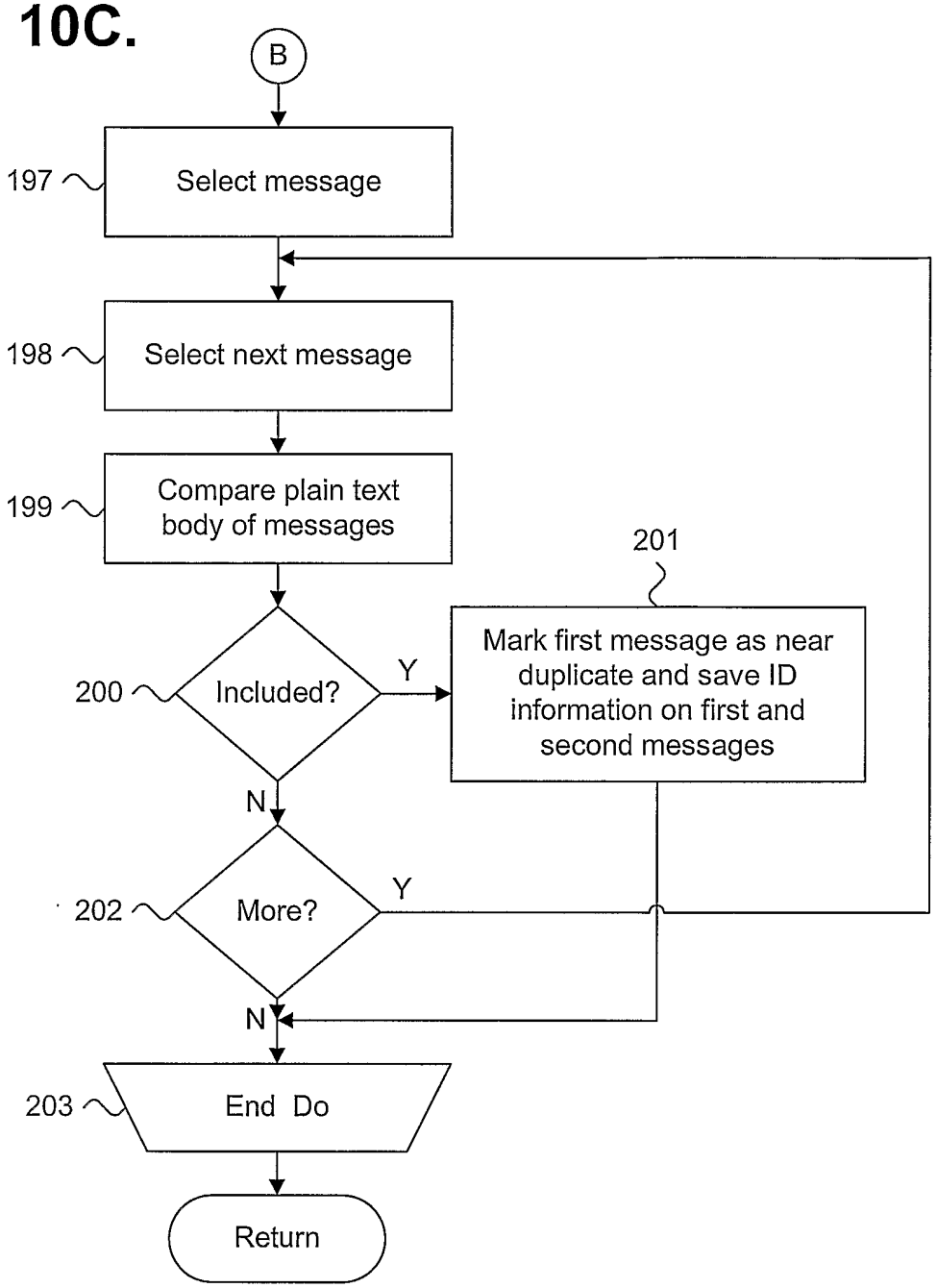
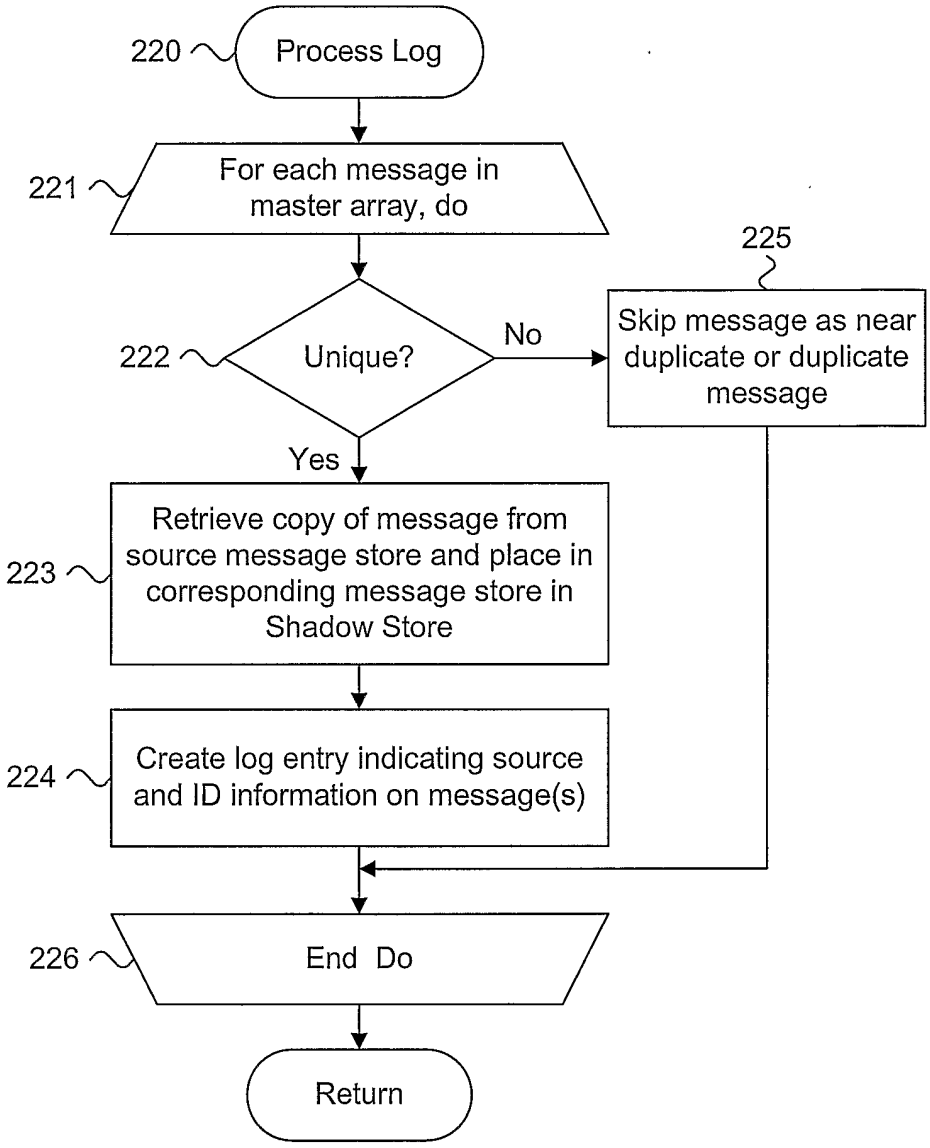

