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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
11 June 2015 (11.06.2015)

WIPO | PCT

(10) International Publication Number
WO 2015/085103 A1

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

(21) International Application Number:
PCT/US2014/068641

(22) International Filing Date:
4 December 2014 (04.12.2014)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

| | | |
|---|---|---|
| 61/912,344 | 5 December 2013 (05.12.2013) | US |
| 14/559,550 | 3 December 2014 (03.12.2014) | US |

(71) Applicant: **ORACLE INTERNATIONAL CORPORATION** [US/US]; 500 Oracle Parkway, M/S 5OP7, Redwood Shores, California 94065-1677 (US).

(72) Inventors: **KALI, Prathab**; 899, First Floor, 10th cross, 22nd Main, JP Nagar 2nd Phase, Bangalore, Karnataka 560078 (IN). **SRINIVASAN, Anand**; E 303 Mantri Elegance, N.S. Palya, Bannerghatta Road, Bangalore, Karnataka 560076 (IN). **BISHNOI, Sandeep**; 29-MITC Colony, Barnala Road, Sirsa, Haryana 125055 (IN).

(74) Agents: **SRIPATHY, Kanchan** et al.; Kilpatrick Townsend & Stockton LLP, Two Embarcadero Center, 8th Floor, San Francisco, California 94111 (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, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

**Published**:

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

# (54) Title: PATTERN MATCHING ACROSS MULTIPLE INPUT DATA STREAMS

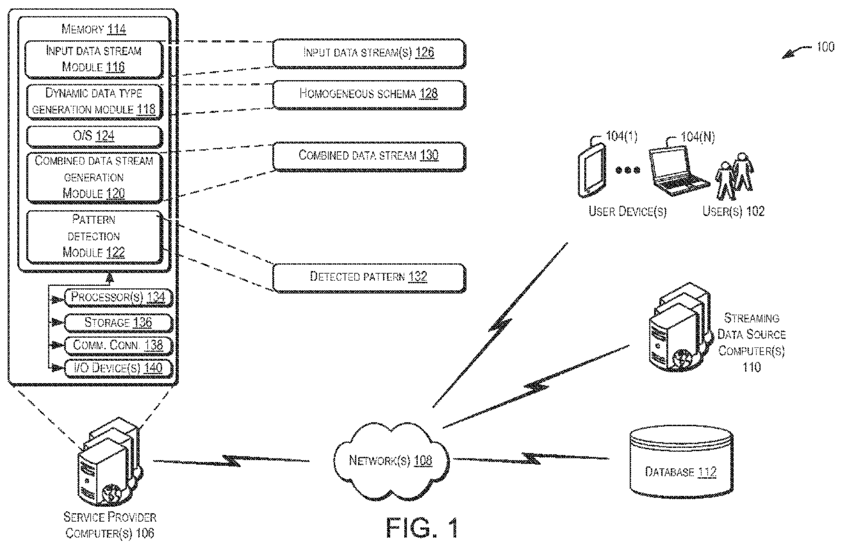

FIG. 1

(57) **Abstract**: A method for detecting patterns across multiple input data streams related to one or more applications is disclosed. The method includes receiving multiple input data streams and generating one or more dynamic data types for one or more attributes of the input data streams. In some embodiments, the method may include combining the input data streams to generate a combined input data stream based on the dynamic data types and processing a continuous query over the combined data stream to detect a pattern.

# PATTERN MATCHING ACROSS MULTIPLE INPUT DATA STREAMS

## CROSS-REFERENCES TO RELATED APPLICATIONS

**[0001]** This application claims priority to U.S. Patent Application 14/559,550, filed December 3, 2014, entitled "PATTERN MATCHING ACROSS MULTIPLE INPUT DATA STREAMS," and to U.S. Provisional Patent Application No. 61/912,344, filed on December 5, 2013, entitled "PATTERN MATCHING ACROSS MULTIPLE INPUT DATA STREAMS," the entire contents of each are incorporated by reference in their entirety for all purposes.

## BACKGROUND

**[0002]** In traditional database systems, data is stored in one or more databases usually in the form of tables. The stored data is then queried and manipulated using a data management language such as a structured query language (SQL). For example, a SQL query may be defined and executed to identify relevant data from the data stored in the database. A SQL query is thus executed on a finite set of data stored in the database. Further, when a SQL query is executed, it is executed once on the finite data set and produces a finite static result. Databases are thus best equipped to run queries over finite stored data sets.

**[0003]** A number of modern applications and systems however generate data in the form of continuous data or event streams instead of a finite data set. Examples of such applications include but are not limited to sensor data applications, financial tickers, network performance measuring tools (e.g. network monitoring and traffic management applications), clickstream analysis tools, automobile traffic monitoring, and the like. Such applications have given rise to a need for a new breed of applications that can process the data streams. For example, a temperature sensor may be configured to send out temperature readings continuously.

**[0004]** Managing and processing data for these types of event stream-based applications involves building data management and querying capabilities with a strong temporal focus. A different kind of querying mechanism is needed that comprises long-running queries over continuous unbounded sets of data. While some vendors now offer product suites geared towards event streams processing, these product offerings still lack the processing flexibility required for handling today's events processing needs.

## SUMMARY

**[0005]** In certain embodiments, techniques are provided (e.g., a method, a system, a non-transitory computer-readable medium storing code or instructions executable by one or more processors) for detecting patterns across multiple input data streams related to one or more applications.

