



- (51) International Patent Classification:
G06F 17/30 (2006.01)
- (21) International Application Number:
PCT/US2016/014547
- (22) International Filing Date:
22 January 2016 (22.01.2016)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:
14/604,477 23 January 2015 (23.01.2015) US
- (63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 14/604,477 (CON)
Filed on 23 January 2015 (23.01.2015)
- (71) Applicant: EBAY INC. [US/US]; 2145 Hamilton Avenue, San Jose, California 95125 (US).
- (72) Inventors: MURTHY, Sharad R.; 2270 Bayleaf Drive, San Ramon, California 94582 (US). AVALANI, Bhaven; 10384 Melissa Court, Cupertino, California 95014 (US). NG, Chun Tung Tony; 908 Bailey Court, San Ramon, California 94582 (US).
- (74) Agents: SCHEER, Bradley, W. et al.; P O Box 2938, Minneapolis, Minnesota 55402 (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, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE,

[Continued on next page]

(54) Title: MESSAGING AND PROCESSING HIGH VOLUME DATA

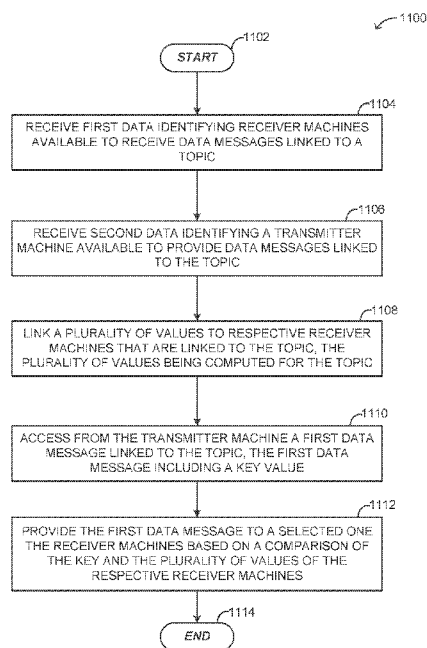


FIG. 11

(57) Abstract: Disclosed are a system comprising a computer-readable storage medium storing at least one program, and a computer-implemented method for providing partial views of event streams over a network. A subscription manager receives subscription data from a client device. A messaging interface module receives an event stream comprising event messages. A normalizer converts the received event stream to a table of entries. The entries correspond to respective event messages. A view processor selects a portion of the converted first event stream based at least on the entries of the selected portion of the event stream matching the subscription data. The view processor provides the selected portion of the converted event stream for transmission as session data to the subscriber.





DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, **Published:**

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).

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

MESSAGING AND PROCESSING HIGH VOLUME DATA

PRIORITY CLAIM

[0001] The application claims priority to U.S. Patent Application No. 14/604,477, entitled “SYSTEMS AND METHODS FOR MESSAGING AND PROCESSING HIGH VOLUME DATA OVER NETWORKS,” filed January 23, 2015, which is incorporated herein by reference in its entirety.

TECHNICAL FIELD

[0002] Example embodiments of the present application relate generally to the technical field of data processing.

BACKGROUND

[0003] Communications between or among machines may be performed using a publisher-subscriber arrangement. A transmitter machine functions as a message publisher, also known as a message producer. The transmitter machine may transmit (e.g., produce or publish) one or more messages using a network. The transmitter machine sends a message via the network to one or more receiver machines. The message, however, is not addressed to any particular receiver machine. Rather, the transmitter machine sends the message using a multicast network protocol that allows multiple receiver machines to each receive the message. The multicast protocol supports one-to-many communication, and the transmitter machine has no information indicating which specific receiver machine will process the message. In this regard, the multicast communication differs from point-to-point (e.g., one-to-one) communication. A receiver machine functions as a message subscriber, also known as a message consumer. The receiver machine may receive (e.g., consume) the message sent from the transmitter machine. The receiver machine monitors the network for messages sent using the multicast protocol.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] In the drawings, which are not necessarily drawn to scale, like numerals can describe similar components in different views. Like numerals having different letter or numeric suffixes can represent different instances of similar components. The drawings illustrate generally, by way of example, but not by way of limitation, various embodiments discussed in the present document.

[0005] FIG. 1 is a network diagram depicting a client-server system, within which one example embodiment can be deployed.

[0006] FIG. 2 is a block diagram illustrating a mobile device, according to an example embodiment.

[0007] FIG. 3 is a block diagram illustrating an example embodiment of a messaging system including multiple modules forming at least a portion of the client-server system of FIG. 1.

[0008] FIG. 4 is a block diagram illustrating an example producer-consumer system, in accordance with an example embodiment.

[0009] FIG. 5 is a block diagram illustrating an example messaging stack, in accordance with an example embodiment.

[0010] FIG. 6 is a block diagram illustrating an example producer-agent-consumer system, in accordance with an example embodiment.

[0011] FIG. 7 is a block diagram illustrating an example data structure of an advertisement, in accordance with an example embodiment.

[0012] FIG. 8 is a block diagram illustrating an example messaging system architecture, in accordance with an example embodiment.

[0013] FIG. 9 is a block diagram illustrating an example monitoring system deployed by the messaging system, in accordance with an example embodiment.

[0014] FIG. 10 is a schematic diagram depicting an example embodiment of interactions of producers and consumers for reconfiguring a consumer cluster, according to an example embodiment.

[0015] FIG. 11 is a flowchart illustrating an example method of cluster messaging, in accordance with an example embodiment.

[0016] FIG. 12 is a block diagram illustrating an example sessionizer system architecture, in accordance with an example embodiment.

[0017] FIG. 13 is a block diagram illustrating example embodiment of a sessionizer including multiple modules.

[0018] FIG. 14 is a block diagram illustrating an example sessionizer architecture, in accordance with an example embodiment.

[0019] FIG. 15 is a flowchart illustrating an example method of generating the sessionized data, in accordance with an example embodiment.

[0020] FIG. 16 is a flowchart illustrating an example method of generating and updating sessions, in accordance with an example embodiment.

[0021] FIG. 17 is a flowchart illustrating an example method of session lifecycle management, in accordance with an example embodiment.

[0022] FIG. 18 is a block diagram illustrating an example stream view management architecture, in accordance with an example embodiment.

[0023] FIG. 19 is a block diagram illustrating an example stream view management system, in accordance with an example embodiment.

[0024] FIG. 20 is a flowchart illustrating an example method of stream view management, in accordance with an example embodiment.

[0025] FIGS. 21A and 21B are block diagrams illustrating example data structures of an event message, in accordance with example embodiments.

[0026] FIG. 22 is a block diagram of a machine in the example form of a computer system within which instructions can be executed for causing the machine to perform any one or more of the methodologies discussed herein.

DETAILED DESCRIPTION

[0027] Reference will now be made in detail to specific example embodiments for carrying out the inventive subject matter. Examples of these specific embodiments are illustrated in the accompanying drawings. It will be understood that they are not intended to limit the scope of the claims to the described embodiments. On the contrary, they are intended to cover alternatives, modifications, and equivalents as can be included within the scope of the disclosure as defined by the appended claims.

[0028] In the following detailed description of example embodiments of the invention, reference is made to the accompanying drawings which form a part hereof, and which is shown by way of illustration only, specific embodiments in which the invention may be practiced. It is to be understood that other embodiments may be utilized and structural changes may be made without departing from the scope of the present invention. Specific details are set forth in order to provide a thorough understanding of the subject matter. Embodiments can be practiced without some or all of these specific details. In addition, well known features may not have been described in detail to avoid unnecessarily obscuring the subject matter.

[0029] In accordance with the present disclosure, components, process steps, and/or data structures are implemented using various types of operating systems, programming languages, computing platforms, computer programs, and/or like machines. In addition, those of ordinary skill in the art will recognize that devices, such as hardwired devices, field programmable gate arrays (FPGAs), application specific integrated circuits (ASICs), or the like, can also be used to exploit one or more technical aspects of the devices without departing from the scope of the concepts disclosed herein. Embodiments can also be tangibly embodied as a set of computer instructions stored on a computer readable medium, such as a memory device, to exploit technical aspects of a computer-instruction based embodiments.

[0030] Example methods and systems for distributing and/or processing data, which are embodied on electronic devices, are described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of example embodiments. It will be evident, however, to one skilled in the art, that the present inventive subject matter can be practiced without these specific details.

[0031] In example embodiments, systems and methods are disclosed for distributing event messages, sessionizing event streams, and managing views of the event streams in real time within networks. For example, real-time complex event processing (CEP) involves processing millions of events per second. In some scenarios, CEP can involve ingestion of event streams at very high velocity (e.g., up to 1 million events per sec), volume (e.g., terabytes of data), and/or

variety (e.g., unstructured and semi structured data). CEP engines are CEP instances used to analyze event streams to compute aggregates for the tuples of information or a combination of tuples of information contained in the event. Sometimes multiple streams can be joined to correlate event streams or detect patterns in the arrival of events. However, a CEP engine running in a single node will not have the processing resources to handle such large volumes.

[0032] As disclosed herein, example embodiments deploy multiple CEP engines in a cluster and deployed on a number of devices. Example embodiments distribute the workload across the cluster of CEP engines. Such an arrangement can provide a scalable system. The system can scale the cluster of CEP engines elastically so that as load increases new CEP engines can be added to the cluster dynamically without impacting the health (e.g., performance, network stability, etc.) of the cluster. The cluster can selfheal in case of an CEP engine failures or a specific instance becoming busy. Furthermore, the system can deploy the cluster across one or more datacenters. In such a scenario, event messages flow over a wide area network. In order to use the communication bandwidth efficiently, the messaging system can batch and compress the messages travelling over the network.

[0033] As used herein, events can refer to messages in a messaging system. It will be appreciated that example embodiments of the messaging system can be used in applications other than the CEP use case.

[0034] FIG. 1 is a network diagram depicting a client-server system 100, within which one example embodiment can be deployed. A networked system 102, in the example form of a network-based marketplace or publication system, provides server-side functionality, via a network 104 (e.g., the Internet or wide area network (WAN)), to one or more clients. FIG. 1 illustrates, for example, a web client 106 (e.g., a browser), and a programmatic client 108 executing on respective client machines 110 and 112. Herein, the client machine 110 can be referred to as a “client device” or “user device” in various applications.

[0035] An application program interface (API) server 114 and a web server 116 are coupled to, and provide programmatic and web interfaces respectively to, one or more application servers 118. The application servers 118 host one or more marketplace applications 120, and payment applications 122. The

application servers 118 are, in turn, shown to be coupled to one or more data processing servers 123 that facilitate processing data and database servers 124 that facilitate access to one or more databases 126.

[0036] The marketplace application(s) 120 can provide a number of marketplace functions and services to users that access the networked system 102. The payment application(s) 122 can likewise provide a number of payment services and functions to users. The payment application(s) 122 can allow users to accumulate value (e.g., in a commercial currency, such as the U.S. dollar, or a proprietary currency, such as "points") in accounts, and then later to redeem the accumulated value for items that are made available via the marketplace application(s) 120.

[0037] The data processing servers 123 can provide a number of functions and services to the networked system 102. In an example embodiment, the data processing servers can deploy a number of producer devices for generating event messages based on data received from the client machines 110. Furthermore, the data processing servers 123 can deploy a number of CEP engines for processing event messages generated by the producer devices. The data processing servers 123 can correspond to a number of servers deploying a distributed architecture. For example, a number of data processing servers 123 can be deployed within a number of datacenters as producer devices. Additionally or alternatively, a number of data processing servers 123 can be deployed within a number of datacenters as CEP engines. As will be described later in connection with FIG. 3, the data processing servers 123 can include additional components for facilitating routing event messages to the respective CEP engine.

[0038] Further, while the system 100 shown in FIG. 1 employs a client-server architecture, the present inventive subject matter is, of course, not limited to such an architecture, and could equally well find application in a distributed, or peer-to-peer, architecture system, for example. The various marketplace and payment applications 120, 122 could also be implemented as standalone software programs, which do not necessarily have networking capabilities.

[0039] In addition, while the various marketplace and payment applications 120, 122 have been described above as having separate functionalities, in

alternative embodiments these functionalities can be performed by any one or more of the various marketplace and payment applications 120, 122.

[0040] The web client 106 accesses the various marketplace and payment applications 120 and 122 via the web interface supported by the web server 116. Similarly, the programmatic client 108 accesses the various services and functions provided by the marketplace and payment applications 120 and 122 via the programmatic interface provided by the API server 114. The programmatic client 108 can, for example, be a seller application (e.g., the TURBOLISTER™ application developed by EBAY INC.™, of San Jose, California) to enable sellers to author and manage listings on the networked system 102 in an off-line manner, and to perform batch-mode communications between the programmatic client 108 and the networked system 102.

[0041] FIG. 1 also illustrates a third party application 128, executing on a third party server 130, as having programmatic access to the networked system 102 via the programmatic interface provided by the API server 114. For example, the third party application 128 can, utilizing information retrieved from the networked system 102, support one or more features or functions on a website hosted by the third party. The third party website can, for example, provide one or more promotional, marketplace, or payment functions that are supported by the relevant applications of the networked system 102.

[0042] FIG. 2 is a block diagram illustrating a mobile device 200, according to an example embodiment. In an example embodiment, the mobile device 200 can correspond to the client machine 110 of FIG. 1. In particular, the mobile device 200 can interact with the networked system based on user input received by the mobile device 200 from a user. Examples of user input can include file requests, page views, clicks, form submissions, “keypress” events, input device events, and/or other client-machine-side interactions. File requests can encompass requesting, e.g., a web page, a component of a webpage, image data, data from the marketplace application 120, and the web resources. The user input can additionally or alternatively correspond to a sequence of interactions, such a click path or stream (e.g., an ordered sequence of client-machine-side interactions).

[0043] The mobile device 200 can include a processor 202. The processor 202 can be any of a variety of different types of commercially available processors specially configured for mobile devices 200 (for example, an XScale architecture microprocessor, a microprocessor without interlocked pipeline stages (MIPS) architecture processor, or another type of processor). A memory 204, such as a random access memory (RAM), a Flash memory, or other type of memory, is typically accessible to the processor 202. The memory 204 can be adapted to store an operating system 206, as well as application programs 208, such as a mobile location-enabled application that can provide location based services (LBSs) to a user. The processor 202 can be coupled, either directly or via appropriate intermediary hardware, to a display 210 and to one or more input/output (I/O) devices 212, such as a keypad, a touch panel sensor, a microphone, and the like. Similarly, in some embodiments, the processor 202 can be coupled to a transceiver 214 that interfaces with an antenna 216. The transceiver 214 can be configured to both transmit and receive cellular network signals, wireless data signals, or other types of signals via the antenna 216, depending on the nature of the mobile device 200. Further, in some configurations, a global positioning system (GPS) receiver 218 can also make use of the antenna 216 to receive GPS signals.

EXAMPLE MESSAGING SYSTEMS

[0044] In example embodiments disclosed herein, a messaging system utilizes a publication-subscription (also referred to as “pub-sub” herein) process by which producer devices (also referred to as “transmitter device” herein) and consumer devices (also referred to as “receiver devices” herein) connected to a network discover each other through advertisements sent by the consumer devices via a relay agent within the network. As stated, the producer devices can be embodied by the data processing servers 123 of FIG. 1. Additionally or alternatively, the producer devices can be embodied by the client machine 110 of FIG. 1. The consumer devices correspond to one or more servers of the data processing servers 123. A consumer device hosts one or more CEP engines for processing event messages produced by the producer devices. A relay agent can

be a module hosted by the data processing servers 123 for interfacing producer devices and consumer devices during runtime.

[0045] For example, in operations the messaging system can identify a number of consumer devices (e.g., forming a “consumer cluster ring”) available to receive and process messages on a given topic that a producer device generates. The producer device maintains a registry of the consumer devices that have been identified as having subscribed to the topic. As consumer devices are removed or added to the consumer cluster ring, the producer device updates the producer’s registry.

[0046] The producer devices generate and send to consumer devices event messages (also referred to as “event data” herein) that are representative of events (e.g., representative of client-machine-side interactions). An event is a collection of tuples of information. A tuple is made up of a key, such as a set of American Standard Code for Information Interchange (ASCII) characters or other suitable string data type, and a corresponding value, such as a primitive data type. Example primitive types include integer, Booleans, floating point numbers, fixed point numbers, characters and/or strings, data range, and/or the like data types that are built-in the programming language. Events can be classified into types based on matching tuples of information of the events. An event stream is a collection of events received over time. There can be an event stream for each event type. In an example embodiment, the collection of tuples of information are representative of one or more user interactions or user events in connection with the user’s interaction with a web resources, such as a web page or an Internet-connected software program executing on the user’s device.

[0047] The producer device, e.g. using consistent hashing, associates a number of hash values to each of the consumer devices of the consumer cluster. The hash values can be viewed as respective consumer nodes on a circle. As such, the assignment of hash values to consumer devices partitions the identified consumer cluster to form a logical ring of consumer nodes for the given topic. In other words, each consumer device is represented by a number of consumer nodes on the logical ring.

[0048] The hash values can be assigned to a consumer device based on an identifier of the corresponding consumer device. Examples of identifiers of the

consumer device include a globally unique identifier (“GUID”) of the consumer device, an application identifier (“APPID”), a combination of the GUID APPID, IP address, and/or the like. The consumer device can provide the identifier to the producer device by providing the identifier within advertisement data as described in connection with FIG. 7.

[0049] The hash values can be assigned to the consumer device in a pseudo random manner using the identifier as a seed. In an example embodiment, the producer device assigns hash values to a consumer device by executing a pseudo random number generator using the identifier of the consumer device as the seed to generate a number of values. The generated values can be assigned as the hash values linked to the consumer device. Assigning hash values in a pseudo random manner can facilitate distributing the consumer nodes approximately evenly on the logical ring. Moreover, assigning hash values in a pseudo random manner can facilitate approximately even distribution while adding or removing consumer devices, for example, without reassigning hash values to the existing consumer nodes.

[0050] The assignment of hash values to consumer devices can be stored in a registry in the producer devices. During operation, the producer device can determine the mapping of a given hash value to the corresponding consumer device by using a hash function.

[0051] As described later in greater detail, in an example embodiment, each producer device publishing messages on a given topic produces the same logical ring. For example, each producer device publishing on a given topic can have the same consumer devices registering to receive event messages in the given topic. The consumer devices will provide the same identifiers to each of the producer devices. Moreover, each producer device can use same mechanism to assign hash values. As was described above, one example mechanism is to assign the hash values pseudo random manner using the identifiers as seeds. Accordingly, each producer device generates the same assignments between hash values and consumer devices.

[0052] The producer device schedules event messages to the consumer devices of the consumer cluster. For example, the producer device uses a key contained in the event message to generate a partition key to select one consumer

device to receive the event message. In one example embodiment, the producer device computes a hash value of the partition key and matches the computed hash value against the hash values representing the consumer nodes of the consumer devices registered with the producer device. The producer device selects one of the consumer devices to receive the event message based on comparing the distance of the hash of the partition key to the respective consumer nodes. For example, the producer device makes the selection by “walking” around the logical ring in a direction (e.g., clockwise or anti-clockwise), starting at the point of the hash of the partition key, until the first consumer node is reached. The produce device selects the consumer device associated with the first consumer node and routes the event message to the selected consumer device for processing.

[0053] Using such an approach can provide an effective way to redistribute the workload to the consumer device in response to a consumer device failure or workload imbalance. For example, if a node fails, the producer device removes or ignores the consumer nodes associated with the failed consumer device. As stated, the distribution of the consumer nodes of the remaining consumer devices remain approximately evenly distributed when the consumer nodes of a consumer device is removed. This can be true since the event messages that would have been scheduled for the failed consumer device are redistributed to the next nearest consumer node of a function consumer device, while the routing of event messages to the remaining consumer devices remain the same. Likewise, when a consumer device becomes busy, the producer device can automatically rebalance workload to other consumers by removing the consumer nodes of the busy consumer device. Producer devices can discover slow consumer devices and send event messages addressed to the consumer device to an off-heap persistent queue to be replayed (e.g., processed) later, as will be described in greater detail later in connection with FIGS 4 and 9. The statistics produced by the messaging system can be used to elastically scale consumer nodes in the cloud.

[0054] Accordingly, one aspect, among others, of example embodiments is that the messaging system provides a scalable infrastructure with self-healing features to facilitate complex event processing at very high volumes in, for

example, a cloud environment. Furthermore, the messaging system provides that event messages with the same partition key are transmitted to the same consumer device in the cloud, thereby facilitating computing aggregates and for watching for patterns and reacting to those patterns. The messaging system can be deployed in a network cloud or other distributed computing environment, as the messaging system can batch, compress, and enable flow control. The messaging system can elastically scale consumer clusters in real time in response to changes in load and can automatically rebalance traffic in case of network congestion on computation machine failures. As such, example embodiment of the messaging system facilitates deploying the messaging system on a network cloud and facilitating complex event processing.

[0055] It will be understood that example embodiments of the messaging system can be used to in an Internet application tracking pipeline and several other use cases deploying, for example, the JETSTREAM CEP framework. The CEP framework can be used in building distributed applications for user behavior analytics, monitoring, advertising, internet marketing, and risk and security use cases. Example illustrative embodiments are described below.

[0056] FIG. 3 is a block diagram illustrating an example embodiment of a messaging system 300 including multiple modules forming at least a portion of the client-server system of FIG. 1. The modules 302-312 of the illustrated data analysis system 300 include an application interface module(s) 302, a relay agent module(s) 304, a scheduler module(s) 306, a provisioning module(s) 308, a database interface module(s) 310, and a database update module(s) 312. The application interface module(s) 302 includes a consumer-facing sub-module(s) 314, a producer-facing sub-module(s) 316, and a third party-facing sub-module(s) 318.

[0057] In some embodiments, the components of the data analysis system 300 can be included in the data processing servers 123 of FIG. 1. However, it will be appreciated that in alternative embodiments, one or more components of the data analysis system 300 described below can be included, additionally or alternatively, in other devices, such as one or more of the marketplace application 120, the payment application 122, the servers 114, 116, 118, 130, the network 104, and/or the client machines 110, 112 of FIG. 1. It will also be

appreciated that the data analysis system 300 can be deployed in systems other than online marketplaces.

[0058] The modules 302-312 of the data analysis system 300 can be hosted on dedicated or shared server machines (not shown) that are communicatively coupled to enable communications between server machines. One or more of the modules 302-312 can be deployed in one or more datacenters. Each of the modules 302-312 are communicatively coupled (e.g., via appropriate interfaces) to each other and to various data sources, so as to allow information to be passed between the modules 302-312 of the data analysis system 300 or so as to allow the modules 302-312 to share and access common data. The various modules of the data analysis system 300 can furthermore access one or more databases 126 via the database server(s) 124.

[0059] The messaging system 300 can facilitate real-time CEP of large-scale event data over a networked and distributed environment. To this end, the data analysis system 300 illustrated in FIG. 3 includes the relay agent module(s) 304, the scheduler module(s) 306, the provision module(s) 308, the database interface module(s) 310, and the database update module(s) 312.

[0060] The application interface module(s) 302 can be a hardware-implemented module that facilitates communication of data between the messaging system 300 and clients, server, and other devices of the network, such as between the data processing servers 123 and one or more of the marketplace application 120, the payment application 122, the servers 114, 116, 118, 130, the network 104, and/or the client machines 110, 112 of FIG. 1. In particular, the application interface module(s) 302 includes the consumer-facing sub-module(s) 314 for providing an interface with consumer devices. The producer-facing sub-module(s) 316 provides an interface with producer devices. The third-party-facing sub-module(s) 318 provides an interface with a number of third-party servers. In example embodiments, the messaging system 300 can interface with third-party applications 128 that provide web-based services, such as, but not limited to, search services, data storage, data management, data mining, web-activity monitoring and analytics, and like services. The messaging system 300 can receive such services by interacting with, for example, the third party application 128.

[0061] In an example embodiment, the producers and consumer devices use a publication-subscription model. As such, the producer-facing sub-module(s) 314 provides an interface for producer devices (e.g., one or more servers of the data processing servers 123 of FIG. 1) to provide broadcast data for indicating topics to which the producer devices publish. The broadcast data identifies the producer device available to provide data messages linked to the topic

[0062] The consumer-facing sub-module(s) 316 provides an interface for consumer devices (e.g., one or more servers of the data processing servers 123 of FIG. 1) to provide advertisement data for indicating topics to which the consumer devices request to subscribe. The advertisement data identifies that the consumer device is available to receive event messages linked to the indicated topics.

