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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
9 June 2016 (09.06.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/089787 A1

(51) International Patent Classification:
G06F 17/30 (2006.01)

(21) International Application Number:
PCT/US2015/063034

(22) International Filing Date:
30 November 2015 (30.11.2015)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/086,111 1 December 2014 (01.12.2014) US
14/954,731 30 November 2015 (30.11.2015) US

(71) Applicant: **INFORMATICAL LLC** [US/US]; 2100 Seaport Blvd., Redwood City, CA 94063-4755 (US).

(72) Inventors: **FUGITT, Jesse, A.**; C/o Informatica Llc, 2100 Seaport Blvd., Redwood City, CA 94063-4755 (US). **CANLI, Turkmen**; C/o Informatical Llc, 2100 Seaport Blvd., Redwood City, CA 94063-4755 (US). **HODA, Sahir**; C/o Informatical Llc, 2100 Seaport Blvd, Redwood City, CA 94063-4755 (US).

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

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU,

[Continued on next page]

(54) Title: MESSAGE BROKER SYSTEM WITH PARALLEL PERSISTENCE

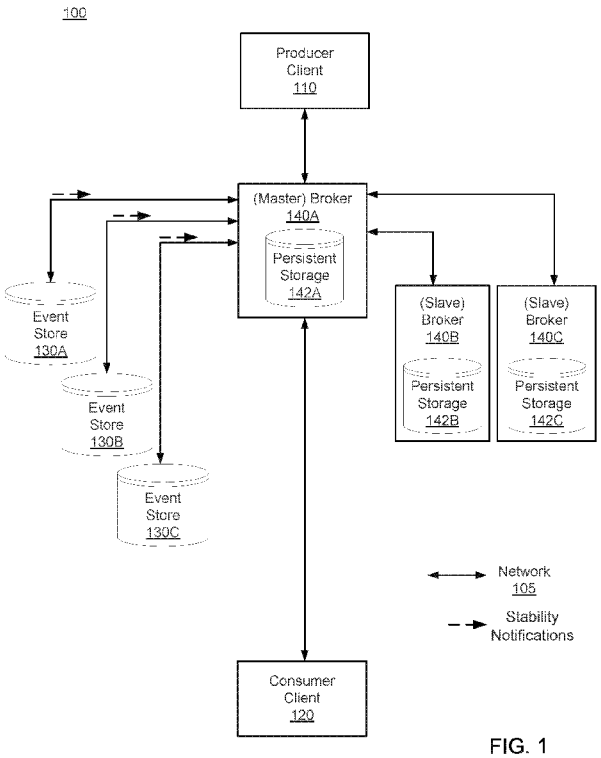

FIG. 1

(57) Abstract: A message broker computer includes a master broker, a plurality of slave message brokers and event stores. A client system sends messages for processing to the master broker. The master broker generates a message event in response to receiving such a message, and distributes the message event in parallel to the slave brokers and the event stores. Each of the event stores store the message event in persistent storage, and notifies the master broker that the message event has been persisted. The master broker considers the message stabilized n a quorum of the event stores. As the master broker does not take action until a messaging event is stabilized, in the event of failover, a new master broker is able to re-construct a broker state of the old master with no loss of data.

WO 2016/089787 A1

TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

MESSAGE BROKER SYSTEM WITH PARALLEL PERSISTENCE

CROSS REFERENCE TO RELATED APPLICATIONS

This application claims the benefit of U.S. Provisional Application No. 62/086,111, filed December 1, 2014; and U.S. Application No. 14/954,731, filed November 30, 2015; each of which are hereby incorporated by reference in their entirety.

BACKGROUND

1. TECHNICAL FIELD

[0001] The subject matter described herein relates generally to message broker computer systems, and in particular, a master/slave message broker computer system with parallel persistence.

2. BACKGROUND INFORMATION

[0002] Message speed and reliability are crucial performance requirements for messaging applications. For example, financial markets compete on speed, and high-speed trading capabilities have become a performance differentiator for the largest financial services firms and some investment funds. There is a demand for an efficient, high-throughput, low-latency and more secure method of linking their worldwide operations. Current solutions generally utilize a master/slave broker architecture that requires a shared database/file system. The shared database/file system, however, is far from ideal as it becomes a single point of failure such that if the shared database/file system fails the entire system goes down – and data (e.g., brokerage orders) may be potentially be lost.

BRIEF DESCRIPTION OF THE DRAWINGS

[0003] FIG. 1 is a high-level block diagram illustrating a networked computing environment suitable for implementing a message broker system, according to one embodiment.

[0004] FIG. 2 is a high-level block diagram illustrating a detailed view of modules within a broker according to one embodiment.

[0005] FIG. 3 is a time chart illustrating uncertainty in determining a last stable message event in case of a failed master broker, according to one embodiment.

[0006] FIG. 4A - C is an interaction diagram of a method of using a message broker system to process a message, according to one embodiment.

[0007] FIG. 5 is a flowchart illustrating a method for transitioning from a slave broker to a master broker, according to one embodiment.

[0008] FIG. 6A is a class diagram of a replication stream and replication event store components of a message broker system to process a message, according to one embodiment.

[0009] FIG. 6B is a class diagram of long term storage components of a message broker system to process a message, according to one embodiment.

[0010] FIG. 6C is an interface between an event store and a message broker of a message broker system to process a message, according to one embodiment.

[0011] FIG. 7 is a high-level block diagram of the components of a computing system suitable for use in the networked environment depicted in FIG. 1, according to one embodiment.

DETAILED DESCRIPTION

[0012] A message broker computer system configured as a master/slave message broker computer system using a shared-nothing architecture provides low latency messaging in combination with high reliability. The message broker computer system includes at least three brokers, one of which is configured as a master broker and the other brokers configured as slave brokers. The message broker system includes a plurality of event stores. Client

producer computer systems send messages for processing to the master broker. The master broker generates a message event in response to receiving such a message, and distributes the message event in parallel to the slave brokers and the event stores. Each of the event stores store the message event in persistent storage, and notifies the master broker that the message event has been persisted. Because the message event is persisted on a quorum of the event stores the master broker considers the message stabilized, the master need not obtain confirmation from the slave brokers. A quorum of event stores is at least two event stores. In some embodiments, a quorum may be the majority of the event stores. After a slave broker persists a messaging event it separately notifies the event stores. As the master broker does not take action until a messaging event is stabilized, in the event of failover, a new master broker is able to re-construct a broker state of the old master with no loss of data.

[0013] In the event of failover (e.g., the master broker terminates operation or is taken down for service), the slave brokers are configured to elect a new master broker from the pool of slave brokers. The newly elected master broker is able to recover any stabilized message event from the event stores. The newly elected master broker retrieves message events from the event stores that it has no record of, and identifies a maximum contiguous message event (MCM) using the message event stream. Based on the MCM, the new master broker identifies a synchronization point using the MCM, the synchronization point being a sequence number pointed to by a back pointer in metadata associated with the MCM. The new master broker then republishes any message events with sequence numbers above that of the synchronization point, and updates the broker state in the message broker to correspond to the MCM. As different slave brokers receive message events at different times, slave brokers may have different broker states (ideally they would be the same – and reflect the broker state of the master broker). The republished message events include metadata that indicates there is a new master broker. The slave brokers are configured to drop message events from the old master broker with synchronization numbers later than the synchronization point. Accordingly, the new master broker is able to completely recover a broker state of the old master broker prior to failure in a manner that results in no loss of stabilized data.

[0014] The Figures (FIGS.) and the following description describe certain embodiments in which the message broker system is applied to a computer network by way of illustration only. One of skill in the art will recognize from the following description that the message

broker system can be applied to other computer based representations of inter-related elements, and that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles described herein. Reference will now be made to several embodiments, examples of which are illustrated in the accompanying figures. It is noted that wherever practicable similar or like reference numbers may be used in the figures and may indicate similar or like functionality.

[0015] FIG. 1 is a block diagram illustrating an embodiment of an environment for facilitating a message brokering computer system 100. The environment includes a producer client 110, a consumer client 120, event stores 130A, 130B, and 130C, and slave brokers 140B, and 140C, each coupled by a network 110 to a master broker 140A. Here, only one producer client 110 and consumer client 120, are illustrated, but in practice there will numerous instances of each of these entities, typically thousands or more, requiring a high speed and reliable mechanism for delivery of messages between them. Moreover, while only two slave brokers and three event stores are illustrated, some embodiments include additional slave brokers and/or event stores. As appreciated and understood by those of skill in the art, the system and methods and methods described herein are inherently and necessarily performed using a physical, tangible computer system operating on electrical coded signals; in no event can the methods and operations described herein be performed by a human being (or group of human beings) or by mental steps. Thus, the terms "master broker," "slave broker," "message broker," "clients," and "message store" are to be understood as referring to physical computer systems, not human beings or mental processes.

[0016] The network 105 provides a communication infrastructure between the producer client 110, the consumer client 120, the event stores 130A, 130B, and 130C, the slave brokers 140B and 140C, the master broker 140A. The network 105 is typically the Internet, but may be any network, including but not limited to a Local Area Network (LAN), a Metropolitan Area Network (MAN), a Wide Area Network (WAN), a mobile wired or wireless network, a private network, or a virtual private network.

[0017] The producer clients 110 are computing devices that execute computer program modules—e.g., client applications—which communicate messages to the master broker 140A. A message is digital data. Messages may be, e.g., brokerage orders, bank

transactions, temperature readings, some other data that may be replicated and persisted by the message brokering system 100, or some combination thereof. The Producer clients 110 may be, e.g., a custom application running on a desktop computer coupled to the master broker 140A that is running on an enterprise grade server. In other embodiments, a producer client 110 is a webserver of an enterprise that generates messages in response to instructions received from users accessing the webserver using a browser application on a personal computer, smart phone, etc. In some embodiments the producer clients 110 may be an ultra-messaging queuing application and/or a JAVA messaging service application. The producer clients 110 provide the message (e.g., a sell order for stock) to the master broker 140A. In some embodiments, the producer clients 110 may communicate with the master broker 140A using Advanced Message Queuing Protocol (AMQP) and/or OPENWIRE protocol available from The Apache Software Foundation™.

[0018] The consumer clients 120 are computing devices that execute computer program modules—e.g., client applications—which communicate messages to the master broker 140A. In some embodiments, a consumer client 120 is a backend server in an enterprise that is processing the messages that it receives from the master broker 140A. In some embodiments, a consumer client 120 may be associated with a regulator or a computing device that sends the message to the regulator. In some embodiments the consumer clients 120 may be an ultra-messaging receiver application and/or a JAVA consumer application. In some embodiments, the consumer clients 120 may communicate with the master broker 140A using AMQP and/or OPENWIRE protocol. As configured for interoperation with a master broker computer system, for example with the AMQP or OPENWIRE protocols, the operations and functions of the clients 110, 120 extend beyond the basic, generic functions of a generic computer system,

[0019] The event stores 130A, 130B, 130C are data stores that each comprise one or more non-transitory computer-readable media and are configured to store message events received from the master broker 140A. Once an event store 130A, 130B, 130C has stored (also referred to as persisted) a message event, the event store 130A, 130B, 130C notifies the master broker 140A that the message event was successfully persisted using a stability notification.

[0020] In embodiments, where a broker is transitioning from slave broker to a master broker, the event stores 130A, 130B, 130C also provide synchronization information to newly elected master broker. The synchronization information includes sequence information for the message, including the highest sequence numbers of message events which has been persisted on each of the plurality of event stores 130A, 130B, 130C. The event stores 130A, 130B, 130C also may provide one or more stored message events to the newly elected master broker.

[0021] The brokers 140A, 140B, 140C, include one master broker and the remaining brokers are slave brokers. In this example, broker 140A is the master broker, and brokers 140B and 140C are slave brokers. However, any broker has the capability of acting as a master broker or a slave broker. Moreover, in the event that the master broker 140A fails, one of the remaining slave brokers 140B, 140C is selected by the slave brokers as the new master broker. The master election process is further described in detail with respect to FIG. 4 below. As configured by the operations and functions described herein, the brokers 140 provide features and functions that extend beyond the basic generic functions of a generic computer system. For example, conventional generic computer systems do not provide as part of their instruction sets, operating system software, utility software, or other applications, functions for the election of a new master broker from plural slave brokers in the event of a failover of an existing master broker.

[0022] Broker 140A, 140B, and 140C each include a persistent storage 142A, 142B, and 142C, respectively. The persistent storages 142A, 142 B, 142C comprises one or more non-transitory computer-readable media and is configured to store message events. The data stored by the persistent storage 142A, 142 B, 142C includes information describing a broker state of the associated broker 140. For example, the persistent storage 142 would include information describing the broker state of the master broker 140A, the persistent storage 142B would include information describing the broker state of slave broker 142B, and so on. The broker state is a representation of the message event stream that is has been persisted at that broker – and in effect describes the net state for each message event.

[0023] As indicated above, the master broker 140A generates message events. A message event may be one of many different types (e.g., enqueue, assign, consume,

transaction events, administration events such as create or delete a destination queue, etc.). The master broker 140A may generate a message event in response to receiving a message from a client producer 110, a consumer producer 120, stabilization notifications from at least two of the event stores 130A, 130B, 130C for a particular message event, etc. The master broker 140A distributes generated message events to the slave brokers 140B, 140C and the event stores 130A, 130B, 130C.

[0024] A message event (may also be referred to as a replication event) is the message received from the producer client 110 along with additional metadata which is added by the master broker 140A. The metadata includes the following data fields: last stable sequence number, last stable epoch, and sequence number. The epoch and sequence number fields represent the unique message event identifier. The epoch is incremented each time a new master broker is elected and the sequence number is incremented each time a new message event is published by that master (and resets to 0 when changing epochs). The "last stable" field indicates the last stable message event known to the master broker at the time it was sending the "new" message event. For example, assume that message event at (epoch 0 sequence, number 17) is an enqueue message event being published by the master broker 140A to indicate a producer client 110 is attempting to add a persistent message to a queue in the broker. Immediately after being distributing this message event to event stores 130A, 130B, 130C, and the slave brokers 140B, 140C, the master broker 140A does not have any information indicating if this message event is stable anywhere. However, once event stores 130A, 130B, 130C send stability notifications back to the master broker 140A, the broker 140A stores information indicating this the message event at (epoch 0, sequence number 17) is stable (on disk) at the one or more of event stores 130A, 130B, 130C. When it publishes its next message event (epoch 0 sequence number 18), the master broker 140A will set the last stable epoch and last stable sequence number to 0 and 17, respectively in this next message event, to indicate that the previous events up to (and including) epoch 0 sequence number 17 are now stable.