**[0006]** In accordance with one embodiment, a method for detecting patterns across multiple input data streams related to one or more applications is disclosed. The method includes receiving a plurality of input data streams comprising a first input data stream and a second input data stream. The method further includes generating a first dynamic data type for the first input data stream and generating a second dynamic data type for the second input data stream. In some embodiments, the first dynamic data type may be generated by identifying a first attribute of the first input data stream as not being present in the second data stream. The first dynamic data type is then generated for the first attribute. The second dynamic data type may be generated by identifying a second attribute of the second input data stream as not being present in the first data stream. The second dynamic data type is then generated for the second attribute. In an embodiment, the first dynamic data type may be configured to store a first data value corresponding to the first attribute of the first input data stream and the second dynamic data type may be configured to store a second data value corresponding to the second attribute of the second input data stream.

**[0007]** In some embodiments, the method may include combining the first input data stream and the second input data stream to generate a combined data stream based on the first dynamic data type and the second dynamic data type. In an embodiment, the method may then include processing a continuous query over the combined data stream to detect a pattern. In some embodiments, a 'pattern' may constitute an occurrence of a first event in a first data stream followed by the occurrence of another event in a second data stream.

**[0008]** In accordance with another embodiment, a system for detecting patterns across multiple input data streams related to one or more applications is disclosed. The system includes a memory for storing a plurality of instructions and a processor configured to access the memory. In an embodiment, the processor is configured to execute instructions to receive a continuous query identifying a first input data stream and a second input data stream. The processor is further configured to execute instructions to identify a first dynamic data type for a first attribute of the first input data stream and a second dynamic data type for a second attribute of the second input data stream. In some embodiments, the processor is configured to generate a combined data stream based on the first dynamic data type and the second dynamic data type and execute the continuous query over the combined data stream to detect a pattern.

**[0009]** In accordance with some embodiments, a non-transitory computer-readable media storing computer executable instructions executable by one or more processors is disclosed. The computer-executable instructions comprise instructions that cause the one or more processors to receive a plurality of input data streams comprising at least a first input data stream and a second input data stream. The computer-executable instructions further comprise instructions that cause the one or more processors to generate a first dynamic data type for the first input data stream and a second dynamic data type for the second input data stream. In some embodiments, the computer-executable instructions comprise instructions to combine the first input data stream and the second input data stream to generate a combined data stream based on the first dynamic data type and the second dynamic data type and process a continuous query over the combined data stream to detect a pattern.

**[0010]** In accordance with some embodiments, a method is disclosed that comprises receiving a plurality of input data streams comprising at least a first input data stream and a second input data stream; generating a first dynamic data type for the first input data stream; generating a second dynamic data type for the second input data stream; combining the first input data stream and the second input data stream to generate a combined data stream based at least in part on the first dynamic data type and the second dynamic data type; and processing a continuous query over the combined data stream to detect a pattern.

**[0011]** In some embodiments, the step of generating the first dynamic data type further comprises identifying a first attribute of the first input data stream as not being present in the second data stream; and generating the first dynamic data type for the first attribute, the first dynamic data type configured to store a first data value corresponding to the first attribute of the first input data stream.

**[0012]** In some embodiments, the step of generating the second dynamic data type further comprises identifying a second attribute of the second input data stream as not being present in first data stream; and generating the second dynamic data type for the second attribute, the second dynamic data type configured to store a second data value corresponding to the second attribute of the second input data stream.

**[0013]** In some embodiments, the method further comprises identifying a common attribute, the common attribute identified as being present in the first input data stream and being present in the second input data stream; generating a homogeneous schema, the homogeneous schema including a representation of one or more attributes of the first input data stream and the second input data stream, the representation including at least the common attribute, the first dynamic data type and the second dynamic data type; and generating the combined data stream based at least in part on the homogeneous schema.

**[0014]** In some embodiments, the method further comprises detecting the pattern based at least in part on analyzing the combined data stream, wherein the pattern identifies a first event in the first input data stream followed by a second event in the second input data stream.

**[0015]** In accordance with some embodiments, a service provider device is provided. The service provider device may comprise an input data stream receiving unit , configured to receive a plurality of input data streams comprising at least a first input data stream and a second input data stream; a first dynamic data type generation unit, configured to generate a first dynamic data type for the first input data stream; a second dynamic data type generation unit, configured to generate a second dynamic data type for the second input data stream; combined data stream generation unit, configured to combine the first input data stream and the second input data stream to generate a combined data stream based at least in part on the first dynamic data type and the second dynamic data type; and a pattern detection unit, configured to process a continuous query over the combined data stream to detect a pattern.

**[0016]** In some embodiments, the first dynamic data type generation unit is further configured to: identify a first attribute of the first input data stream as not being present in the second data stream; and generate the first dynamic data type for the first attribute.

**[0017]** In some embodiments, the first dynamic data type is configured to store a first data value corresponding to the first attribute of the first input data stream.

**[0018]** In some embodiments, the second dynamic data type generation unit is further configured to identify a second attribute of the second input data stream as not being present in first data stream; and generate the second dynamic data type for the second attribute, the second dynamic data type configured to store a second data value corresponding to the second attribute of the second input data stream.

**[0019]** In some embodiments, the service provider device may further comprise a common attribute identification unit , configured to identify a common attribute, the common attribute identified as being present in the first input data stream and being present in the second input data stream; and homogeneous schema generation unit , configured to generate a homogeneous schema, the homogeneous schema including a representation of one or more attributes of the first input data stream and the second input data stream, the representation including at least the common attribute, the first dynamic data type and the second dynamic data type.