[0063] The relay agent module(s) 304 (also referred to as “identification module(s)” herein) can be a hardware-implemented module that facilitates linking producer devices and consumer devices. The relay agent module(s) can be embodied by one or more servers of the data processing servers 123 of FIG. 1. In operation, the relay agent module(s) 304 receives broadcast data from producer devices via the producer-facing sub-module 314 and advertisement data from consumer devices via the consumer facing sub-module(s) 316. Based on the received data from the producer and consumer devices, the relay agent module(s) 304 can serve to link the producer and the consumer devices that have matching topics.

[0064] The scheduler module(s) 306 can be a hardware-implemented module that facilitates distributing event messages from a producer device to the consumer devices of linked to the producer device. Example scheduling schemes include round robin and consistent hashing, among others. When a producer device provides a broadcast message to register with the relay agent module(s) 304, the scheduler module(s) 306 instantiates a scheduler instance hosted on the producer device. Each instance for a given topic can be the same, and is thus said that “the scheduler is bound to the topic.”

[0065] The provisioning module(s) 308 can be a hardware-implemented module that facilitates scheduling event messages to the consumer devices in a scalable, fault-tolerant manner. The provisioning module(s) 308 links a plurality

of values, such as hash values, to respective consumer devices that are linked to the topic. For example, the provisioning module(s) 308 can generate a plurality of values based on the corresponding consumer device identifier. The generating of the plurality of values can be in response to receiving respective request messages (e.g., advertisement data as will be described in greater in connection with FIG. 7). The plurality of values can be computed per topic. The provisioning module(s) 308 can include a pseudo-random number generator to generate the hash values for use with a consistent hashing scheduler.

[0066] In an example embodiment, the scheduler module(s) 308 determines the hash values linked to the respective consumer devices in a way that is independent of the producer device. In this way, the same hash values are provided to each producer device publishing on a given topic.

[0067] In operation, the scheduler module(s) 306 accesses (e.g., receives or retrieves) from the producer device an event message linked to the topic. The event message includes a key value, and the scheduler module(s) 306 provides the event message to a selected one of the consumer devices based on a comparison of the key and a plurality of values of the respective consumer devices. For example, the scheduler module(s) 306 computes a hash value of the key value and then compares the computed hash value with the hash values of the consumer devices. For example, the scheduler module(s) 306 makes the selection by “walking” around the logical ring in a direction (e.g., clockwise or anti-clockwise), starting at the point of the hash of the key value until the consumer hash value is reached, and the associated consumer device is selected.

[0068] The database interface module(s) 310 can be a hardware-implemented module that facilitates accessing data for the messaging system 300. In an example embodiment, the database interface module(s) 310 can interface with the database 126 of FIG. 1.

[0069] The database update module(s) 312 can be a hardware-implemented module that facilitates updating the databases supporting the messaging system 300. For example, the database update module(s) 312 can facilitate updating databases to aggregate data and to initiate database maintenance activities in response to lifecycle events of sessions (e.g., start and end session events).

[0070] FIG. 4 is a block diagram illustrating an example producer-consumer system 400, in accordance with an example embodiment. The producer consumer system 400 includes a consumer device 402, a producer device 404, a messaging service 406, a natty transport 408, a Zookeeper transport 410, an advisory listener 412, a persistent queue 414, and a replayer 416.

[0071] As stated, the producer device 404 corresponds to a device that is a producer of event messages and can be referred to as a “publisher” in the messaging paradigm. The consumer device 402 corresponds to a device that consumes messages and can be referred to as a “subscriber” of messages.

[0072] In operation, the producer device 404 and the consumer device 402 establish a “contract” which establishes a communication channel between the publisher and subscriber. The address for the channel is referred to as a topic. Topics are strings of the form “id1.kind1/id2.kind2/name.” The ‘/’ is a context separator. A topic is made up of one or more contexts (e.g., “id1.kind1” and “id2.kind2”). Subsequently-listed topics (e.g., id2.kind2) can inherit attributes, such as quality of service characteristics, linked to the root context. The first context is called the root context. In some example embodiments, a topic can have only a root context.

[0073] As an illustrative example embodiment, an example topic can be named “topic = Rtbd.RTD/rtdEvent” that is being subscribed to and published on. For example, the producer device 404 interfaces with the messaging service 406 and invokes an API call to an example function `publish(topic, event)` that is implemented by the producer-facing sub-module(s) 316 of FIG. 3.

Additionally, the consumer device 402 interfaces with the messaging service 406 and invokes an API call to an example function `subscribe(topic)`. The root context in this topic is “Rtbd.RTD”. It is bound to a specific Netty transport instance, such as available from the APACHE SOFTWARE FOUNDATION™, and which will be described in greater detail below. This context can be bound to a consistent hashing scheduler. Accordingly, by publishing on a topic bound to “Rtbd.RTD,” the messaging service 406 uses consistent hashing scheduling.

[0074] The implementation exposes a singleton service interface that can be used by producer devices to publish messages and consumer devices to consume messages. The messaging service 406 has a pluggable transport architecture.

The transport instances can be dynamically plugged into the messaging service 406 at runtime. The messaging service 406 can support at least two types of transports, such as a Netty transport 408 and a Zookeeper transport 410, such as available from the APACHE SOFTWARE FOUNDATION™.

[0075] The messaging service 406 can use the Zookeeper transport 410 to send control messages. Example control messages include discovery, advertisement, and advisory messages. The Zookeeper transport 410 can communicate with a distributed Zookeeper ensemble which acts as a relay agent to pass control messages between producer and consumer message service instances. The Zookeeper transport 410 binds to a topic and controls message flows over this topic.

[0076] The messaging service 406 can use the Netty transport 408 for transporting event messages. The Netty transport 408 can be built on top of Netty non-block input-output (NIO) facility. It provides a completely async transport over TCP. It transports plain old JAVA objects (POJOs) and uses Kryo object serializer for marshalling the JAVA objects. The messaging service 406 can deploy one or more Netty Transport instances.

[0077] In an example embodiment, each instance of the Netty Transport 408 binds to one or more root context of a topic. This binding is provisioned with the rule that no two transport instances bind to the same root context. Each root context is bound to a scheduler module. All topics bound to the context are then bound to the scheduler that is bound to the context. Each transport instance has its own communication properties, which can substantially correspond to TCP properties which can be tuned at runtime. The advisory listener 412 can direct undelivered event messages to the persistent queue 414 to be resent later by the replayer 416. Advisory listener 412 and replayer 416 will be described later in greater detail in connection with FIGS. 8-10.

[0078] FIG. 5 is a block diagram illustrating an example messaging stack 500, in accordance with an example embodiment. The messaging stack 500 includes a producer-side stack 502, which includes a publisher 506, a scheduler 508, an encoder 510, a batcher 512, and a compressor 524. The messaging stack 600 also includes a consumer-side stack 504, which includes a decompressor 526, a decoder 528, and a subscriber 530.

[0079] The producer-side stack 502 can be embodied by a server of the data processing servers 123. The scheduler 508, the encoder 510, the batcher 512, and the compressor 524 can be embodied by the scheduler module(s) 306. The consumer-side stack 504 can be embodied by a server of the data processing servers 123.

[0080] The publisher 506 can correspond to a producer device 404 of FIG. 4. The publisher 506 provides the scheduler event messages to be published to one or more subscribers (e.g., consumer devices).

[0081] The scheduler 508 can correspond to the scheduler module(s) 306 of FIG. 3. The producer-side of the messaging stack is provisioned with the pluggable message scheduler 508. The scheduler 508 can be one of two types: weighted round robin or consistent hashing scheduling algorithms. Schedulers provide quality of service (QOS). The scheduler 508 is bound to a root context of a topic in example embodiments. Accordingly, the publisher can pick QOS by simply picking the topic bound to the root context.

[0082] Weighted round robin: When the weighted round robin scheduler is provisioned, event messages are distributed amongst the discovered consumers using a weighted round robin algorithm. With this approach if weights for all consumers are same, then the messages flowing down the producer side stack are equally distributed amongst the set of consumers bound to that root context. If a weight is assigned to a consumer node then events are distributed to that node taking the weight into account. For example, if the weight is set to 20 for a specific consumer instance, then only 20% of overall messages per second (MPS) is scheduled to that instance and the balance gets equally distributed between the remaining of the instances. Workload distribution can be done per topic.

[0083] Consistent hashing: When a consistent hashing scheduler is provisioned, a consistent hashing ring is created per topic. For example, when a consumer advertisement arrives, a provisioned number of hashes are computed using the identifier of the consumer device. For example, the identifier can be contained in the advertisement. This operation is performed for all discovered consumers. The hashes are distributed across a logical ring. There is one ring for each topic. The algorithm is implemented such that the producer devices

publishing on the same topic have the same view of the ring. When a message arrives at the scheduler, the message is decorated with a key that can be used to create an affinity between the message and a consuming node. The scheduler 508 computes a hash on the key, and places the hashed value on the ring. The scheduler 508 then traverses the ring in an anticlockwise or clockwise direction until the scheduler 508 meets the first hash of a consumer node. There is a mapping of hashes to consuming devices. The scheduler 508 can perform a lookup in the registry to find the consumer device associated with the matched hash. The scheduler 508 then schedules the event message to the consumer node associated with the matched hash.

[0084] A listener (e.g., the listener 412 of FIG. 4) can be plugged into the consistent hashing scheduler so that it can listen to ring change notifications with a view of the new ring. The listener can be used in systems that are sensitive to ring changes. Examples of ring changes include adding a consumer device, removing a consumer device, redefining QOS definitions of the consumer devices, and/or the like. Listeners will be described in greater detail later in connection with FIG. 9.

[0085] The encoder 510 performs an encoding process on the event message. For example, the encoder 510 receives event messages from the scheduler 508, encodes the received event messages, and provides the encoded event messages to the batcher 512.

[0086] The producer-side 502 can be provisioned with the batcher 512 to batch messages per context. Topics under the selected context can be batched. The batch size is also provisioned and this can be changed at runtime. The batcher 512 in the stack accumulates event messages delivered from upstream. Event messages are scheduled downstream when either the batch size is reached or a timer expires. The timer provides a mechanism to inhibit substantial delays by the batcher 512 caused by a low rate of received event messages. It will be appreciated that the batcher 512 can be omitted in alternative embodiments. In an example embodiment, whether or not the batcher 512 is included can be determined during provisioning of the .

[0087] The compressor 524 can compress the event messages. Compression is driven through advertisements from the consumer (e.g., the subscriber 530).

When the consumer signals to the producer that it expects the messages in a compressed form, the producer can compress the messages. Both batched and non-batched event messages can be compressed. In one example, SNAPPY compression is supported.

[0088] On the consumer-side of the stack 504, the decompressor 526 and the decoder 528 reverses the process of the compressor 526 and the encoder 510.

[0089] FIG. 6 is a block diagram illustrating an example producer-agent-consumer system 600, in accordance with an example embodiment. The producer-agent-consumer system 600 includes a producer device 600, a relay agent module(s) 604, and a consumer device 606. . FIG. 6 shows the interactions of the producer device 602, the relay agent module(s) 604, and the consumer device 606 during the publication-subscription process, and for the sake of brevity additional components of the messaging system architecture are not shown. An example embodiment of the messaging system architecture is described in greater detail in connection with FIG. 8.

[0090] In operation, the producer device 602 and consumer device 606 can start up out of order. Both the producer device 602 and the consumer device 606 register with the relay agent module(s) 604. The relay agent module(s) 604 can be deployed as a cluster across datacenters. A group of the relay agents (not shown) can be configured to be an active ensemble. The remainder of the group is designated as observers. The relay agent module(s) 604 can be used as a message router/distributor. The producer devices and consumer devices publish messages through the relay agent module(s) 604 using a topic based address as previously described.

[0091] If a consumer device 606 initiates registration, the consumer device 606 posts an advertisement through the relay agent module(s) 604 to all interested producers at interaction 610. The relay agent module(s) 604, in response, posts the advertisement with the producer device 602 at interaction 612. In response, the producer device 602 can build a registry containing all known consumer devices that has registered with the producer device 602.

[0092] In some embodiments, the consumer device 606 can provide to the producer 602 a number of advisors through the relay agent module(s) 604. For example, the consumer device 606 can provide advisories to indicate a state of

the consumer device 606, such as the consumer device 606 is processing event messages slowly, lacks resources to process event messages, has a surplus of resources for its current workload, is requesting reinstating workload, and/or like conditions that indicate reducing or increasing the workload to the consumer device.

[0093] As an illustration, in operation the consumer device 606 can provide the relay agent module(s) 604 the advisory message at interaction 614. The advisory message can include a consumer device identifier, a topic identifier, and an advisory identifier that is indicative of the type of advisory (e.g., usable by the producer device 602 and/or the relay agent module(s) 604 to determine to increase or decrease workload). After receiving the advisory message, the relay agent module(s) 604 can route the advisory message to the producer device 602 and other producer devices linked to the topic at interaction 616. In turn, the producer device 602 can update its registry of consumer identifiers based on the advisory. For example, the producer device 602 can remove or disable the consumer identifier from its registry so that the consumer device 606 is no longer available to receive event messages for processing.

[0094] In further operation, if the consumer device detects that the consumer device is available to process event messages, the consumer device 606 can send another advisory to the relay agent module(s) 604 to indicate to the produce device 602 that the consumer device 606 is available for receiving event messages.

[0095] When a producer device 602 initiates registration, the producer device 602 sends out a discover message through the relay agent module(s) 604 at interaction 618. The discovery message is broadcasted to all interested consumer devices, such as consumer device 606, at interaction 620. The consumer device 606, in response to receiving the discover message, can respond with an advertisement, such as was described above in connection with interactions 610, 612. The advertisement message is also sent through the relay agent module(s) 604.

[0096] FIG. 7 is a block diagram illustrating an example data structure of an advertisement 700, in accordance with an example embodiment. The advertisement 700 travels from consumer device to producer device, as

described above in connection with FIG. 6. The consumer device generates a unique consumer identifier data field 702 when it starts up. The advertisement 700 comprises an address data field 704, which can correspond to the consumers IP address and/or port binding. The advertisement 700 also includes a QOS data field 706 that can be indicative of any suitable quality of service characteristic, such as compression, serialization format, data rate, and the like. The advertisement 700 can also include a timestamp data field 708 that is indicative of when the consumer device posted the advertisement 700. The advertisement 700 can also include, a weight data field 710 that is indicative of a relative workload requested by the consumer device. The advertisement 700 can also include a topic data field 712 that is indicative of the topics to which the consumer device subscribing. When producer device receives the advertisement, the producer device can update its consumer registry with the advertisement 700.

[0097] FIG. 8 is a block diagram illustrating an example messaging system architecture 802, in accordance with an example embodiment. The messaging system architecture 802 can be deployed across datacenters. For example, a first datacenter can include a producer device 802, a relay agent module(s) 804, a replayer 806, and a queue 808, such as a Kafka queue. A second datacenter can include a portion of a cluster ring 810 comprising the consumer devices 812A-C, a relay agent module(s) 814, a queue 818, such as a Kafka queue, and a replayer 822. A third datacenter can include the remaining portion of the cluster ring 810 comprising the consumer devices 812D-F, the relay agent module(s) 816, a queue 820, such as a Kafka queue, and a replayer 824. The producer device 802 passes event messages to the cluster ring 810 through a scheduler module(s) 826 that determines the routing of the event messages within the cluster ring 810. Similarly, the replayer 806 provides event messages to the cluster ring 810 through the scheduler module(s) 826.

[0098] The relay agent 804,, 814, 816 correspond to instances of the relay agent module(s) 304 of FIG. 3. The scheduler module(s) 826 can correspond to the scheduler module(s) 306 of FIG. 3.

[0099] The relay agent modules 804, 814, 816 are deployed across the datacenters. Some of the relay agent modules in one of the datacenters are designated as active ensemble (e.g., relay agent modules 804, 814, 816) and the

remaining as observers (not shown). The messaging stack in the producing device 802 and consuming devices 812A-F register with an available relay agent module. From then on, the producing device 802 and consuming devices 812A-F communicate with the same relay agent module until the connection is broken to that relay agent.

[00100] The producer device 802 sends discovery messages through its relay agent 804, and the consumer devices 812A-F send advertisement messages through their relay agents 814, 816. A discovery message can include a topic identifier and an identifier of the producer device 802. The discovery and advertisement messages can be sent and received across datacenters. Accordingly, producer devices can discover consumer devices across datacenters. Furthermore, the scheduler module(s) 826, utilizing a consistent hashing arrangement, can facilitate routing event messages to cluster ring 810 having consumer devices that span across multiple datacenters.

[00101] FIG. 9 is a block diagram illustrating an example monitoring system 900 deployed by the messaging system, in accordance with an example embodiment. The monitoring system 900 includes a messaging service 902, which includes a Netty transport 904 containing virtual queues 906A-C and a virtual queue monitor 908. The messaging service 902 interfaces with one or more consumer devices 910A-C, an alert listener 912, and an advisory listener 914 connected to a queue 916.

[00102] In one aspect, the monitoring system 900 can support self-healing clusters. For example, when a consumer device 910A-C fails or becomes busy, the traffic being scheduled to that consumer device should be redirected to another consumer device. This reconfiguration can be achieved through the messaging system described herein. In particular, the producer side of the messaging stack can sense when a consumer device has failed. In such a scenario, the failed consumer device can be removed from the producer device's registry and no more traffic is scheduled to that failed consumer device. Instead, the traffic is rebalanced across the other instances in the cluster ring which are still operational.

[00103] The producer side of the messaging stack maintains a virtual queue 906A-C per consumer device 910A-C in its registry, as shown in FIG. 9. The

virtual queue monitor 908 monitors the virtual queues 906A-C to detect slowness of the consumer devices 910A-C. This typically happens when the network between the producer device and a consumer device becomes slow or when a consumer device operates slowly and cannot empty its communication socket buffers. In such a case, the messaging service 902 emits advisory messages to the advisory listener 914 or other listener components that are subscribing to advisory messages. The advisory message contains the original message with the topic over which it was published. The advisory listener 914 takes the event message, adds metadata and pushes the event to a Kafka queue 916. A replayer device, such as the replayers 822, 824 of FIG. 8, is setup to listen to the queue 916 and replay the message directly to the consumer device.

[00104] The producer side can also be provisioned with the alert listener 912. When the virtual queue monitor 908 detects a slow consumer device, the virtual queue monitor 908 will signal the alert listener 912. For example, the virtual queue monitor 908 can provide the alert listener 912 information about the network address of the slow consumer device along with the topic. This can be used as a signal to detect that corresponding producer device. An example method of reconfiguring a consumer cluster ring is described below in connection with FIG. 10.

EXAMPLE METHODS OF MESSAGING SYSTEMS

[00105] FIG. 10 is a schematic diagram depicting an example embodiment of interactions 1000 of producers and consumers for reconfiguring a consumer cluster, according to an example embodiment.

[00106] At interaction lines 1002-1004, the producer-side stack 502 transmits event messages to the consumer-side stack 504. At interaction line 1006, the consumer-side stack 504 monitors upstream queue depth to detect slowness of the consumer application. At interaction line 1008, the consumer-side stack 504 senses that the upstream queue in the consumer messaging stack has built up beyond a first threshold value, and at interaction line 1010 it sends advisories to all producer devices to stop sending messages to the consumer side stack 504. At interaction line 1012, the producer-side stack 502 reacts to the

advisory message by rebalancing traffic destined to this consumer instance and distributing this traffic across the cluster ring.

[00107] At interaction line 1014, the consumer-side stack 504 detects that the upstream queue has dropped below a second threshold value, and the consumer-side stack 504 sends another advisory message to producers to start sending messages to the consumer-side stack 504 again. In an example embodiment, the first and second threshold values can be different values. At interaction line 1020, the producer-side stack 502 resumes transmission of the event messages to the consumer-side stack 504.

[00108] FIG. 11 is a flowchart illustrating an example method 1100 of cluster messaging, in accordance with an example embodiment. In this example, the method 1100 can include operations such as receive first data identifying consumer devices available to receive data messages (block 1104), receive second data identifying a transmitter machine available to provide data messages (block 1106), link a plurality of values to respective consumer devices (block 1108), access from the transmitter machine a first data message (block 1110), and provide the first data message to a selected one the consumer devices (block 1112). The example method 1100 will be described below, by way of explanation, as being performed by certain modules. It will be appreciated, however, that the operations of the example method 1100 can be performed in any suitable order by any number of the modules shown in FIG. 3.

[00109] In an example embodiment, the method 1100 starts at block 1102 and proceeds to block 1104, at which the relay agent module(s) 304 receives first data identifying consumer devices available to receive data messages linked to a topic. The first data can correspond to one or more advertisements sent by consumer devices.

[00110] At block 1106, the relay agent module(s) 304 receives second data identifying a producer device (or “transmitter device”) that is available to provide data messages linked to the topic. For example, the relay agent module(s) 304 can receive a discovery message from a producer device that indicates that producer device is publishing event messages on an identified topic.

[00111] At block 1108, the provisioning module(s) 308 links a plurality of values to respective consumer devices. For example, the values can be hash values generated by a consistent hashing scheduler. For example, an identifier of a consumer device can be used as a seed in a pseudo-random number generator to generate a number of values that will be assigned to the consumer device as its hash values. Thus, a number of hash values can be linked to each consumer device. The plurality of values can be computed for the topic.

[00112] At block 1110, the scheduler module(s) 306 can access from the producer device a first event message linked to the topic. For example, the scheduler module(s) 306 can retrieve the first event message from data memory (e.g., when implemented in the producer device) or receive it via a data communication channel from the producer device (e.g., when implemented in a device external to the producer device). The first event message includes a key value. At block 1112, the scheduler module(s) 306 provides the first event message to a selected one of the consumer devices based on a comparison of the key and the plurality of values of the respective consumer devices. As stated, the comparison can include evaluating the key using a hash function and then comparing the resulting hash value with the values linked to the consumer devices. The comparison can be made on the perimeter of a logical ring finding the closest match in a clockwise or anti-clockwise direction. At block 1114, the method 1100 can end.

EXAMPLE SESSIONIZATION SYSTEMS

[00113] In one aspect, among others, example embodiments disclosed herein provide a system for facilitating sessionization of network data, such as for real-time analytics of Internet and mobile applications, and also for monitoring, security, Internet bot detection, and the like applications. Sessionization is a process of grouping events containing a specific identifier and that occur during a time window referred to as session duration. A visit or session is defined as a series of page requests, image requests, and/or the like web resource requests from the same uniquely identified client. Each time window starts when an event is first detected with the unique identifier. The time window terminates when no events have arrived with that specific identifier for the specified

duration. For analytics of web applications, sessionization can define the session duration as 30 minutes of inactivity. However, for analytics of mobile device flows and other applications, sessionization can use different session duration.

[00114] Sessionization facilitates storing selected fields extracted from the event stream and also computing and storing aggregated counts of fields or events over the lifetime of the session as events flow through the network. Example embodiments disclosed herein are described in the context of sessionizing user behavior data. It will be appreciated that machine behavior can be sessionized in alternative embodiments, as well as other types of data suitable sessionization.

[00115] The sessionizing system described herein comprise multi-stage distributed-stages pipelined together. The first stage is a collection tier which ingests events over multiple interfaces from different sources. The sessionizer is the second stage of the pipeline. In one aspect, among others, the sessionizer system creates and sustains sessionized data per tenant (or referred to as “subscriber”) and produces lifecycle events as the session transitions through its start and end state. A tenant is a client device requesting sessionized data.

[00116] To achieve a fault-tolerant distributed environment operating across multiple datacenters, the sessionizing system uses the messaging system 300 of FIG. 3 to maintain and manipulate the state of a session. Furthermore, the sessionization system can create hierarchical sessions where one session spans multiple sub sessions and/or multiple channels.

[00117] FIG. 12 is a block diagram illustrating an example sessionizer system architecture 1200, in accordance with an example embodiment. The sessionizer system architecture 1200 includes an application(s) 1202 that transmits event messages by a first messaging system 1204 to a collector cluster ring 1206 that includes one or more CEP engines 1208A-F. The collector cluster ring 1206 is interconnected with Zookeeper transports 1210, 1212 and queues 1214, 1216. Additionally, the collector cluster ring 1206 connected to a second messaging system 1224 that routes and passes event messages to a sessionization cluster ring 1226 formed by one or more consumer devices 1228A-F. Additionally, the sessionization cluster ring 1226 is interconnected with Zookeeper transports

1230, 1232 and data queues 1234, 1236. Furthermore, the sessionization clustering 1226 can be interconnected with a subscriber(s) 1240.