B
197
Select message
198
Select next message
199
Compare plain text body of messages
201
200
Included?
Y
Mark first message as near duplicate and save ID information on first and second messages
N
202
More?
Y
N
203
End Do
Return

Fig. 11.


220
Process Log
221
For each message in master array, do
222
Unique?
No
225
Skip message as near duplicate or duplicate message
Yes
223
Retrieve copy of message from source message store and place in corresponding message store in Shadow Store
224
Create log entry indicating source and ID information on message(s)
226
End Do
Return

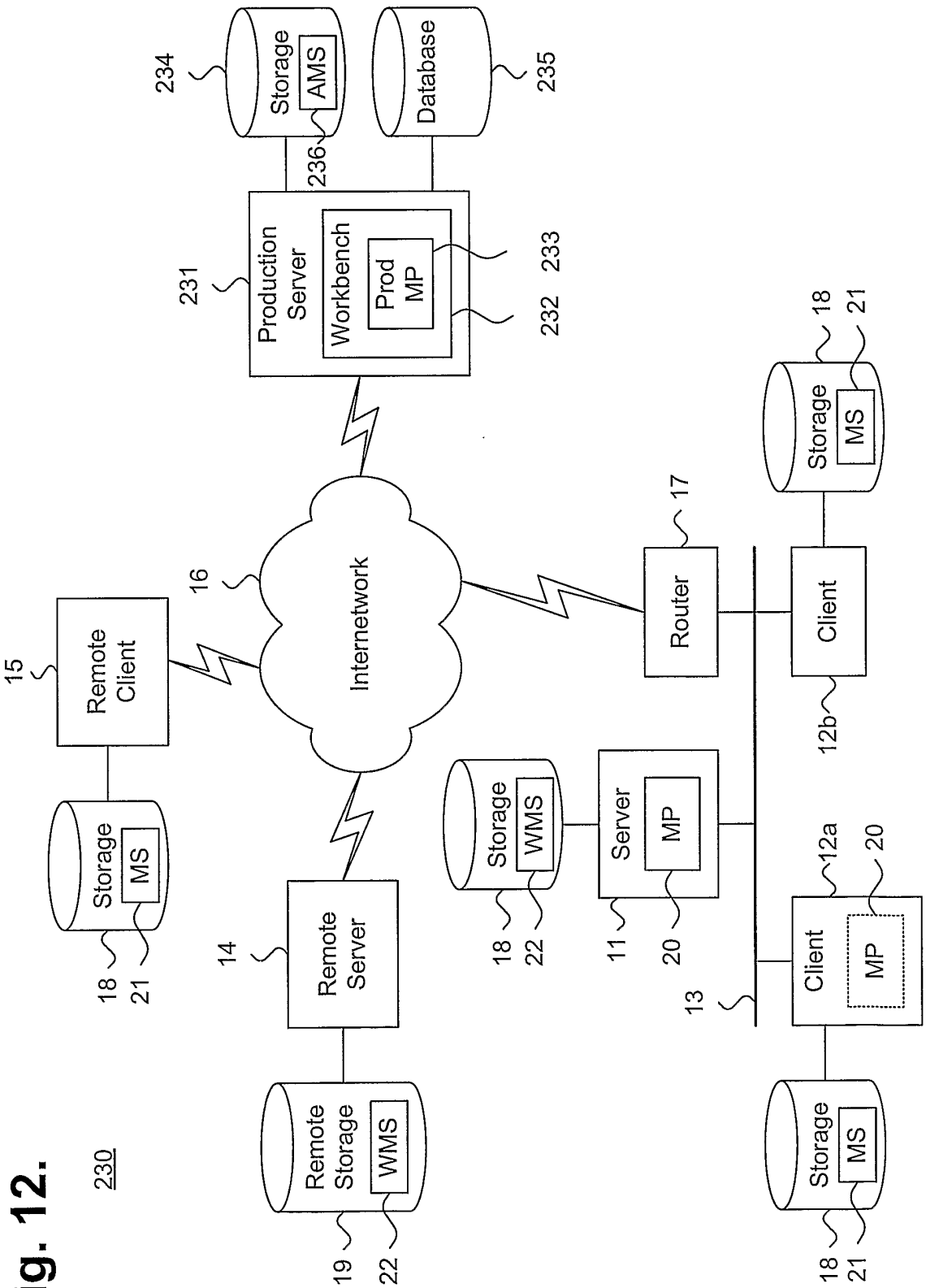

230
Fig. 12.
Storage
AMS
234
236
Database
235
Production Server
231
Workbench
Prod MP
232
233
Internetwork
16
Remote Client
15
Storage
MS
18
21
Remote Server
14
Remote Storage
WMS
19
22
Storage
WMS
18
22
Server
MP
11
20
Router
17
Storage
MS
18
21
Client
12b
Client
MP
12a
20
13
Storage
MS
18
21

Fig. 13.