[0025] The master broker 140A is the only entity in the messaging system that is configured to process the metadata of the message events and selectively act in response to such metadata. This simplifies and reduces that number of states that can occur in messaging

systems where each entity is configured with processing logic to selectively act upon the message metadata. In one embodiment, every determination by the master broker 140A is made by waiting for at least two different event stores 130A, 130B or 130C to indicate that an event has been stabilized. Once at least two different message stores 130A, 130B or 130C have indicated that the message event is stabilized, the master broker 140A is configured to take external action on the message event and update its broker state in persistent storage 142A. In addition, the master broker 140A distributes to the slave brokers 140B, 140C, and the event stores 130A, 130B, 130C information describing the last messaging event that it has determined to be stabilized by periodically sending out a stability event that identifies the sequence number and the epoch number of the last messaging event which has been stabilized. In states of constant activity, the master broker 140A does not need to send out the periodic stability event because it will include this information as part of each outgoing message event. Accordingly slave brokers 140B, 140C can safely apply message events being held in memory to their disk persistence layer (i.e., persistent storage), and slave brokers may utilize this information to identify a clean cutover point when becoming a master during the failover process.

[0026] The master broker 140A and the slave brokers 140B, 140C, each maintain a broker state. As discussed above, the broker state is a representation of the message event stream that is has been persisted at that broker – and in effect describes the net state for each message event. For example for a particular message, an enqueue message event, followed by an assign message event, followed by a consume message event might result in a broker state that was totally empty for that message. In contrast, for a second message, only an enqueue message event may have been generated such that the broker state shows an enqueue event being the current state associated with the second message.

[0027] FIG. 2 is a high-level block diagram illustrating a detailed view of modules within brokers 140A, 140B, and 140C (referred to in this section as broker 140) depicted in FIG. 1. Some embodiments of the broker 140 have different modules than those described here. Similarly, the functions can be distributed among the modules in a different manner than is described here. The broker 140 is comprised of modules including persistent storage 142, persistence module 220, master-election module 230, and recovery module 240.

[0028] The persistent storage 142 comprises one or more non-transitory computer-readable media and is configured to store data used by the broker 140 in persisting message events. The data stored by the persistent storage 142 includes information describing the broker state. Additional functions of the persistent storage 142 include storing messaging events based on a broker state, periodically rolling up a stream of messaging events into a state that represents the snapshot at that time and then storing the snapshot to the persistent storage, retrieving a snapshot of a messaging event before applying a new messaging event, applying replication events from a snapshot or the events stored in memory and other such functions.

[0029] The persistence module 220 controls operation of the broker 140 during operation either as a master broker 140A or a slave broker 140B or 140C. In one embodiment, the persistence module 220 can perform functions for a master broker. The functions for a master broker include generating transaction events such as add, assign or consume a message event or adding or deleting a destination event. As the message events are generated, the master broker persistence module 220 sends a replication event message to the slave brokers and the event stores in parallel. Once the replication event messages are sent out, the persistence module 220 waits for a stability notification from the event stores. A stability notification is an assurance from a slave broker or an event store that the message event has been persisted in their storage. On receiving stability notifications from a quorum, i.e. a predefined number of event stores (e.g. 2 event stores), the persistence module 220 determines that the message event has been persisted and communicates the same to a producer client 110 of the message event. The number that makes the quorum is configurable, i.e. it can be set to 3, 4, or any other number of event stores.

[0030] For every message event that the persistence module 220 of a master broker generates, it applies information associated with the previously stabilized message event to the metadata of the message event. Additionally, the persistence module 220 of a master broker periodically generates stability message events that include information associated with the last stable message event.

[0031] The persistence module 220 further sends the message event to the consumer client 120 and waits for an acknowledgement from the consumer client 120 that it has consumed the message. The persistence module 220 then generates remove message event

that it distributes to the slave brokers and the event stores. On receiving the stability notifications for the remove message events, the persistence module 220 of the master broker removes the message event from the internal memory of the master broker 140A.

[0032] In one embodiment, the persistence module 220 can perform functions for the slave broker. On receiving the message event from a master broker, the persistence module 220 of the slave broker 140B or 140C holds it in the internal memory. The received message event includes metadata that contains information associated with the last stable event known to the master broker. The persistence module 220 extracts the information included in the metadata to retrieve a back pointer that points to the last stable message event number. The persistence module 220 of the slave broker applies the back pointer along with the current message event, also known as a snapshot of the current message event stream, to the persistent storage 142.

[0033] In case a slave broker is broken or has to be restarted, the slave broker can be ahead or behind the other brokers and thus have an inconsistent state. The persistence module 220 of the slave broker helps in achieving consistency of the broker state when the slave broker comes back up again via the replication stream protocol. For example, in case a slave broker received only a part of a replication message event before any other broker or event store, the message event is not applied before receiving the additional information related to the snapshot of the message from the master broker. If the slave broker goes down for a small period of time, the message event stream at the slave broker is almost in sync with the replication message event stream distributed by the master broker. The persistence module 220 can recover the lost message events from one of the event stores, leading to the slave broker being consistent with the ongoing replication message event stream. In case the slave broker is down for a long period of time, the slave broker falls far behind the current replication message event stream, a recovery module 240 is used to recover the state of the slave by a slave onboarding process.

[0034] The recovery module 240 can recover the broker state of the prior master broker in the event of failover of the master broker or recover the broker state of a slave broker in the event of a failover or restart of a slave broker. In a failover event of a master broker, a master election module 230, described in detail below, elects one of the slave brokers as the new master. Once a new master broker is elected, the recovery module 240 assists in

transitioning the slave broker state to the prior master's broker state. The recovery module 240 determines a replication stream cutover point, i.e. a sequence number from a replication stream that defines the last event that the master broker marked as stable, i.e. the master broker received a quorum of stability notifications for the determined message event in the replication stream.

[0035] Since the master broker has failed, the master broker either needs to be restarted or repaired. This makes retrieving the last stable message event from the master broker not convenient and time consuming. Based on the replication stream protocol, the master broker generates information associated with the last stable message event with each message event. The message events may be in flight when the master broker fails. Based on the message events in flight, an embodiment is described below to determine the last stable message event.

[0036] FIG. 3 is a time chart illustrating uncertainty in determining a last stable message event in case of a failed master broker, according to one embodiment. In the case 1 of the example embodiment of FIG. 3, the event store UMP_3 is the only event store that has all the messages in flight. If the master broker and the event store UMP_3 fail at the same time, there is no way to determine that m_s is the last stable message 310. In the case above, the last good point 315 is m_x.

[0037] In the case 2 of the example embodiment, the event store UMP_2 has some messages that are in flight, the last message it has, message m_k, points back to last stable message 310 $\mathbf{m_s}$. The event store UMP_3 has all the messages in flight. If the master broker and the event store UMP_3 fail at the same time, message m_k is the latest available message amongst the event stores UMP daemons, and it points back to the last stable message m_s. The statuses of the messages $[m_{s+1}, m_k]$ will not be known. The last good point is m_s.

[0038] From the above example embodiments, it is noted that there is uncertainty in determining the last stable message. The message events after the known good point may or may not be stabilized. To overcome the uncertainty of the status of the message events, if there are gaps, i.e. none of the event stores have a particular message event, the range of message events that may or may not be stabilized after the known good point, is from the next message event of the known good point up to the last message before the gap.

[0039] Thus, for recovering the message event stream, a maximum contiguous message (MCM) is observed, i.e. a highest sequence numbered message that is observed before encountering a sequence number gap. A synchronization point is further determined as the back pointer of the MCM. The synchronization point is the stream cutover point. The new master will republish all the message events from the synchronization point up to the MCM, with a new epoch number determined for the elected master broker. Further, the new master will update its snapshot on the persistence storage up to the MCM, and reconstruct its broker memory state to be consistent with the MCM. Once the broker state is recovered, the master broker can start generating message events and resume role of a master.

[0040] In another embodiment, the recovery module 240 recovers the broker state of a slave broker in the event of a failover or restart of a slave broker. A broken slave broker is treated like a new slave broker that has no persistence or broker state. A disk snapshot from an existing slave broker that is not broken is copied over to the new or broken slave broker. The new broker that has no state will receive the live stream and will temporarily suspend the periodic updates to its disk snapshot. The existing broker will continue receiving the replication stream and will also temporarily suspend the periodic updates to its disk snapshot. The existing broker will then attempt to send the entire disk snapshot to the new broker. If the transmission succeeds, both slaves will have the same disk snapshot and both will begin applying their "buffered" events from memory to the disk snapshot and will eventually both be caught up to the live stream.

[0041] The master election module 230 determines that failover occurs and elects one of the slave brokers 140B, 140C as the new master broker. In some embodiments, the master broker sends a heartbeat signal to the master election module 230 at periodic intervals, for example, a heartbeat is sent every microsecond. The periodic interval is predefined and is configurable. The master election module 230 detects a failure of a master broker via lack of the heartbeat signal for a predefined timeout interval, for example, if no signal is received for a second.

[0042] The master election module 230 further keeps track a list of eligible slave brokers that can be elected as master brokers. On detection of a broken master, the master election module 230 elects a new master from the tracked list of eligible brokers. In some embodiments, the master election module 230 utilizes the RAFT protocol to elect the new

master from the tracked list. In the time that the master election module 230 elects a new master, a temporary broker state is created by the recovery module 240 that can help the slave brokers to recover the failed master broker's message events, and recover the broker state of the failed master broker to apply it to the newly elected master broker.

[0043] FIG. 4A - C is an interaction diagram of a method of using a message broker system to process a message, according to one embodiment. In various embodiments, the method may include different and/or additional steps than those described in conjunction with FIG. 4A - C. Additionally, in some embodiments, the steps may be performed in a different order than described in conjunction with FIG. 4A - C.

[0044] The producer client 110 sends a message 402 to the master broker 140A. The master broker 140A generates an enqueue event based on the received message. The generated enqueue event includes metadata including a sequence number of the last stable message event, an epoch number associated with the master broker 140A who sent the last stable message event, a sequence number for the enqueue event (e.g., sequence number 1), and an epoch number associated with the master broker 140A. . The master broker 140A distributes 404 the enqueue event in parallel to the slave brokers 140B, 140C, and the messaging sources 130A, 130B, and 130C. In some embodiments, the distribution of the enqueue event may be done in a serial manner, or some combination of parallel and serial distribution.

[0045] The master broker 140A holds 406 the enqueue event in memory. Similarly, once the slave brokers 140B, 140C receive the distributed enqueue event, they also hold 408, 410, the enqueue event in memory.

[0046] The event stores 130A, 130B, 130C receive the enqueue event, and store 412, 414, 416 the enqueue event in their respective permanent memories. Once an event store stores the enqueue event in permanent memory it sends notification 418 to the master broker 140A that the enqueue event has been persisted. Thus the master broker 140A may receive none, one, some, or all of the notifications 418, depending on whether the message stores 130A, 130B or 130C were able to successfully persist the enqueue event. Once the master broker 140A has received at least two notifications 418 the master broker 140A determines that the enqueue event is stabilized and so it stores 420 the enqueue event in its long term

storage (i.e., persistent storage 142). Responsive to determining that the enqueue event is stabilized the master broker 140A sends a notification to the producer client 110 that the message is persisted 422.

[0047] The master broker 140A generates an assign event. The generated assign event includes metadata including the sequence number of the last stable message event (the enqueue event), the epoch number associated with the master broker who sent the last stable message event (master broker 140A), a sequence number for the assign event (e.g., sequence number 2), and the epoch number associated with the master broker 140A. The master broker 140A distributes 424 the assign event in parallel to the slave brokers 140B, 140C, and the messaging sources 130A, 130B, and 130C.

[0048] The master broker 140A holds 426 the assign event in memory. Once the slave brokers 140B, 140C receive the distributed assign event they update 428, 430their respective broker states. Specifically, each slave broker 140B, 140C persists (e.g., move to persistent storage 142) the enqueue event that was being held in their respective memories, and then holds the assign event in their respective memories. Once a slave broker 140B, 140C persists the enqueue event, the slave broker 140B, 140C sends an acknowledgement 432 of the persisted event to all of the event stores 130A, 130B, 130C. The event stores 130A, 130B, 130C store the acknowledgments 432.

[0049] The event stores 130A, 130B, 130C receive the assign event, and store 434, 436, 438 the assign event in their respective permanent memories. Once an event store stores the assign event in permanent memory it sends notification 440 to the master broker 140A that the assign event has been persisted. Once the master broker 140A has received at least two notifications 440 the master broker 140A determines that the assign event is stabilized and so stores 442 assign event the assign event in its long term storage. In alternate embodiments, the master broker 140A may simply overwrite 442 the enqueue event in long term storage with the assign event.

[0050] Responsive to determining that the assign event is stabilized the master broker 140A sends 444 the message to a consumer client 120 (e.g., a backend server in an enterprise that is processing the orders that it receives from the master broker 140A). The consumer client 120 sends an acknowledgement 446 that it has received the message.

[0051] The master broker 140A generates a consume event. The generated consume event includes metadata including the sequence number (e.g., sequence number 2) of the last stable message event (the assign event), the epoch number associated with the master broker who sent the last stable message event (master broker 140A), a sequence number for the assign event (e.g., sequence number 3), and the epoch number associated with the master broker 140A. The master broker 140A distributes 448 the consume event in parallel to the slave brokers 140B, 140C, and the messaging sources 130A, 130B, and 130C.

[0052] The master broker 140A holds 450 the consume event in memory. Once the slave brokers 140B, 140C receive the distributed consume event they update 452, 454 their respective broker states. Specifically, each slave broker 140B, 140C persists the assign event that was being held in their respective memories, and then holds the consume event in their respective memories. In alternate embodiments, the slave brokers 140B, 140C may simply overwrite the enqueue event in long term storage with the assign event. Once a slave broker 140B, 140C persists the assign event, the slave broker 140B, 140C sends an acknowledgement 456 of the persisted event to all of the event stores 130A, 130B, 130C. The event stores 130A, 130B, 130C store the acknowledgments 356.

[0053] The event stores 130A, 130B, 130C receive the consume event, and store 458, 460, 462 the consume event in their respective permanent memories. Once an event store stores the consume event in permanent memory it sends notification 464 to the master broker 140A that the consume event has been persisted. Once the master broker 140A has received at least two notifications 464 the master broker 140A considers the consume event to be stabilized and stores 466 the consume event in its long term storage. In alternate embodiments, the master broker 140A may simply delete the consume event from memory and remove 466 the enqueue event and/or assign event from long term storage.