**[0020]** In some embodiments, the homogeneous schema comprises at least one of a stream name identifier attribute, a first timestamp attribute associated with the first input data stream or a second timestamp attribute associated with the second input data stream.

**[0021]** In some embodiments, the combined data stream generation unit may be further configured to select a first set of tuples from the first data stream, the first input data stream identified by the homogeneous schema; select a second set of tuples from the second input data stream, the second input data stream identified by the homogeneous schema; and process a sub-query over the first set of tuples and the second set of tuples to generate the combined data stream.

**[0022]** In some embodiments, the pattern may be detected based at least in part on analyzing the combined data stream, wherein the pattern identifies a first event in the first input data stream followed by a second event in the second input data stream.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0023]** The detailed description is set forth with reference to the accompanying figures. In the figures, the left-most digit(s) of a reference number identifies the FIG. in which the reference number first appears. The use of the same reference numbers in different FIGS. indicates similar or identical items.

**[0024]** FIG. 1 depicts a simplified example system or architecture in which techniques for identifying patterns across multiple input data streams may be implemented.

**[0025]** FIG. 2 illustrates a simplified block diagram with which features for the detection of patterns across multiple input data streams may be described.

**[0026]** FIG. 3 is an exemplary illustration of performing pattern recognition using a CQL query that identifies multiple input data streams, in accordance with one embodiment of the present disclosure.

**[0027]** FIG. 4 is an exemplary illustration of performing pattern recognition using a CQL query that identifies multiple input data streams, in accordance with another embodiment of the present disclosure.

**[0028]** FIG. 5 is a high level flowchart depicting a process for detecting patterns across multiple input data streams, in accordance with one embodiment of the present disclosure.

**[0029]** FIG. 6 is a high level flowchart depicting a process for generating a homogenous schema, in accordance with one embodiment of the present disclosure.

**[0030]** FIG. 7 depicts a simplified high level diagram of an event processing system that may incorporate an embodiment of the present disclosure.

**[0031]** FIG. 8 depicts a simplified diagram of a distributed system for implementing one of the embodiments.

**[0032]** FIG. 9 is a simplified block diagram of one or more components of a system environment by which services provided by one or more components of an embodiment system may be offered as cloud services, in accordance with an embodiment of the present disclosure.

**[0033]** FIG. 10 illustrates an exemplary computer system, in which various embodiments of the present invention may be implemented.

**[0034]** FIG. 11 illustrates a simplified block diagram of an exemplary service provider device, in which various embodiments of the present invention may be implemented.

## DETAILED DESCRIPTION

**[0035]** In the following description, various embodiments will be described. For purposes of explanation, specific configurations and details are set forth in order to provide a thorough understanding of the embodiments. However, it will also be apparent to one skilled in the art that the embodiments may be practiced without the specific details. Furthermore, well-known features may be omitted or simplified in order not to obscure the embodiment being described.

**[0036]** In some applications, data may take the form of continuous, unbounded data streams, rather than finite stored data sets. Examples of such data streams may include stock tickers in financial applications, performance measurements in network monitoring and traffic management, log records or click-streams in web tracking and personalization, data feeds from sensor applications, network packets and messages in firewall-based security, call detail records in telecommunications, and the like. Due to their continuous nature, these data streams may typically be queried using continuous queries rather than traditional one-time SQL queries.

**[0037]** In general, a continuous data stream (also referred to as an event stream) may include a stream of data or events that may be continuous or unbounded in nature with no explicit end. Logically, an event or data stream may be a sequence of data elements (also referred to as events), each data element having an associated timestamp. A continuous event stream may be logically represented as a bag or set of elements (s, T), where "s" represents the data portion, and "T" is in the time domain. The "s" portion is generally referred to as a tuple or event. An event stream may thus be a sequence of time-stamped tuples or events.

**[0038]** In some aspects, the timestamps associated with events in a stream may equate to a clock time. In other examples, however, the time associated with events in an event stream may be defined by the application domain and may not correspond to clock time but may, for example, be represented by a sequence of numbers instead. Accordingly, the time information associated with an event in an event stream may be represented by a number, a timestamp, or any other information that represents a notion of time. For a system receiving an input event stream, the events arrive at the system in the order of increasing timestamps. There could be more than one event with the same timestamp.

**[0039]** In some examples, an event in an event stream may represent an occurrence of some worldly event (e.g., when a temperature sensor changed value to a new value, when the price of a stock symbol changed, etc.) and the time information associated with the event may indicate when the worldly event represented by the data stream event occurred.

**[0040]** For events received via an event stream, the time information associated with an event may be used to ensure that the events in the event stream arrive in the order of increasing timestamp values. This may enable events received in the event stream to be ordered based upon their associated time information. In order to enable this ordering, timestamps may be associated with events in an event stream in a non-decreasing manner such that a later-generated event has a later timestamp than an earlier-generated event. As another example, if sequence numbers are being used as time information, then the sequence number associated with a later-generated event may be greater than the sequence number associated with an earlier-generated event. In some examples, multiple events may be associated with the same timestamp or sequence number, for example, when the worldly events represented by the data stream events occur at the same time. Events belonging to the same event stream may generally be processed in the order imposed on the events by the associated time information, with earlier events being processed prior to later events.

**[0041]** The time information (e.g., timestamps) associated with an event in an event stream may be set by the source of the stream or alternatively may be set by the system receiving the stream. For example, in certain embodiments, a heartbeat may be maintained on a system receiving an event stream, and the time associated with an event may be based upon a time of arrival of the event at the system as measured by the heartbeat. It is possible for two events in an event stream to have the same time information. It is to be noted that while timestamp ordering is specific to one event stream, events of different streams could be arbitrarily interleaved.