[00118] The application(s) 1202 can correspond to web resources executed on user devices and can serve as producer devices for the collector cluster ring 1206. The messaging system 1204 can route event messages from the application(s) 1202 to the collector cluster ring 1206 using a round-robin scheduler module. Event messages are then routed by the second messaging system 1224 to the sessionization cluster ring 1226 for processing to generate sessionized data. The sessionization clustering 1226 provides the sessionized data to a corresponding subscriber(s) 1240, which is a device(s) that requests particular sessionized data.

[00119] The collector cluster ring 1206, the CEP engines 1208A-F, the Zookeeper transports 1210, 1212 and the queues 1214, 1216 for a first tier of the sessionizer system 1200 can be referred to as the collector tier. The collector tier can be deployed using multiple datacenters. For example, a first datacenter can deploy the CEP engines 1208A-C, the Zookeeper transport 1210, and the data queue 1214. A second datacenter can deploy the CEP engines 1208D-F, the Zookeeper transport 1212, and the data queue 1216.

[00120] The collector tier receives event messages over multiple interfaces from different 1202 sources, such as the application(s) 1202, and schedules the event messages to a second tier, referred to as the sessionizer tier, through the second messaging system 1224. Accordingly, the CEP engines 1208A-F serve as consumer devices with respect to the application(s) 1202 and serve as producer devices with respect to the sessionization tier 1226. In an example embodiment, the second messaging system 1224 can correspond to the messaging system 300 of FIG. 3.

[00121] The sessionizer tier comprises the sessionization cluster ring 1226, the consumer devices 1228A-F, the Zookeeper transports 1230, 1232, and the data queues 1234, 1236. As such, the sessionizer tier can be deployed as a cluster ring that encompasses multiple datacenters. For example, the first datacenter can deploy the consumer devices 1228A-C, the Zookeeper transport 1230, and the data queue 1234, and a second datacenter can deploy the consumer devices 1228D-F, the Zookeeper transport 1232, and the data queue 1236. To provide

messaging across multiple datacenters, the Zookeepers 1210, 1212, 1230, 1232 can embody relay agent module(s) 304.

[00122] The subscriber(s) 1240 provides to the sessionization cluster ring 1226 a request message to receive sessionized data. The subscriber(s) can be referred to as a “tenant.” The sessionization cluster ring 1226 performs sessionization on a per-tenant basis. In other words, the sessionization cluster ring 1226 can generate sessionized data specifically for a particular tenant.

[00123] Accordingly, each tenant-specific session has a unique identifier derived from one or more tuples in the event stream. The collector tier and the second messaging system 1224 can partition the sessionization cluster ring 1226 based on a key included in the event messages. For example, the key can correspond to a globally unique identifier (“GUID”) of the event messages. A GUID is unique to the device that is the source of the event messages (e.g., the user device browsing a website). It will be appreciated that other data of the event messages can be used as the key in alternative example embodiments. For example, an application identifier (“APPID”) and the GUID can be combined (e.g., concatenated or joined) to form the session identifier.

[00124] As stated, the collector tier can receive event messages over multiple interfaces. In an example embodiment, the collector tier additionally normalizes the received event messages prior to sending the event message downstream. The CEP engines 1208A-F hosted in the collector tier can filter out Internet bot traffic. For example, the CEP engines 1208A-F can look up Internet bot signature (e.g., patterns of events) and mark the event messages that match an Internet bot signature as a “BOTSignatureEvent” type. The event message can be marked by adding metadata indication the “BOTSignatureEvent” type. After filtering, the event message stream is then scheduled for the sessionizer tier by using the key within the event message, such as by using the GUID of the event message, as will be described in greater detail in connection with FIG. 14.

[00125] The sessionizing system can facilitate tenants defining the session duration specific to their sessions. For example, session duration can be defined as 30 minutes of inactivity. In alternative examples, session duration can be shorter (e.g., 5-10 minutes) or longer (e.g., 30-60 minutes).

[00126] FIG. 13 is a block diagram illustrating example embodiment of a sessionizer system 1300 including multiple modules. The illustrated sessionizer system 1300 includes a counter updater module(s) 1302, an event decorator module(s) 1304, a bot detector module(s) 1306, a session updater module(s) 1308, and a metadata updater module(s) 1310.

[00127] In some embodiments, the components of the sessionizer system 1300 can be included in the data processing servers 123 of FIG. 1 and/or the cluster ring 1226 of the sessionizer ring . However, it will be appreciated that in alternative embodiments, one or more components of the sessionizer system 1300 described below can be included, additionally or alternatively, in other devices, such as one or more of the marketplace application 120, the payment application 122, the servers 114, 116, 118, 130, the network 104, and/or the client machines 110, 112 of FIG. 1. It will also be appreciated that the sessionizer system 1300 can be deployed in other machines interconnected with the network 104 and in systems other than online marketplaces.

[00128] The modules 1302-1310 of the sessionizer system 1300 can be hosted on dedicated or shared server machines (not shown) that are communicatively coupled to enable communications between server machines. One or more of the modules 1302-1310 can be deployed in one or more datacenters. Each of the modules 1302-1310 are communicatively coupled (e.g., via appropriate interfaces) to each other and to various data sources, so as to allow information to be passed between the modules 1302-1310 of the sessionizer system 1300 or so as to allow the modules 1302-1310 to share and access common data. The various modules of the sessionizer system 1300 can furthermore access one or more databases 126 via the database server(s) 124.

[00129] The counter updater module(s) 1302 can be a hardware-implemented module that facilitates the counting of the occurrence of user-defined fields in received event messages and/or count the events represented by the event messages. In operation as event messages arrive, the counter updater module(s) 1302 maintains a count of the occurrence of user defined fields in those event messages or counts the events. These counters are maintained in session data storage.

[00130] The event decorator module(s) 1304 can be a hardware-implemented module that facilitates combining sources of information external to the event message with the event messages. For example, other sources of data with valuable information can be combined with an event stream, such as for example, a user behavior stream. Examples of data that can be added with the event message data includes geographical information, device classification, demographics, and segment data.

[00131] In an example embodiment, the event decorator 1304 combines supplemental data with the event message streams in real-time as the event messages flow through the sessionizer system 1300. For example, the event decorator 1304 looks up a data store using one of the attributes of the event message as keys. Caching data can be used locally on the processing node or externally in a fast lookup cache. Adding the supplemental data to event message streams in real-time facilitates scalable data stores that can be queried at the rates experienced in large-scale systems.

[00132] The bot detection module(s) 1306 can be a hardware-implemented module that facilitates processing event messages generated by Internet bot programs.

[00133] The bot detection module(s) 1306 can identify in real-time event messages that correspond to activities of Internet bot programs. Although an application tier that is upstream of the bot detection module(s) 1306 can look up bot signatures in the producing applications (e.g., signatures of self-declared bots and those detected during offline processing can be stored in a cache for looked up), the application tier may fail to identify some Internet bot activities. The bot detection module(s) 1306 detects Internet bot program activities by observing the rates at which the Internet bot programs are accessing the site using signatures. The bot detection module(s) 1306 uses probabilistic frequency estimation techniques measured over rolling windows of time. The bot detection module(s) 1306 can serve to reduce the processing resources consumed by event messages generated by Internet bot programs. As the sessionization system 1300 detects bot signatures, the sessionization system 1300 updates the bot signature cache. This cache can be provided by the collector tier to enforce bot filtering.

[00134] The session updater module(s) 1308 can be a hardware-implemented module that facilitates updating session information. For example, the session updater module(s) 1308 can post a session start marker event to the corresponding session when a session is created and a session end marker event to the corresponding session when a session ends. The session start and end marker events contain metadata derived from event streams and geographical enriched data. The sessionizer system 1300 can support cross-channel (e.g., across multiple devices) sessions referred to as super sessions. The session identifier of a super session is the user identifier.

[00135] The metadata updater module(s) 1310 can be a hardware-implemented module that facilitates extracting data from streams of event messages. For example, the metadata updater module(s) 1310 can extract data from the event messages of a session and store the extracted data in a session record in the form of session metadata. Examples of session metadata include session identifier data, a page identifier data, geographical location data (e.g., city, region, country, continent, longitude, latitude, Internet service provider), browser type data, OS type data, and device type data.

[00136] In response to receiving a message event, the sessionizer system 1300 looks up the cache to determine if a key for the session exists. For example, the event message has metadata indicating the key to use for cache lookup. The event message is evaluated by the consumer device of the sessionization tier and, for respective tenants, metadata is extracted and updated into the respective tenant's session if the session exists (e.g., the key for the session exists in the cache). If session does not exist (e.g., the key for the session does not exist in the cache) a new session instance is created and a session key is minted and injected into the session. The metadata extracted from the event is then updated into the session record.

[00137] The sessionizer system 1300 can implement an interface for providing sessionization capabilities. In an example embodiment, the sessionizer system 1300 implements a number of annotations extending the EPL (Esper's event processing language). Example annotations are listed below:

TABLE 1: ANNOTATION LISTING

@BotSession – Hint for create/load bot session

```
@BotSession("IP")
select 'IP/' || ip as _pk_, ip
from BotFeedbackEvent(category='IP' and ip is not null);
```

Create/Load a bot session and use the IP address as the bot signature.

@UpdateCounter – Increase bot session counter.

```
@UpdateCounter(name="bounceCount", category="IP")
select * from SessionEndEvent(eventCount = 1);
```

Increase the session counter bounceCount.

@PublishBotSignature – Detect a bot signature and publish it

```
@PublishBotSignature("IP")
select 123 as _bottype_
from SessionEndEvent(ipSession.counter("bounceCount") > 50);
```

Publish the bot signature.

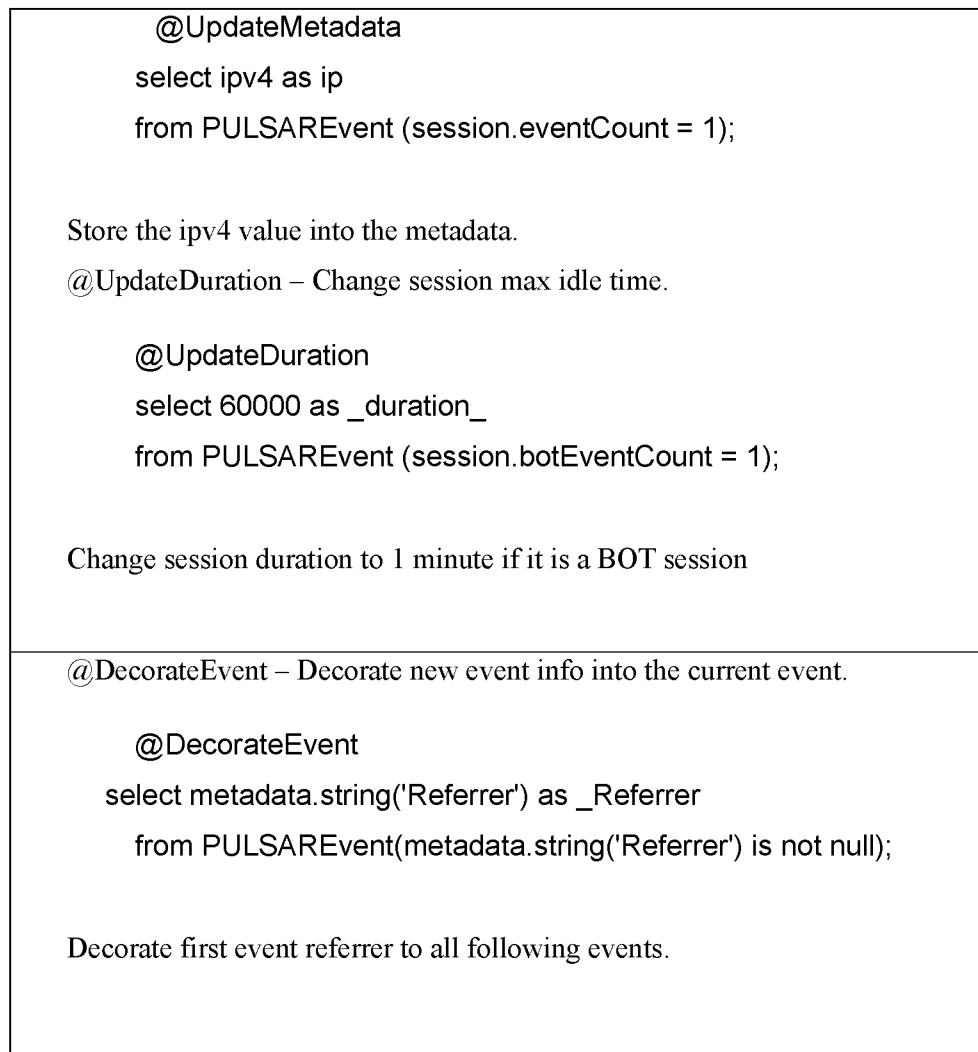
@DebugSession – Used for debugging

@Session –Provides hint for sessionizer to create/load session

```
@Session("SOJMainSession")
select si as _pk_, _ct as _timestamp_, __sessionTTL as
_duration_
from PULSAREvent(si is not null and _ct is not null);
```

The statement returns _pk_ as the identifier of the session. The statement can also return optional _timestamp_ as the event timestamp and _duration_ as the session max inactivity time.

@SubSession – This annotation is used to provide hint for sessionizer to create/load sub session <pre> @SubSession("AppSession") select app as _pk_ from PULSAREvent(p is not null); </pre> The statement returns <code>_pk_</code> as the identifier of the sub session and also can return an optional <code>_duration_</code> as the sub session max inactivity time.
@UpdateState – Save state to session <pre> @UpdateState select p as page from PULSAREvent; </pre> Store the value of p tag into session variable page.
@UpdateCounter – Create/Increase session counter <pre> @UpdateCounter("HomePageView") select * from PULSAREvent(pageGroup = 'HomePage'); </pre> Increase the session counter HomePageView if the pageGroup is HomePage.
@AppendState – Append data into a list variable <pre> @AppendState(name="pageList ", colname="page", unique="true") select p as page from PULSAREvent; </pre> Append the current page to session pageList variable.
@UpdateMetadata – Save to session metadata



[00138] FIG. 14 is a block diagram illustrating an example sessionizer architecture 1400, in accordance with an example embodiment. The sessionizer architecture 1400 can correspond to the sessionizer tier described above in connection with FIG. 12. The sessionizer architecture 1400 includes an inbound message channel (IMC) 1402, a dispatcher 1404, a sessionizer 1406, an outbound message channel (OMC) 1408, a bot detector 1410, and a bot OMC 1412. The sessionizer 1406 can implement the counter update module(s) 1302, the event decorator module(s) 1304, the session updater module(s) 1308, and the metadata updater module(s) 1310.

[00139] In an example embodiment, the sessionizer architecture 1400 is implemented using a JETSTREAM container. The JETSTREAM container

provides a mechanism to build dynamic pipelines declaratively that can be changed at run time. The pipeline of the sessionizer architecture 1400 can be hosted in a JETSTREAM Application container and is wired as shown in the illustrated embodiment of FIG. 14.

[00140] The IMC 1402 serves as a first stage of the pipeline that receive event messages from the collector tier of FIG. 12. The event messages arriving at the IMC are then forwarded to the dispatcher 1404. In the example embodiment, the dispatcher 1404 corresponds to an Esper CEP engine that evaluates the event messages and makes routing decisions. Event messages marked as type BOTSignatureEvent are forwarded to the bot detector 1410, which processes the event message and then provides the processed data to subscribers interested in receiving events which contain metrics for different bot types.

[00141] In response a determination that the received event message marked as an event type that does not require sessionization, the dispatcher 1404 forwards the received event message to the OMC 1408, thereby bypassing the sessionizer 1406.

[00142] Accordingly, the dispatcher 1404 passes to the sessionizer 1406 event messages that have bot activities filtered out. An example embodiment, the sessionizer 1406 is another CEP engine instance that manages session lifecycle and provides a fourth generation language (4GL) programmatic interface which allows extraction of data from event and update of sessions. In an example embodiment, the sessionizer 1406 is implemented using Esper's EPL. Additionally or alternatively, the sessionizer 1406 can be implementing using the annotation listed in Table 1.

[00143] Additionally or alternatively, the sessionizer 1406 creates new sessions for the specified combination of tuples of information contained in the incoming event message. The sessionizer architecture provides users an interface for writing user-defined rules for enforcing tenancy-based sessionization in structured query language (SQL). An example for achieving this using SQL is shown below:

[00144] INSERT INTO EWASESSION SELECT appid, guid, 30 AS sessionduration, * FROM TRACKING_EVENT WHERE appid is not null;

[00145] @OutputTo("sessionizer") SELECT * FROM SESSIONINFO;

[00146] In this example, the SQL instructions define that the APPID and GUID form the session identifiers and that 30 minutes as the session duration.

[00147] Providing users the ability to define rules can be met by providing a 4GL programming construct so users can implement these rules in 4GL and submit the rules. For example, SQL can be adopted as a 4GL programming construct to implement and submit rules using POWERBUILDER™, STATISTICAL ANALYSIS SYSTEM™ (SAS), STATISTICAL PACKAGE FOR SOCIAL SCIENCES™ (SPSS), STATA™, and/or the like 4GL programs. The JETSTREAM framework, which has an embedding the JETSTREAM framework in the CEP engines, can allow the sessionizer to create new SQL rules and apply it on the fly.

[00148] In example embodiments, the sessionizer architecture 1400 can track and generate "super sessions" that span across multiple channels (for example, one session for a user using multiple screens and devices). A super session is identified by an identifier referred to as "ActorId." Events can have a unique identifier for the session referred to as a session GUID. As the user changes from one channel to another channel, the user receives multiple session identifiers (e.g., multiple GUIDs). The user's identity can be recognized during the user's interaction with the system, and the ActorId is included into the event messages as an indicator of the user.

[00149] The sessionizer 1406 can detect that an event message includes an ActorId. If the sessionizer 1406 detects an ActorId in the event message, the sessionizer 1406 forwards the event message back into the sessionizer cluster ring 1226 over a different topic specifying the ActorId as the affinity key. The event is marked as being replayed to process ActorId. The sessionizer 1406 now creates a new session for the ActorId. With this approach, aggregates attributed to the channel session are now also attributed to the super session. Accordingly, if a mobile event message is received, the event message will be attributed to the mobile session and the super session. The super session can have a different life cycle compared to the sessions that are linked to it. The linked session and the super session can also have aggregates.

[00150] Additionally or alternatively, the sessionizer architecture 1400 can

also partition the session itself into multiple sub-sessions. In such a case, the sessionizer architecture 1400 creates a sub-session per partition, which allows the sessionizer architecture 1400 to maintain aggregates and metadata per sub-session. The life cycle of the sub sessions is encapsulated within the lifecycle of the parent session. So when parent session expires, the sub-sessions expire too. Aggregate and metadata updates in the sub-session can be performed in EPL. For example, subsessions can facilitate experimentation by creating subsessions for respective experiment trials as a way to measure trial results and to attribute effect of independent variables of the experiment.

[00151] Additionally or alternatively, the sessionizer architecture 1400 can track and manage session lifecycle events, such as session start and session end events. For example, when an event message arrives at the sessionizer architecture 1400, the sessionizer 1406 passes the event message through the dispatcher 1404 to determine where to forward the event message. If the event is to be sessionized, the dispatcher 1404 forwards the event message to the sessionizer processor 1406. As stated, the sessionizer 1406 is a CEP engine which has a set of rules written in SQL per tenant. The event message is evaluated by the CEP engine and, for each tenant, metadata is extracted and updated into the tenant's session if the corresponding session exists. If session does not exist, a new session instance is created and a session key is minted and injected into the session. The metadata extracted from the event is then updated into the session record. The newly created session is also updated with geographic and demographic information. A lifecycle event called "session start marker event" is generated to signal the start of a new session. This event contains session start time and all the data accumulated in the session at creation time. This event is forwarded to interested subscribers.

[00152] As more events arrive into the session, the aggregates are updated in the session. The event itself is decorated with geographic and demographic information along with the session key and sent to interested subscribers.

[00153] Session records can be stored in a cache, such as an off heap cache that can work on a very large memory region outside the JAVA heap. This cache is backed by a replicated distributed backing store deployed across multiple datacenters. The cache component has a runner that monitors the

activity for each record in the cache. When a session record is updated in the cache, its last update time stamp is recorded along with an expiry time, which can be calculated in the process. The runner sweeps the entire cache every few seconds. When it encounters that a session record that has expired, it removes it from cache and generates a life cycle event called “session end marker event.” This event contains the session key, the data recorded in the session along with the aggregate counts, and session start and end times.

[00154] Subscribers can subscribe to session life cycle and raw events.

[00155] Additionally or alternatively, the sessionizer architecture 1400 can facilitate dynamic scaling and fault accommodations. For example, the consumer devices 1228A-F (also referred to as “sessionizer nodes”) of FIG. 12 can be automatically discovered by the CEP engines 1204 (e.g., the producer devices of the collector cluster ring 1206). The sessionizer cluster ring 1226 can grow to hundreds of nodes, and as new nodes are added to the cluster, traffic automatically rebalances. When a node in the sessionizer cluster ring 1226 fails or a new node is added to the ring, traffic is rebalanced so that all the events flowing to that particular sessionizer node is now scheduled to other nodes in the cluster ring. As traffic enters other nodes, the session state associated with that event is restored from the distributed cache.

[00156] The cluster ring facilitates disaster recovery. An event is generated in response to detecting that a change due to node failure or addition of a new node. A listener can listen to this event. The listener then queries the distributed cache for sessionizer identifiers (also referred to as “keys” herein) that were inserted from the failed node. The sessionizer identifiers that have expired (e.g., the most recent event occurred after the duration window of the session elapsed) are then processed and closed out. As part of the process of closing out the expired sessions, a life cycle event called “session end marker event” is generated.

[00157] When a new session is created a session key is minted and bound to the session, and the binding is stored in a cache. In an example embodiment, an off-heap cache technology with a distributed backing store is used. This type of design allows recovery from failures and restore state. An off-heap cache can be used that has a backing store in a server farm to store the session data.

[00158] When an event message is received, the cache is checked to see if the

key exists. The event message has metadata indicating the key to use for cache lookup. If the key is not found, the backing store is checked. If the key is found in the backing store, the session is restored from cache, the count is incremented, and the backing store is updated.

[00159] When a sessionizer node dies, the session that were supposed to expire on the node should be closed. A record of these sessions are stored in the distributed cache. The sessionizer architecture 1400 maintains a segment in the cache that contains a set of keys that were written to the cache from the sessionizer nodes accumulated over a window (e.g., 200 milliseconds). In this way, a record is kept of the keys written from a specific node. The key for each of these bucket entries is created by combining the time segment and hostId, which usable to identify and/or address the device that hosts the sessionizer node. Each sessionizer node has a listener that listens to ring changes. When there is failure detected on one host, other hosts have a leader that reads the keys and their corresponding sessions from the expired node and closes them out. If the session is still valid, the session is kept; otherwise, the “session end marker event” is sent out.

[00160] FIG. 15 is a flowchart illustrating an example method 1500 of generating the sessionized data, in accordance with an example embodiment. In this example, the method 1500 can include operations such as receiving data indicative of a subscription request for sessionized data (block 1504), allocating a sessionizer bank linked to the subscription request (block 1506), provisioning identifier linked to the respective processing engines of the sessionizer bank (block 1508), registering the allocated sessionizer bank as available to process event messages (block 1510), receiving event messages (block 1512), processing the received event messages (block 1514), and providing generated sessionized data (block 1516). The example method 1500 will be described below, by way of explanation, as being performed by certain modules. It will be appreciated, however, that the operations of the example method 1500 can be performed in any suitable order by any number of the modules shown in FIGS. 3 and 13.

[00161] The method 1500 starts at block 1502 and proceeds to block 1504, at which a subscription interface receives data indicative of a subscription request for sessionized data. The subscription request can include a subscriber identifier,

the sessions of interest, and the like. The subscription request can be transmitted by the subscriber(s) 1240 of FIG. 12. The subscription interface can be implemented at the sessionizer cluster ring 1226 of FIG. 12. In an example embodiment, the subscription interface can correspond to the relay agent module(s) 304 of FIG. 3.