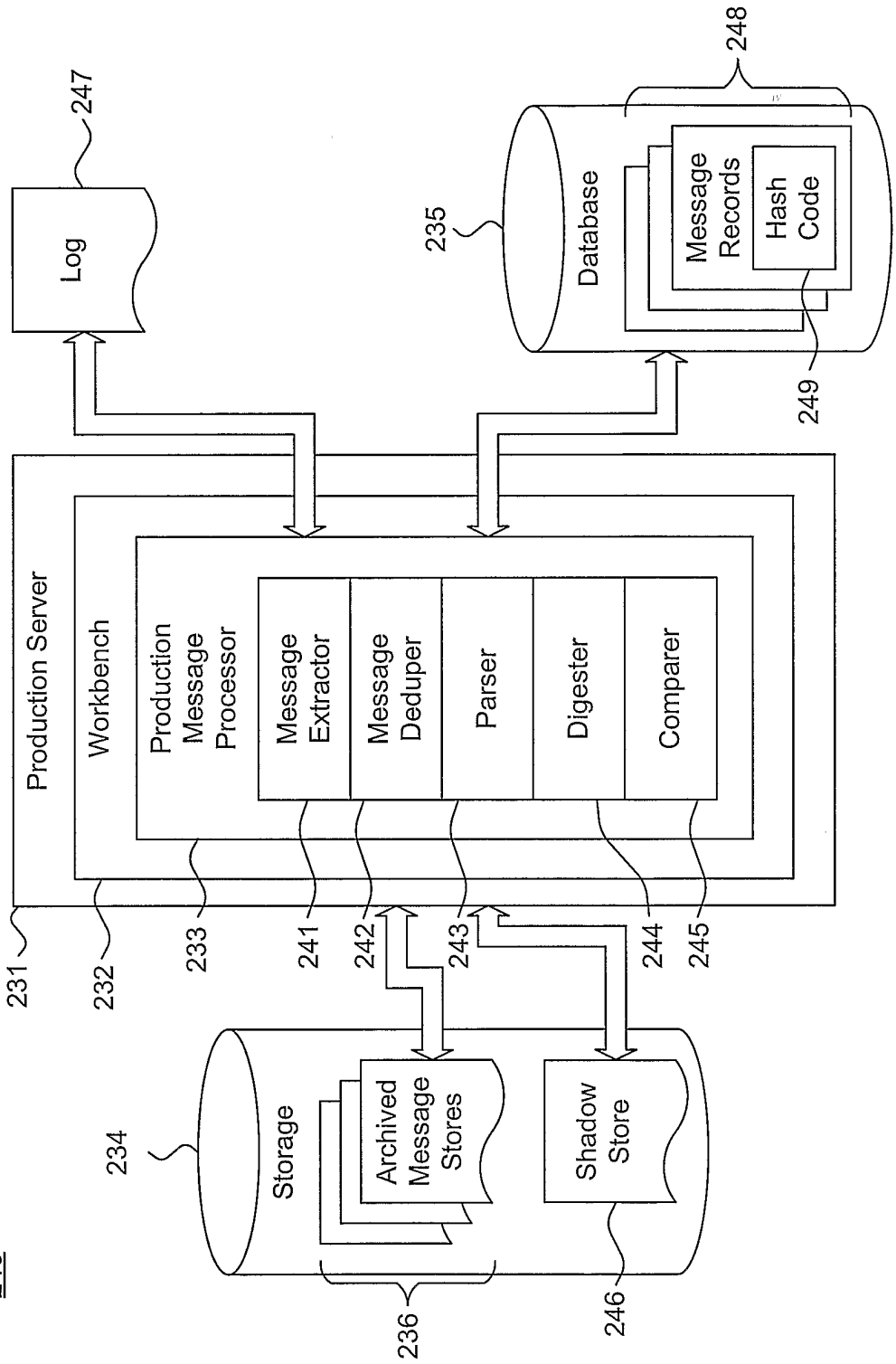
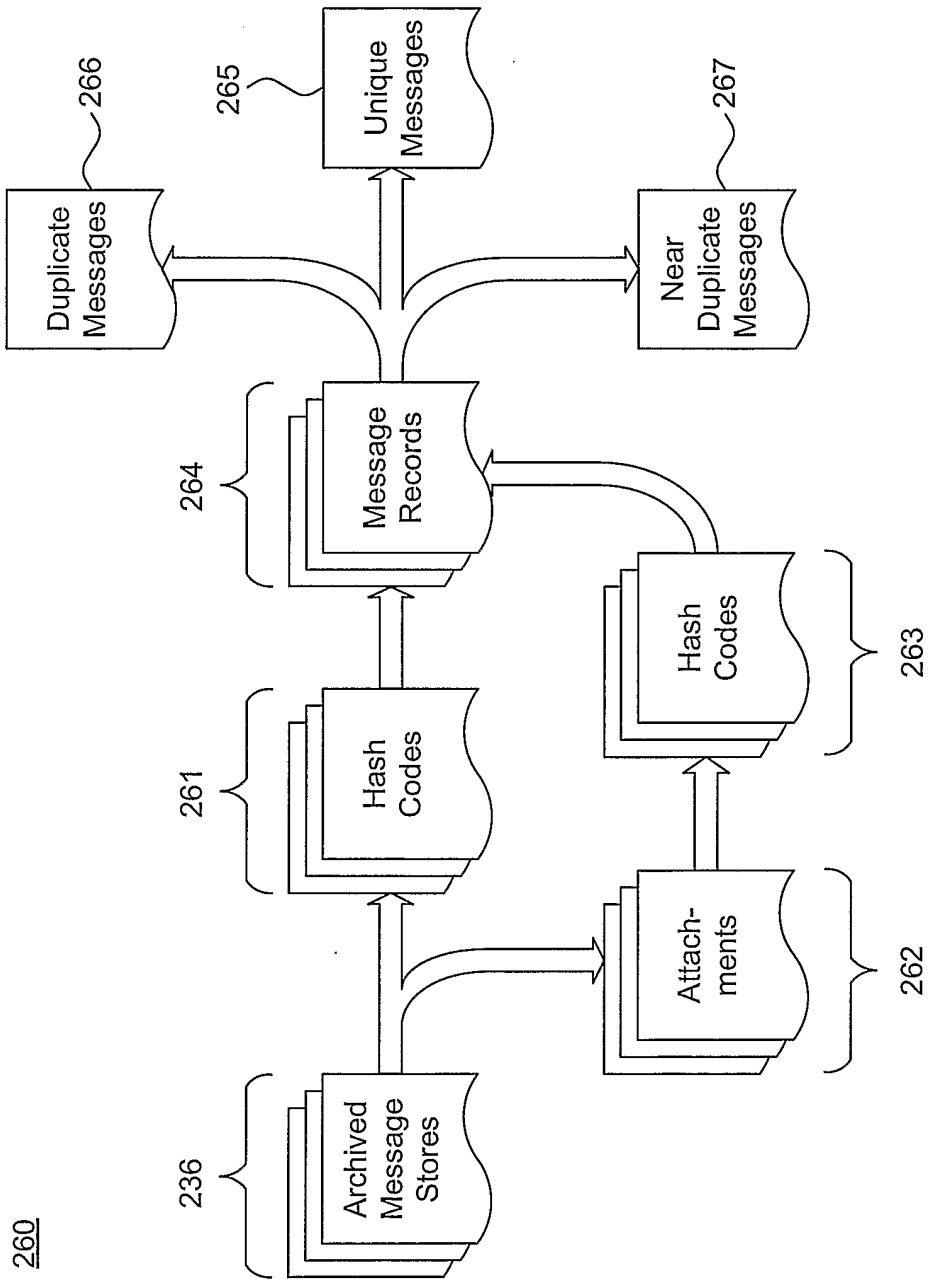