**[0042]** An event stream may have an associated schema "S," the schema comprising time information and a set of one or more named attributes. All events that belong to a particular event stream conform to the schema associated with that particular event stream. Accordingly, for an event stream (s, T), the event stream may have a schema 'S' as (<time_stamp>, <attribute(s)>), where <attributes> represents the data portion of the schema and can comprise one or more attributes. For example, the schema for a stock ticker event stream may comprise attributes <stock symbol>, and <stock price>. Each event received via such a stream will have a time stamp and the two attributes. For example, the stock ticker event stream may receive the following events and associated timestamps:

```
...
(<timestamp_N>, <NVDA,4>)
(<timestamp_N+1>, <ORCL,62>)
(<timestamp_N+2>, <PCAR,38>)
(<timestamp_N+3>, <SPOT,53>)
(<timestamp_N+4>, <PDCO,44>)
(<timestamp_N+5>, <PTEN,50>)
...
```

In the above stream, for stream element (<timestamp_N+1>, <ORCL,62>), the event is <ORCL,62> with attributes "stock_symbol" and "stock_value." The timestamp associated with the stream element is "timestamp_N+1". A continuous event stream is thus a flow of events, each event having the same series of attributes.

**[0043]** As noted, a stream may be the principle source of data that CQL queries may act on. A stream S may be a bag (also referred to as a “multi-set”) of elements (s, T), where “s” is in the schema of S and “T” is in the time domain. Additionally, stream elements may be tuple-timestamp pairs, which can be represented as a sequence of timestamped tuple insertions. In other words, a stream may be a sequence of timestamped tuples. In some cases, there may be more than one tuple with the same timestamp. And, the tuples of an input data stream may be requested to arrive at the system in order of increasing timestamps. Alternatively, a relation (also referred to as a “time varying relation,” and not to be confused with “relational data,” which may include data from a relational database) may be a mapping from the time domain to an unbounded bag of tuples of the schema R. In some examples, a relation may be an unordered, time-varying bag of tuples (i.e., an instantaneous relation). In some cases, at each instance of time, a relation may be a bounded set. It can also be represented as a sequence of timestamped tuples that may include insertions, deletes, and/or updates to capture the changing state of the relation. Similar to streams, a relation may have a fixed schema to which each tuple of the relation may conform. Further, as used herein, a continuous query may generally be capable of processing data of (i.e., queried against) a stream and/or a relation. Additionally, the relation may reference data of the stream.

**[0044]** In some examples, business intelligence (BI) may help drive and optimize business operations at particular intervals (e.g., on a daily basis in some cases). This type of BI is usually called operational business intelligence, real-time business intelligence, or operational intelligence (OI). Operational Intelligence, in some examples, blurs the line between BI and business activity monitoring (BAM). For example, BI may be focused on periodic queries of historic data. As such, it may have a backward-looking focus. However, BI may also be placed into operational applications, and it may therefore expand from a mere strategic analytical tool into the front lines in business operations. As such, BI systems may also be configured to analyze event streams and compute aggregates in real time.

**[0045]** In some examples, a continuous query language service (CQ Service) may be configured to extend a BI analytics server to handle continuous queries and enable real-time alerts. The CQ Service, in some aspects, may provide integration with a BI analytics server and a CQL engine. By way of example only, a BI analytics server may delegate continuous queries to the CQ Service and the CQ Service may also act as a logical database (DB) gateway for a CQL engine. In this way, the CQL engine may be able to leverage the BI analytics server for its analytics capabilities and semantic modeling.

**[0046]** In some examples, the CQ Service may provide, among other things, the following functionalities:

- Remoting service for BI Analytics Server as CQL engine Gateway;
- Event source/sink adapter;
- Generate data definition languages (DDLs) from logical SQL plus CQL extensions;

- Provide unified model for all types of continuous queries and implementation selections;
- Maintain metadata and support restartability; and
- High availability and scalability support.

**[0047]** Additionally, in some examples, OI is a form of real-time dynamic, business analytics that can deliver visibility and insight into business operations. OI is often linked to or compared with BI or real-time BI, in the sense that both help make sense out of large amounts of information. But there are some basic differences: OI may be primarily activity-centric, whereas BI may be primarily data-centric. Additionally, OI may be more appropriate for detecting and responding to a developing situation (e.g., trend and pattern), unlike BI which may traditionally be used as an after-the-fact and report-based approach to identifying patterns.

**[0048]** In some examples, a business event analysis and monitoring (BEAM) system may include a CQL engine to process and/or receive in-flight data. For example, a CQL engine may be an in-memory real-time event processing engine configured to query or otherwise process incoming real-time information (e.g., BI or OI). The CQL engine may utilize or understand temporal semantics and be configured to allow definition of a window of data to process. Utilizing a CQL engine may, in some cases, involve always running a query on incoming data.

**[0049]** In some aspects, the CQL engine may include a full blown query language. As such, a user may specify computations in terms of a query. Additionally, the CQL engine may be designed for optimizing memory, utilizing query language features, operator sharing, rich pattern matching, rich language constructs, etc. Additionally, in some examples, the CQL engine may process both historical data and streaming data. For example, a user can set a query to send an alert when California sales hit above a certain target. Thus, in some examples, the alert may be based at least in part on historical sales data as well as incoming live (i.e., real-time) sales data.