[00162] At block 1506, the allocation module(s) allocates a sessionizer bank (such as the sessionizer cluster ring 1226 of FIG. 12) for servicing the subscription request. The sessionizer bank comprises processing engines available to service the subscription request. The sessionizer bank is linked to the subscription request and the subscriber(s) 1240.

[00163] At block 1508, a messaging interface module provisions identifiers linked to the respective processing engines of the sessionizer bank. For example, the messaging interface module can be implemented by the messaging system 1224 of FIG. 12. Furthermore, the messaging system 1224 can implement the messaging system 300 described in connection with FIG. 3 for providing consistent hashing scheduling. As such, the identifiers linked to the respective processing engines can correspond to a number of hash values assigned to each of the processing engines of the sessionizer bank.

[00164] At block 1510, the messaging interface module registers with a collection server the allocated sessionizer bank as available to process event messages (or “event data”) matching the subscription request by providing the provisioned identifiers. For example, the consumer devices 1228A-F of FIG. 12 provides advertisements to the messaging system 1224, relay agent modules (e.g., the Zookeeper transports 1230, 1232) interfaced with the collector cluster ring 1206, or a server (not shown) connected with the collector cluster ring 1206 configured to receive advertisements. As a result, the CEP engines 1208A-F can serve as producer devices with respect to the sessionizer cluster ring 1226.

[00165] At block 1510, the messaging interface module receives event messages from a processing engine linked to the collection server. For example, one of the CEP engines 1208A-F transmits an event message through the messaging system 1224 to a selected one of the consumer devices 1228A-F. The selection of the consumer device can be based on a consistent hashing scheduler.

[00166] At block 1514, the selected one of the consumer devices 1228A-F processes received event messages in accordance with session rule data linked to the subscription request to generate sessionized data. The session rule data correspond to one or more attributes (e.g., tuples of information) that the subscriber requested. At block 1516, providing the generated sessionized data to the subscriber(s) 1240. At block 1518, the method 1500 ends.

[00167] FIG. 16 is a flowchart illustrating an example method 1600 of generating and updating sessions, in accordance with an example embodiment. In an example embodiment, the blocks 1602-1652 of the method 1600 can be performed by the consumer devices 1228A-F of FIG. 12.

[00168] The method 1600 can be triggered by receiving an event message. For example, the method 1600 can start at block 1602 in response to receiving an event message of a raw event type. For example, a raw event type corresponds to receiving an event message from a producer device that is not marked as containing an Internet bot program signature and/or if it is not marked to be bypassed by the sessionizer. At block 1604, the method 1600 determines whether a session exists for the raw event. In response a determination that a session does not exist, the method 1600 can proceed to block 1606 to determine whether a read is to be performed. In an example embodiment, a read is to be performed in response to a sessionizer node failure. If a read is to be performed, the method 1600 can proceed to block 1608 for loading a session and to block 1610 for determining whether a load is pending. If the load is not pending, the method performs an asynchronous load of a session at block 1612. If the load is pending, the method 1600 queues the received raw event at block 1614.

[00169] Returning to the decision of block 1606, if the method 1600 determines that a read is not required then the method 1600 proceeds to block 1616 for creating a new session. The method 1600 proceeds to blocks 1618-1624 for decorating the received event message with geographic and demographic information, extracting metadata from the received event message, executing a sessionizer on the event message, and sending a session begin marker. At blocks 1626-1634, the method 1600 updates the counters associated with the decorated event message, executes a sessionizer, saves and synchronizes the session, sends the sessionized event data to the subscriber, and

processes pending events. After processing pending events at block 1634, the method 1600 can return to block 1626 for repeating the actions of block 1626-1634.

[00170] Returning to the decision of block 1604, in response a determination that a session for the received event message exists, the method 1600 proceeds to block 1636 for determining whether the existing session is long based on the session duration. For example, an existing session is long when it has expired based on the latest cached event message being past the session duration. Based on a determination that the session is long, the method 1600 proceeds to block 1602 to end the old session and create a new session at block 1616, after which the method 1600 executes block 1618-1634 as described above. If instead the method determines that the session is not long at block 1636, the method proceeds to block 1642 to decorate the event message with geographic and demographic information and then proceeds to block 1626-1634 as described above.

[00171] The method 1600 can also be triggered by receiving data indicative of a session load event, such as a request to load a selected session. The method 1600 can start at block 1650 and proceed to block 1650 to determine whether or not the session exists. Based on a determination that the session does exist the method 1600 proceeds to block 1636 to determine whether or not the session is long and continues as described above. Based on a determination that the session does not exist, the method 1600 proceeds to block 1638 to end the old session and proceeds to block 1616 as described above.

[00172] FIG. 17 is a flowchart illustrating an example method of session lifecycle management, in accordance with an example embodiment. In an example embodiment, the blocks 1702-1752 of the method 1700 can be performed by the consumer devices 1228A-F of FIG. 12. The execution of the method 1700 can be executed in response to a number of events, such as a timer event, a session check event, and a session recover event. In response to a timer event, the method 1700 starts at block 1702 and proceeds to block 1704 to handle a timeout read request. At block 1706, the method 1700 handles the session timeout and then proceeds to block 1708 to check the affinity. For example, an affinity exists if there is a sessionizer node that is assigned to

receive event messages for the corresponding session and tenant of the time out event. Based on a determination that there is no affinity, the method 1700 proceeds to block 1710 to send a session check event and then ends at block 1712 (and, e.g., repeating the method 1700 starting at block 1722).

Alternatively, based on a determination that there is an affinity, the method 1700 proceeds to block 1714 to execute the sessionizer. The method then proceeds to blocks 1716, 1718 to send a session end marker and then to delete the session. Afterwards, the method 1700 proceeds to block 1719 to end.

[00173] In response to either a session recover event (block 1720) or session check event (block 1722), the method 1700 proceeds to block 1724 to determine whether a local session exists. A local session is session data stored in an off heap cache of a sessionizer node and which is backed up in a backing store, such as the Kafka queues 1234, 1236 of FIG. 12. Based on a determination that a local session does exist, the method 1700 proceeds to block 1726 and ends. Alternatively, based on a determination that a local session does not exist, the method 1700 proceeds to block 1714 to execute the sessionizer and then proceed to blocks 1716-1719 as described above.

EXAMPLE VIEW MANAGEMENT SYSTEMS

[00174] In a real-time analytics environment, streams of event messages can carry hundreds of tuples of information. In systems where events are generated at a very high rate (e.g., approximately millions of events/sec or greater) and the payload size is large (e.g., approximately 1 kilobyte or more), the network bandwidth and computational resources to process the streams can be high. The system can be even more expensive if the streams flow over a wide area network. However, in many situations, the consumer devices of these streams are interested in consuming only a small amount of information from these streams and only when certain trigger conditions are satisfied. This is driven, in part, by cost considerations, as consuming a large volume of event messages at high rates can involve significant cost for the processing nodes and also can impact network bandwidth.

[00175] Example view management systems and methods disclosed herein can provide a cost effective way of dynamically creating partial views of an

event stream. A partial view of an event stream refers to a portion of the event stream that is selected for providing to a subscriber. The portion is selected based on a filtering rule that is defined by the subscriber and applied to the tuples of information of the event messages of the event stream.

[00176] Partial views can be generated based on subscription request received in a publication-subscription model. In this way, the system can efficiently distribute information through a wide area network, using point-to-point communication, and using different qualities of service. The tuples of information that are served to subscribers are regulated such that the subscriber receives information tuples that they are authorized to see and information tuples that the subscriber is not authorized to see are filtered out of the partial view.

[00177] In particular, in an example embodiment, the view management system is a single stage cloud based distributed real time system. At its ingress, the view management system receives high volume of events through multiple interfaces in real time. The view management system supports representation state transfer (REST), Kafka, and the like cluster messaging solutions. The architecture is capable to be adapted to other communication channels by building and deploying new adaptors. New adaptors can be built to support a communication channel configured to receive messages of the new channel type and to provide to a normalizer (which will be described in greater detail below) a data definition that specifies how to flatten out the event messages of the new channel type to a map or table having entries that are keys paired with respective values without nested maps or tables. The new adaptor can be deployed by having the new adaptor register with the normalizer so that the normalizer becomes configured to receive event messages from the new adaptor. As part of the registration, the new adaptor can provide the normalizer the corresponding data definition. Normalizing data will be described in greater detail later in connection with FIG. 21.

[00178] In this way, in one aspect, among others, of an example embodiment, the view management system is opaque to data. For example, the view management system is designed to handle semi-structured and unstructured data. Internally, the view management system handles Map data type (e.g., a table) where the key is of type string and the corresponding value can be a primitive

data type. The view management system can handle values of complex type if the data type is defined to the view management system and also built into the application.

[00179] FIG. 18 is a block diagram illustrating an example view management architecture 1800, in accordance with an example embodiment. The view management architecture 1800 includes a view application 1802 that comprises a number of input adaptors, such as an inbound message channel 1804, an inbound REST channel 1806, and an inbound Kafka channel 1808. The view application 1802 further includes a normalizer 1810, a view processor 1812, and a number of output adapters, such as an outbound message channel 1814, an outbound REST channel 1816, and an outbound Kafka channel 1818. The view application 1802 is interconnected with a subscriber module(s) 1820, a subscription manager module(s) 1822, an authentication system 1824, and a rules database 1826. The authentication system 1824 and subscription manager module(s) 1822 are coupled with a policy database 1828.

[00180] As event messages are received by the input adaptors, the normalizer 1810 normalizes the event message data for the processing by a CEP engine. In one example aspect, nested tables are folded into the outer table such that resulting normalized event message corresponds to a flat table with no nested objects. Normalization will be described in greater detail later in connection with FIGS. 21A, 21B. The normalized event messages are provided to the view processor 1812, which corresponds to a CEP engine provisioned with mutation and filtering rules per subscriber.

[00181] An example aspect of example embodiments, the view processor 1812 treats the normalized event message stream as a database table. The structure of the events are defined to the view processor 1812 similar to the way a table is defined to a database management application. The view processor 1812 can perform SQL queries against the stream of event messages. The stream of event messages change over time, so the query is performed in real-time on the streams of event messages. The view processor achieves this by using a window of time over which the SQL queries are performed. That is the SQL queries are applied to the portion of the event stream occurring within the window of time. The window can be either a tumbling window (e.g., a number

of fixed-sized, non-overlapping and contiguous time intervals) or a rolling window (e.g., a fixed-size time interval that shifts with time). An example embodiment, the view management architecture 1800 can handle any type of event as long as the event is defined to the normalizer 1810.

[00182] The subscription manager module(s) 1822 provides an API for subscribers 1820 to subscribe to views. In operation, the subscriber 1820 provides the subscription manager module(s) 1822 subscription data to subscribe to a view of the stream of event messages. The subscription data includes the parent stream name and a list of tuples from the parent stream that the subscriber 1820 selects to view. The subscription data also includes the channel, the channel address, and the QOS associated with the channel. Some channels might not support QOS control.

[00183] Additionally or alternatively, the view management architecture 1800 enforces authentication and authorization processes as part of subscription. For example, the subscription manager module obtains authorization information for the subscriber 1820 from the authentication system 1824 once the subscriber 1820 has successfully authenticated. The view management architecture 1800 supports a simple authentication and security level (SASL) based interface enabling the view management architecture 1800 to work with many authentication systems. Once the authentication system 1824 determines that the subscriber 1820 is authenticated and once authorization information successfully retrieved, the subscription manager module(s) 1822 generates SQL statements to select the specified tuples from the original stream and also adds the predicate to the statement based on the filtering rules provided in the subscription. The SQL statement is generated if the subscriber 1820 is authorized to see those tuples; and the SQL statement is not generated if the subscriber 1820 is not authorized to see those tuples. The subscription request will fail if the authorization check fails.

[00184] If authorization passes, the SQL statements are committed to the configuration system and the system is updated with the new SQL statements and the view becomes active. For example, the SQL statements can be stored in the rules database 826 is a set of rules that can be accessed by the view processor 1812 during operation.

[00185] The view management architecture 1800 utilizes the JETSTREAM framework which provides a distributed CEP infrastructure. For example, the view processor 1812 can correspond to JETSTREAM's Esper Processor. Accordingly, the view processor 1812 receives SQL statements on the fly and the statement are compiled and applied to the CEP engine at run time.

[00186] In example embodiments, the subscription manager module(s) 1822 provides a portal, such as a graphical user interface, for a human user to interact with the view management architecture 1800 and setup a subscription manually. This requires the user to be authenticated along with a subscriber application whose credentials need to be provided.

[00187] FIG. 19 is a block diagram illustrating an example view management system 1900, in accordance with an example embodiment. The view management system 1900 includes one or more applications 1902 coupled to a messaging system 1904 to connect to the cluster ring 1906 of one or more view application nodes 1908A-F. Load balancers 1910, 1912 interconnect applications 1914, 1916 with the cluster ring 1906. Furthermore, the applications 1914, 1916 are connected to respective data queues 1918, 1920. The data queues 1918, 1920 can correspond to Kafka queues in an example embodiment. Zookeeper transports 1922, 1924 also connected to the cluster ring 1906. Subscribers 1926 are coupled to the clustering 1906.

[00188] The application 1902 can serve as producer device generating event messages delivered to the cluster ring 1906 through the messaging system 1904. The messaging system 1904 can employ a round-robin scheduler in an example embodiment. It will be appreciated that the messaging system 1904 can, but need not, correspond to the messaging system 300 of FIG. 3. The view application nodes 1908A-F can correspond to respective instances of the view application 1802 of FIG. 18. Furthermore, each node of the view application nodes 1908A-F can be provisioned in response to a view description provided by the subscriber 1926.

[00189] The view application nodes 1908A-F treat the real-time stream as a database table and run queries against the stream. A stream is made of a sequence of event messages of a given type. In this way, each stream is similar to a database table. Each individual event message in the stream is similar to a

row in a database table. A technical effect is that persistent storage of streams can be avoided in example embodiments. The view applications 1908A-F use CEP engines to provide query processing capability. A schema of the event message can be declared to the CEP engine, e.g., by the application 1902. The view applications 1908A-F apply queries at run time on behalf of corresponding subscribers. An example query follows:

```
[00190]  @OutputTo("outboundMessageChannel")
[00191]  @PublishOn(topics="Trkng.RR1/bisEvent")
[00192]  Select
nqt,flgs,t,p,itm,app,mav,sid,g,uc,aa,cat,tcatid,gf,lfcac,cpnip,sQr,leaf,type,b
ti,quan,binamt,bidamt,curprice,incr_price,bi,st,pri,l1,l2,meta,plmt,trkp,car
t_itm,itm_qty,ul,rdt,dn,osv from SOJEvent(p in
('2047935','2052268','1468719','1673582','5408','2056116','2047675','434
0'));
```

[00193] In this example, the query selects a set of fields from the stream named SOJEvent after using the filters specified by the IN() clause. The output is then directed to one of the endpoints in the Directed Cyclical Graph. At the SQL level, the subscribers control the flow of events through the pipeline and also specify the address over which the information will be published. For example, the @outputTo() annotation specifies the channel.

[00194] FIG. 20 is a flowchart illustrating an example method 2000 of stream view management, in accordance with an example embodiment. In this example, the method 2000 can include operations such as receiving subscription data from a client device (block 2004), receiving a first event stream (block 2006), converting the received first event stream to a table of entries (block 2008), selecting a portion of the converted first event stream based at least on the entries of the selected portion of the event stream matching the at least on attribute (block 2010), and providing the selected portion of the converted event stream for transmission as session data to the client device (block 2012). The example method 2000 will be described below, by way of explanation, as being performed by certain modules and components. It will be appreciated, however, that the operations of the example method 2000 can be performed in any suitable order by any number of the modules shown in FIGS. 3, 18, and 19.

[00195] The method 2000 starts at block 2002 and proceeds to block 2004, at which the subscription manager module(s) 1822 receives subscription data from a client device. The client device can correspond to a subscriber such as the subscriber 1820 a FIG. 18. The subscription data comprises an event stream identifier to identify an event stream and at least one attribute to select events from the event stream. In example embodiments, the subscription data further comprise data indicative of a requested channel, channel address, and QOS associated with the requested subscription. At least a portion of the subscription data can be stored in the rules database 1826. For instance, the at least one attribute to select events can be stored as a set of rules (e.g., SQL type query rules) linked to the subscriber (e.g., the registered user of the client device).

[00196] In an example embodiment, a view processor 1812 of FIG. 18 is provisioned in response to receiving the subscription data. The view processor 1812 is configured based on a number of rules based on the subscription data. For example, the provisioned view processor 1812 accesses the rules database 1826 to retrieve and apply a number of SQL type query rules.

[00197] In example embodiments, the subscription manager module(s) 1822 and the authentication system 1824 authenticate the user and authorize the user to receive the data requested, as described above. For example, in response to a successful authentication process, the subscription manager module(s) 1822 compares the subscription data with authorization data linked to the user device. The subscription manager module(s) 1822 determines whether the user device is authorized to receive data indicated by the at least one attribute of the subscription data.

[00198] At block 2006, a messaging interface module receive a first event stream from a producer device. For example, the event stream can correspond to a stream of event messages received from at least one of the inbound message channel 1804, the inbound REST channel 1806, or the inbound Kafka channel of FIG. 19. Furthermore, the received stream of event messages match the event stream identifier.

[00199] At block 2008, a normalizer module, such as the normalizer 1810 of FIG. 18, converts the received first event stream to a table of entries. The entries

of the table correspond to respective event messages. In other words, the normalizer module flattens out the data structure of the received event messages.

[00200] At block 2010, a view processor 1812 selects a portion of the converted first event stream based at least on the entries of the selected portion of the event stream matching the at least on attribute of the subscription data. The selecting of the portion of the converted first event stream includes performing a database search query against the converted first event stream. At block 2012, the view processor engine 1812 provides the selected portion of the converted event stream for transmission as session data to the client device.

[00201] FIGS. 21A and 21B are block diagrams illustrating example data structures 2100A, 2100B of an event message, in accordance with example embodiments. The data structure 2100A represents an illustrative example embodiment of an event message 2102 received at the input of the normalizer 1810 of FIG. 18. The data structure 2100B represents an illustrative example embodiment of the data structure 2100A after its structure has been flattened by the normalizer 1810 during operation. It will be appreciated that the example embodiments of FIG. 21A, 21B are described by way of illustration only and are not intended to be limiting. Furthermore, the scope of the disclosure herein encompasses other received event-message data structures in alternative embodiments.

[00202] The data structure 2100A includes the received event message 2102, which includes a header data field 2104 and one or more entries, such as entries 2106-2110. The entries 2106-2110 represent tuples of information of the event message. Each of the entries 2106-2110 includes a key paired with a value. For example, the entry 2106 includes KEY_1 matched with VALUE_1. The entry 2108 includes KEY_2 matched with VALUE_2. The entry 2110 includes KEY_3 matched with VALUE_3.

[00203] The header data field 2104 can include data that is descriptive of the contents of the event message 2102. For example, the header data field 2104 can include an identifier of the event message type (e.g., channel type), data indicative of the number of entries contained by the event message 2102, a start address of the entries 2106- 2110, an end address of the entries 2106-2110, a start address of each of the entries 2106-2110, an end address of each of the

entries 2106-2110, a timestamp indicating the time that the event message 2102 was sent and/or received, and/or the like.

[00204] Furthermore, the header data field 2104 can include data that describes the data types or data structures of the values of the entries 2106-2110. This data can be useful because values of the entries 2106-2110 of the received event message 2102 need not correspond to primitive data types. One or more of the values of the entries 2106-2110 can correspond to a nested table or map. For example, in the illustrated example embodiment, VALUE_2 of the entry 2108 corresponds to a nested table 2112 that includes a number of entries, such as entries 2114-2118. In particular, the entry 2114 includes KEY_A matched with VALUE_A; the entry 2116 includes KEY_B matched with VALUE_B; and the entry 2118 includes KEY_C matched with VALUE_C. The nested table can, for example, describe attributes of a tuple of information.

[00205] The header data field 2104 can include data that is descriptive of the structure of the values of the entries 2106-2110. For example, the header data field 2104 can include data that indicates the length (e.g., the number of entries) of each of the values of the entries 2106-2110. In the illustrated example embodiment of FIG. 21A, VALUE_1 and VALUE_3 are primitive data types.

[00206] Turning to FIG. 21B, the data structure 2100B shows the flattened version of the data structure 2100A. For example, during operation, the normalizer 1810 can convert the event message 2102 to the data structure 2100B of FIG. 21B which corresponds to a flat map or table. Accordingly, the data structure 2100B includes a number of entries such as 2120-2128. The entry 2120 corresponds to the entry 2106 of the event message 2102. The entries 2122-2126 of the data structure 2100B correspond to the nested table 2112 that the normalizer 1810 has brought to the top level of the table to remove nested tables. To maintain the key mapping, the normalizer 1810 combines each of the keys KEY_A, KEY_B, KEY_C of the nested table 2112 to the key (e.g., KEY_2) that is matched to the nested table 2112 (e.g., VALUE_2). In an alternative embodiment, the order of the appended keys can be reversed (e.g., [KEY_A KEY_1]).

[00207] The normalizer 1810 can combine the keys of the different levels in a number of ways. The combination can be formed by appending and/or

concatenating each key or a portion of each key. For example, the entry 2122 corresponds to the first entry 2114 of the nested table 2112. Accordingly, the normalizer 1810 appends KEY_A to KEY_2 to form KEY_2A, which is matched to the value VALUE_A of the first entry 2114. That is, KEY_2A can be represented as [KEY_2 KEY_A]. The normalizer 1810 can continue appending the key of each entry of the nested table 2112 to KEY_2 to form the keys of the entries 2124, 2126.

[00208] Accordingly during operation, the normalizer 1810 generates a normalized event message in response to receiving the event message. For example, the normalizer 1810 determines whether an entry of the event message has a nested table or map as its value. In response to a determination that the entry includes a nested table, the normalizer 1810 combines the key of the entry with the respective keys of the nested table. The normalizer then matches the resulting keys with the respective entries of the nested table to form the entries of the normalized event message that correspond to the nested table. This processes can occur recursively so that tables of tables of tables, and so on, are flattened out.

[00209] Additionally or alternatively, the normalizer 1810 can omit the header data field 2104 of the event message 2102 in normalizing the event message 2102 and generating the data structure 2100B. For example, the normalizer 1810 can generate new header data describing characteristics of the data structure 2100B, such as its length e.g., number of entries). In alternative embodiment, the new header data can be omitted from the data structure 2100B (e.g., by using an end of table marker to signal the end of the data structure 2100B).

[00210] Certain embodiments are described herein as including logic or a number of components, modules, or mechanisms. Modules can constitute either software modules (e.g., code embodied (1) on a non-transitory machine-readable medium or (2) in a transmission signal) or hardware-implemented modules. A hardware-implemented module is a tangible unit capable of performing certain operations and can be configured or arranged in a certain manner. In example embodiments, one or more computer systems (e.g., a standalone, client or server computer system) or one or more processors can be configured by software (e.g.,

an application or application portion) as a hardware-implemented module that operates to perform certain operations as described herein.

[00211] In various embodiments, a hardware-implemented module can be implemented mechanically or electronically. For example, a hardware-implemented module can comprise dedicated circuitry or logic that is permanently configured (e.g., as a special-purpose processor, such as a field programmable gate array (FPGA) or an application-specific integrated circuit (ASIC)) to perform certain operations. A hardware-implemented module can also comprise programmable logic or circuitry (e.g., as encompassed within a general-purpose processor or other programmable processor) that is temporarily configured by software to perform certain operations. It will be appreciated that the decision to implement a hardware-implemented module mechanically, in dedicated and permanently configured circuitry, or in temporarily configured circuitry (e.g., configured by software) can be driven by cost and time considerations.

[00212] Accordingly, the term "hardware-implemented module" should be understood to encompass a tangible entity, be that an entity that is physically constructed, permanently configured (e.g., hardwired) or temporarily or transitorily configured (e.g., programmed) to operate in a certain manner and/or to perform certain operations described herein. Considering embodiments in which hardware-implemented modules are temporarily configured (e.g., programmed), each of the hardware-implemented modules need not be configured or instantiated at any one instance in time. For example, where the hardware-implemented modules comprise a general-purpose processor configured using software, the general-purpose processor can be configured as respective different hardware-implemented modules at different times. Software can accordingly configure a processor, for example, to constitute a particular hardware-implemented module at one instance of time and to constitute a different hardware-implemented module at a different instance of time.