240
Production Server
Workbench
Production Message Processor
Message Extractor
Message Deduper
Parser
Digester
Comparer
Log
247
Database
235
248
Message Records
Hash Code
249
Storage
234
Archived Message Stores
Shadow Store
236
246
231
232
233
241
242
243
244
245

Fig. 14.


260
236
Archived Message Stores
261
Hash Codes
262
Attach-ments
263
Hash Codes
264
Message Records
265
Unique Messages
266
Duplicate Messages
267
Near Duplicate Messages

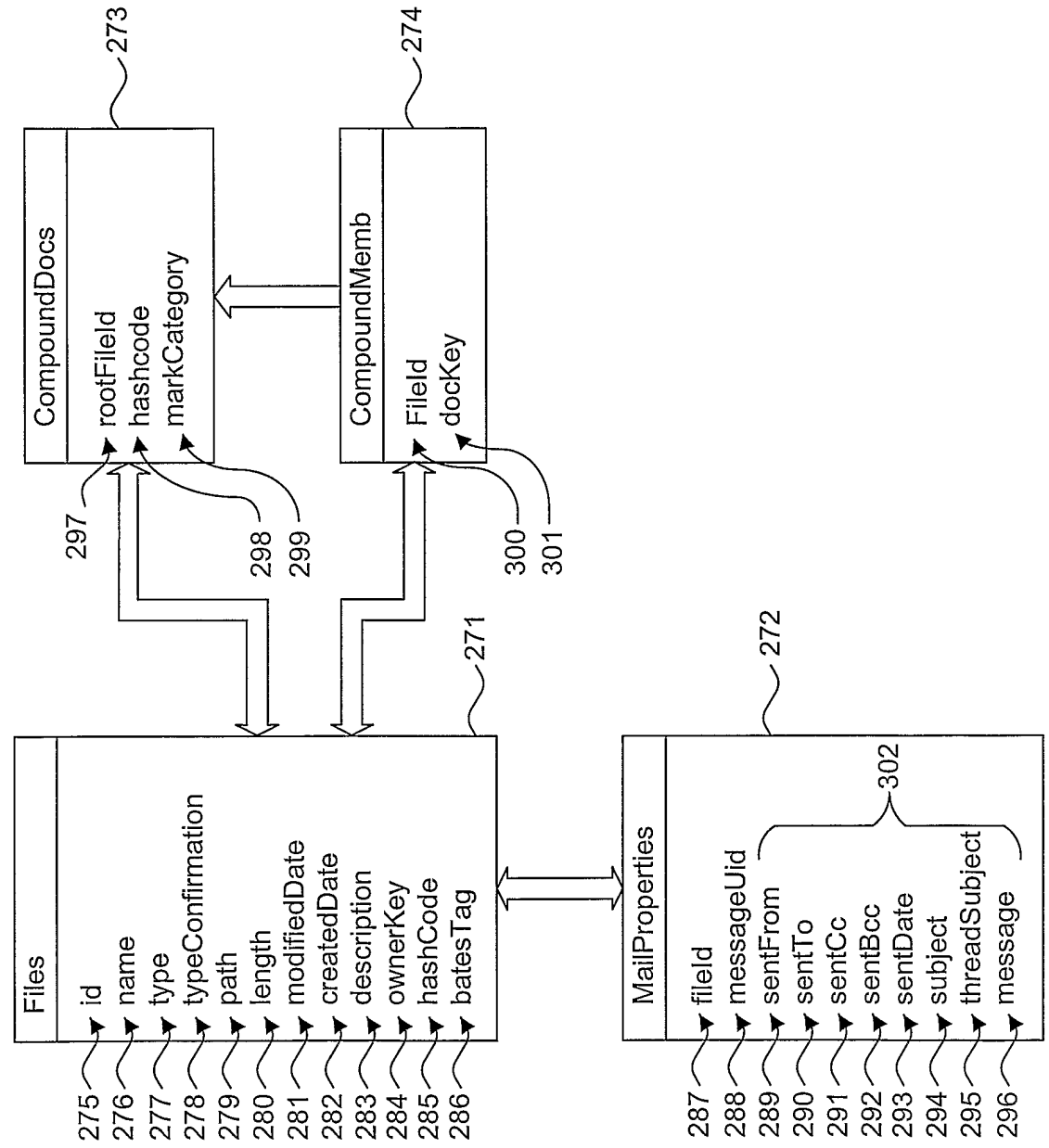

270
Files
id
name
type
typeConfirmation
path
length
modifiedDate
createdDate
description
ownerKey
hashCode
batesTag
MailProperties
fileId
messageUid
sentFrom
sentTo
sentCc
sentBcc
sentDate
subject
threadSubject
message
CompoundDocs
rootFileId
hashcode
markCategory
CompoundMemb
FileId
docKey
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302

Fig. 15.

Fig. 16.

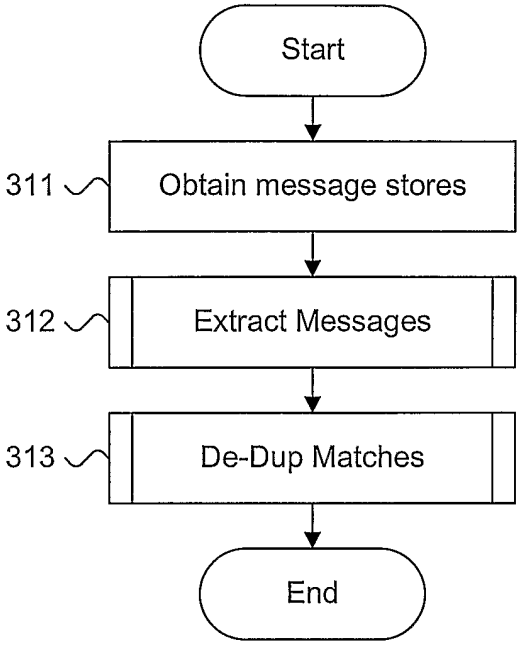


Fig. 17A.

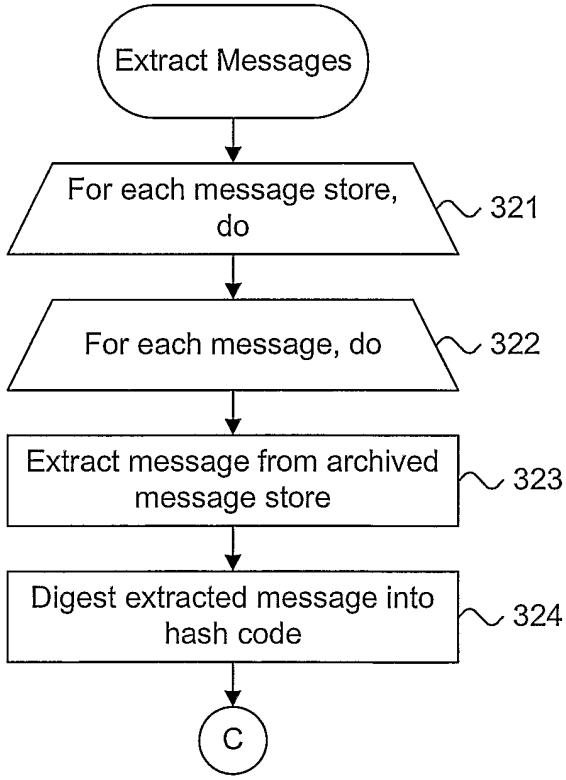


Fig. 17B.