[0054] Note that at this point in the processing flow, the master has removed the stabilized consume message event, but the slave brokers 140B, 140C still are holding the consume message event in memory. In periods of high activity, the next message event generated by the master broker 140A and distributed to the slave brokers 140B, 140C would cause them to update their respective broker states. However, if there is a delay between message events, the master broker 140A may generate and distribute a stability event 468 to the slave brokers 140B, 140C, and the event stores 130A, 130B, 130C.

[0055] The stability message includes metadata including the sequence number (e.g., sequence number 3) of the last stable message event (the consume event), the epoch number associated with the master broker who sent the last stable message event (master broker 140A), a sequence number for the stability event (e.g., sequence number 4), and the epoch number associated with the master broker 140A. The master broker 140A distributes 468 the stability event in parallel to the slave brokers 140B, 140C, and the messaging sources 130A, 130B, and 130C.

[0056] Once the slave brokers 140B, 140C receive the distributed stability event they update their respective broker states. Specifically, each slave broker 140B, 140C persists the consume event that was being held in their respective memories. In alternate embodiments, the slave brokers 140B, 140C may simply remove the consume event from memory and remove the enqueue event and assign event from long term storage. The slave brokers 140B, 140C may also simply overwrite the enqueue event and/or assign event with the consume event. Once a slave broker 140B, 140C, persists 470, 472 the consume event, the slave broker 140B, 140C sends an acknowledgement 474 of the persisted event to all of the event stores 130A, 130B, 130C. The event stores 130A, 130B, 130C store the acknowledgments 474.

[0057] FIG. 5 is a flowchart illustrating a method for transitioning from a slave broker to a master broker, according to one embodiment. In one embodiment, the process of FIG. 5 is performed by the broker 140. Other entities may perform some or all of the steps of the process in other embodiments. Likewise, embodiments may include different and/or additional steps, or perform the steps in different orders.

[0058] The broker 140 receives 505 a notification of a change in broker status from slave broker to master broker. For example, the previous master broker 140A may have failed, and after a specific time has passed since the last communication the slave brokers 140B, 140C elect using their respective master election modules 240 one of the slave brokers (e.g., 140B) to be the new master broker – referred to in this section as the broker 140. The epoch number is incremented to indicate that a new broker 140 epoch has been initiated.

[0059] The broker 140 identifies 510 a base value, the base value being a highest sequence number of a message event in persistent storage on the messaging broker. For

example, sequence information may indicate that {509} is the highest value of a sequence number of a message that is in the persistent storage 142 of the broker 140.

[0060] The broker 140 retrieves 515 sequence information from a plurality of event stores 130A, 130B, 130C. Sequence information describes the highest sequence numbers of message events which has been persisted on each of the plurality of event stores 130A, 130B, 130C. For example, the highest sequence numbers may be values of {510, 512, 514} for event stores 130A, 130B, and 130C, respectively.

[0061] The broker 140 determines 520 a set of message events to retrieve based on the base value and the sequence information. Messages with sequence numbers below the base value are already tracked by the broker 140. If the base value equals the highest sequence number described in the sequence information, then the broker 140 is up to date. However, it is typically the case that before the old master broker 140A went offline there were some messages sent out which had yet to be stabilized. Accordingly, the broker state of the broker 140 is likely slightly behind the broker state of the old master broker 140A. By using the sequence information from the various message stores, the new elected broker 140 is able to recover the state of the older master broker 140A. Continuing the example above, the sequence information includes values of {510, 512, 514} for event stores 130A, 130B, and 130C, respectively, and the base value is {509}. Accordingly, the broker determines that it should retrieve message events {510-514} from one or more of the event stores 130A, 130B, and 130C, since this set of messages covers the entire set of messages from the base value message at sequence number {509} to the highest stabilized message at sequence number {514}. For example, Table 1 illustrates which message events that are stored in each of the respective event stores 130A, 130B, and 130C. The cell values in Table 1 indicate a sequence number and a back pointer associated with a message event, the back pointer being to the last stabilized message event. For example, "509(507)" represents a sequence number {507} and a back pointer to a sequence number {507} which is associated with the last stabilized message event when event stores 130A, 130B, and 130C stored the message event associated with sequence number 509.

Event store	Sequence Number (Back pointer)	Sequence Number (Back pointer)	Sequence Number (Back pointer)	Sequence Number (Back pointer)	Sequence Number (Back pointer)	Sequence Number (Back pointer)
130A	509 (507)	510 (508)				
130B	509 (507)	510 (508)	511 (509)	512 (510)		
130C	509 (507)	510 (508)	511 (509)	512 (510)	513 (512)	514 (512)

Table 1: Example allocation of messages – no gap in sequence numbers.

[0062] The broker 140 retrieves 525 the set of message events from one or more of the plurality of event stores. In some embodiments, the broker 140 retrieves the set of message events in a round robin fashion, such that the broker 140 retrieves one message from a particular event store, the next message event from a different event store, and the next message event from a different event store, and so on, thus, reducing the load on any one store. The broker 140 does not request a message event from a message broker 130 if the requested message event has a sequence number higher than that of any of the message events stored on the event store. Continuing the example, above, the broker 140 would request message events {513} and {514}only from event store 130C, as event stores 130A and 130B only had message events up {510} and {511}, respectively, thus distributing the load among the event stores 130A, 130B, and 130C. In this example, the retrieved set of messages may include messages events with sequence numbers {511-514}. Note, the message event associated with sequence number 509 need not be requested as the base number is 509 – accordingly broker state of the broker 140 is already updated to the messaging event associated with the sequence number 509. In alternate events, all the message events may be retrieved from a single store (e.g., message store 130C).

[0063] The broker 140 assembles 530 a message event stream based in part on the retrieved set of message events. The broker 140 creates the message event stream by ordering the message events in the received set by their respective sequence numbers. For example, a possible message event stream may be ordered {510, 511, 512, 513, 514}. The message events are ordered to identify any gaps in sequence numbers. Note that in some instances, there may be gaps in sequence numbers received from the multiple message stores 130A, 130B, 130C if one or more of the requested set of messages is missing. For example, a

possible message event stream including a gap may be {510, 512, 513, 514}, which is missing messaging event {511}. A gap may occur if, for example, one or message stores fails to receive a message event from the master broker 140.

[0064] The broker 140 identifies 535 a maximum contiguous message event (MCM) using the message event stream. The MCM is a stabilized message event with the highest sequence number that the broker 140 can recover before encountering a gap in the sequence numbers. In cases where there is no gap, the MCM is simply the stabilized message event with the highest sequence number in the message event stream. For example, assuming there is no gap, the message event associated with sequence number {512} is the MCM as it is the message event with the highest sequence number that is also stabilized (stored on at least two event stores). Alternatively, if the message event stream included a gap – e.g., the same stream as depicted in Table 1 – except that there are no entries for sequence number {511}, the MCM would be the message event associate with the sequence number {510} as it is the highest stabilized message before a gap occurs in the sequence numbers.

[0065] The broker 140 identifies 540 a synchronization point using the MCM. The MCM includes metadata that includes a back pointer to the last stable message (i.e., the sequence number of last stable message). The synchronization number of the messaging event pointed to by the back pointer of the MCM. Continuing with the above example where the MCM has a sequence number of {512}, the broker 140 would retrieve a back pointer in the metadata associated with that message that identifies the last stable message (e.g., the message event having a sequence number of {510}). Accordingly, the synchronization point would be the sequence number {510}.

[0066] The broker 140 republishes 545 any message events with sequence numbers above that of a synchronization point. The broker 140 republishes message events by re-generating the message events with sequence numbers above that of a synchronization point using a new epoch number and a new synchronization number, and then distributing the regenerated message events to all of the slave brokers and all of the event stores 130A, 130B, 130C. For example, assuming the synchronization point is the message with a sequence number of {503}, the broker 140 would republish message events with sequence numbers {504, 506, 507} to all of the event stores 130A, 130B, 130C and the remaining slave broker 140C. Slave brokers (e.g., 140C), responsive to receiving the republished message events

with the different epoch number, discard all messages events post the synchronization point, and ignore any future messages from the prior master broker. In parallel, the broker 140 updates 550 its broker state up to the MCM. At this point, all of the brokers 140A, 140B, and 140C, and all of the message stores 130A, 130B, and 130 are fully synchronized. This was achieved even though there is no one shared central repository of messages.

[0067] FIG. 6A is a class diagram of a replication stream and replication event store components of a message broker system to process a message, according to one embodiment. The replication stream event store components include a UMPersistenceAdapter class 605, a ReplicatingMessageStore class 610, a ReplicationStream class 615 and a ReplicatingTransaction Store class 620. The UM PersistenceAdapter 605 implements ReplicatingMessageStore 610 and Replicating Transaction Store 620 to ensure that the persistent data messages that are being written to disk are first replicated using the UMP interface to support a master/slave message broker system without a centralized repository. An ActiveMQ broker will call the ReplicatingMessageStore's methods (i.e. addMessage, removeMessage, etc) as the producer or consumer clients interact with the master broker to send message events. The ReplicationStream class 615 methods are used by all the ReplicatingMessageStores 620 of the UM PersistenceAdapter 605. For example, the "addMessage" command from the broker will create an AddMessageReplicationEvent in the ReplicatingMessageStore 620 and the event is sent to the ReplicationStream 615 class's "replicate" method. The replicate method in the ReplicationStream 615 will distribute a message event from the source to the plurality of event stores and slave brokers in the system. Thus, the ReplicationStream class 615 is used to send/receive the replicated events that occur in a master broker.

[0068] The master broker performs various administrative events that are replicated to a slave through the standard event store interface (such as "create a new queue destination" or "remove an existing queue destination"). The decisions made by the master broker will be sequenced and replicated by the replication stream protocol.

[0069] The actual disk persistence occurs as a result of receiving the replication events from the ReplicationStream 615. A long term storage interface is implemented to allow for different implementations writing data to the disk. FIG. 6B illustrates an example class diagram of long term storage components of a message broker system to process a message,

according to one embodiment. The long term storage components include a ReplicationStreamLongTermStorage class 635, a PersistenceAdapterBasedLongTermStorage class 640 and a LongTermStorageReplicationStreamListener 645 class. Once the replication event messages are sent to each slave broker, the receiver at the slave broker sends the message event to the ReplicationStreamListeners 645. One of the ReplicationStreamListener, i.e. the LongTermStorageReplicationStreamListener 645 will take each message event and pass it to the long term storage persistence adapter, i.e. PersistenceAdapterBasedLongTermStorage 640. The PersistenceAdapterBasedLongTermStorage 640 class will perform the actual persistence of the message event to the long term storage using the addMessage method.

[0070] FIG. 6C is an example of an interface 650 between an event store and a message broker of a message broker system to process a message, according to one embodiment. The messaging queue replication interface, as illustrated in FIG. 6C, is used by the persistence adapter and other messaging and event store components to perform methods related to the replication stream protocol. For example, a RepublishReplicationEvent method 655 is used by the replication stream in the event of transitioning a slave broker to a master broker, to republish message events once a synchronization point is determined. Similarly other methods described in the replication interface are used by one of the messaging system components when performing the replication stream protocol.

[0071] FIG. 7 is a high-level block diagram illustrating an example computer 700 for implementing the entities shown in FIG. 1. Illustrated are at least one processor 705 coupled to a chipset 710. Also coupled to the chipset 710 are a memory 715, a storage device 720, a keyboard 725, a graphics adapter 730, a pointing device 735, and a network adapter 740. A display 745 is coupled to the graphics adapter 530. In one embodiment, the functionality of the chipset 710 is provided by a memory controller hub 750 and an I/O controller hub 755. In another embodiment, the memory 715 is coupled directly to the processor 705 instead of the chipset 710.

[0072] The storage device 720 is any non-transitory computer-readable storage medium, such as a hard drive, compact disk read-only memory (CD-ROM), DVD, or a solid-state memory device. The memory 715 holds instructions and data used by the processor 705. The pointing device 735 may be a mouse, track ball, or other type of pointing device,

and is used in combination with the keyboard 725 to input data into the computer 700. The graphics adapter 730 displays images and other information on the display 745. The network adapter 740 couples the computer 700 to a network (e.g., the network 105 of FIG. 1).

[0073] As is known in the art, a computer 700 can have different and/or other components than those shown in FIG. 7. In addition, the computer 700 can lack certain illustrated components. For example, a computer 700 may lack a keyboard 725, pointing device 735, graphics adapter 730, and/or display 745. As another example, a computer 700 configured to display the stadium view visualization may be a tablet or smartphone with a touch screen interface and thus lack a keyboard 725 and pointing device 735. Moreover, the storage device 720 can be local and/or remote from the computer 700.

[0074] As is known in the art, the computer 700 is adapted to execute computer program modules for providing functionality described herein. As used herein, the term "module" refers to computer program logic utilized to provide the specified functionality. Thus, a module can be implemented in hardware, firmware, and/or software. In one embodiment, program modules are stored on the storage device 720, loaded into the memory 715, and executed by the processor 705.

[0075] Embodiments of the physical components described herein can include other and/or different modules than the ones described here. In addition, the functionality attributed to the modules can be performed by other or different modules in other embodiments. Moreover, this description occasionally omits the term "module" for purposes of clarity and convenience.

ADDITIONAL CONFIGURATION CONSIDERATIONS

[0076] Some portions of above description describe the embodiments in terms of algorithmic processes or operations. These algorithmic descriptions and representations are commonly used by those skilled in the data processing arts to convey the substance of their work effectively to others skilled in the art. These operations, while described functionally, computationally, or logically, are understood to be implemented by computer programs comprising instructions for execution by a processor or equivalent electrical circuits, microcode, or the like. Furthermore, it has also proven convenient at times, to refer to these

arrangements of functional operations as modules, without loss of generality. The described operations and their associated modules may be embodied in software, firmware, hardware, or any combinations thereof.

[0077] As used herein any reference to "one embodiment" or "an embodiment" means that a particular element, feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment. The appearances of the phrase "in one embodiment" in various places in the specification are not necessarily all referring to the same embodiment.

[0078] Some embodiments may be described using the expression "coupled" and "connected" along with their derivatives. It should be understood that these terms are not intended as synonyms for each other. For example, some embodiments may be described using the term "connected" to indicate that two or more elements are in direct physical or electrical contact with each other. In another example, some embodiments may be described using the term "coupled" to indicate that two or more elements are in direct physical or electrical contact. The term "coupled," however, may also mean that two or more elements are not in direct contact with each other, but yet still co-operate or interact with each other. The embodiments are not limited in this context.