[00213] Hardware-implemented modules can provide information to, and receive information from, other hardware-implemented modules. Accordingly, the described hardware-implemented modules can be regarded as being communicatively coupled. Where multiple of such hardware-implemented

modules exist contemporaneously, communications can be achieved through signal transmission (e.g., over appropriate circuits and buses) that connect the hardware-implemented modules. In embodiments in which multiple hardware-implemented modules are configured or instantiated at different times, communications between such hardware-implemented modules can be achieved, for example, through the storage and retrieval of information in memory structures to which the multiple hardware-implemented modules have access. For example, one hardware-implemented module can perform an operation, and store the output of that operation in a memory device to which it is communicatively coupled. A further hardware-implemented module can then, at a later time, access the memory device to retrieve and process the stored output. Hardware-implemented modules can also initiate communications with input or output devices, and can operate on a resource (e.g., a collection of information).

[00214] The various operations of example methods described herein can be performed, at least partially, by one or more processors that are temporarily configured (e.g., by software) or permanently configured to perform the relevant operations. Whether temporarily or permanently configured, such processors can constitute processor-implemented modules that operate to perform one or more operations or functions. The modules referred to herein can, in some example embodiments, comprise processor-implemented modules.

[00215] Similarly, the methods described herein can be at least partially processor-implemented. For example, at least some of the operations of a method can be performed by one or more processors or processor-implemented modules. The performance of certain of the operations can be distributed among the one or more processors, not only residing within a single machine, but deployed across a number of machines. In some example embodiments, the processor or processors can be located in a single location (e.g., within a home environment, an office environment or as a server farm), while in other embodiments the processors can be distributed across a number of locations.

[00216] The one or more processors can also operate to support performance of the relevant operations in a "cloud computing" environment or as a "software as a service" (SaaS). For example, at least some of the operations can be performed by a group of computers (as examples of machines including

processors), these operations being accessible via a network 104 (e.g., the Internet) and via one or more appropriate interfaces (e.g., application program interfaces (APIs).)

[00217] Example embodiments can be implemented in digital electronic circuitry, or in computer hardware, firmware, software, or in combinations of them. Example embodiments can be implemented using a computer program product, e.g., a computer program tangibly embodied in an information carrier, e.g., in a machine-readable medium for execution by, or to control the operation of, data processing apparatus, e.g., a programmable processor, a computer, or multiple computers.

[00218] A computer program can be written in any form of programming language, including compiled or interpreted languages, and it can be deployed in any form, including as a stand-alone program or as a module, subroutine, or other unit suitable for use in a computing environment. A computer program can be deployed to be executed on one computer or on multiple computers at one site or distributed across multiple sites and interconnected by a communication network 104.

[00219] In example embodiments, operations can be performed by one or more programmable processors executing a computer program to perform functions by operating on input data and generating output. Method operations can also be performed by, and apparatus of example embodiments can be implemented as, special purpose logic circuitry, e.g., a field programmable gate array (FPGA) or an application-specific integrated circuit (ASIC).

[00220] The computing system can include clients and servers. A client and server are generally remote from each other and typically interact through a communication network 104. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other. In embodiments deploying a programmable computing system, it will be appreciated that both hardware and software architectures merit consideration. Specifically, it will be appreciated that the choice of whether to implement certain functionality in permanently configured hardware (e.g., an ASIC), in temporarily configured hardware (e.g., a combination of software and a programmable processor), or a combination of

permanently and temporarily configured hardware can be a design choice.

Below are set out hardware (e.g., machine) and software architectures that can be deployed, in various example embodiments.

[00221] FIG. 22 is a block diagram of a machine in the example form of a computer system 2200 within which instructions 2224 can be executed for causing the machine to perform any one or more of the methodologies discussed herein. In alternative embodiments, the machine operates as a standalone device or can be connected (e.g., networked) to other machines. In a networked deployment, the machine can operate in the capacity of a server or a client machine 110 in server-client network environment, or as a peer machine in a peer-to-peer (or distributed) network environment. The machine can be a personal computer (PC), a tablet PC, a set-top box (STB), a personal digital assistant (PDA), a cellular telephone, a web appliance, a network router, switch or bridge, or any machine capable of executing instructions 2224 (sequential or otherwise) that specify actions to be taken by that machine. Further, while only a single machine is illustrated, the term “machine” shall also be taken to include any collection of machines that individually or jointly execute a set (or multiple sets) of instructions 2224 to perform any one or more of the methodologies discussed herein.

[00222] The example computer system 2200 includes a processor 2202 (e.g., a central processing unit (CPU), a graphics processing unit (GPU) or both), a main memory 2204 and a static memory 2206, which communicate with each other via a bus 2208. The computer system 2200 can further include a video display unit 2210 (e.g., a liquid crystal display (LCD) or a cathode ray tube (CRT)). The computer system 2200 also includes an alphanumeric input device 2212 (e.g., a keyboard or a touch-sensitive display screen), a user interface (UI) navigation (or cursor control) device 2214 (e.g., a mouse), a disk drive unit 2216, a signal generation device 2218 (e.g., a speaker) and a network interface device 2220.

[00223] The disk drive unit 2216 includes a computer-readable medium 2222 on which is stored one or more sets of data structures and instructions 2224 (e.g., software) embodying or utilized by any one or more of the methodologies or functions described herein. The instructions 2224 can also reside, completely or

at least partially, within the main memory 2204 and/or within the processor 2202 during execution thereof by the computer system 2200, the main memory 2204 and the processor 2202 also constituting machine-readable media 2222.

[00224] While the computer-readable medium 2222 is shown, in an example embodiment, to be a single medium, the term "computer-readable medium" can include a single medium or multiple media (e.g., a centralized or distributed database, and/or associated caches and servers) that store the one or more instructions 2224 or data structures. The term "computer-readable medium" shall also be taken to include any non-transitory, tangible medium that is capable of storing, encoding or carrying instructions 2224 for execution by the machine and that cause the machine to perform any one or more of the methodologies of the present inventive subject matter, or that is capable of storing, encoding or carrying data structures utilized by or associated with such instructions 2224. The term "computer-readable medium" shall accordingly be taken to include, but not be limited to, solid-state memories, and optical and magnetic media.

Specific examples of computer-readable media 2222 include non-volatile memory, including by way of example semiconductor memory devices, e.g., erasable programmable read-only memory (EPROM), electrically erasable programmable read-only memory (EEPROM), and flash memory devices; magnetic disks such as internal hard disks and removable disks; magneto-optical disks; and CD-ROM and DVD-ROM disks.

[00225] The instructions 2224 can further be transmitted or received over a communications network 2226 using a transmission medium. The instructions 2224 can be transmitted using the network interface device 2220 and any one of a number of well-known transfer protocols (e.g., hypertext transfer protocol (HTTP)). Examples of communication networks 2226 include a local area network (LAN), a WAN, the Internet, mobile telephone networks, plain old telephone (POTS) networks, and wireless data networks (e.g., WiFi and WiMax networks). The term "transmission medium" shall be taken to include any intangible medium that is capable of storing, encoding or carrying instructions (e.g., instructions 2224) for execution by the machine, and includes digital or analog communications signals or other intangible media to facilitate

communication of such software. A transmission medium is one embodiment of a computer readable medium.

[00226] The following numbered examples comprise embodiments.

[00227] 1. A system for providing partial views of event streams over a network, the system comprising:

a subscription manager module configured to receive subscription data from a client device, the subscription data comprising an event stream identifier to identify an event stream and at least one attribute to select events from the event stream;

a messaging interface module configured to receive a first event stream comprising event messages, the first event stream matching the event stream identifier;

a normalizer module configured to convert the received first event stream to a table of entries, the entries corresponding to respective event messages; and

a view processor engine configured to select a portion of the converted first event stream based at least on the entries of the selected portion of the event stream matching the at least one attribute, the view processor engine being configured to provide the selected portion of the converted event stream for transmission as session data to the client device.

[00228] 2. The system of example 1, wherein the converting of the received first event stream includes flattening out nested tables within the received first event stream.

[00229] 3. The system of example 1 or example 2, wherein the view process engine is provisioned with subscription rules based on the received subscription data.

[00230] 4. The system of any one of examples 1 to 3, the selecting of the portion of the converted first event stream includes performing a database search query against the converted first event stream.

[00231] 5. The system of any one of examples 1 to 4, wherein the received subscription data further includes a channel identification data.

[00232] 6. The system of example 5, wherein the channel identification data is usable by the subscription manager module to determine a quality-of-service characteristic.

[00233] 7. The system of any one of examples 1 to 6, further comprising an authentication module interfaced with the subscription module such that in response to the subscription manager receiving the subscription data the authentication module performs an authentication process on a user of the user device.

[00234] 8. The system of example 7, wherein in response to a successful authentication process, the subscription module compares the subscription data with authorization data linked to the user device, the subscription module being configured to determine whether the user device is authorized to receive data indicated by the at least one attribute.

[00235] 9. The system of any one of examples 1 to 8, wherein the view processor corresponds to an instance of an Esper Processor.

[00236] 10. The system of any one of examples 1 to 9, wherein the subscription module is further configured to provide data for rendering a graphical interface on the client device, the graphical interface to receive user input to form at least a portion of the subscription data.

[00237] 11. A method for providing partial views of event streams over a network, the method comprising:

receiving subscription data from a client device, the subscription data comprising an event stream identifier to identify an event stream and at least one attribute to select events from the event stream;

receiving a first event stream comprising event messages, the first event stream matching the event stream identifier;

converting the received first event stream to a table of entries, the entries corresponding to respective event messages;

selecting, using one or more processors, a portion of the converted first event stream based at least on the entries of the selected portion of the event stream matching the at least one attribute; and

providing the selected portion of the converted event stream for transmission as session data to the client device.

[00238] 12. The method of example 11, wherein the converting of the received first event stream includes flattening out a nested table within the received first event stream.

[00239] 13. The method of example 11 or example 12, wherein the selecting of the portion of the converted first event stream is performed by a view process engine that is provisioned with subscription rules based on the received subscription data.

[00240] 14. The method of any one of examples 11 to 13, the selecting of the portion of the converted first event stream includes performing a database search query against the converted first event stream.

[00241] 15. The method of any one of examples 11 to 14, wherein the received subscription data further includes a channel identification data.

[00242] 16. A machine-readable storage medium for providing partial views of event streams over a network, the machine-readable storage medium embodying instructions that, when executed by a machine, cause the machine to perform operations comprising:

receiving subscription data from a client device, the subscription data comprising an event stream identifier to identify an event stream and at least one attribute to select events from the event stream;

receiving a first event stream comprising event messages, the first event stream matching the event stream identifier;

converting the received first event stream to a table of entries, the entries corresponding to respective event messages;

selecting a portion of the converted first event stream based at least on the entries of the selected portion of the event stream matching the at least one attribute; and

providing the selected portion of the converted event stream for transmission as session data to the client device.

[00243] 17. The machine-readable storage medium of example 16, wherein the converting of the received first event stream includes flattening out a nested table within the received first event stream.

[00244] 18. The machine-readable storage medium of example 16 or example 17, wherein the selecting of the portion of the converted first event stream is

performed by a view process engine that is provisioned with subscription rules based on the received subscription data.

[00245] 19. The machine-readable storage medium of any one of examples 16 to 18, the selecting of the portion of the converted first event stream includes performing a database search query against the converted first event stream.

[00246] 20. The machine-readable storage medium of any one of examples 16 to 19, wherein the received subscription data further includes a channel identification data.

[00247] Although the inventive subject matter has been described with reference to specific example embodiments, it will be evident that various modifications and changes can be made to these embodiments without departing from the broader scope of the inventive subject matter. Accordingly, the specification and drawings are to be regarded in an illustrative rather than a restrictive sense. The accompanying drawings that form a part hereof, show by way of illustration, and not of limitation, specific embodiments in which the subject matter can be practiced. The embodiments illustrated are described in sufficient detail to enable those skilled in the art to practice the teachings disclosed herein. Other embodiments can be utilized and derived therefrom, such that structural and logical substitutions and changes can be made without departing from the scope of this disclosure. This Detailed Description, therefore, is not to be taken in a limiting sense, and the scope of various embodiments is defined only by the appended claims, along with the full range of equivalents to which such claims are entitled.

[00248] Such embodiments of the inventive subject matter can be referred to herein, individually and/or collectively, by the term “invention” merely for convenience and without intending to voluntarily limit the scope of this application to any single invention or inventive concept if more than one is in fact disclosed. Thus, although specific embodiments have been illustrated and described herein, it should be appreciated that any arrangement calculated to achieve the same purpose can be substituted for the specific embodiments shown. This disclosure is intended to cover any and all adaptations or variations of various embodiments. Combinations of the above embodiments, and other

embodiments not specifically described herein, will be apparent to those of skill in the art upon reviewing the above description.

CLAIMS

1. A system for providing partial views of event streams over a network, the system comprising:

a subscription manager module configured to receive subscription data from a client device, the subscription data comprising an event stream identifier to identify an event stream and at least one attribute to select events from the event stream;

a messaging interface module configured to receive a first event stream comprising event messages, the first event stream matching the event stream identifier;

a normalizer module configured to convert the received first event stream to a table of entries, the entries corresponding to respective event messages; and

a view processor engine configured to select a portion of the converted first event stream based at least on the entries of the selected portion of the event stream matching the at least one attribute, the view processor engine being configured to provide the selected portion of the converted event stream for transmission as session data to the client device.

2. The system of claim 1, wherein the converting of the received first event stream includes flattening out nested tables within the received first event stream.

3. The system of claim 1, wherein the view process engine is provisioned with subscription rules based on the received subscription data.

4. The system of claim 1, the selecting of the portion of the converted first event stream includes performing a database search query against the converted first event stream.

5. The system of claim 1, wherein the received subscription data further includes a channel identification data.

6. The system of claim 5, wherein the channel identification data is usable by the subscription manager module to determine a quality-of-service characteristic.
7. The system of claim 1, further comprising an authentication module interfaced with the subscription module such that in response to the subscription manager receiving the subscription data the authentication module performs an authentication process on a user of the user device.
8. The system of claim 7, wherein in response to a successful authentication process, the subscription module compares the subscription data with authorization data linked to the user device, the subscription module being configured to determine whether the user device is authorized to receive data indicated by the at least one attribute.
9. The system of claim 1, wherein the view processor corresponds to an instance of an Esper Processor.
10. The system of claim 1, wherein the subscription module is further configured to provide data for rendering a graphical interface on the client device, the graphical interface to receive user input to form at least a portion of the subscription data.
11. A method for providing partial views of event streams over a network, the method comprising:
- receiving subscription data from a client device, the subscription data comprising an event stream identifier to identify an event stream and at least one attribute to select events from the event stream;
 - receiving a first event stream comprising event messages, the first event stream matching the event stream identifier;
 - converting the received first event stream to a table of entries, the entries corresponding to respective event messages;

selecting, using one or more processors, a portion of the converted first event stream based at least on the entries of the selected portion of the event stream matching the at least on attribute; and

providing the selected portion of the converted event stream for transmission as session data to the client device.

12. The method of claim 11, wherein the converting of the received first event stream includes flattening out a nested table within the received first event stream.

13. The method of claim 11, wherein the selecting of the portion of the converted first event stream is performed by a view process engine that is provisioned with subscription rules based on the received subscription data.

14. The method of claim 11, the selecting of the portion of the converted first event stream includes performing a database search query against the converted first event stream.

15. The method of claim 11, wherein the received subscription data further includes a channel identification data.

16. A machine-readable medium for providing partial views of event streams over a network, the machine-readable storage medium embodying instructions that, when executed by a machine, cause the machine to perform operations comprising:

receiving subscription data from a client device, the subscription data comprising an event stream identifier to identify an event stream and at least one attribute to select events from the event stream;

receiving a first event stream comprising event messages, the first event stream matching the event stream identifier;

converting the received first event stream to a table of entries, the entries corresponding to respective event messages;

selecting a portion of the converted first event stream based at least on the entries of the selected portion of the event stream matching the at least on attribute; and

providing the selected portion of the converted event stream for transmission as session data to the client device.

17. The machine-readable storage medium of claim 16, wherein the converting of the received first event stream includes flattening out a nested table within the received first event stream.

18. The machine-readable storage medium of claim 16, wherein the selecting of the portion of the converted first event stream is performed by a view process engine that is provisioned with subscription rules based on the received subscription data.

19. The machine-readable storage medium of claim 16, the selecting of the portion of the converted first event stream includes performing a database search query against the converted first event stream.

20. The machine-readable storage medium of claim 16, wherein the received subscription data further includes a channel identification data.