**[0050]** In some examples, the CQL engine or other features of the below described concepts may be configured to combine a historical context (i.e., warehouse data) with incoming data in a real-time fashion. Thus, in some cases, the present disclosure may describe the boundary of database stored information and in-flight information. Both the database stored information and the inflight information may include BI data. As such, the database may, in some examples, be a

BI server or it may be any type of database. Further, in some examples, the features of the present disclosure may enable the implementation of the above features without users knowing how to program or otherwise write code. In other words, the features may be provided in a feature-rich user interface (UI) or other manner that allows non-developers to implement the combination of historical data with real-time data.

**[0051]** In certain embodiments, events received in a continuous data stream may be processed at run time to detect occurrences of a specified pattern in the data stream. A 'pattern' may constitute a sequence of consecutive events or tuples in the continuous data stream, each satisfying certain conditions. As an example, an occurrence of an event such as a change in the trading volume of a stock that results in the occurrence of another event such as a change in pricing of the stock value and may constitute a 'pattern' in a continuous data stream that receives 'stock tick' events related to a financial application.

**[0052]** In the context of multiple continuous data streams, a 'pattern' may constitute an occurrence of a first event in a first data stream followed by the occurrence of another event in a second data stream, and so on. As an example, consider a first business process that is driven by a first sequence of events, wherein the events relate to a car rental reservation application. Similarly, consider a second business process that is driven by a second sequence of events, wherein the events relate to a flight reservation application. Additionally, assume that the first sequence of events is received via a first continuous input data stream and that the second sequence of events is received via a second continuous input data stream. In such a situation, it may be desirable to output a pattern match when the arrival of an event (e.g., a car reservation by a user) in the first continuous input data stream is followed by the arrival of another event (e.g., a flight ticket issuance to the user) in the second continuous input data stream.

**[0053]** In one embodiment, pattern matching across multiple continuous input data streams may be performed by applying a continuous query (e.g., a CQL query) to the incoming input data streams. In one approach, pattern matching across multiple continuous input data streams may be performed by first performing a UNION of all or a subset of all of the relevant input data streams over which pattern matching is to be performed with the result defining a view corresponding to an intermediate stream. The pattern to be matched can then be specified over this single intermediate stream. In one embodiment, the CQL language construct MATCH_RECOGNIZE clause may be used for performing pattern recognition in a CQL query that identifies multiple input data streams. The pattern may then be matched to all the streams included in the view.

**[0054]** As an example, consider the following CQL query, Q1, wherein Q1 is a continuous query that specifies a pattern to be matched across a first input data stream S1 and a second input data stream, S2.

```
Q1:
SELECT *
FROM
(SELECT cleintId, p1, -1L as p2 from S1 UNION ALL SELECT clientId, -1L as
p1, p2 from S1) AS S MATCH_RECOGNIZE
(
  PARTITION BY clientId
  MEASURES
     A.p1 as p1,
B.P2 AS P2
PATTERN (A b)
DEFINE
A as (P1 !=-1l)
b as (P2 !=-1l)
)
```

**[0055]** Additionally, assume that stream S1 is defined by a first schema: S1(int p1, int clientId) and stream S2 is defined by a second schema: S2(int p2, int clientId), where p1, p2 and clientId correspond to one or more attributes of stream S1 and S2.

**[0056]** Query Q1 comprises a FROM clause that specifies a UNION of the streams S1 and S2. In order for the UNION ALL query to combine the result sets of the streams S1 and S2, each CQL SELECT statement within the CQL UNION ALL query is typically required to have the same number of fields in the results sets with similar data types. In the example of query Q1 shown above, the schema of stream S1 is different from the schema of stream S2. In one approach, the streams S1 and S2 may be combined by normalizing the schemas of streams S1 and S2.

**[0057]** In one example, the normalization of the schemas may be performed by adding an additional column to each of the schemas of streams S1 and S2. In the example of query Q1 shown above, a column is added to each of the schemas of streams S1 and S2 and populated with a hard coded value such as '-1L' to normalize the schemas. However, introducing hard-coded data values into each stream is a manual process which is prone to error since hard-coded values have to be entered each time a CQL query that identifies multiple input data streams with different schemas is processed.

**[0058]** In one embodiment of the present disclosure, a homogeneous schema representing one or more participating input data streams is generated. In some embodiments, the homogeneous schema may be generated by creating one or more dynamic data types for one or more attributes of the participating input data streams. The generation of the dynamic data types and the homogenous schema enables each SELECT statement within the UNION ALL sub-query to include the same number of fields and similar data types so that the result sets of the streams S1 and S2 may be combined using the SELECT statements. Additionally, the generation of the dynamic data type for the attributes of the participating input data streams is performed in real-time, and hard-coded values need not be introduced into the participating input data streams each time a CQL query is processed. In one example, and as will be discussed in detail below, the dynamic data types may refer to a composite data type identified for one or more attributes of the input data streams.

**[0059]** In certain embodiments, the input data streams may then be combined to generate a combined data stream based on the homogenous schema. A continuous query may then be processed over the combined data stream to detect a pattern across the input data streams. In one embodiment, the CQL language construct MATCH_RECOGNIZE clause may be used for performing pattern recognition in a CQL query that identifies multiple input data streams. Additional details of the manner in which the homogeneous schema and the dynamic data types may be generated and utilized to perform pattern recognition in a CQL query that identifies multiple input data streams is discussed in detail in FIGS. 1-4 below.

**[0060]** Using a MATCH_RECOGNIZE clause, a user can define conditions on the attributes of incoming events and identify conditions for pattern matching by using identifiers called correlation variables. As discussed above, a sequence of consecutive events or tuples in an input data stream, each satisfying certain conditions constitutes a pattern. The pattern recognition functionality allows a user to define conditions on the attributes of incoming events or tuples and to identify these conditions by using string names called correlation variables.