[0079] As used herein, the terms "comprises," "comprising," "includes," "including," "has," "having" or any other variation thereof, are intended to cover a non-exclusive inclusion. For example, a process, method, article, or apparatus that comprises a list of elements is not necessarily limited to only those elements but may include other elements not expressly listed or inherent to such process, method, article, or apparatus. Further, unless expressly stated to the contrary, "or" refers to an inclusive or and not to an exclusive or. For example, a condition A or B is satisfied by any one of the following: A is true (or present) and B is false (or not present), A is false (or not present) and B is true (or present), and both A and B are true (or present).

[0080] In addition, use of the "a" or "an" are employed to describe elements and components of the embodiments herein. This is done merely for convenience and to give a general sense of the disclosure. This description should be read to include one or at least one and the singular also includes the plural unless it is obvious that it is meant otherwise.

[0081] Upon reading this disclosure, those of skill in the art will appreciate still additional alternative structural and functional designs for message brokering system. Thus, while particular embodiments and applications have been illustrated and described, it is to be understood that the described subject matter is not limited to the precise construction and components disclosed herein and that various modifications, changes and variations which will be apparent to those skilled in the art may be made in the arrangement, operation and details of the method and apparatus disclosed herein. The scope of the invention is to be limited only by the following claims.

CLAIMS

1. A method for a message broker computer system to process a message, the method comprising:

a master broker computer system receiving a message from a client producer;

the master broker computer system, responsive to receiving the message from the client producer, generating a message event, the message event uniquely identifying the message with an epoch value associated with the master broker computer system and a sequence number associated with the message, the message event further including a pointer to a last message stored by at least two of a plurality of event stores in associated persistent storage systems;

the master broker computer system distributing the message event to a plurality of event stores and a plurality of slave broker computer systems, wherein each event store has an associated persistent storage system, and each slave broker computer system has an associated memory;

each slave broker computer system holding the message event in its associated memory;

each event store that received the message event from the master broker computer system storing the message event in its associated persistent storage system, wherein once the message event is stored, each event store sends a stabilization notification to the master broker computer system indicating that the message event has been stored in the event store;

the master broker computer system, responsive to receiving stabilization notifications from at least two event stores, storing the message event in an associated persistent storage, and sending a notification to the client producer;

the master broker computer system generating a second message event indicating that the message event has been stabilized;

the master broker computer system distributing the second message event to the plurality of event stores and the plurality of slave broker computer systems; and

each slave broker computer system, responsive to receiving the second message event, storing the message event from its memory to a respective persistent storage system.

2. The method of claim 1, further comprising a broker computer system state for each messaging broker computer system, wherein a broker computer system state is a representation of the message event stream that has been persisted at the broker computer system.

3. The method of claim 2, further comprising updating the broker computer system state of each messaging broker computer system on receiving a message event from the master broker computer system, wherein updating the broker computer system state includes performing at least one of removal of a message event from the persistent storage or overwriting the message event with a received message event from the master broker computer system.

4. The method of claim 1, further comprising the master broker computer system removing the message event from an internal memory of the master broker computer system once the message event is stabilized.

5. The method of claim 1, further comprising removing the message event from an internal memory of a slave broker computer system on receiving at least one of a second message event from the master broker computer system or a stability message event from the master broker computer system, wherein a stability message event includes information associated with a last known stable message event at the master broker computer system.

6. The method of claim 1, wherein a second message event includes a message event, a sequence number associated with the last stable message event, an epoch value associated with the master broker computer system that sent the last stable message event, a sequence number associated with the message event and an epoch value associated with the master broker computer system.

7. The method of claim 1, wherein the master broker computer system distributes the message event to the plurality of slave broker computer systems and the plurality of event stores in parallel.

8. A method, in a messaging system comprising a master broker computer system, a plurality of slave broker computer systems, and a plurality of event stores, for a broker computer system to change its status from a slave broker computer system to a master broker computer system, the method comprising:

storing in persistent storage a plurality of message events, each message event comprising a message received from a client producer and metadata, the metadata uniquely identifying the message with an epoch value associated with a prior master broker computer system and a sequence number associated with the message event, and including a back pointer to a last message having been stored by at least two of the plurality of event stores in associated persistent storage systems;

receiving a notification to change status from a slave broker computer system to master broker computer system;

identifying a base value for the plurality of message events, the base value being a highest one of the sequence numbers of the message events stored by the broker computer system;

retrieving, from the plurality of event stores, sequence information describing the highest sequence numbers of message events which have been persisted on each of the plurality of event stores;

determining a set of message events to retrieve based on the base value and the sequence information;

retrieving the set of message events from one or more of the plurality of event stores;

assembling a message event stream based in part on the retrieved set of message events;

identifying a maximum contiguous message event (MCM) using the message event stream, wherein MCM is a message event with the highest sequence number that is observed before encountering a sequence number gap, a metadata of the MCM message event including a back pointer;

identifying a synchronization point using the MCM, the synchronization point being a sequence number pointed to by a back pointer in the metadata associated with the MCM;

republishing any message events with sequence numbers above that of a synchronization point with a new epoch number determined for the new master broker computer system, to each of the event stores and to a plurality of slave broker computer systems; and

updating a broker computer system state in the new master broker computer system to correspond to the MCM, wherein the broker computer system state indicates a state of the old master broker computer system prior to failure including information associated with stabilized message events corresponding to the MCM.

9. The method of claim 8, further comprising incrementing an epoch number on updating status of a slave broker computer system to a master broker computer system, wherein incrementing the epoch number indicates that a new broker computer system epoch has been initiated.

10. The method of claim 8, further comprising determining a failure of a master broker computer system wherein a failure is detected in response to one or more slave broker computer systems not receiving a message from a master broker computer system for a predetermined threshold of time.

11. The method of claim 8, wherein identifying a MCM event further comprises:

identifying a gap between sequence numbers of a retrieved set of stabilized message events; and

determining a maximum contiguous message event (MCM), wherein MCM is the stabilized message event with the highest sequence number that a broker computer system can recover before encountering the gap in the sequence numbers.

12. The method of claim 8, wherein republishing any message events further comprises:

regenerating the message events with sequence numbers above a synchronization point by using a new epoch number associated with the master broker computer system and a new synchronization number; and

distributing, in parallel, the regenerated message events to a plurality of slave broker computer systems and a plurality of event stores.

13. The method of claim 12 further comprising, ignoring, by a plurality of slave broker computer systems, any message event associated with a prior master broker computer system.

14. The method of claim 12 further comprising, discarding, by a plurality of slave broker computer systems, a message event that has a sequence number post the synchronization point.

15. A message broker computer system comprising:

a master broker computer system configured to receive a message from a client producer;

the master broker computer system, responsive to receiving the message from the client producer, generating a message event, the message event uniquely identifying the message with an epoch value associated with the master broker computer system and a sequence number associated with the message, the message event further including a pointer to a last message stored by at least two of a plurality of event stores in associated persistent storage systems;

the master broker computer system distributing the message event to a plurality of event stores and a plurality of slave broker computer systems, wherein each event store has an associated persistent storage system, and each slave broker computer system has an associated memory;

each slave broker computer system holding the message event in its associated memory;

each event store that received the message event from the master broker computer system storing the message event in its associated persistent storage system, wherein once the message event is stored, each event store sends a stabilization notification to the master broker computer system indicating that the message event has been stored in the event store;

the master broker computer system, responsive to receiving stabilization notifications from at least two event stores, storing the message event in an associated persistent storage, and sending a notification to the client producer;

the master broker computer system generating a second message event indicating that the message event has been stabilized;

the master broker computer system distributing the second message event to the plurality of event stores and the plurality of slave broker computer systems; and

each slave broker computer system, responsive to receiving the second message event, storing the message event from its memory to a respective persistent storage system.

16. The computer system of claim 15, further comprising a broker computer system state for each messaging broker computer system, wherein a broker computer system state is a representation of the message event stream that has been persisted at the broker computer system.

17. The computer system of claim 16, wherein the message broker computer system is further configured to update the broker computer system state of each messaging broker computer system on receiving a message event from the master broker computer system, wherein updating the broker computer system state includes performing at least one of removal of a message event from the persistent storage or overwriting the message event with a received message event from the master broker computer system.

18. The computer system of claim 15, wherein the master broker computer system is configured to remove the message event from an internal memory of the master broker computer system once the message event is stabilized.

19. The computer system of claim 15, wherein the master broker computer system is configured to remove the message event from an internal memory of a slave broker computer system on receiving at least one of a second message event from the master broker computer system or a stability message event from the master broker computer system, wherein a stability message event includes information associated with a last known stable message event at the master broker computer system.

20. The computer system of claim 15, wherein a second message event includes a message event, a sequence number associated with the last stable message event, an epoch value associated with the master broker computer system that sent the last stable message event, a sequence number associated with the message event and an epoch value associated with the master broker computer system.

21. The computer system of claim 15, wherein the master broker computer system distributes the message event to the plurality of slave broker computer systems and the plurality of event stores in parallel.

22. A messaging system comprising a master broker computer system, a plurality of slave broker computer systems, and a plurality of event stores, for a broker computer system to change its status from a slave broker computer system to a master broker computer system, the messaging system further comprising:

a persistent storage to store a plurality of message events, each message event comprising a message received from a client producer and metadata, the metadata uniquely identifying the message with an epoch value associated with a prior master broker computer system and a sequence number associated with the message event, and including a back pointer to a last message having been stored by at least two of the plurality of event stores in associated persistent storage systems;

a computer processor configured to:

receive a notification to change status from a slave broker computer system to master broker computer system;

identify a base value for the plurality of message events, the base value being a highest one of the sequence numbers of the message events stored by the broker computer system;

retrieve, from the plurality of event stores, sequence information describing the highest sequence numbers of message events which have been persisted on each of the plurality of event stores;

determine a set of message events to retrieve based on the base value and the sequence information;

retrieve the set of message events from one or more of the plurality of event stores;

assemble a message event stream based in part on the retrieved set of message events;

identify a maximum contiguous message event (MCM) using the message event stream, wherein MCM is a message event with the highest sequence number that is observed before encountering a sequence number gap, a metadata of the MCM message event including a back pointer;

identify a synchronization point using the MCM, the synchronization point being a sequence number pointed to by a back pointer in the metadata associated with the MCM;

republish any message events with sequence numbers above that of a synchronization point with a new epoch number determined for the new master broker computer system, to each of the event stores and to a plurality of slave broker computer systems; and

update a broker computer system state in the new master broker computer system to correspond to the MCM, wherein the broker computer system state indicates a state of the old master broker computer system prior to failure including information associated with stabilized message events corresponding to the MCM.

23. The messaging system of claim 22, wherein the computer processor is further configured to increment an epoch number on updating status of a slave broker computer system to a master broker computer system, wherein incrementing the epoch number indicates that a new broker computer system epoch has been initiated.

24. The messaging system of claim 22, wherein the computer processor is further configured to determine a failure of a master broker computer system wherein a failure is detected in response to one or more slave broker computer systems not receiving a message from a master broker computer system for a predetermined threshold of time.

25. The messaging system of claim 22, wherein identifying a MCM event further comprises:

identifying a gap between sequence numbers of a retrieved set of stabilized message events; and

determining a maximum contiguous message event (MCM), wherein MCM is the stabilized message event with the highest sequence number that a broker computer system can recover before encountering the gap in the sequence numbers.

26. The messaging system of claim 22, wherein republishing any message events further comprises:

regenerating the message events with sequence numbers above a synchronization point by using a new epoch number associated with the master broker computer system and a new synchronization number; and

distributing, in parallel, the regenerated message events to a plurality of slave broker computer systems and a plurality of event stores.

27. The messaging system of claim 26 wherein a plurality of slave broker computer systems are configured to ignore any message event associated with a prior master broker computer system.

28. The messaging system of claim 26 wherein a plurality of slave broker computer systems are configured to discard a message event that has a sequence number post the synchronization point.

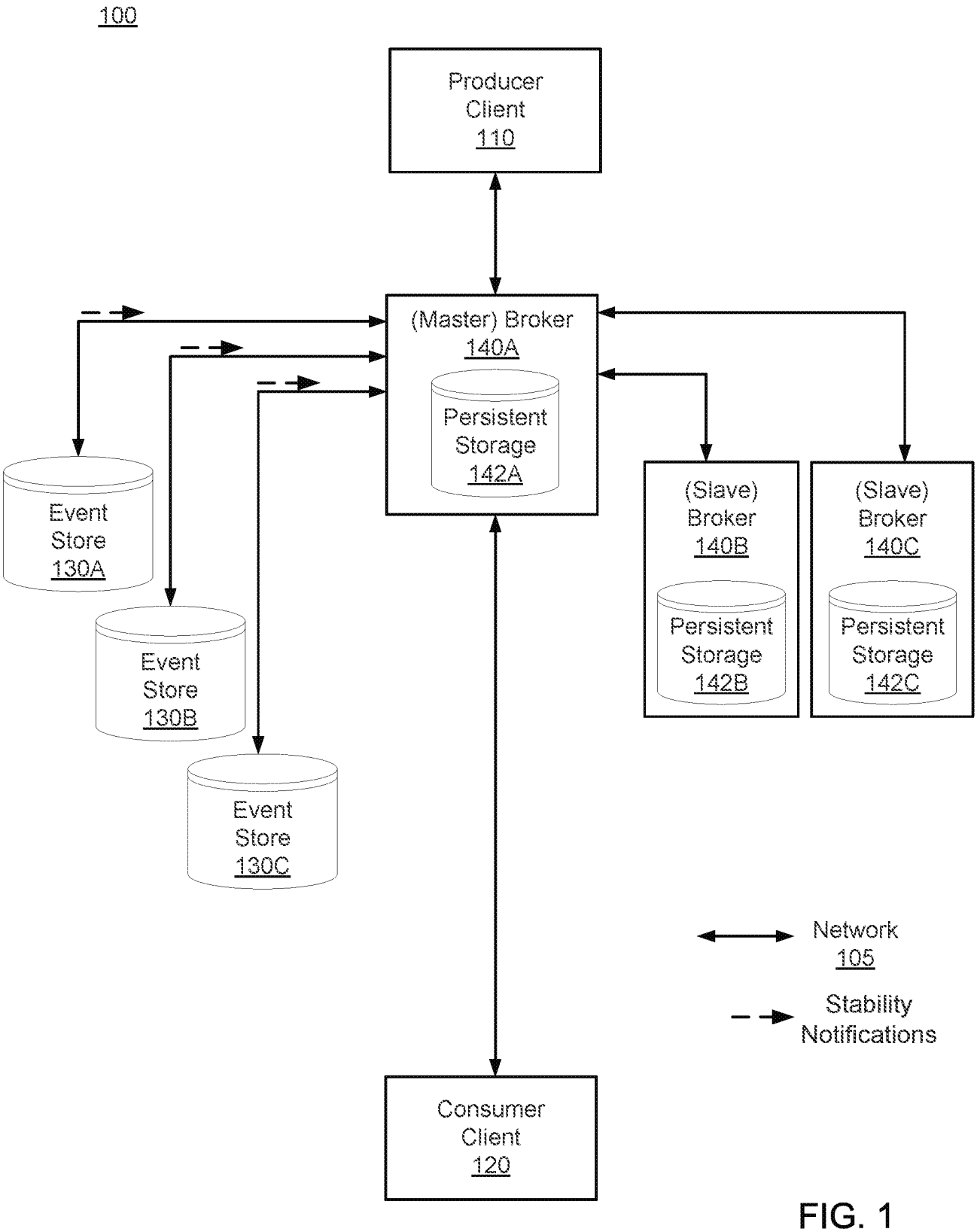
100
Producer Client 110
(Master) Broker 140A
Persistent Storage 142A
Event Store 130A
Event Store 130B
Event Store 130C
(Slave) Broker 140B
Persistent Storage 142B
(Slave) Broker 140C
Persistent Storage 142C
Consumer Client 120
Network 105
Stability Notifications