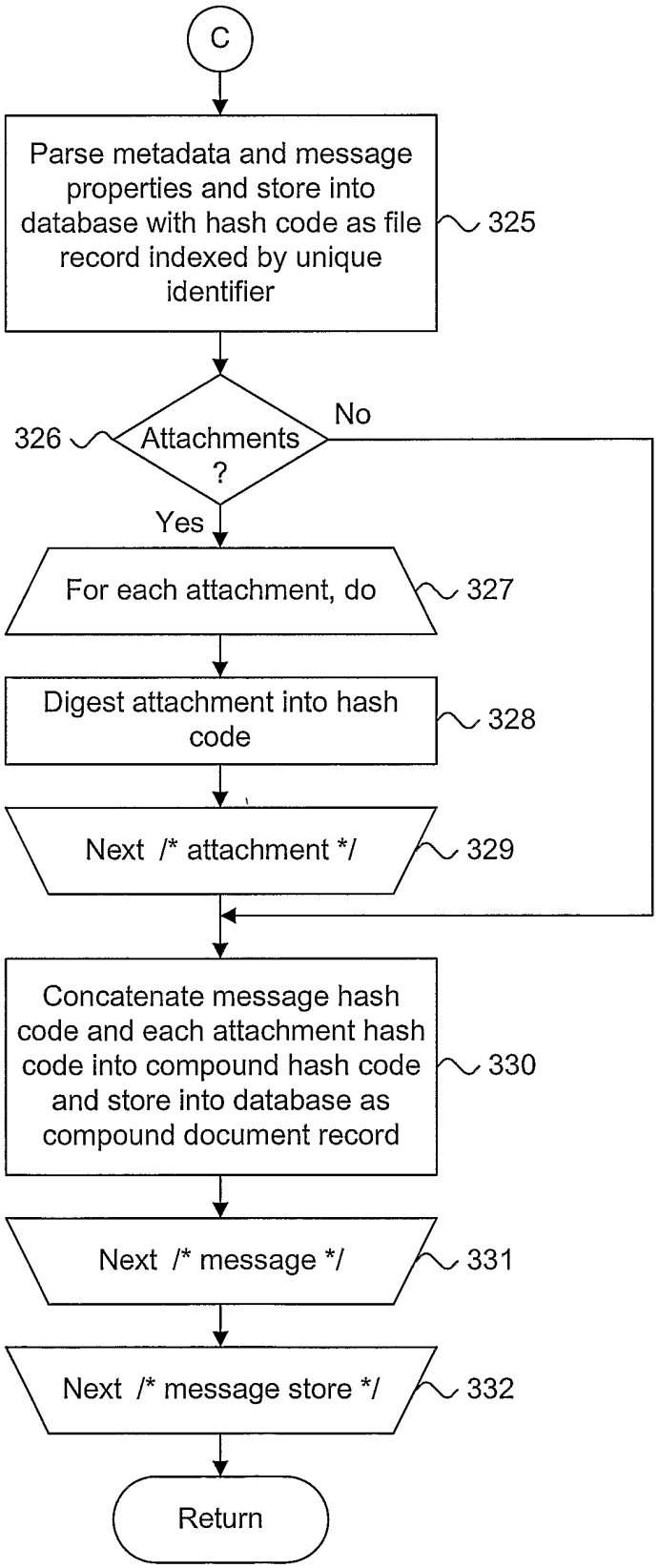

C
Parse metadata and message properties and store into database with hash code as file record indexed by unique identifier
325
326
Attachments ?
No
Yes
For each attachment, do
327
Digest attachment into hash code
328
Next /* attachment */
329
Concatenate message hash code and each attachment hash code into compound hash code and store into database as compound document record
330
Next /* message */
331
Next /* message store */
332
Return

Fig. 18A.