**[0061]** In the query Q1 shown above, 'A' and 'B' are the correlation variables. The pattern to be matched is specified as a regular expression over these correlation variables and it determines the sequence or order in which conditions should be satisfied by different incoming events to be recognized as a valid match. A sequence of consecutive events in the input data stream satisfying these conditions constitutes a pattern. In one embodiment, the output of a MATCH_RECOGNIZE query is a stream. In the query Q1 shown above, the MATCH_RECOGNIZE clause also includes several sub-clauses.

**[0062]** The DEFINE sub-clause specifies the Boolean condition for each correlation variable. This may be specified as any logical or arithmetic expression and may apply any single-row or aggregate function to the attributes of events that match a condition. On receiving a new event via the input data stream, the conditions of the correlation variables that are relevant at that point in time are evaluated. An event is said to have matched a correlation variable if it satisfies its defining condition. A particular input can match zero, one, or more correlation variables. The relevant conditions to be evaluated on receiving an input event are determined by the processing logic governed by the PATTERN clause regular expression and the state in the pattern recognition process that has been reached after processing the earlier inputs. The condition can refer to any of the attributes of the schema of the stream or view that evaluates to a stream on which the MATCH_RECOGNIZE clause is being applied. A correlation variable in the PATTERN clause need not be specified in the DEFINE clause: the default for such a correlation variable is a predicate that is always true. Such a correlation variable matches every event.

**[0063]** The PARTITION BY sub-clause specifies the stream attributes by which a MATCH_RECOGNIZE clause should partition its results. Without a PARTITION BY clause, all stream attributes belong to the same partition. When a PARTITION BY clause is present along with pattern matching, the input data stream is logically divided based on the attributes mentioned in the partition list and pattern matching is done within a partition. In the query Q1 shown above, the 'clientId' attribute (which is an attribute that is common to the streams S1 and S2) is specified in the PARTITION BY clause by which the MATCH_RECOGNIZE clause may partition its results.

**[0064]** The MEASURES sub-clause exports (e.g., makes available for inclusion in the SELECT clause) one or more attribute values of events that successfully match the pattern specified and also enables expressions to be specified on those attribute values. This clause may be used to define expressions over attributes of the events in the event stream that match the conditions (correlation variables) in the DEFINE clause and to alias these expressions so that they can suitably be used in the SELECT clause of the main query of which this MATCH_RECOGNIZE condition is a part. The attributes of an event stream may be referred to either directly or via a correlation variable.

**[0065]** The PATTERN sub-clause specifies the pattern to be matched as a regular expression over one or more correlation variables. Incoming events must match these conditions in the order given (from left to right). The regular expression may be composed of correlation variables and pattern qualifiers such as:

* : 0 or more times

+ : 1 or more times

? : 0 or 1 time, etc.

**[0066]** In certain embodiments, the one-character pattern quantifiers shown above are maximal or "greedy"; they will attempt to match as many instances of the regular expression on which they are applied as possible. The pattern quantifiers can also be two characters, which are minimal or "reluctant"; they will attempt to match as few instances of the regular expression on which they are applied as possible. Examples of two character quantifiers include without limitation:

*? : 0 or more times

+? : 1 or more times

?? : 0 or 1 time

**[0067]** As an example of pattern matching, consider the following pattern:

PATTERN (AB*C)

This pattern clause means a pattern match will be recognized when the following conditions are met by consecutive incoming input events:

State 1: Exactly one event tuple matches the condition that defines correlation variable A, followed by

State 2: Zero or more tuples that match the correlation variable B, followed by

State 3: Exactly one tuple that matches correlation variable C.

**[0068]** The states State 1, State 2, and State 3 may represent the various states for the pattern (AB*C), with State 3 being the final state for the pattern. When a pattern match is in a particular state and can either remain in the same particular state or can transition from the particular state to the next state due to the next event, it may imply that the binding can grow. A pattern may be considered matched if the binding is in the final state. While in state 2, if a tuple or event arrives that matches both the correlation variables B and C (since it satisfies the defining conditions of both of them) then as the quantifier * for B is greedy, that tuple may be considered to have matched B instead of C. Accordingly, due to the greedy property B may get preference over C and a greater number of B may be matched. Had the pattern expression be A B*? C, one that uses a lazy or reluctant quantifier over B, then a tuple matching both B and C may be treated as matching C only. Thus, in that example, C may get preference over B and a fewer number of B may be matched.

**[0069]** In the query Q 1 shown above, the pattern (AB) is matched when:

State 1: Exactly one event tuple matches the condition that defines correlation variable A, is followed by

State 2 (Final state): Exactly one tuple that matches correlation variable B.

The states State 1 and State 2 represent the various possible states for the pattern (AB), with State 2 being the final state for the pattern.

**[0070]** The techniques described above and below may be implemented in a number of ways and in a number of contexts. Several example implementations and contexts are provided with reference to the following figures, as described below in more detail. However, the following implementations and contexts are but a few of many.