FIG. 1

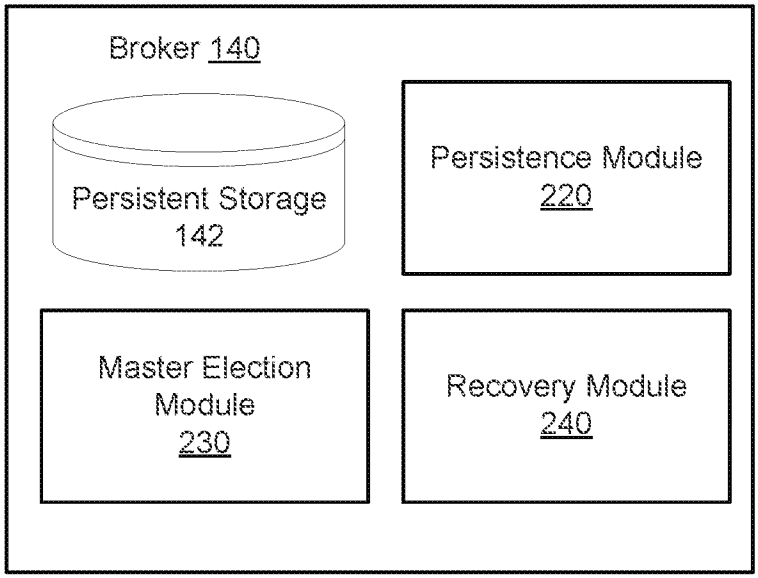
Broker 140
Persistent Storage 142
Persistence Module 220
Master Election Module 230
Recovery Module 240

FIG. 2

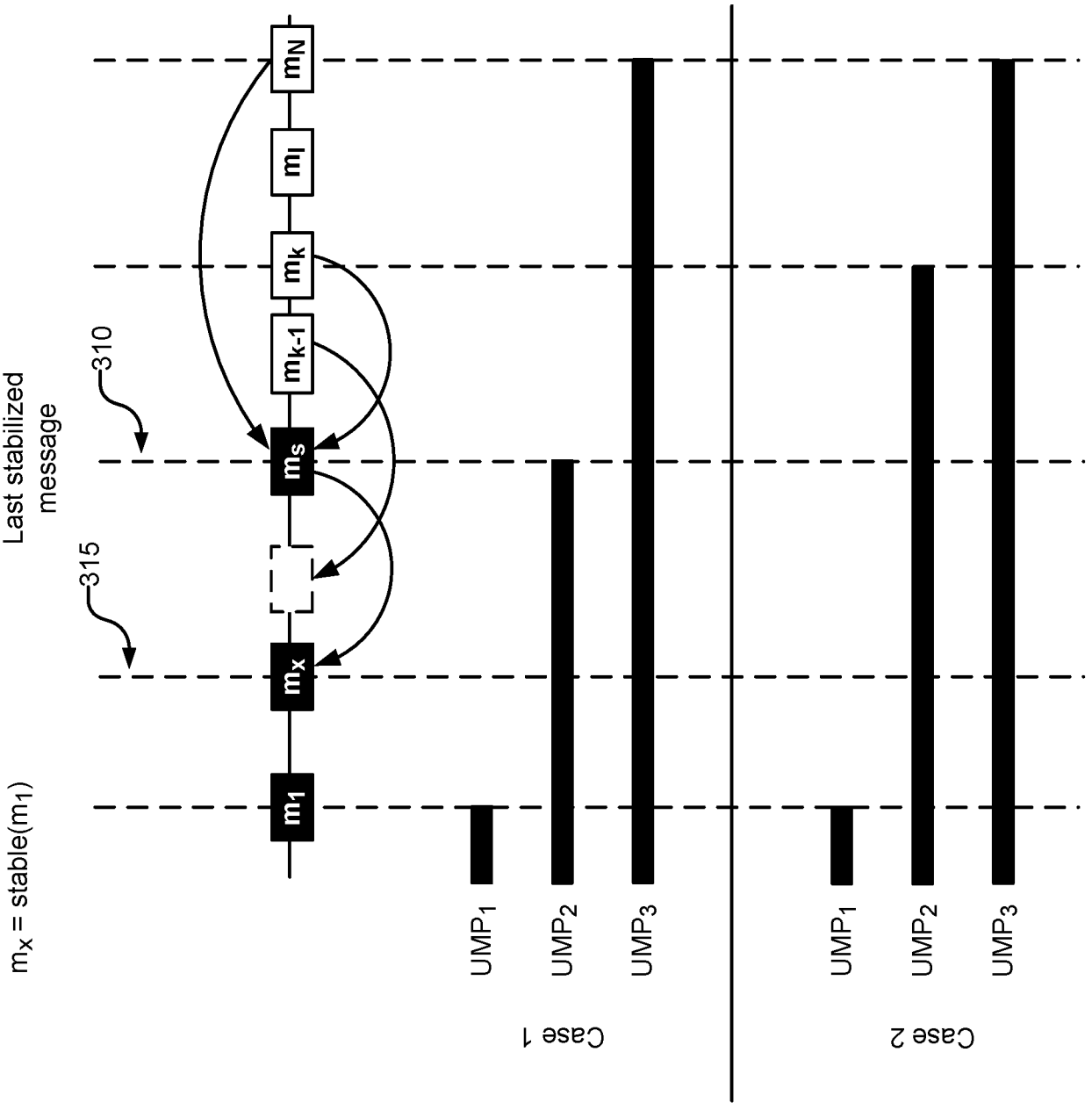
m_N
m_l
m_k
m_{k-1}
m_s
m_x
m_1
310
315
Last stabilized message
m_x = stable(m_1)
UMP_1
UMP_2
UMP_3
Case 1
Case 2

FIG. 3

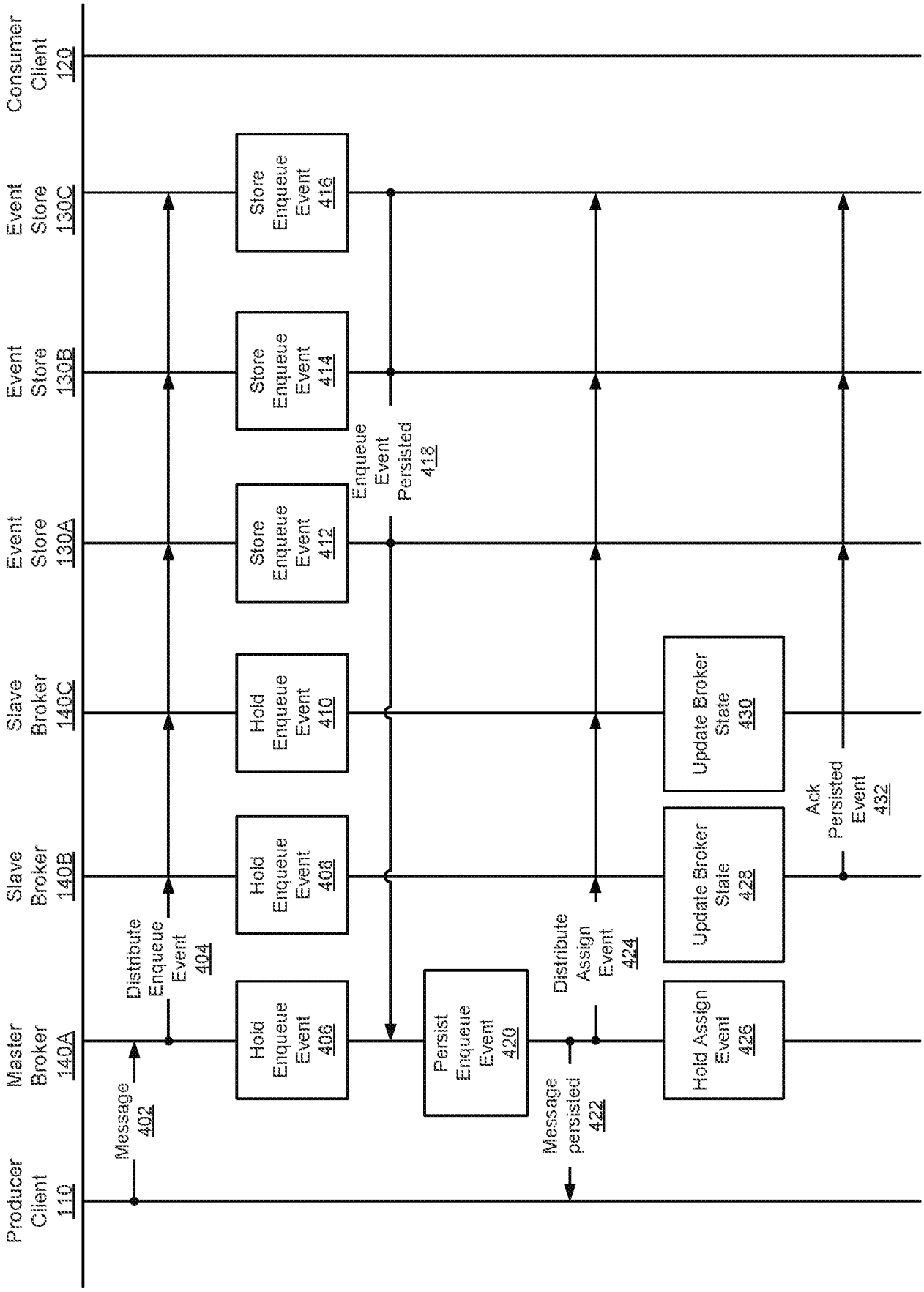
Producer Client 110
Master Broker 140A
Slave Broker 140B
Slave Broker 140C
Event Store 130A
Event Store 130B
Event Store 130C
Consumer Client 120
Message 402
Distribute Enqueue Event 404
Hold Enqueue Event 406
Hold Enqueue Event 408
Hold Enqueue Event 410
Store Enqueue Event 412
Store Enqueue Event 414
Store Enqueue Event 416
Enqueue Event Persisted 418
Persist Enqueue Event 420
Message persisted 422
Distribute Assign Event 424
Hold Assign Event 426
Update Broker State 428
Update Broker State 430
Ack Persisted Event 432

FIG. 4A

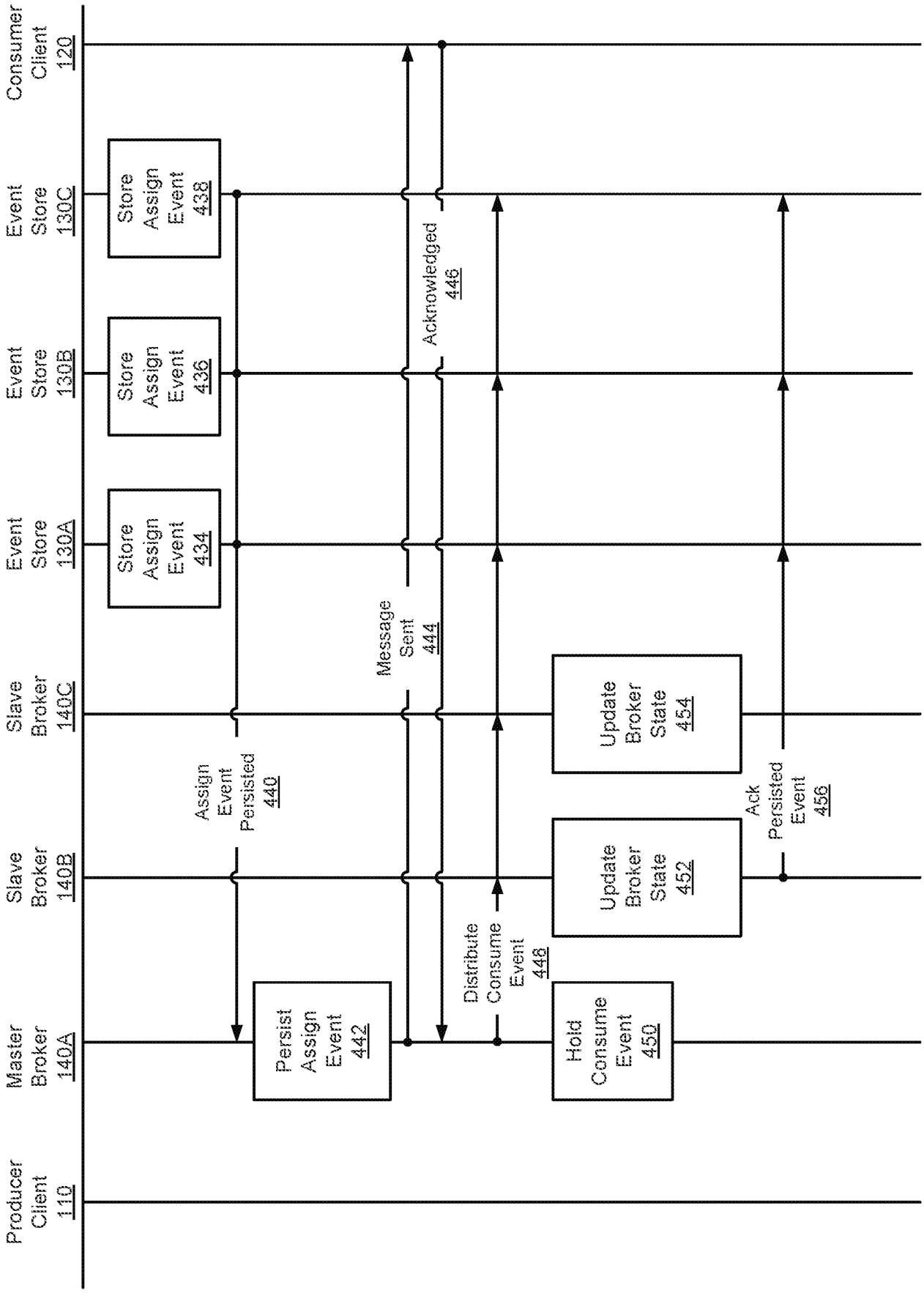
Producer Client 110
Master Broker 140A
Slave Broker 140B
Slave Broker 140C
Event Store 130A
Event Store 130B
Event Store 130C
Consumer Client 120
Store Assign Event 434
Store Assign Event 436
Store Assign Event 438
Assign Event Persisted 440
Persist Assign Event 442
Message Sent 444
Acknowledged 446
Distribute Consume Event 448
Hold Consume Event 450
Update Broker State 452
Update Broker State 454
Ack Persisted Event 456