1/21

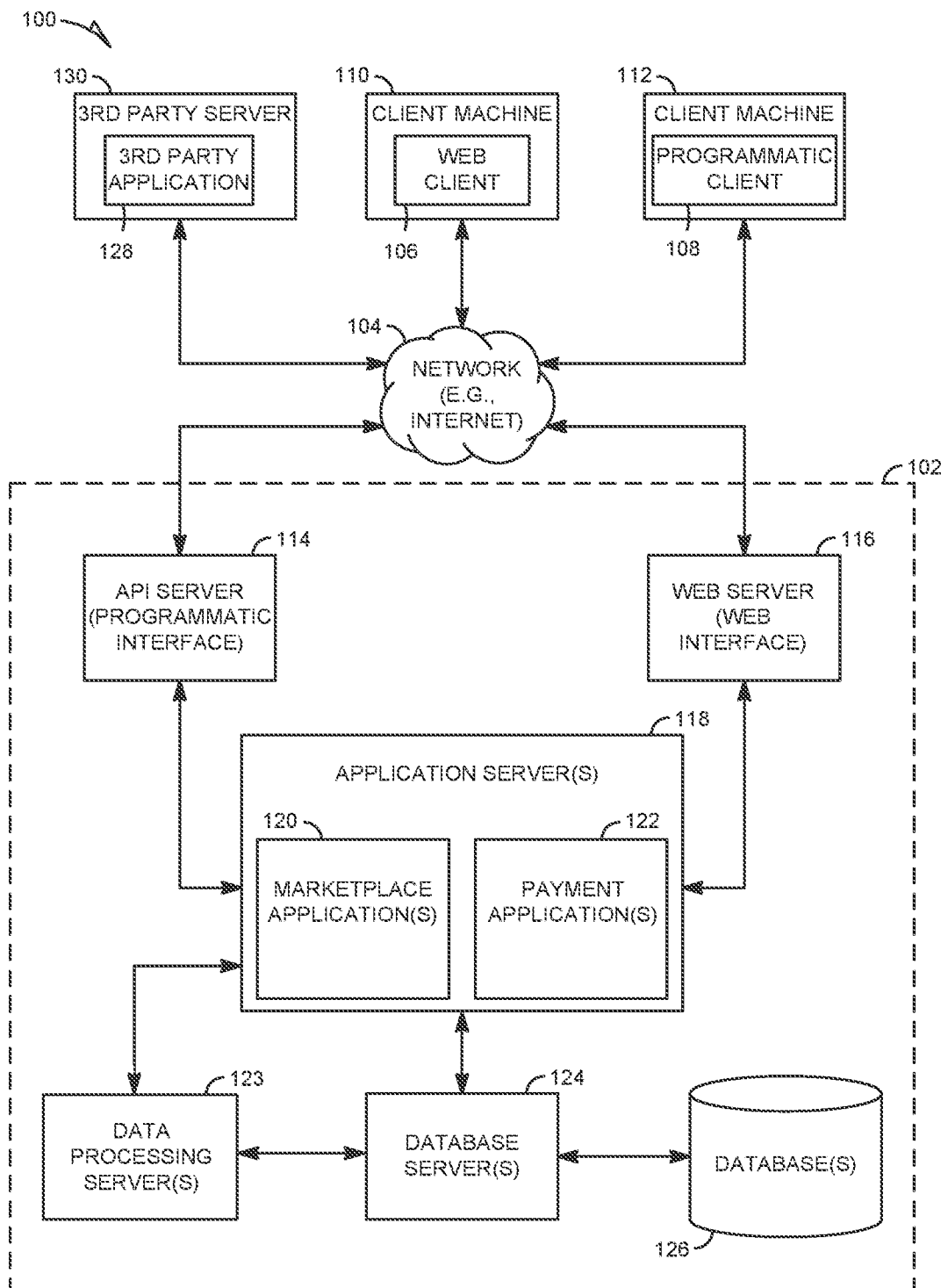


FIG. 1

2/21

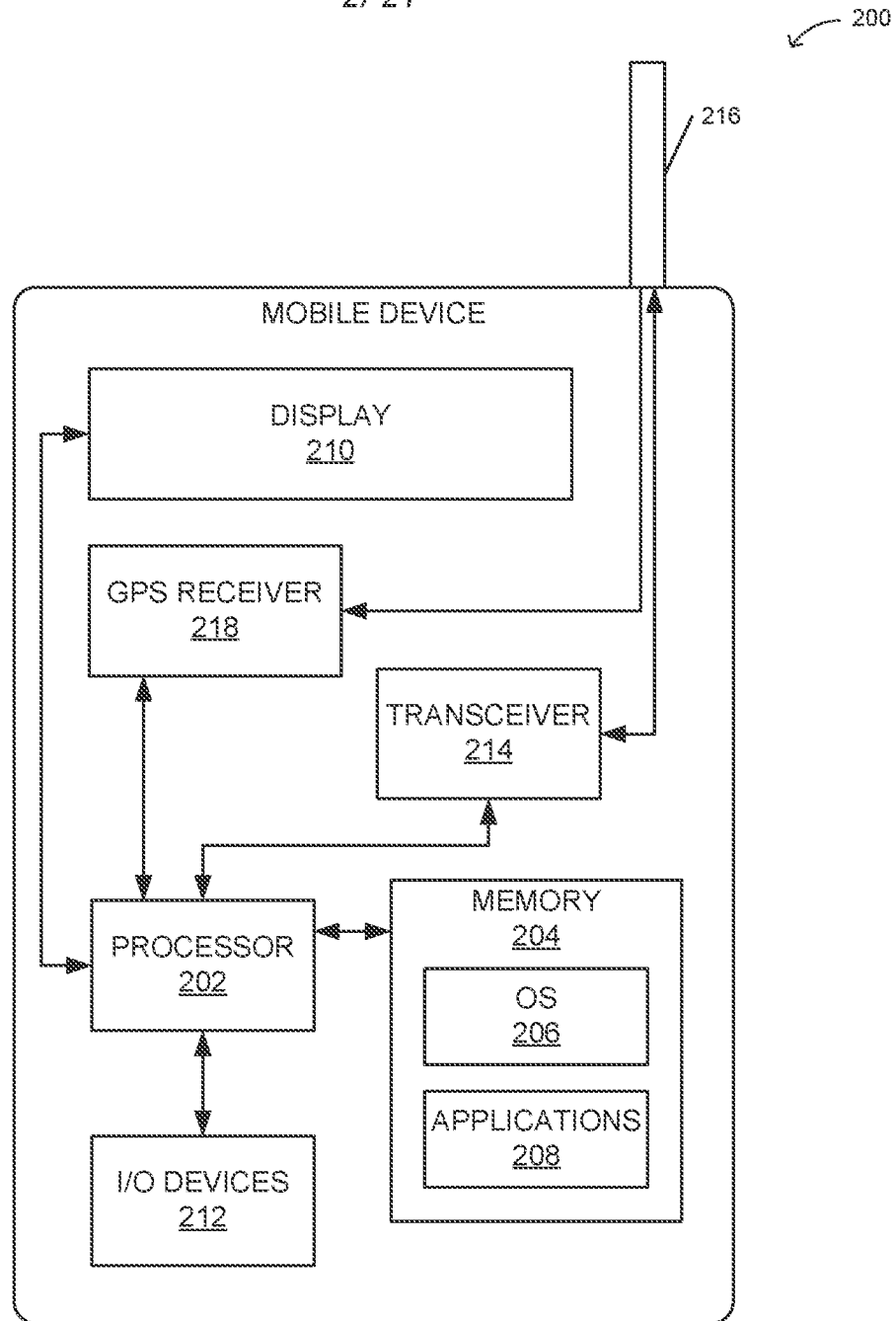


FIG. 2

3/21

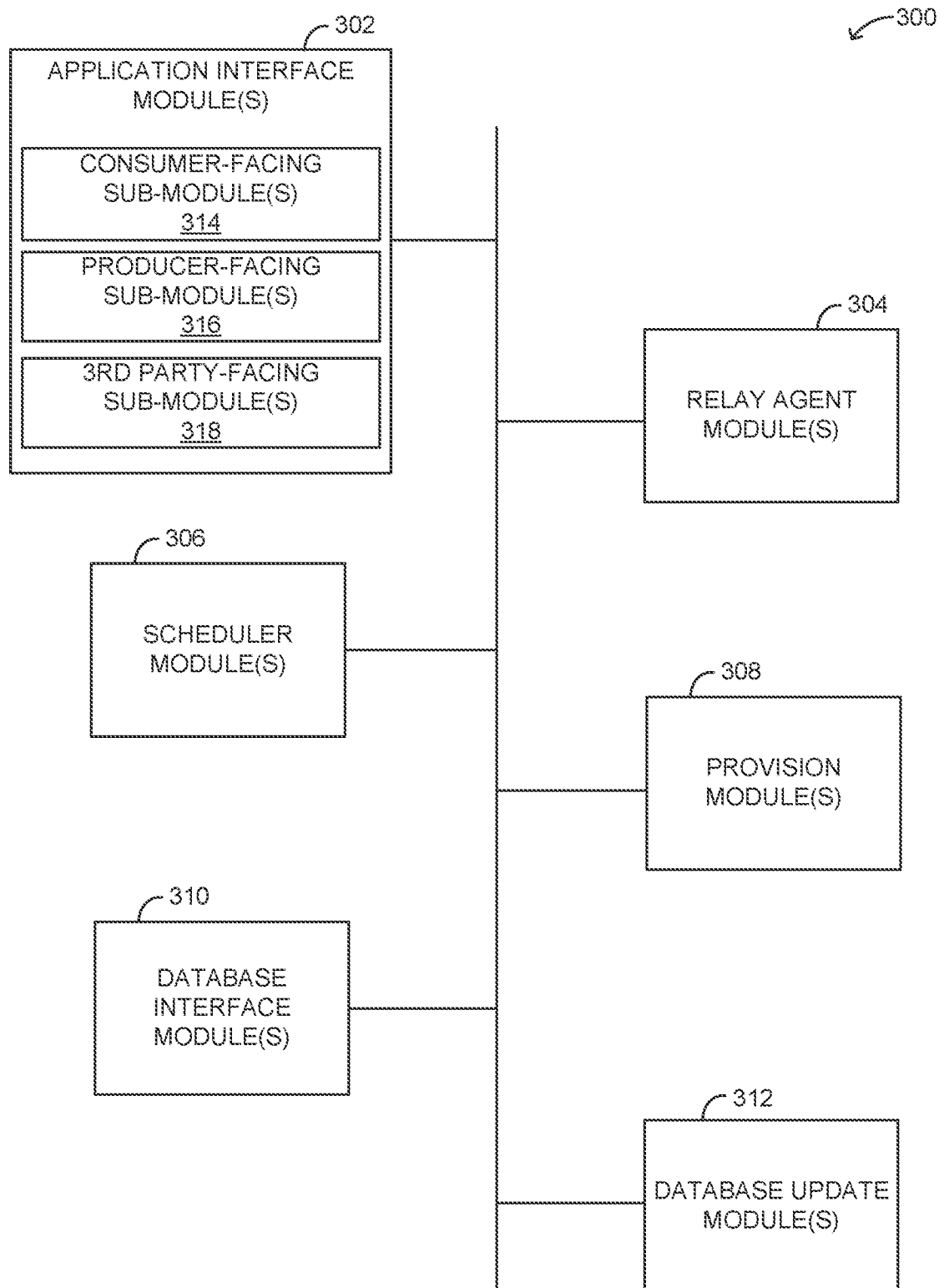


FIG. 3

400 →

4/21

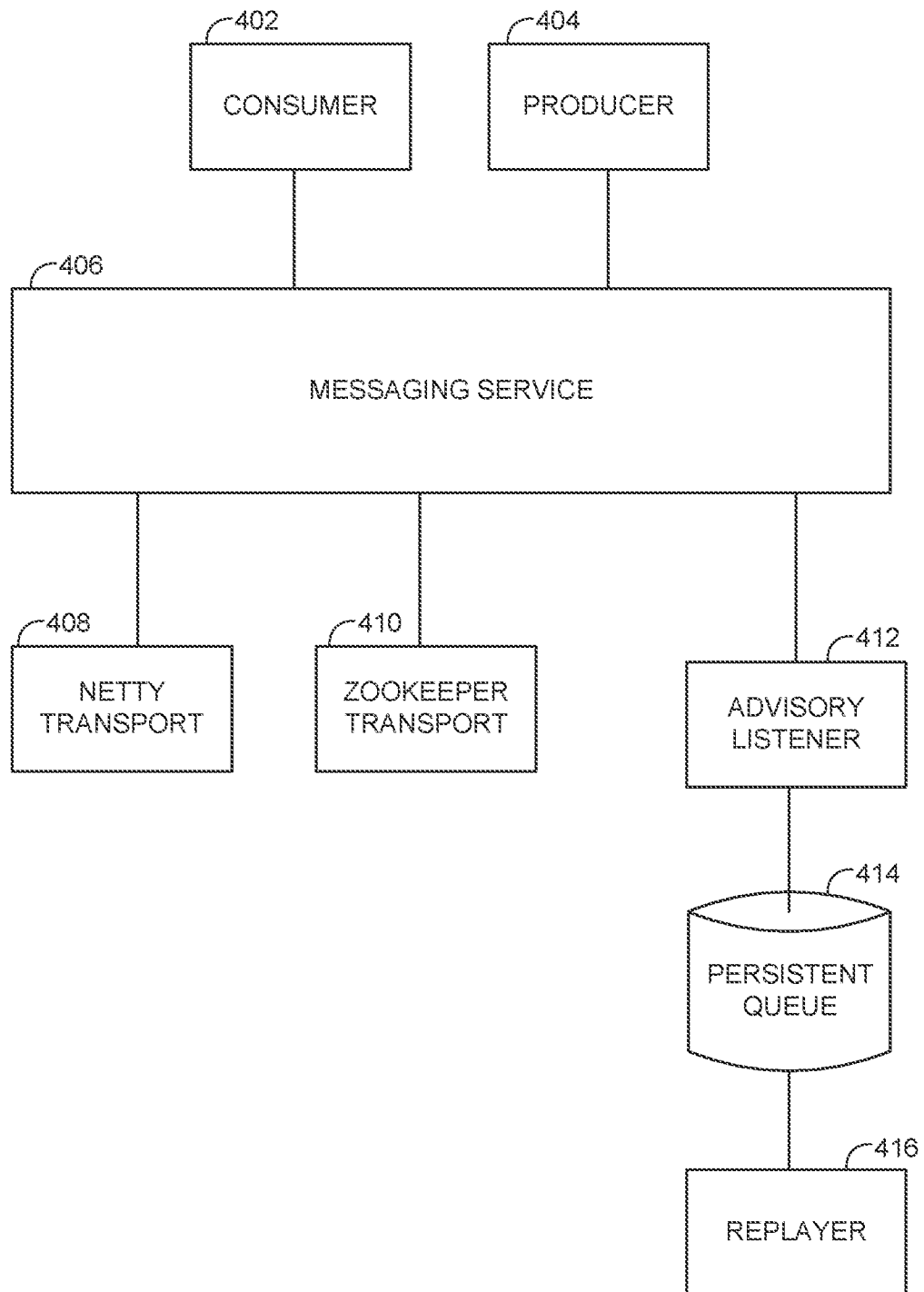


FIG. 4

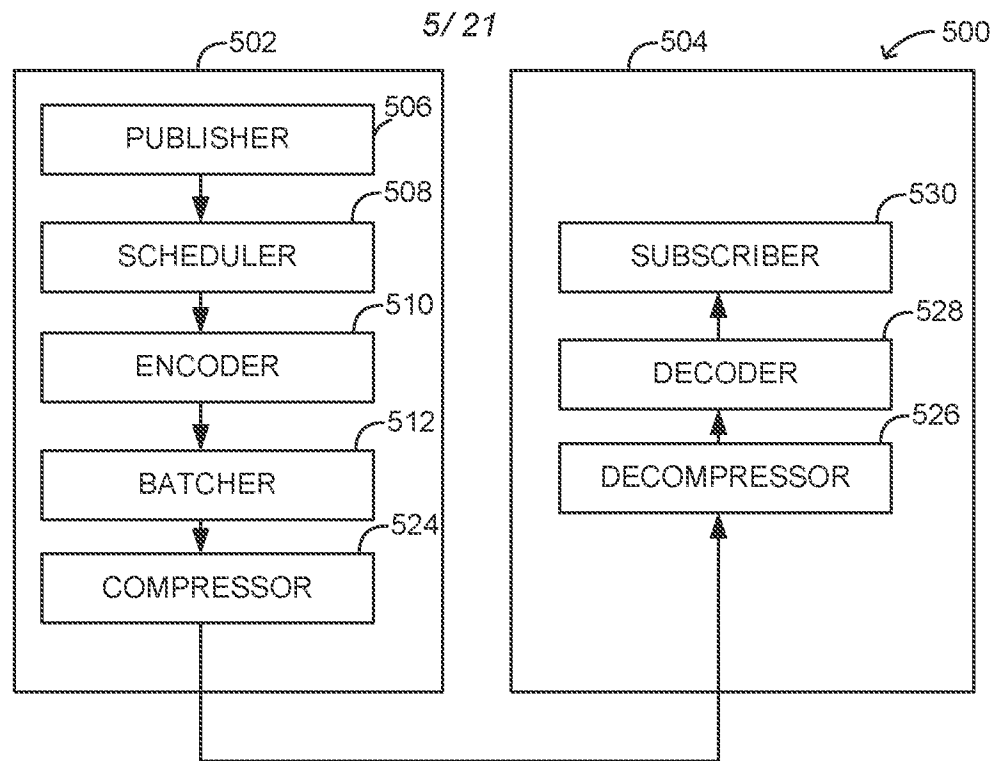


FIG. 5

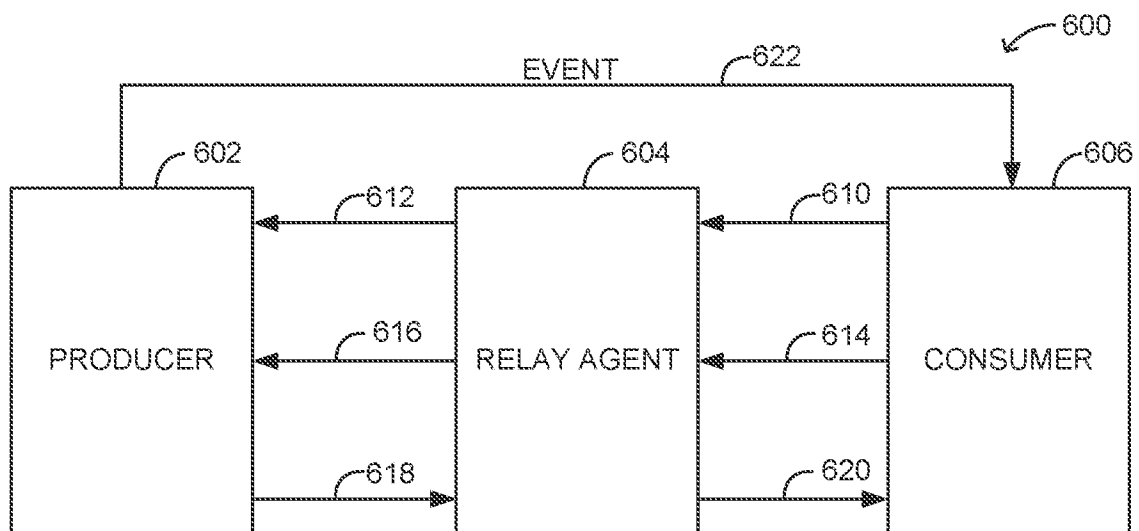


FIG. 6

6/21

700

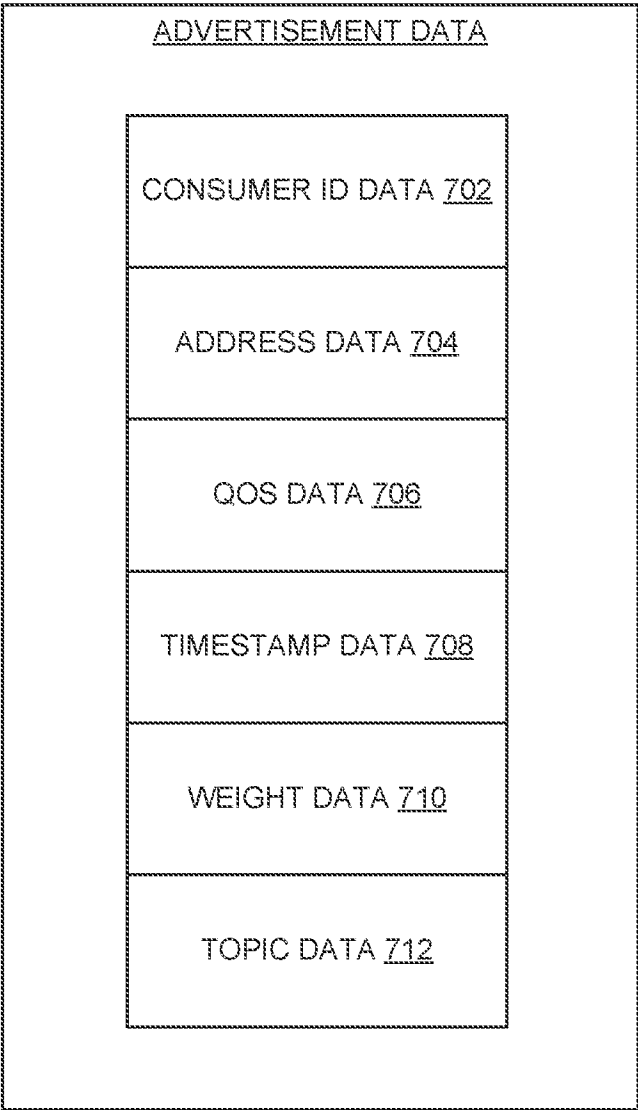
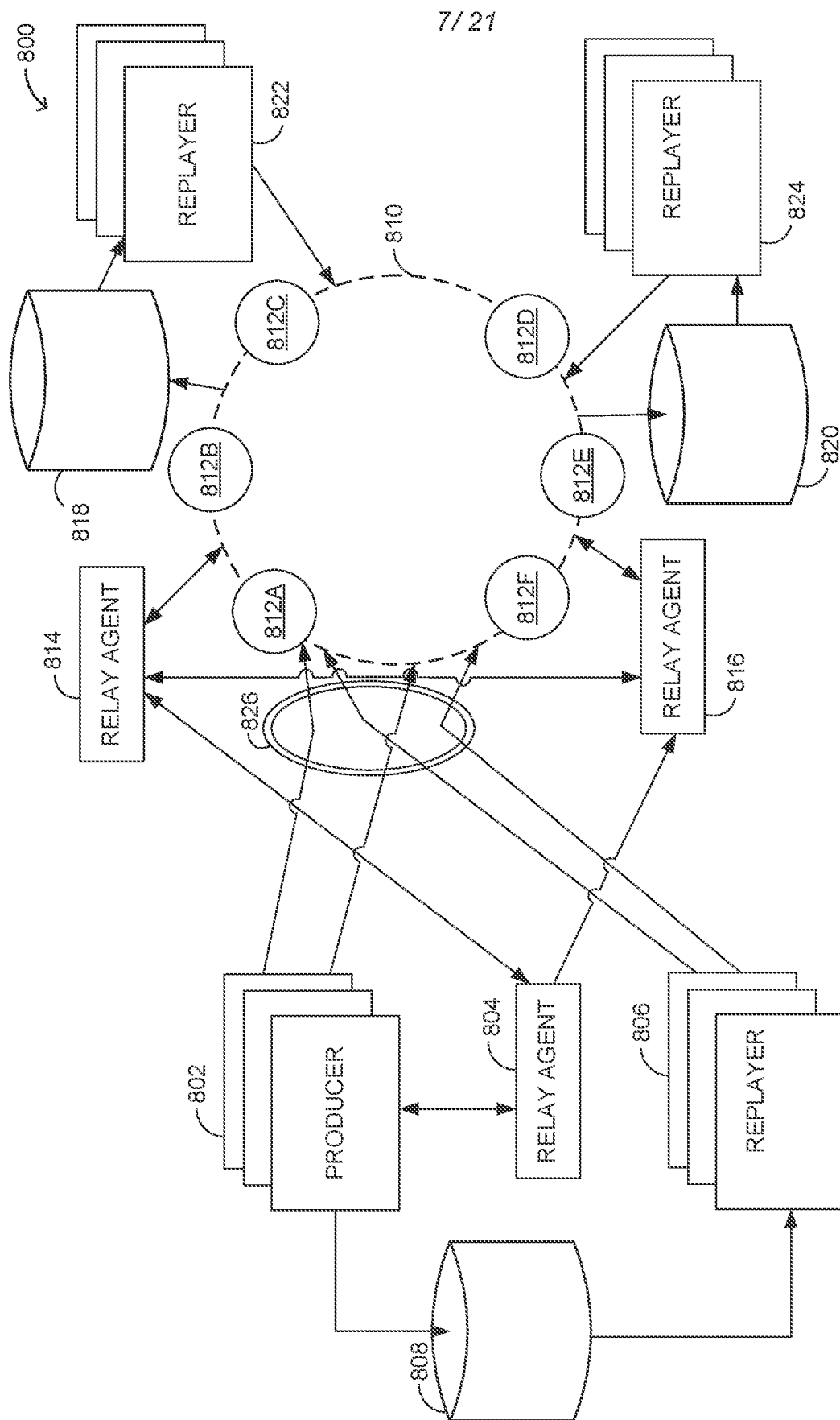