**[0071]** FIG. 1 depicts a simplified example system or architecture 100 in which techniques for performing pattern matching across multiple input data streams may be implemented. In architecture 100, one or more users 102 (e.g., account holders) may utilize user computing devices 104(1)-(N) (collectively, "user devices 104") to access one or more service provider computers 106 via one or more networks 108. In some aspects, the service provider computers 106 may also be in communication with one or more streaming data source computers 110 and/or one or more databases 112 via the networks 108. For example, the users 102 may utilize the service provider computers 106 to access or otherwise manage data of the streaming data source computers 110 and/or the databases 112 (e.g., queries may be run against either or both of 110, 112). The databases 112 may be relational databases, SQL servers, or the like and may, in some examples, manage historical data, event data, relations, archived relations, or the like on behalf of the users 102. Additionally, the databases 112 may receive or otherwise store data provided by the streaming data source computers 110. In some examples, the users 102 may utilize the user devices 104 to interact with the service provider computers 106 by providing queries (also referred to as "query statements") or other requests for data (e.g., historical event data, streaming event data, etc.). Such queries or requests may then be executed by the service provider computers 106 to process data of the databases 112 and/or incoming data from the streaming data source computers 110. Further, in some examples, the streaming data source computers 110 and/or the databases 112 may be part of an integrated, distributed environment associated with the service provider computers 106.

**[0072]** In some examples, the networks 108 may include any one or a combination of multiple different types of networks, such as cable networks, the Internet, wireless networks, cellular networks, intranet systems, and/or other private and/or public networks. While the illustrated example represents the users 102 accessing the service provider computers 106 over the networks 108, the described techniques may equally apply in instances where the users 102 interact with one or more service provider computers 106 via the one or more user devices 104 over a landline phone, via a kiosk, or in any other manner. It is also noted that the described techniques may apply in other client/server arrangements (e.g., set-top boxes, etc.), as well as in non-client/server arrangements (e.g., locally stored applications, etc.).

**[0073]** The user devices 104 may be any type of computing device such as, but not limited to, a mobile phone, a smart phone, a personal digital assistant (PDA), a laptop computer, a desktop computer, a thin-client device, a tablet PC, etc. In some examples, the user devices 104 may be in communication with the service provider computers 106 via the networks 108, or via other network connections. Further, the user devices 104 may also be configured to provide one or more queries or query statements for requesting data of the databases 112 (or other data stores) to be processed.

**[0074]** In some aspects, the service provider computers 106 may also be any type of computing devices such as, but not limited to, mobile, desktop, thin-client, and/or cloud computing devices, such as servers. In some examples, the service provider computers 106 may be in communication with the user devices 104 via the networks 108, or via other network connections. The service provider computers 106 may include one or more servers, perhaps arranged in a cluster, as a server farm, or as individual servers not associated with one another. These servers may be configured to perform or otherwise host features described herein including, but not limited to, the management of CQL relations, generation of input relations, configurable window operators associated with the input relations, and the generation of output relations, described herein. Additionally, in some aspects, the service provider computers 106 may be configured as part of an integrated, distributed computing environment that includes the streaming data source computers 110 and/or the databases 112.

**[0075]** In one illustrative configuration, the service provider computers 106 may include at least one memory 114 and one or more processing units (or processor(s)) 134. The processor(s) 134 may be implemented as appropriate in hardware, computer-executable instructions, firmware, or combinations thereof. Computer-executable instruction or firmware implementations of the processor(s) 134 may include computer-executable or machine-executable instructions written in any suitable programming language to perform the various functions described.

**[0076]** The memory 114 may store program instructions that are loadable and executable on the processor(s) 134, as well as data generated during the execution of these programs. Depending on the configuration and type of service provider computers 106, the memory 114 may be volatile (such as random access memory (RAM)) and/or non-volatile (such as read-only memory (ROM), flash memory, etc.). The service provider computers 106 or servers may also include additional storage 136, which may include removable storage and/or non-removable storage. The additional storage 136 may include, but is not limited to, magnetic storage, optical disks, and/or tape storage. The disk drives and their associated computer-readable media may provide non-volatile storage of computer-readable instructions, data structures, program modules, and other data for the computing devices. In some implementations, the memory 114 may include multiple different types of memory, such as static random access memory (SRAM), dynamic random access memory (DRAM), or ROM.

**[0077]** The memory 114, the additional storage 136, both removable and non-removable, are all examples of computer-readable storage media. For example, computer-readable storage media may include volatile or non-volatile, removable or non-removable media implemented in any method or technology for storage of information such as computer-readable instructions, data structures, program modules, or other data. The memory 114 and the additional storage 136 are all examples of computer storage media.

**[0078]** The service provider computers 106 may also contain communications connection(s) 138 that allow the service provider computers 106 to communicate with a stored database, another computing device or server, user terminals, and/or other devices on the networks 108. The service provider computers 106 may also include input/output (I/O) device(s) 140, such as a keyboard, a mouse, a pen, a voice input device, a touch input device, a display, one or more speakers, a printer, etc.

**[0079]** Turning to the contents of the memory 114 in more detail, the memory 114 may include an operating system 124 and one or more application programs or services for implementing the features disclosed herein including at least an input data stream module 116, a dynamic data type generation module 118, a combined data stream generation module 120 and a pattern detection module 122. As used herein, modules may refer to programming modules executed by servers or clusters of servers that are part of a service. In this particular context, the modules may be executed by the servers or clusters of servers that are part of the service provider computers 106. In other embodiments, the modules may be executed by a CQL Engine and/or a CQ Service 200 (shown in FIG. 2) that may be part of the service provider computers 106. In various embodiments, and as will be discussed in detail below, the modules 116, 118, 120 and 122 may be configured to perform functionality that enables the detection of patterns across a plurality of input data streams. These modules may be implemented in hardware, or software, or combinations thereof. The various modules depicted in FIG. 1 are meant for illustrative purposes and are not intended to limit the scope of embodiments of the present invention. Alternative embodiments may include more or fewer modules than those shown in FIG. 1.