FIG. 4B

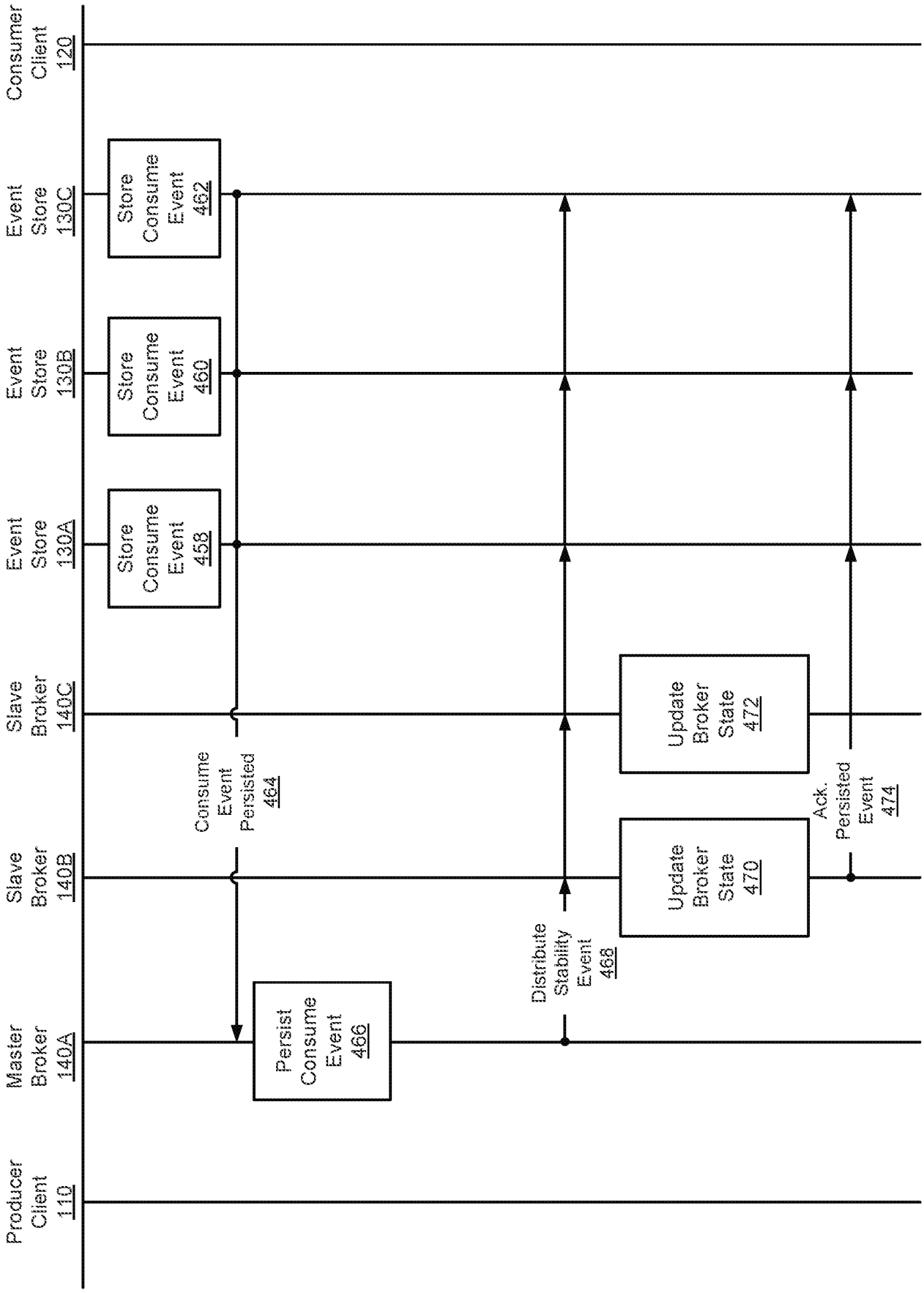
Consumer Client 120
Event Store 130C
Event Store 130B
Event Store 130A
Slave Broker 140C
Slave Broker 140B
Master Broker 140A
Producer Client 110
Store Consume Event 462
Store Consume Event 460
Store Consume Event 458
Consume Event Persisted 464
Persist Consume Event 466
Distribute Stability Event 468
Update Broker State 470
Update Broker State 472
Ack. Persisted Event 474

FIG. 4C

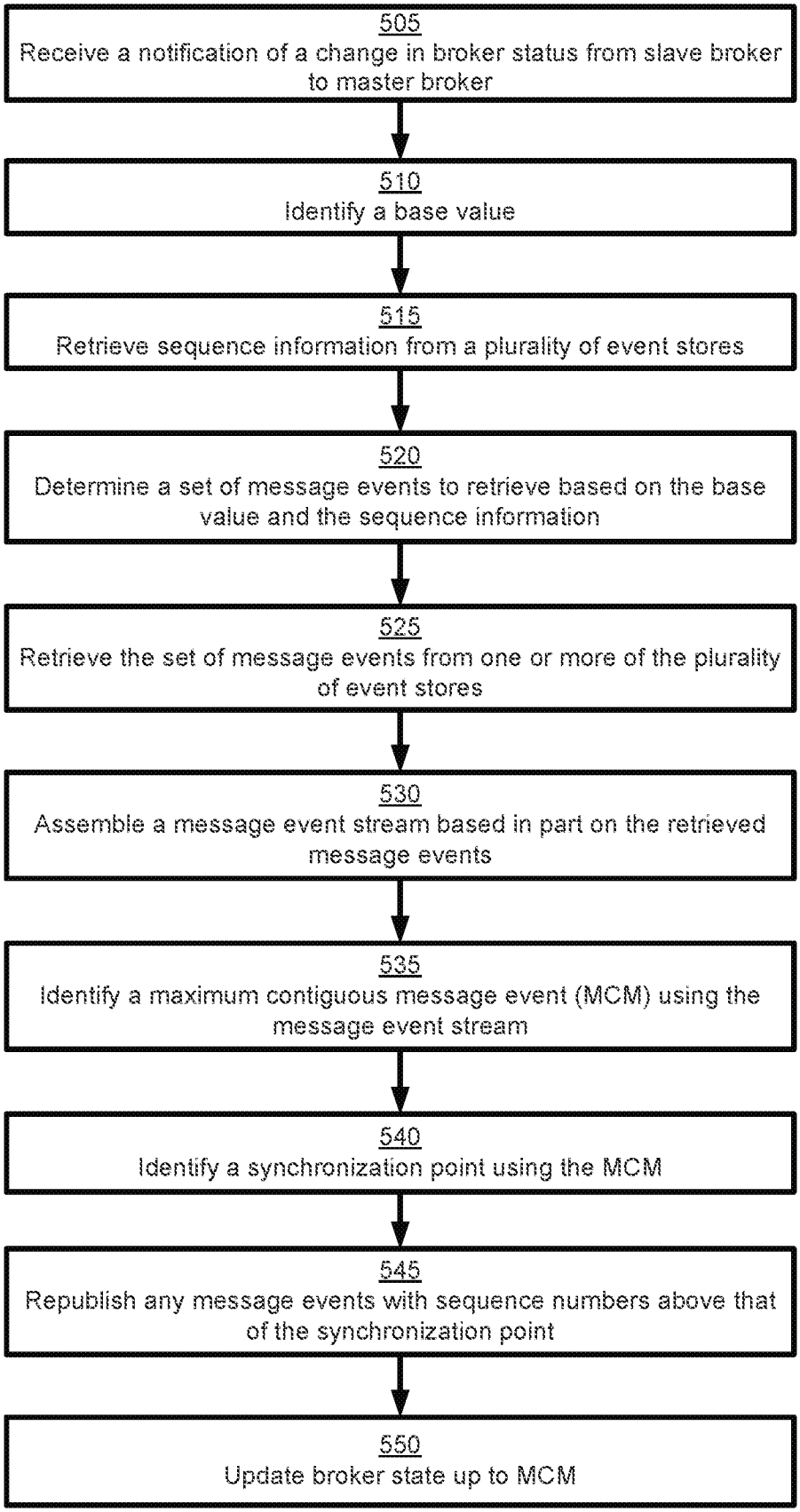
505
Receive a notification of a change in broker status from slave broker to master broker
510
Identify a base value
515
Retrieve sequence information from a plurality of event stores
520
Determine a set of message events to retrieve based on the base value and the sequence information
525
Retrieve the set of message events from one or more of the plurality of event stores
530
Assemble a message event stream based in part on the retrieved message events
535
Identify a maximum contiguous message event (MCM) using the message event stream
540
Identify a synchronization point using the MCM
545
Republish any message events with sequence numbers above that of the synchronization point
550
Update broker state up to MCM

FIG. 5

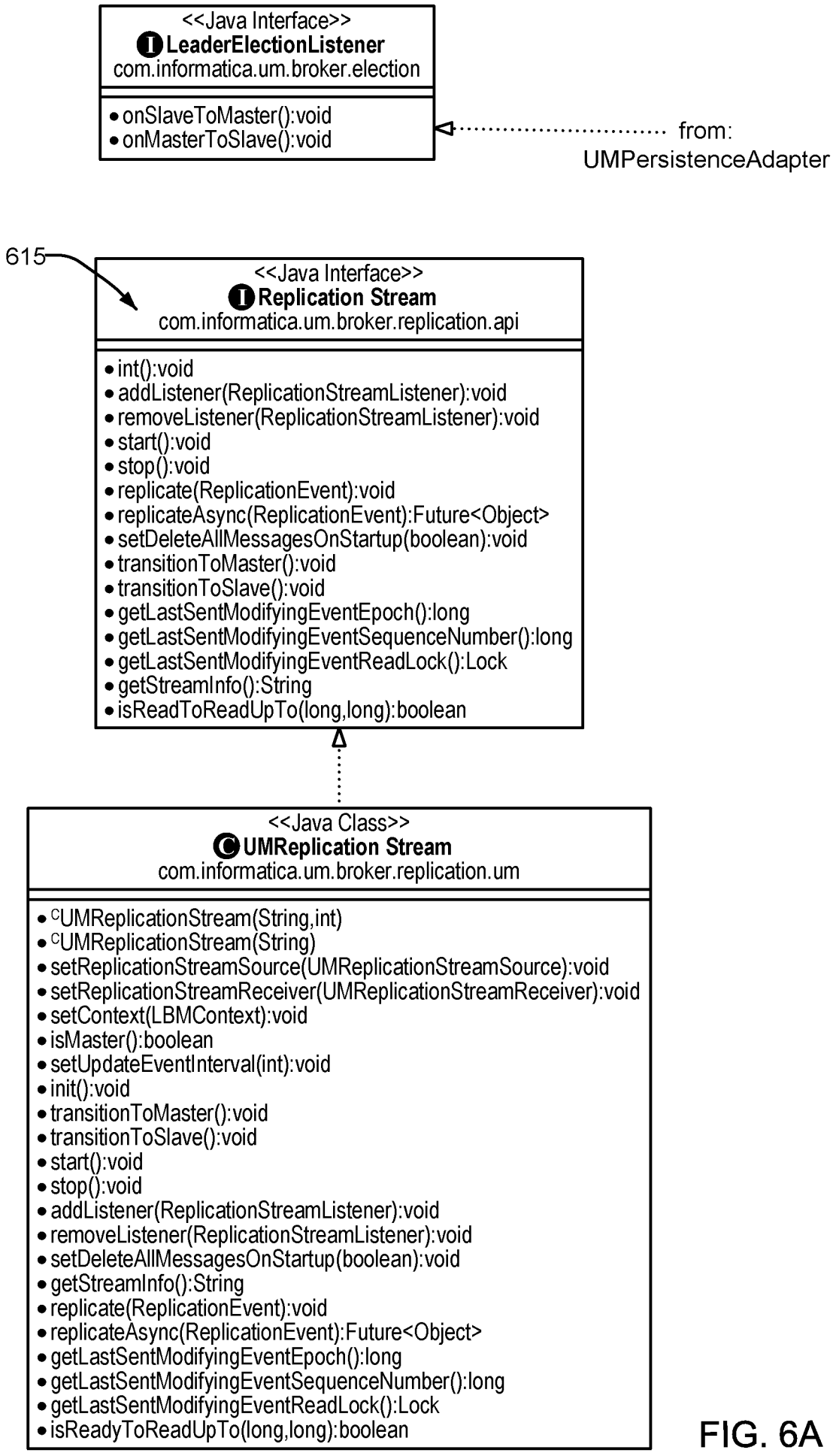
<<Java Interface>>
LeaderElectionListener
com.informatica.um.broker.election
• onSlaveToMaster():void
• onMasterToSlave():void
from:
UMPersistenceAdapter
615
<<Java Interface>>
Replication Stream
com.informatica.um.broker.replication.api
• int():void
• addListener(ReplicationStreamListener):void
• removeListener(ReplicationStreamListener):void
• start():void
• stop():void
• replicate(ReplicationEvent):void
• replicateAsync(ReplicationEvent):Future<Object>
• setDeleteAllMessagesOnStartup(boolean):void
• transitionToMaster():void
• transitionToSlave():void
• getLastSentModifyingEventEpoch():long
• getLastSentModifyingEventSequenceNumber():long
• getLastSentModifyingEventReadLock():Lock
• getStreamInfo():String
• isReadToReadUpTo(long,long):boolean
<<Java Class>>
UMReplication Stream
com.informatica.um.broker.replication.um
• cUMReplicationStream(String,int)
• cUMReplicationStream(String)
• setReplicationStreamSource(UMReplicationStreamSource):void
• setReplicationStreamReceiver(UMReplicationStreamReceiver):void
• setContext(LBMContext):void
• isMaster():boolean
• setUpdateEventInterval(int):void
• init():void
• transitionToMaster():void
• transitionToSlave():void
• start():void
• stop():void
• addListener(ReplicationStreamListener):void
• removeListener(ReplicationStreamListener):void
• setDeleteAllMessagesOnStartup(boolean):void
• getStreamInfo():String
• replicate(ReplicationEvent):void
• replicateAsync(ReplicationEvent):Future<Object>
• getLastSentModifyingEventEpoch():long
• getLastSentModifyingEventSequenceNumber():long
• getLastSentModifyingEventReadLock():Lock
• isReadyToReadUpTo(long,long):boolean