FIG. 7



8/21

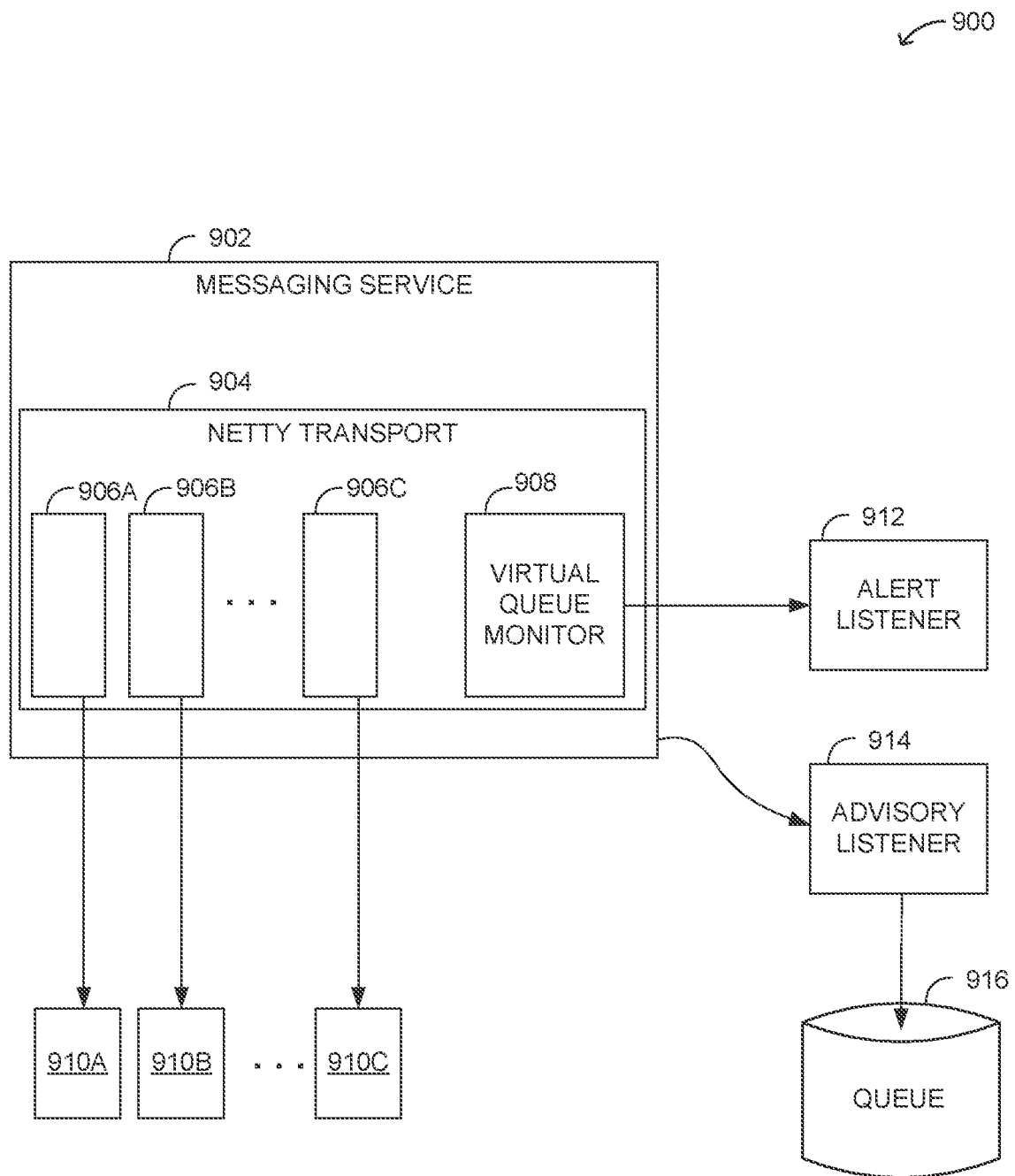


FIG. 9

9/21

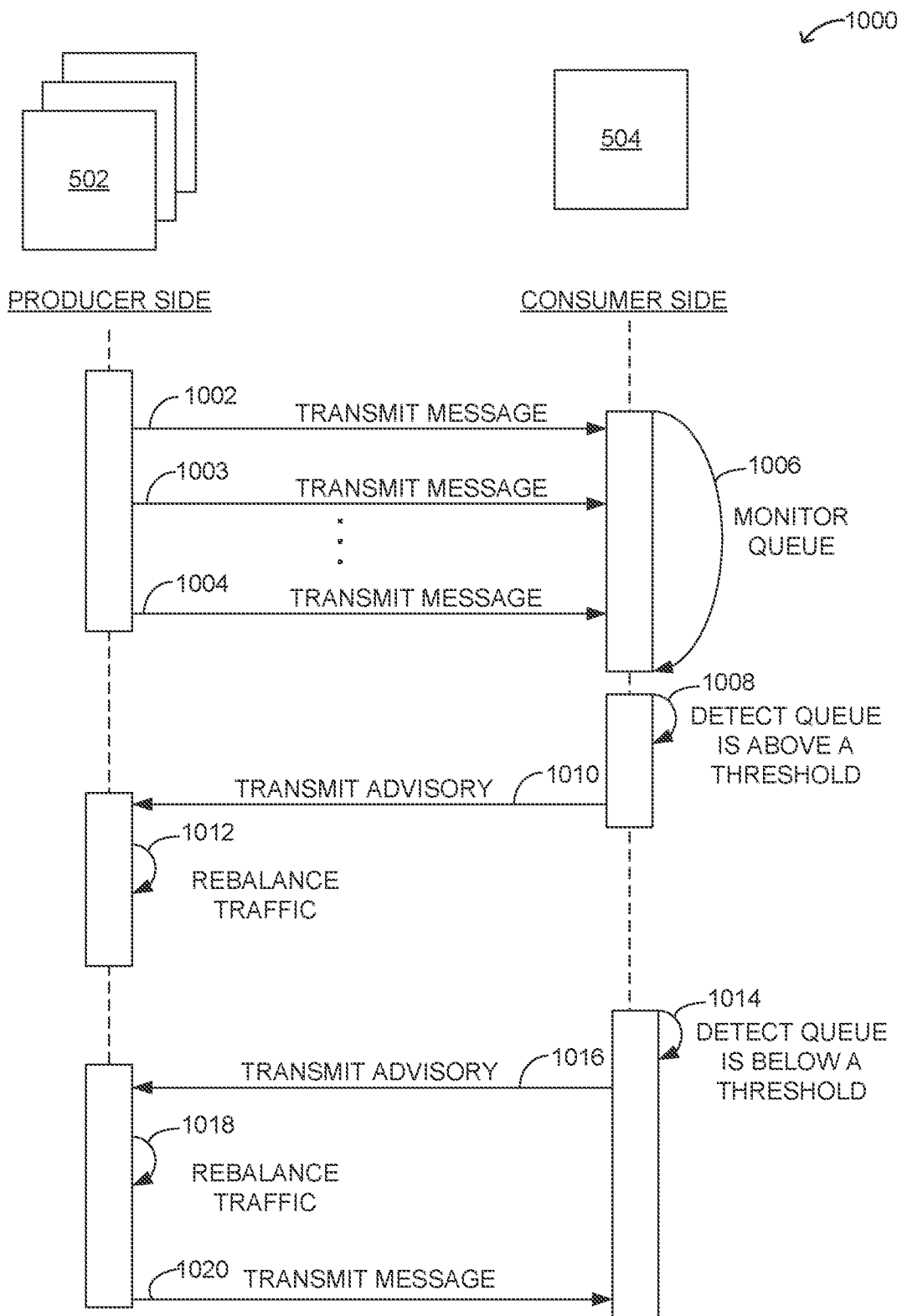


FIG. 10

10/21

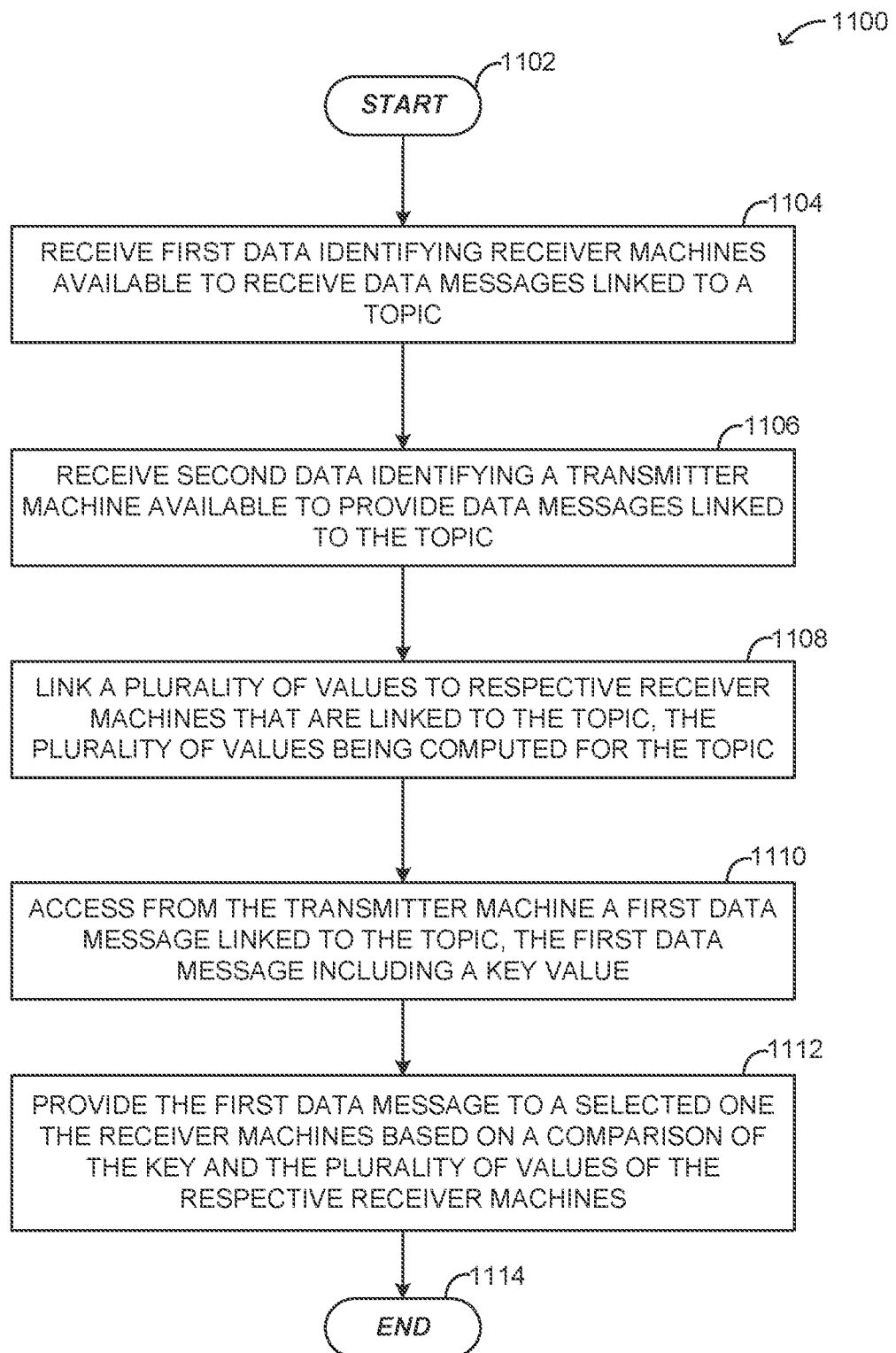


FIG. 11

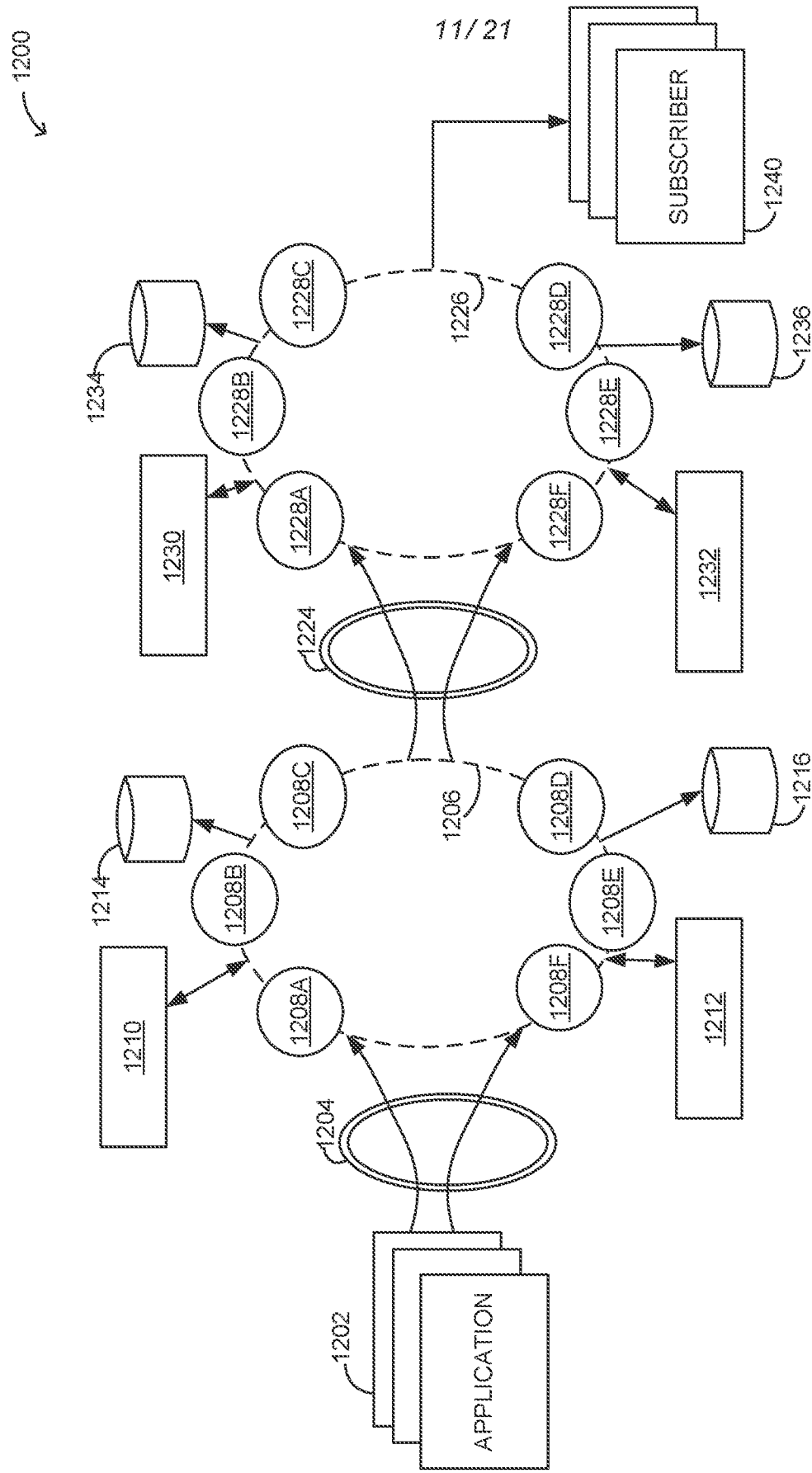


FIG. 12

12/21

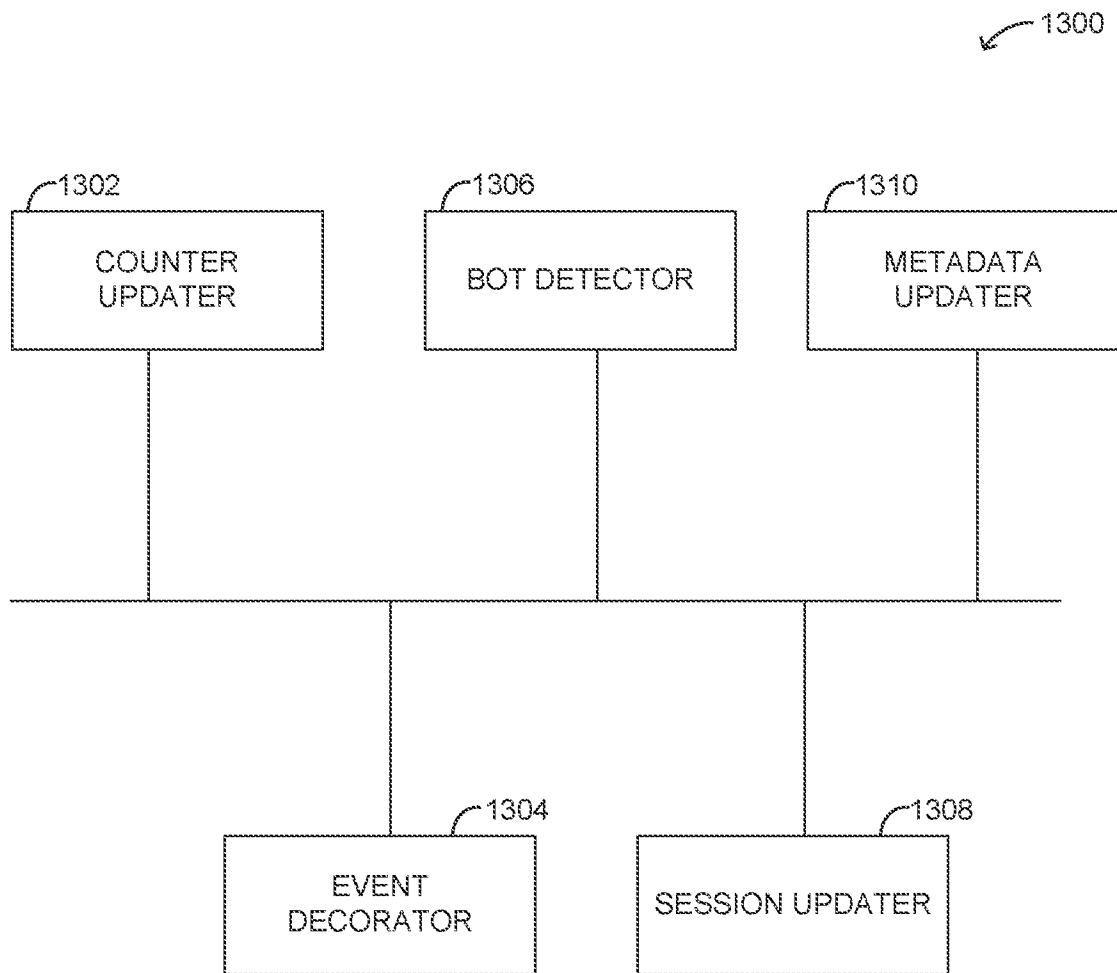


FIG. 13

1400 ↙

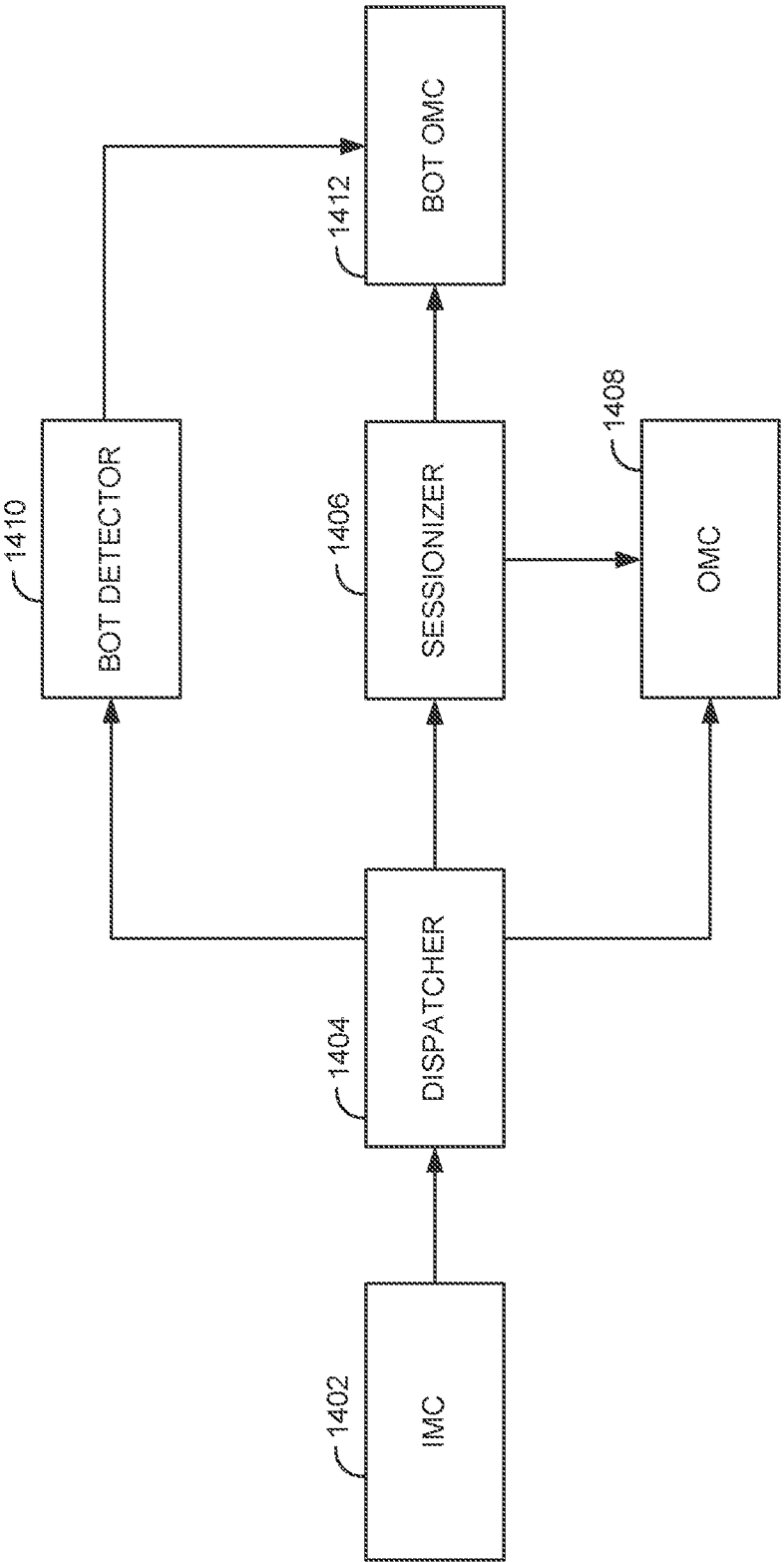


FIG. 14

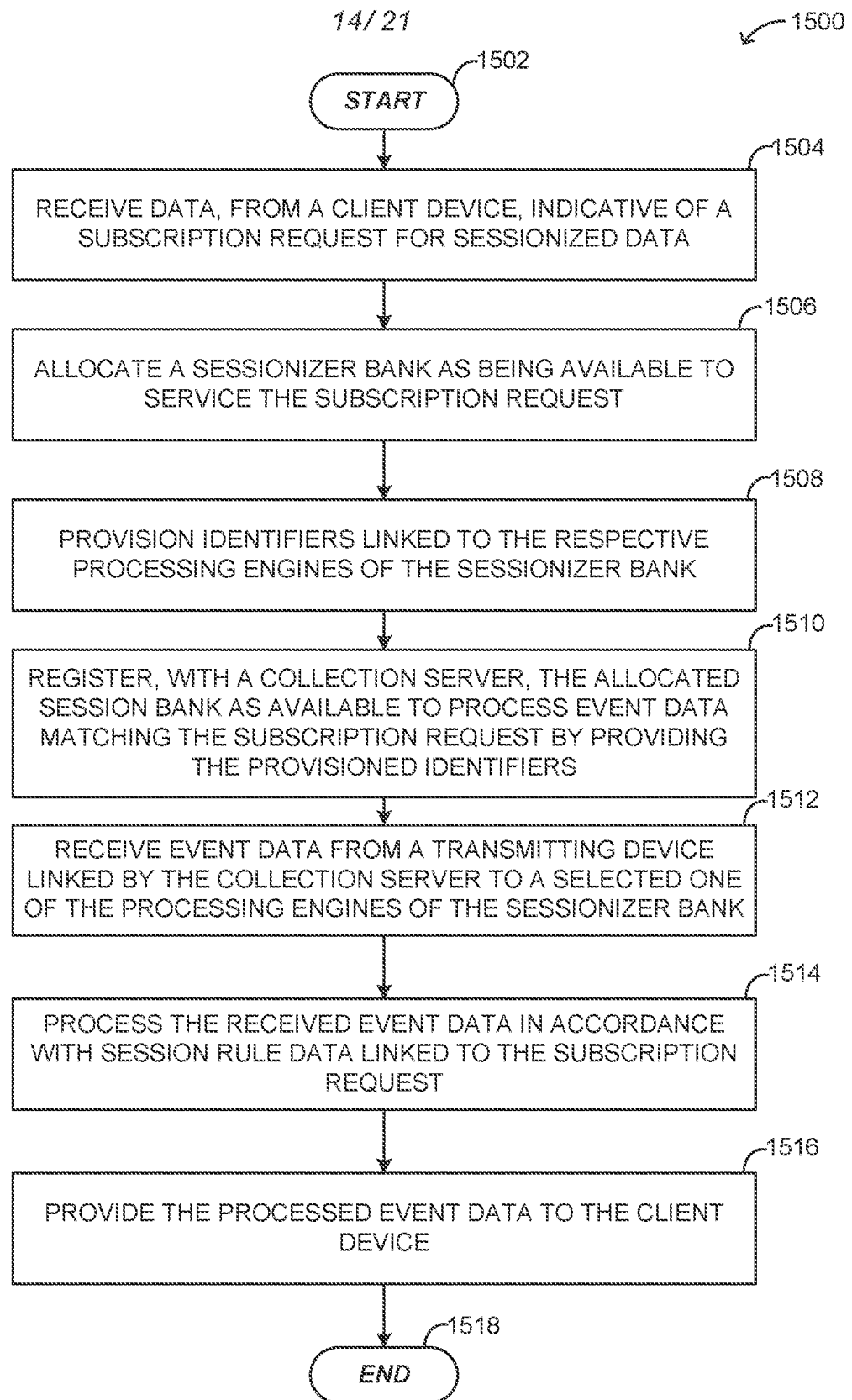
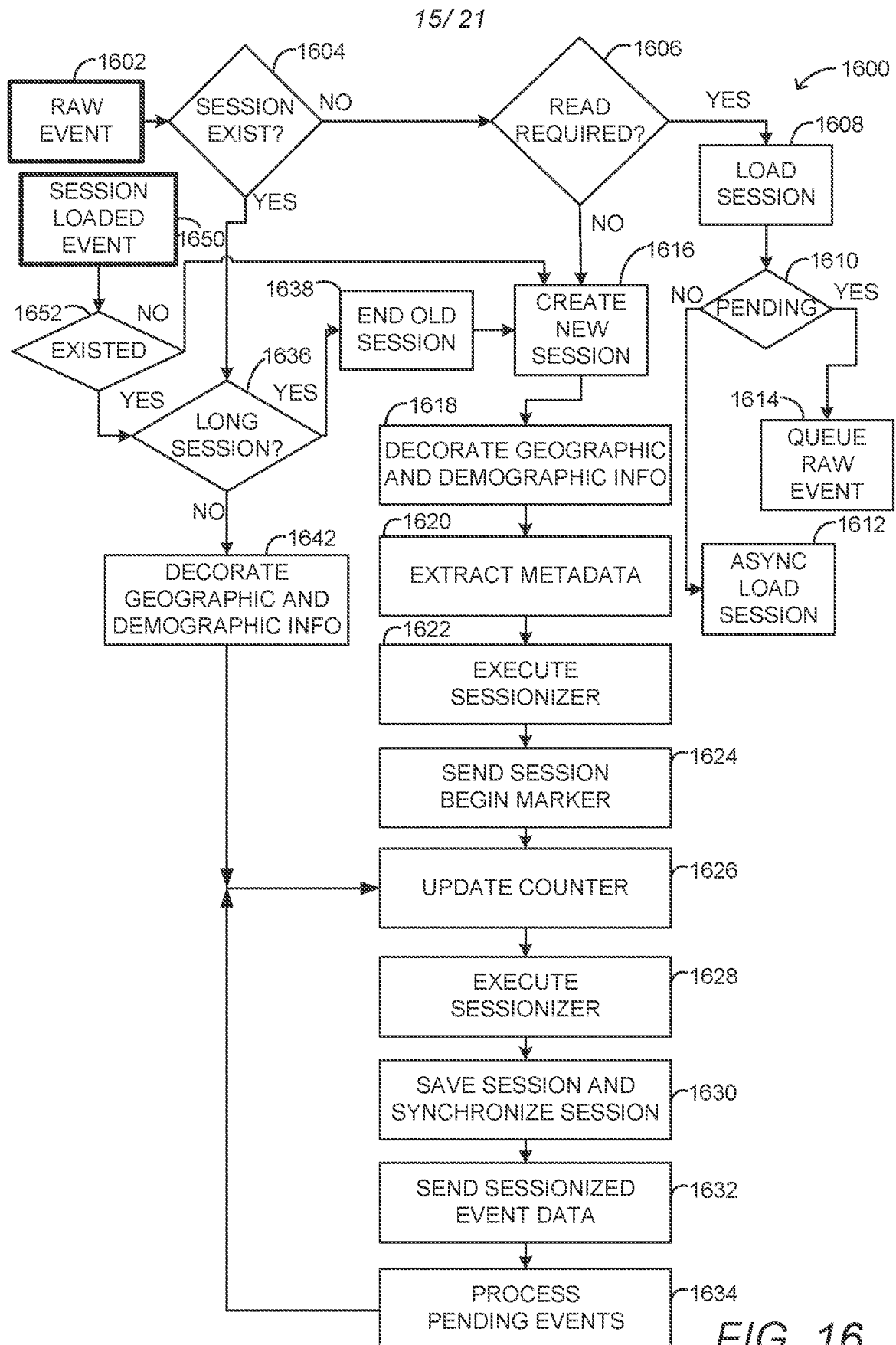