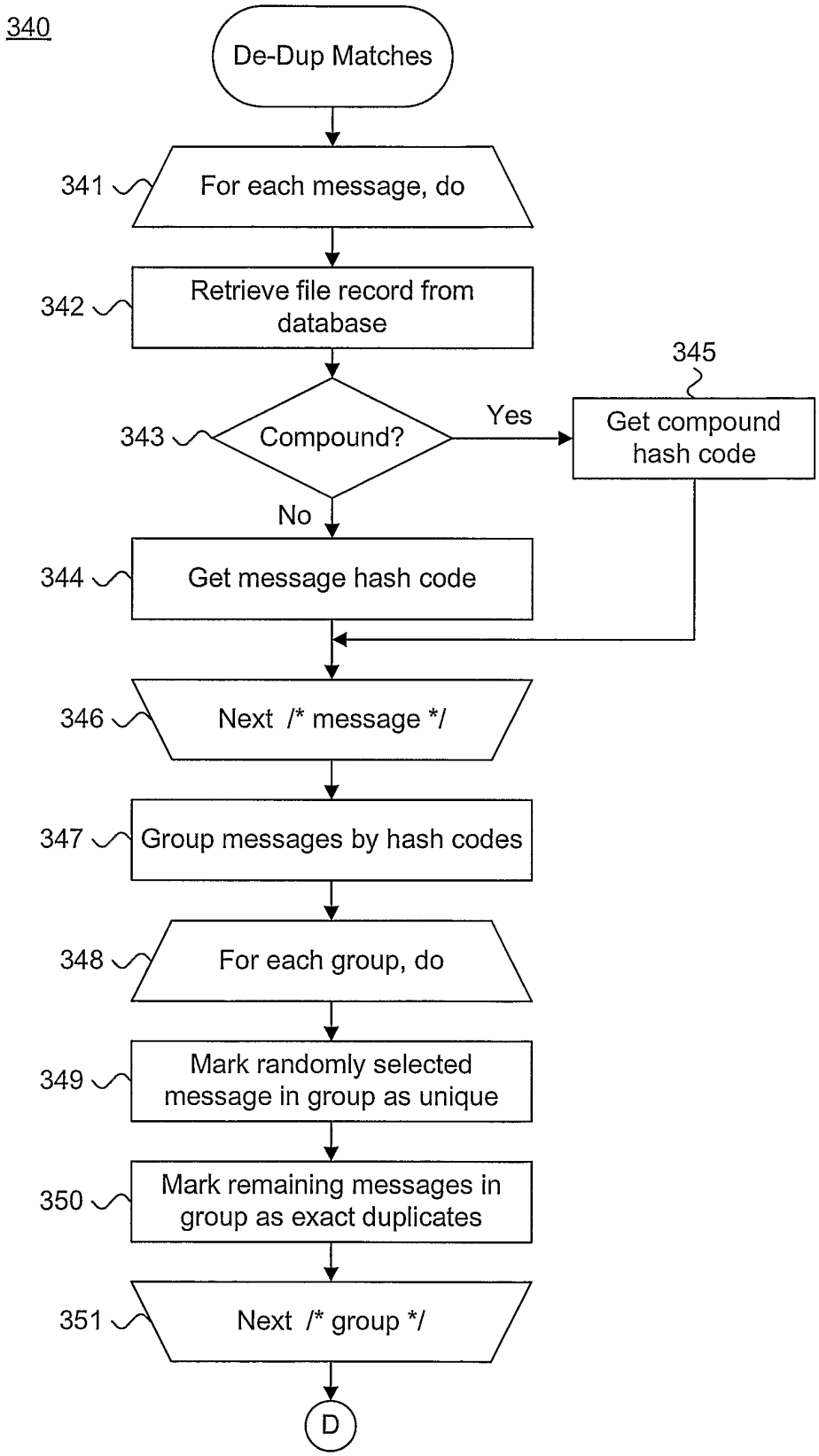

340
De-Dup Matches
341
For each message, do
342
Retrieve file record from database
343
Compound?
Yes
345
Get compound hash code
No
344
Get message hash code
346
Next /* message */
347
Group messages by hash codes