FIG. 6A

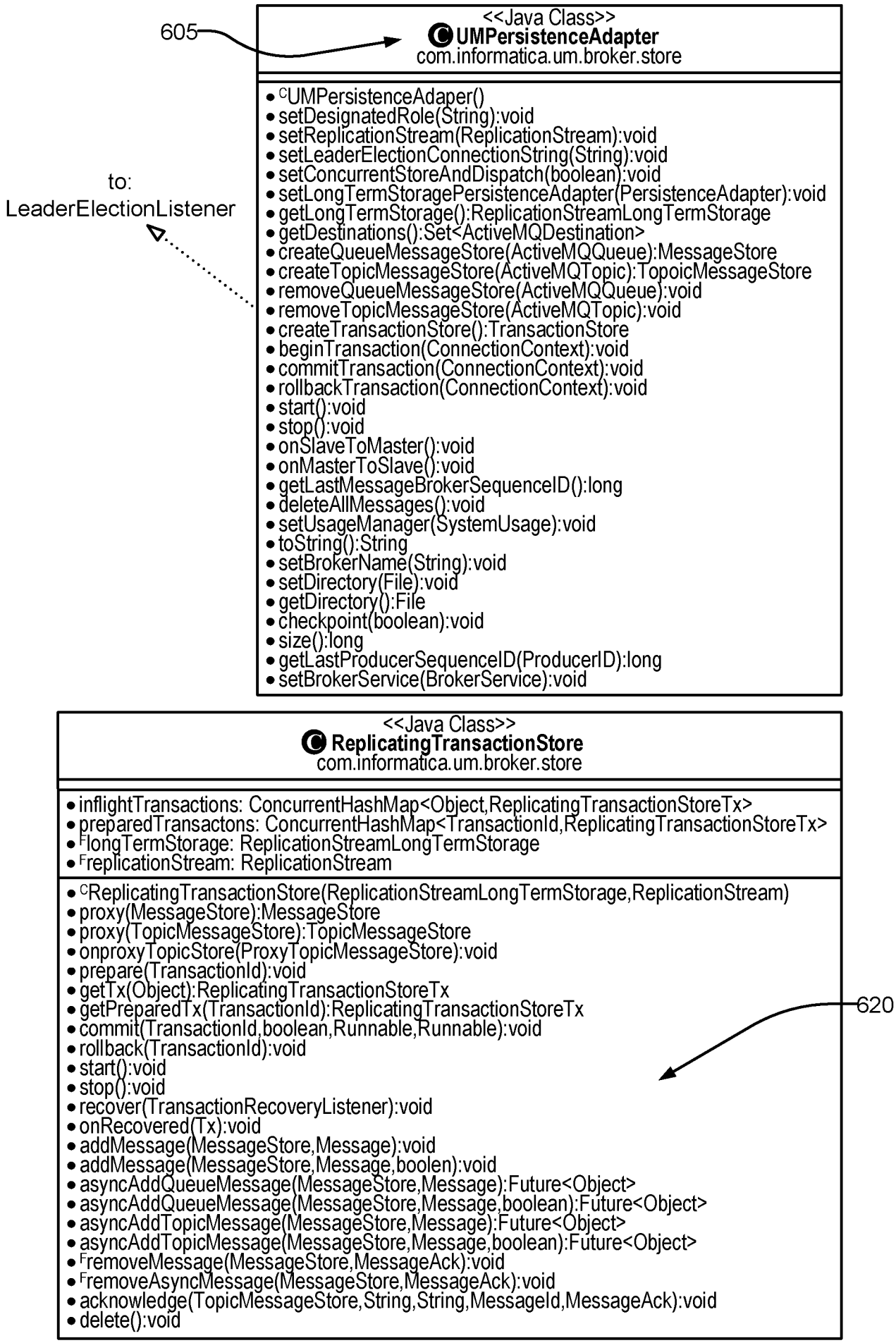

FIG. 6A (Continued)

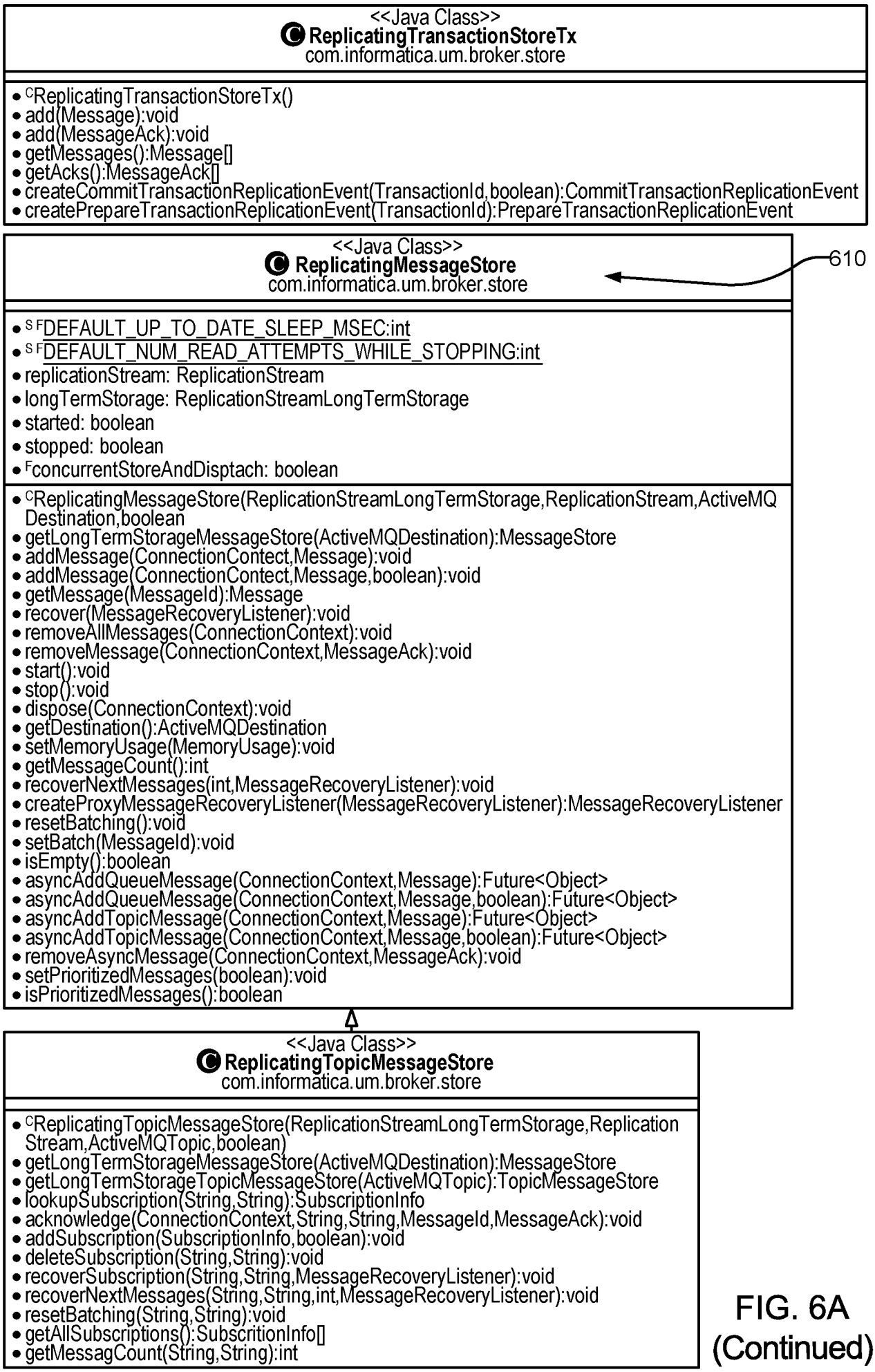

FIG. 6A (Continued)

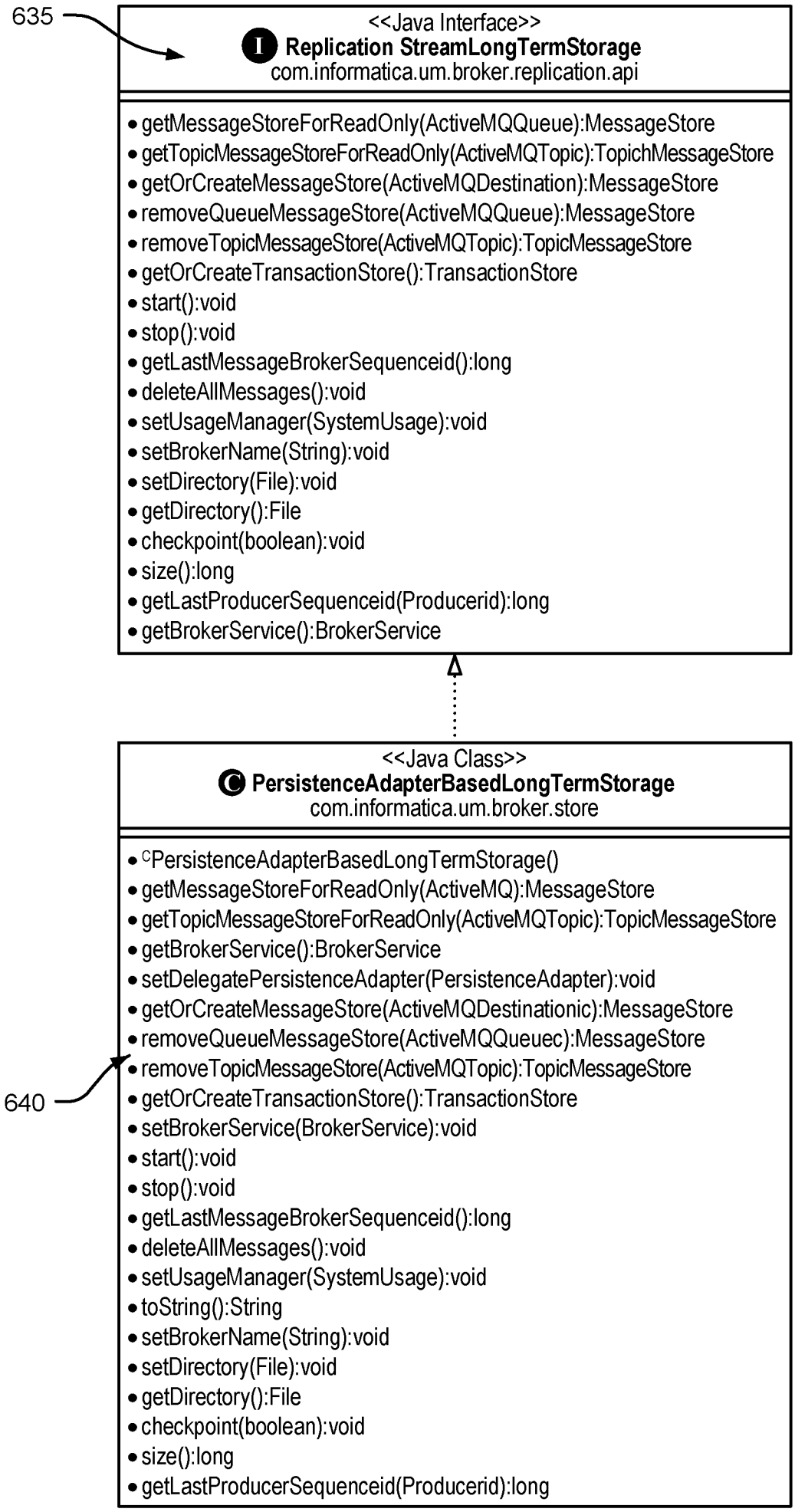
635
<<Java Interface>>
Replication StreamLongTermStorage
com.informatica.um.broker.replication.api
• getMessageStoreForReadOnly(ActiveMQQueue):MessageStore
• getTopicMessageStoreForReadOnly(ActiveMQTopic):TopichMessageStore
• getOrCreateMessageStore(ActiveMQDestination):MessageStore
• removeQueueMessageStore(ActiveMQQueue):MessageStore
• removeTopicMessageStore(ActiveMQTopic):TopicMessageStore
• getOrCreateTransactionStore():TransactionStore
• start():void
• stop():void
• getLastMessageBrokerSequenceid():long
• deleteAllMessages():void
• setUsageManager(SystemUsage):void
• setBrokerName(String):void
• setDirectory(File):void
• getDirectory():File
• checkpoint(boolean):void
• size():long
• getLastProducerSequenceid(Producerid):long
• getBrokerService():BrokerService
<<Java Class>>
PersistenceAdapterBasedLongTermStorage
com.informatica.um.broker.store
• CPersistenceAdapterBasedLongTermStorage()
• getMessageStoreForReadOnly(ActiveMQ):MessageStore
• getTopicMessageStoreForReadOnly(ActiveMQTopic):TopicMessageStore
• getBrokerService():BrokerService
• setDelegatePersistenceAdapter(PersistenceAdapter):void
• getOrCreateMessageStore(ActiveMQDestinationic):MessageStore
• removeQueueMessageStore(ActiveMQQueuec):MessageStore
• removeTopicMessageStore(ActiveMQTopic):TopicMessageStore
640
• getOrCreateTransactionStore():TransactionStore
• setBrokerService(BrokerService):void
• start():void
• stop():void
• getLastMessageBrokerSequenceid():long
• deleteAllMessages():void
• setUsageManager(SystemUsage):void
• toString():String
• setBrokerName(String):void
• setDirectory(File):void
• getDirectory():File
• checkpoint(boolean):void
• size():long
• getLastProducerSequenceid(Producerid):long