FIG. 15



16/21

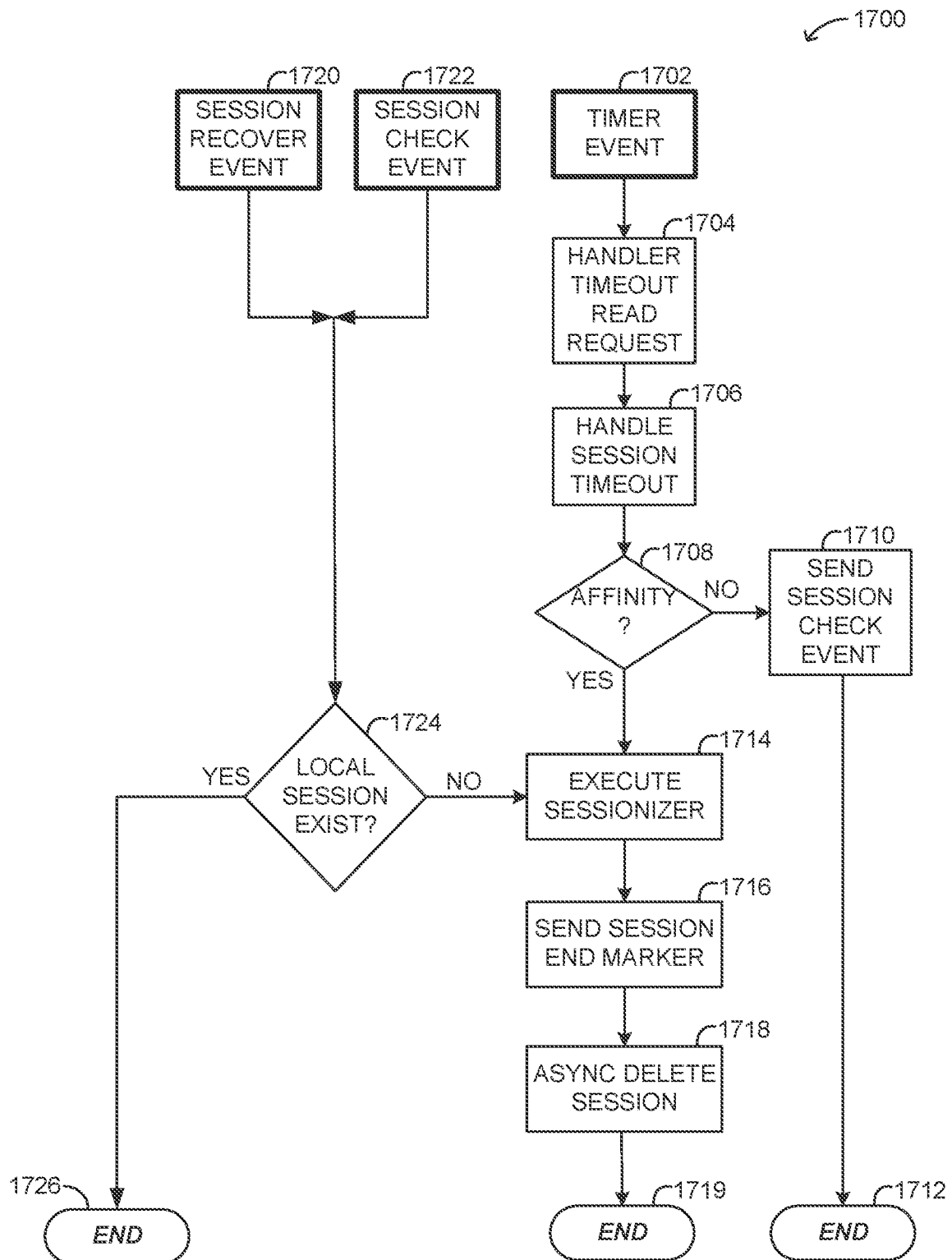


FIG. 17

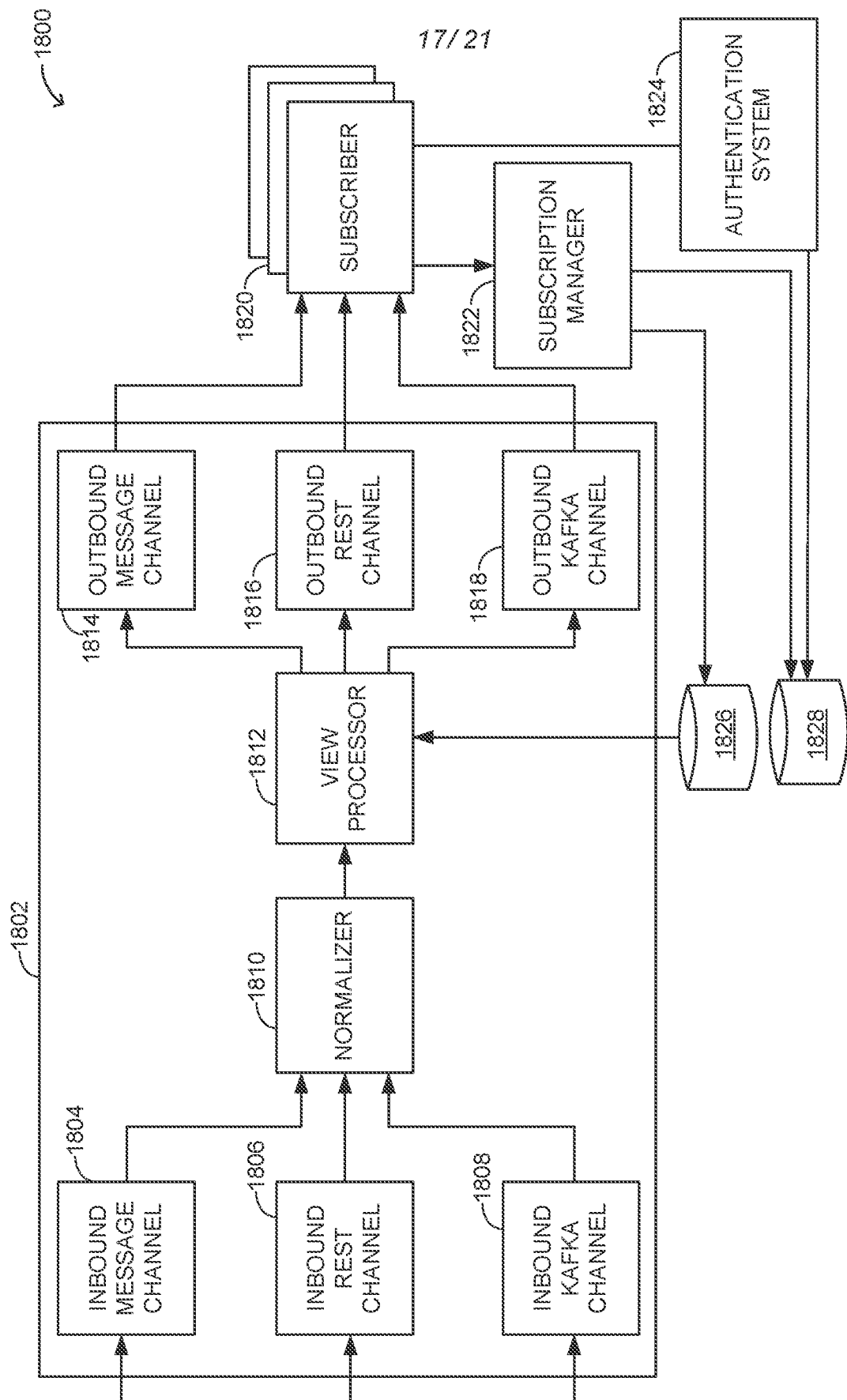


FIG. 18

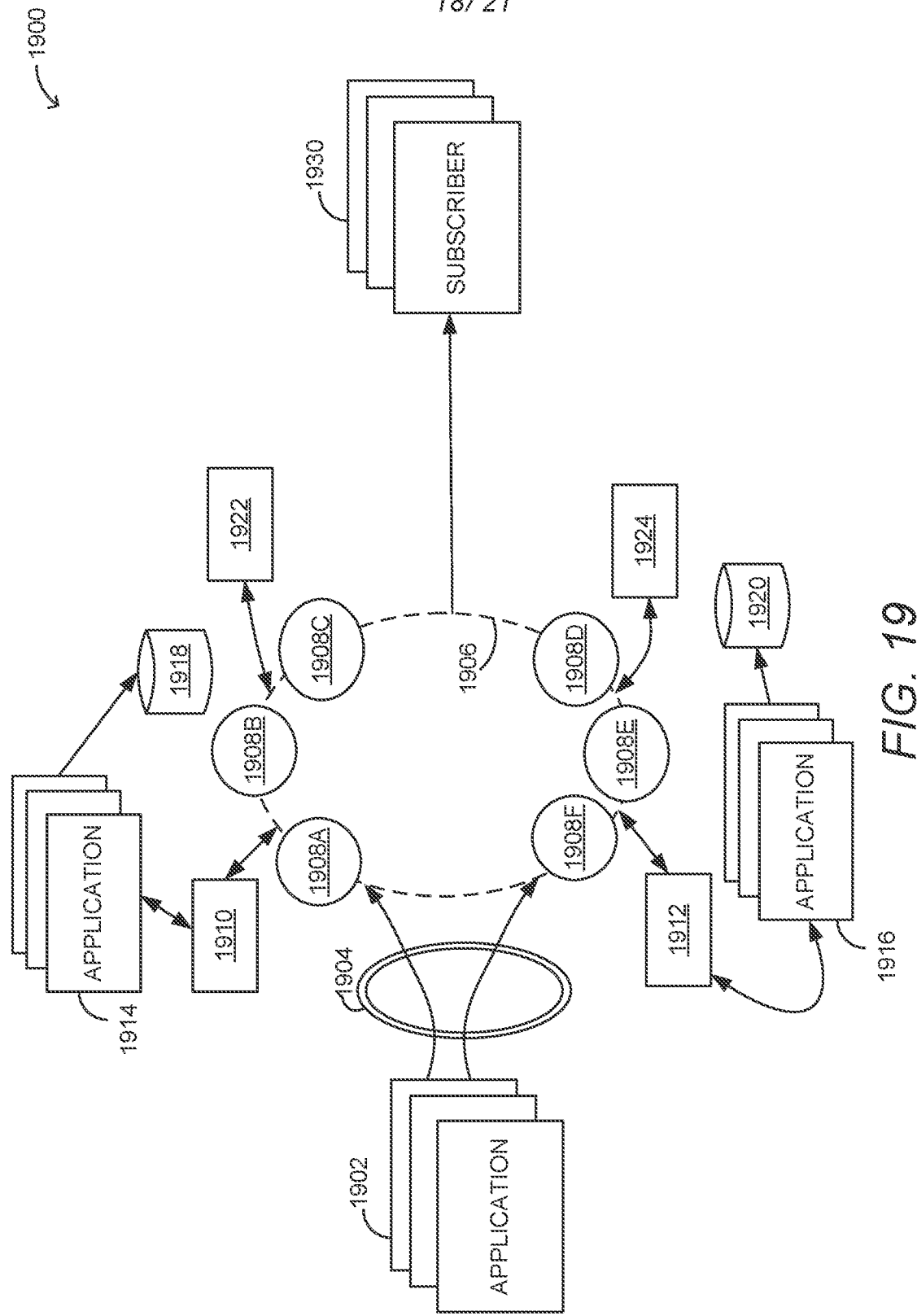


FIG. 19

19/21

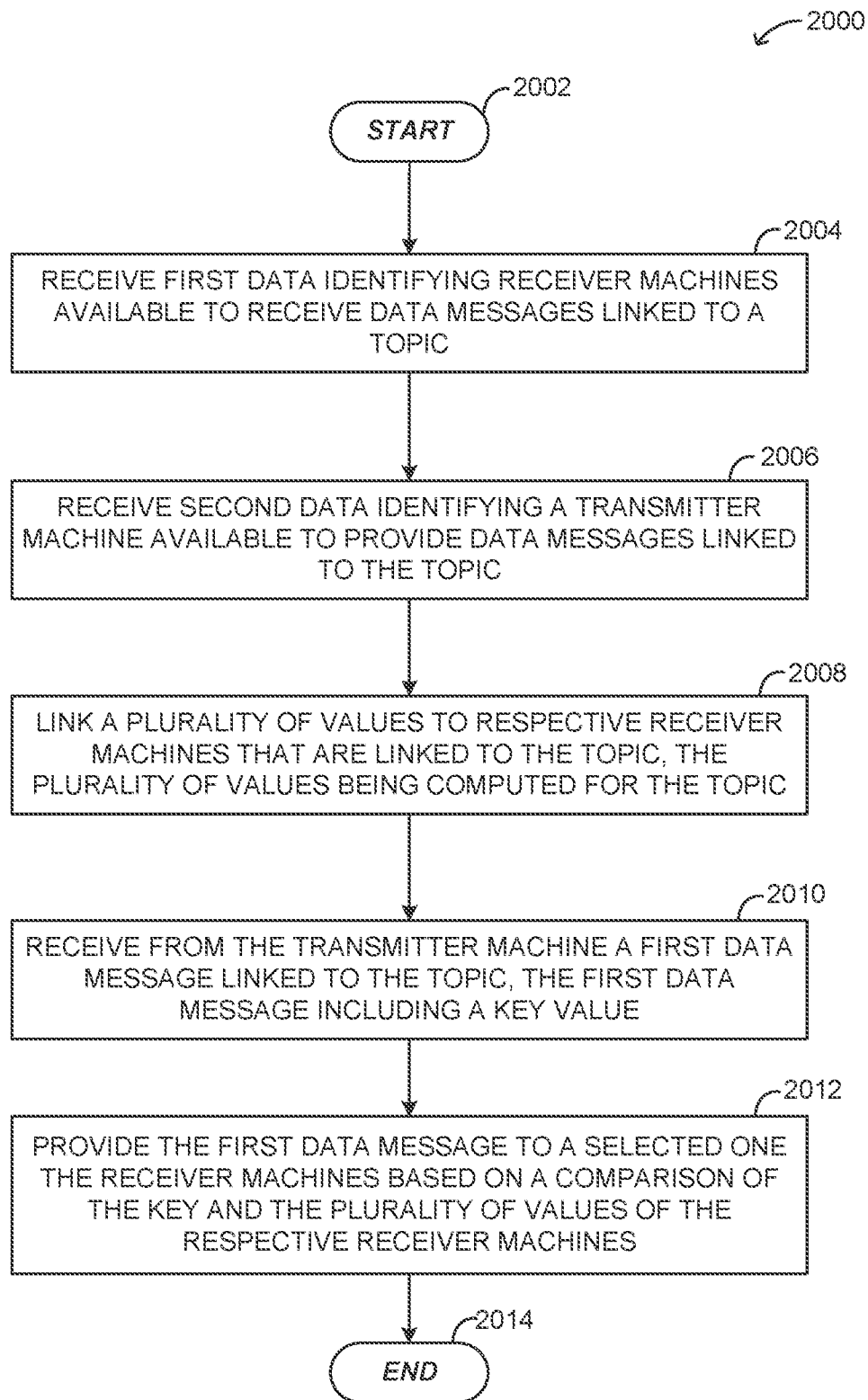


FIG. 20

20/21

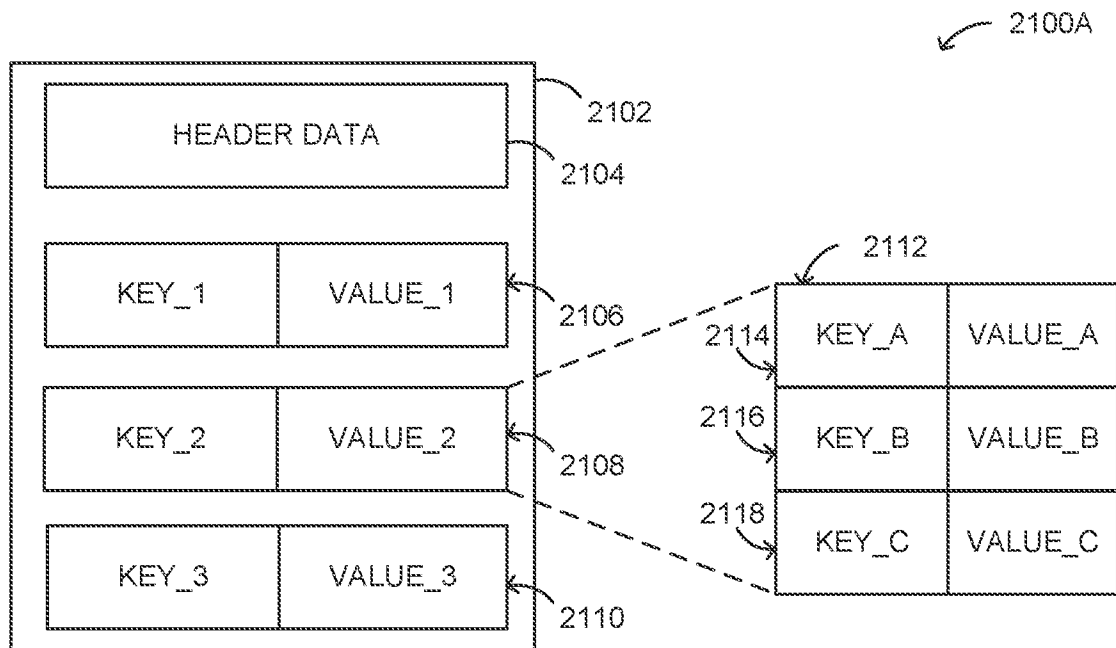


FIG. 21A

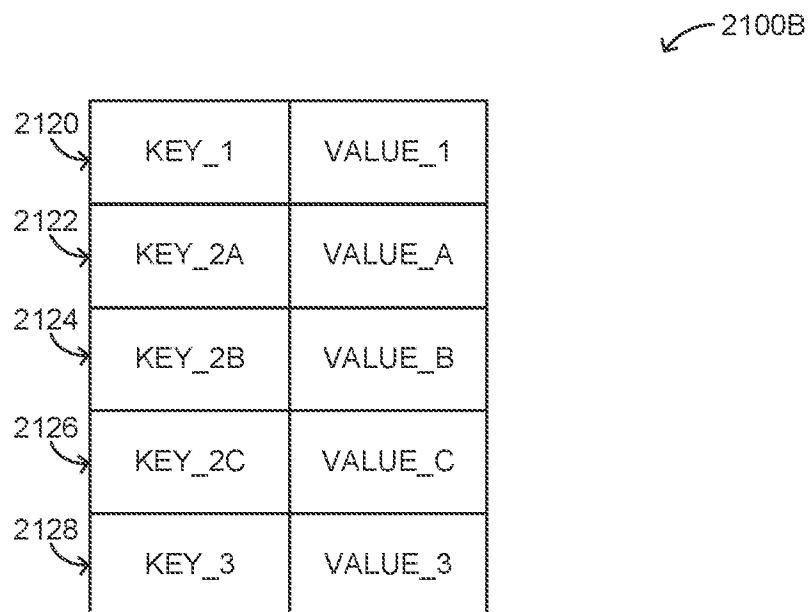


FIG. 21B

21/21

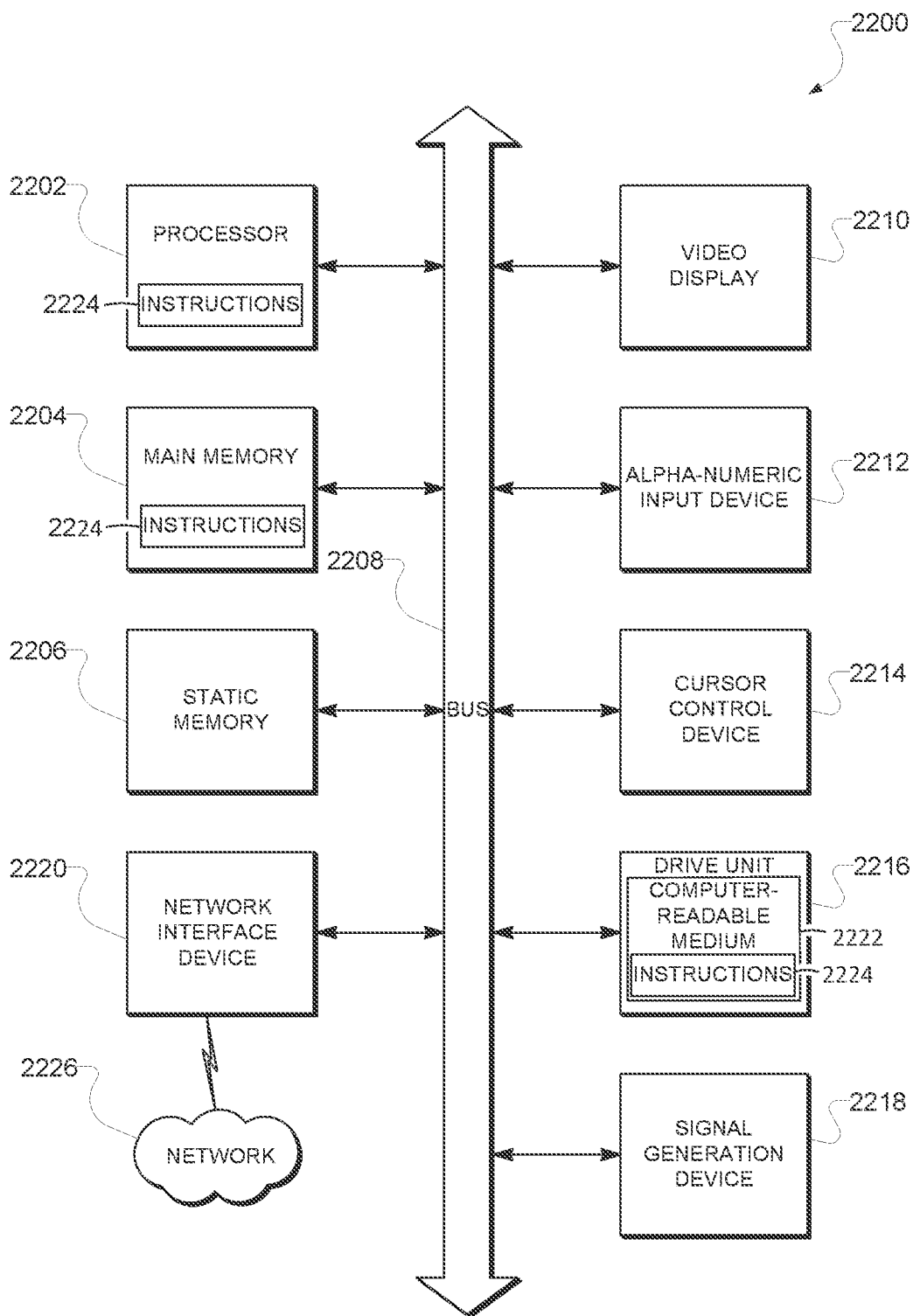


FIG. 22

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US2016/014547

A. CLASSIFICATION OF SUBJECT MATTER

IPC(8) - G06F17/30 (2016.01)

CPC - G06F17/3089 (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) - G06F15/16, /173, /177; F17/00, /20, /28, /30; F19/00; F3/00, /048; F7/00; Q40/00; G10L15/02, /14; H04L12/18 (2016.01)
CPC - G06F17/28, /30064, /3033, /30516, /30545, /30719, /3089; F19/322, /325, /327, /328; F21/6254, /6263; Q10/10; (2016.01)

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

US: 704/236, /239, /245, /251; 705/3, /35, /36.00R; 707/747, /803, /829, /956; 709/201, /204, /207; 715/201, /704, /751, /853
Q40/00, /04; Q50/01, /22, /24; G10L15/02, /14; G11B27/105, /11, /28; H04L12/1859, /1895, /5694; L41/5093 (keyword delimited)

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

Orbit, Google Patents, Google Scholar, Google

Search terms used: messages, event streaming, client, device, subscription, table, view, partial, summary, nested, flatten, esper

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 8,296,303 B2 (NAVAS) 23 October 2012 (23.10.2012) entire document	1, 3-11, 13-16, 18-20

Y		2, 12, 17
Y	US 2007/0106537 A1 (MOORE) 10 May 2007 (10.05.2007) entire document	2, 12, 17
A	US 6,662,206 B1 (BANAVAR et al) 09 December 2003 (09.12.2003) entire document	1-20
A	US 2014/0172427 A1 (ROBERT BOSCH GMBH) 19 June 2014 (19.06.2014) entire document	1-20
A	US 8,275,816 B1 (PEGG) 25 September 2012 (25.09.2012) entire document	1-20
A	US 7,752,552 B2 (PENNINGTON et al) 06 July 2010 (06.07.2010) entire document	1-20
P,A	US 9,166,892 B1 (PRADO et al) 20 October 2015 (20.10.2015) entire document	1-20
A	US 8,768,805 B2 (TAYLOR et al) 01 July 2014 (01.07.2014) entire document	1-20

☐ 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

17 March 2016

Date of mailing of the international search report

24 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