348
For each group, do
349
Mark randomly selected message in group as unique
350
Mark remaining messages in group as exact duplicates
351
Next /* group */
D

Fig. 18B.

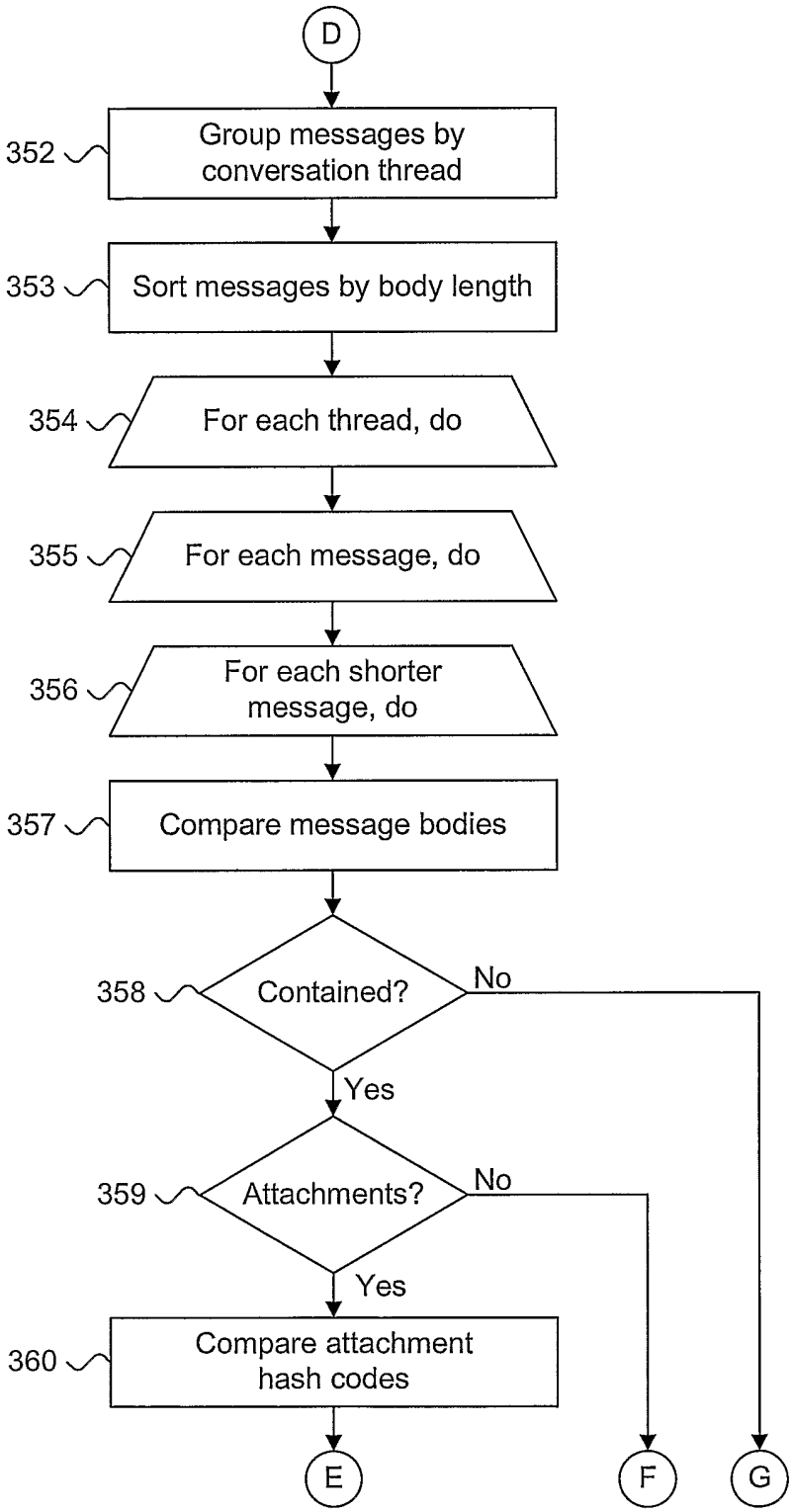
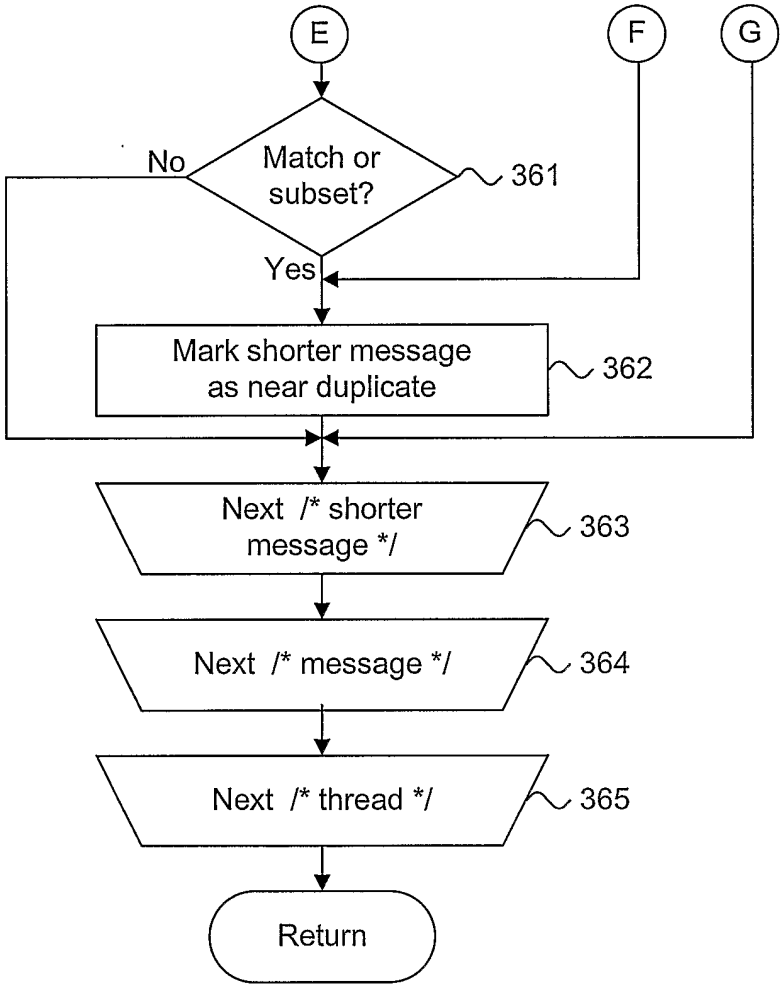