FIG. 6B

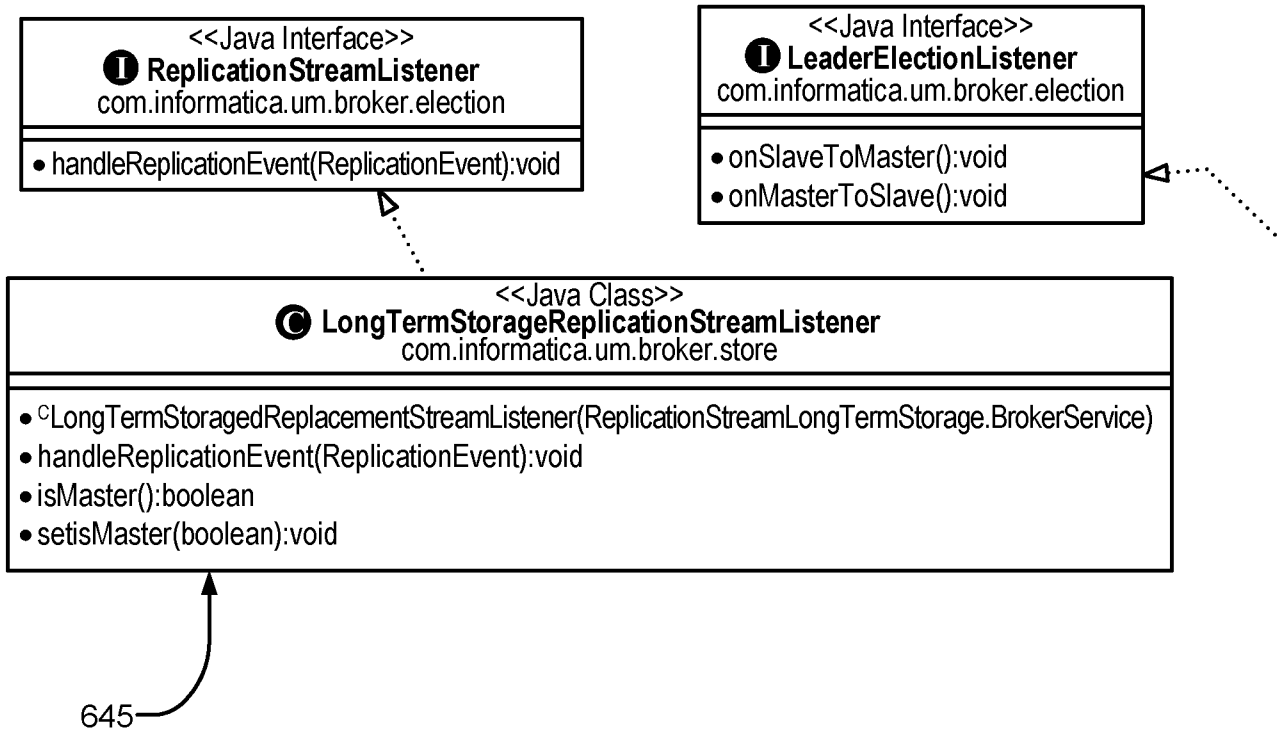

FIG. 6B (Continued)



FIG. 6B (Continued)

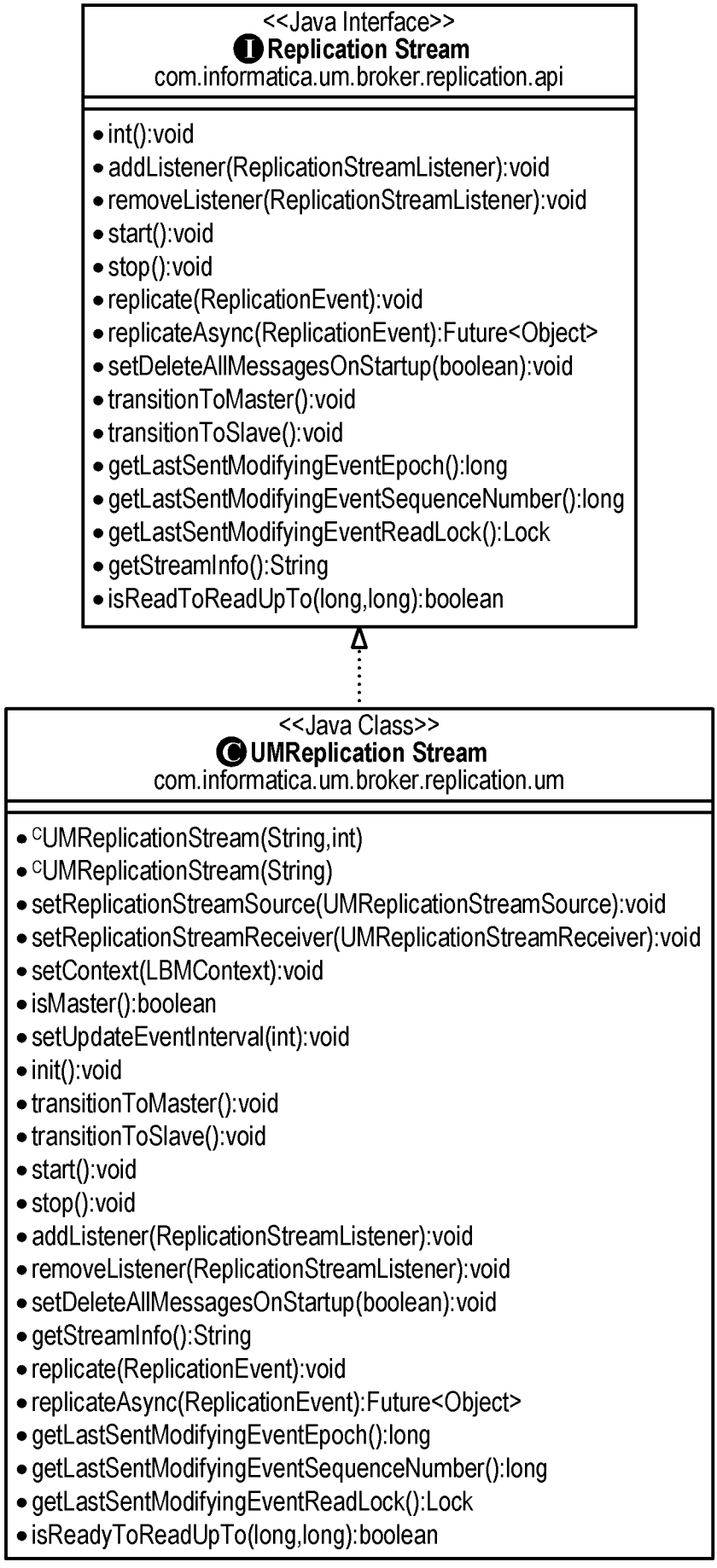

FIG. 6B
(Continued)


650
com.informatica.um.broker.replication.activemq.event
AddMessageReplicationEvent.java
AddMessageWithOptimizeHintReplicationEvent.java
AddSubscriptionReplicationEvent.java
BaseReplicationEvent.java
BaseReplicationEventWithDestination.java
ChangeEpochReplicationEvent.java
CommitTransactionReplicationEvent.java
CreateQueueDestinationReplicationEvent.java
CreateTopicDestinationReplicationEvent.java
DeleteSubscriptionReplicationEvent.java
DisposeReplicationEvent.java
PrepareTransactionReplicationEvent.java
RemoveAllMessagesReplicationEvent.java
RemoveMessageReplicationEvent.java
RemoveQueueDestinationReplicationEvent.java
RemoveTopicDestinationReplicationEvent.java
RepublishReplicationEvent.java
655
RollbackTransactionReplicationEvent.java
SubscriberAcknowledgeReplicationEvent.java
UnknownReplicationEvent.java
UpdateStableSequenceNumberReplicationEvent.java

FIG. 6C

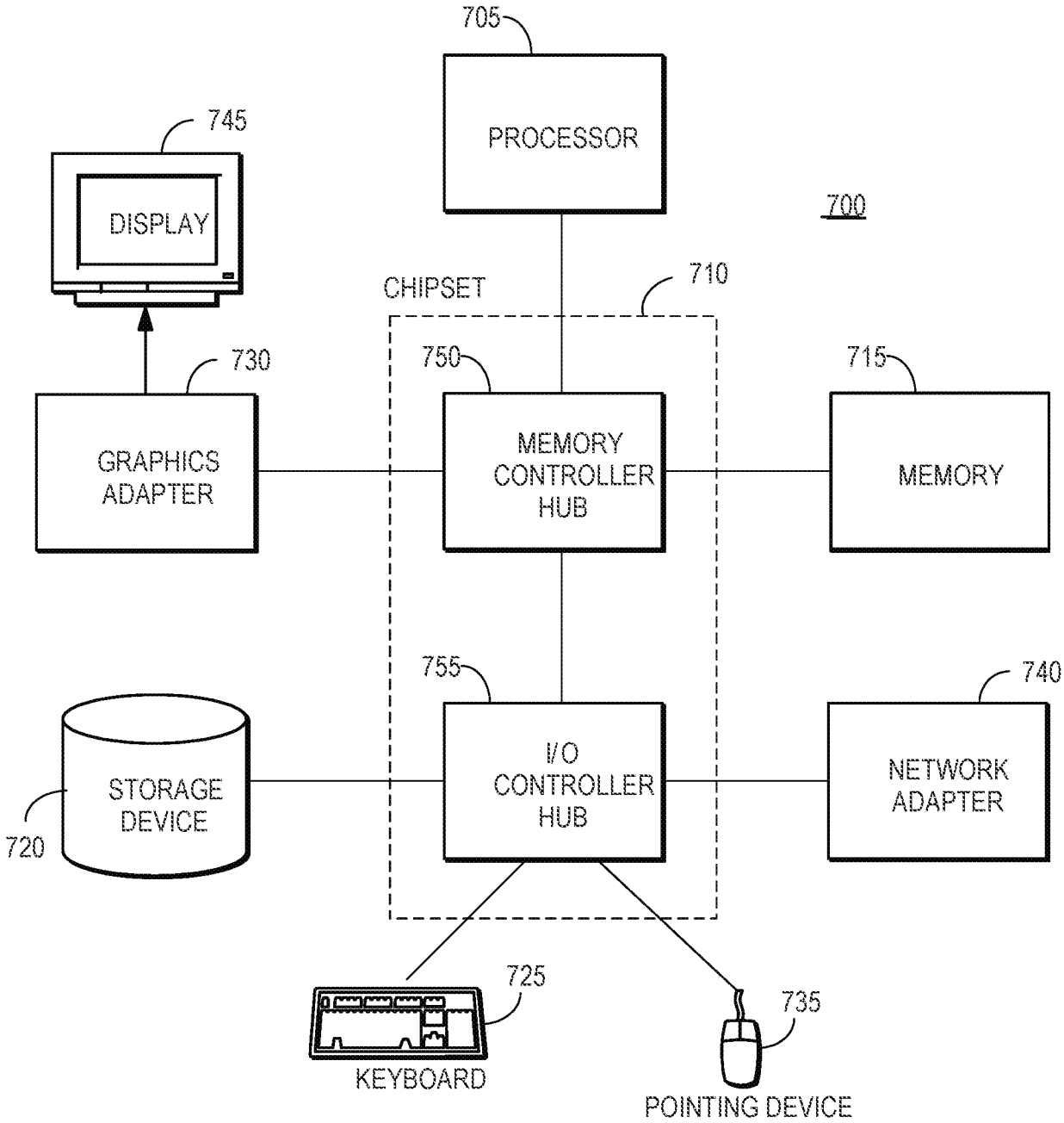
705
PROCESSOR
745
DISPLAY
700
CHIPSET
710
730
750
715
GRAPHICS ADAPTER
MEMORY CONTROLLER HUB
MEMORY
755
740
STORAGE DEVICE
720
I/O CONTROLLER HUB
NETWORK ADAPTER
725
735
KEYBOARD
POINTING DEVICE

FIG. 7

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/063034

Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:
See supplemental page

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying additional fees, this Authority did not invite payment of additional fees.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☒ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:
1-7, 15-21

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.

☐ The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.

☐ No protest accompanied the payment of additional search fees.

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/063034

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - G06F 17/30 (2016.01)
CPC - G06F 17/30575 (2016.01)
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(8) - G06F 15/177, 17/30, 17/40 (2016.01)
CPC - G06F 17/30348, 17/30371, 17/30575 (2016.01)

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
USPC 707/E17.032, 770, 802 (keyword delimited)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
PatBase, Google Patents, Google Scholar, Google
Search terms used: message, event, broker, master

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2008/0301098 A1 (BERNARD) 04 December 2008 (04.12.2008) entire document	1-7, 15-21
Y	US 2004/0267807 A1 (BARABAS et al) 30 December 2004 (30.12.2004) entire document	1-7, 15-21
A	US 20110276636 A1 (CHENG et al) 10 November 2011 (10.11.2011) entire document	1-7, 15-21
A	US 2005/0080796 A1 (MIDGLEY) 14 April 2005 (14.04.2005) entire document	1-7, 15-21
A	US 2014/0075017 A1 (SALESFORCECOM INC) 13 March 2014 (13.03.2014) entire document	1-7, 15-21

☐ Further documents are listed in the continuation of Box C. ☐ See patent family 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 application or patent 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
10 March 2016	29 MAR 2016
Name and mailing address of the ISA/ Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, VA 22313-1450 Facsimile No. 571-273-8300	Authorized officer Blaine R. Copenheaver PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774

Form PCT/ISA/210 (second sheet) (January 2015)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/063034

Continued from Box No. III Observations where unity of invention is lacking

This application contains the following inventions or groups of inventions which are not so linked as to form a single general inventive concept under PCT Rule 13.1. In order for all inventions to be examined, the appropriate additional examination fees must be paid.

Group I, claims 1-7,15-21, drawn to a message broker computer system to process a message.
Group II, claims 8-14,22-28, drawn to a messaging system and method comprising a master broker computer system.

The inventions listed as Groups I-II do not relate to a single general inventive concept under PCT Rule 13.1 because, under PCT Rule 13.2, they lack the same or corresponding special technical features for the following reasons: the special technical feature of the Group I invention: each event store sends a stabilization notification to the master broker computer system indicating that the message event has been stored in the event store; the master broker computer system, responsive to receiving stabilization notifications from at least two event stores, storing the message event in an associated persistent storage, and sending a notification to the client producer as claimed therein is not present in the invention of Group II. The special technical feature of the Group II invention: change its status from a slave broker computer system to a master broker computer system; receiving a notification to change status from a slave broker computer system to master broker computer system as claimed therein is not present in the invention of Group I.

Groups I and II lack unity of invention because even though the inventions of these groups require the technical feature of a messaging system comprising a master broker computer system, a plurality of slave broker computer systems, and a plurality of event stores; storing in persistent storage a plurality of message events, this technical feature is not a special technical feature as it does not make a contribution over the prior art.

Specifically, US 2005/0080796 A1 (MIDGLEY) 14 April 2005 (14.04.2005) teaches a messaging system comprising a master broker computer system, a plurality of slave broker computer systems, and a plurality of event stores (Para. 46 and Fig. 9); storing in persistent storage a plurality of message events (Para. 41).

Since none of the special technical features of the Group I or II inventions are found in more than one of the inventions, unity of invention is lacking.