D
352
Group messages by conversation thread
353
Sort messages by body length
354
For each thread, do
355
For each message, do
356
For each shorter message, do
357
Compare message bodies
358
Contained?
No
Yes
359
Attachments?
No
Yes
360
Compare attachment hash codes
E
F
G

Fig. 18C.


E
F
G
No
Match or subset?
361
Yes
Mark shorter message as near duplicate
362
Next /* shorter message */
363
Next /* message */
364
Next /* thread */
365
Return

INTERNATIONAL SEARCH REPORT

PCT/US2004/023946

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 H04L12/58 G06F17/60

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, WPI Data, PAJ, IBM-TDB, INSPEC, COMPENDEX

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
P,Y	US 2002/161788 A1 (MCDONALD DAVID T) 31 October 2002 (2002-10-31) the whole document	1-25
P,Y	US 6 292 880 B1 (BEGUELIN ADAM ET AL) 18 September 2001 (2001-09-18) abstract column 8, line 37 - column 10, line 22	1-25
A	"DISTRIBUTION DUPLICATION PREVENTION MECHANISM" RESEARCH DISCLOSURE, KENNETH MASON PUBLICATIONS, HAMPSHIRE, GB, no. 336, 1 April 1992 (1992-04-01), page 261, XP000304807 ISSN: 0374-4353 the whole document	1,13,25

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
16 December 2004	29/12/2004

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Eraso Helguera, J

INTERNATIONAL SEARCH REPORT

PCT/US2004/023946

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	JP 2000 187668 A (HITACHI LTD) 4 July 2000 (2000-07-04) abstract	1,13,25
P,A	-& US 6 381 601 B1 (USHIJIMA KAZUTOMO ET AL) 30 April 2002 (2002-04-30) abstract	1,13,25
A	"CHAINING MAIL REPOSITORY ITEMS OVER TIME" IBM TECHNICAL DISCLOSURE BULLETIN, IBM CORP. NEW YORK, US, vol. 37, no. 2B, 1 February 1994 (1994-02-01), pages 113-114, XP000433779 ISSN: 0018-8689 the whole document	1,13,25
A	"AUTO-ELIMINATION OF DUPLICATE IN-BASKET ITEMS IN ELECTRONIC MAIL" IBM TECHNICAL DISCLOSURE BULLETIN, IBM CORP. NEW YORK, US, vol. 36, no. 3, 1 March 1993 (1993-03-01), pages 403-405, XP000354825 ISSN: 0018-8689 the whole document	1,13,25
A	US 5 905 863 A (LEWIS DAVID DOLAN ET AL) 18 May 1999 (1999-05-18) column 7, line 42 - column 12, line 13	1,13,25
A	EP 0 886 227 A (DIGITAL EQUIPMENT CORP) 23 December 1998 (1998-12-23) column 8, line 25 - column 15, line 13	1,13,25

INTERNATIONAL SEARCH REPORT

Information on patent family members

PCT/US2004/023946

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2002161788 A1	31-10-2002	CA 2441390 A1	14-11-2002
		EP 1371195 A2	17-12-2003
		WO 02091701 A2	14-11-2002
		US 2004221295 A1	04-11-2004
US 6292880 B1	18-09-2001	NONE	
JP 2000187668 A	04-07-2000	US 6381601 B1	30-04-2002
US 6381601 B1	30-04-2002	JP 2000187668 A	04-07-2000
US 5905863 A	18-05-1999	WO 9746962 A1	11-12-1997
EP 0886227 A	23-12-1998	CA 2240556 A1	16-12-1998
		DE 69818549 D1	06-11-2003
		DE 69818549 T2	05-08-2004
		EP 0886227 A1	23-12-1998
		JP 11015759 A	22-01-1999