**[0080]** In one embodiment, the input data stream module 116 may be configured to, receive, identify, generate, or otherwise provide one or more input data stream(s) 126. In one example, the input data stream(s) 126 may include an incoming continuous data stream of data or events related to one or more applications. In some embodiments, the input data streams 126 may be received from the data source computers 110 and/or the database 112.

**[0081]** In some examples, the input data stream(s) 126 may include a sequence of time stamped tuples or data records related to one or more applications. For example, each data record in the input data stream(s) 126 may include an event stream entry that may be represented by the following schema: (<time_stamp>, <attribute(s)>), where <attributes> represents the data portion of the schema and can comprise one or more attributes.

**[0082]** As an example, consider that the input data stream(s) 126 include a first data stream and a second data stream. For purposes of this example, further consider that the first input data stream includes a first set of events related to a first application (e.g., a car rental reservation application) and that the second input data stream includes a second set of events related to a second application (e.g., a flight reservation application). Additionally consider that the first input data stream is defined by a first schema, S1: (c1, integer, c2 integer) and that the second input data stream is defined by a second schema, S2: (c1, integer, c3 integer).

**[0083]** In one embodiment, the first input data stream and the second input data stream may be analyzed to detect a pattern, wherein a 'pattern' may constitute the occurrence of an event in a first data stream followed by the occurrence of another event in a second data stream. In one example, pattern matching may be performed by applying a continuous query (e.g., a CQL query) to the incoming continuous data streams. FIG. 2 describes at least one implementation of the CQL Engine and/or a CQ Service 200 for performing pattern matching by applying a CQL query over multiple input data streams. Although the above discussion relates to the detection of detecting a pattern across a first data stream and a second data stream, it is to be appreciated that the disclosed technique may be applied to detecting patterns across multiple continuous input data streams, in at least some embodiments.

**[0084]** In some embodiments, the analysis of the input data streams may initially be performed by the dynamic data type generation module 118. In one embodiment, the dynamic data type generation module 118 may be configured to receive the first input data stream and the second input data stream from the input data stream module 116 and identify one or more attributes in the input data streams. In some examples, the dynamic data type generation module 118 may be configured to identify a first attribute of the first data stream and a second attribute of the second data stream. In one example, the first attribute may be identified as an attribute that is not present in the second data stream and the second atribute may be identified as an attribute that is not present in the first data stream.

**[0085]** Per the example of the data streams S1 and S2 discussed above, in one embodiment, the dynamic data type generation module 118 may identify a first attribute 'c2' as an attribute in the first data stream S1 that is not present in the second data stream S2. Similarly, the dynamic data type generation module 118 may identify a second attribute 'c3' as an attribute in the second data stream S2 that is not present in the first data stream S1. Although the above discussion relates to a first attribute and a second attribute identified in streams S1 and S2, it is to be appreciated that the dynamic data type generation module 118 may be configured to identify additional attributes from the streams S1 and S2, in other embodiments.

**[0086]** In some embodiments, the dynamic data type generation module 118 may then be configured to generate a first dynamic data type for the first attribute and a second dynamic data type for the second attribute. In one example, the first dynamic data type is configured to store a first data value corresponding to the first attribute of the first data stream and the second dynamic data type is configured to store a second data value corresponding to the second attribute of the second data stream.

**[0087]** In some embodiments, the dynamic data type may be implemented as a composite data type that includes one or more member types, referred to herein as 'fields.' In one example, a member field may include a name and an associated type. The field type may be any CQL native type, such as, for example, a CQL extensible type. In some examples, a service (e.g., the CQL Engine and/or CQ Service 200 shown in FIG. 2) may be configured to implement a CQL extensible type for the dynamic data type by providing one or more application programing interfaces (APIs) by which the dynamic data type may be created and defined by specifying its public fields (e.g., field name, data type of field).

**[0088]** In certain embodiments, the dynamic data type generation module 118 may be configured to identify a common attribute, which may be identified as an attribute that is present in both the first data stream and the second data stream. In one embodiment, the dynamic data type generation module 118 may then be configured to generate a homogeneous schema 128 to represent the first input data stream and the second input data stream. In one example, the homogeneous schema 128 may include a representation of one or more attributes of the first input data stream and the second input data stream including the common attribute and the first dynamic data type and the second dynamic data type.

**[0089]** Per the example of the first input data stream S1 and the second input data stream S2 shown above, in one embodiment, a homogeneous schema S for the streams S1 and S2 may be generated as follows:

Homogeneous Schema S: ($streamName char, c1 int, S1 Typ1@DynamicTypeCartridge, S2 Typ2@DynamicTypeCartridge)

**[0090]** Here, '$streamName' refers to an attribute having a data type char in the homogenous schema. The '$streamName' attribute identifies the input data stream source of the received input tuple (event) from the input data stream. For an event 'e', the '$streamName' attribute may store the name of the stream alias in which 'e' is an event. Thus, the '$streamName' attribute enables the identification of a particular input data stream to which the current input event (tuple) belongs. 'c1' refers to a common attribute in the homogeneous schema, wherein the common attribute is identified as an attribute that has the same datatype and the same name in every participating input data stream. For example, the common attribute 'c1' may refer to a user attribute that identifies a user accessing the first application and the second application related to the input data streams S1 and S2. In some examples, a homogeneous schema including a common attribute may enable the specification of a PARTITION by clause to create a sub-stream which itself could contains events from different input data streams.

**[0091]** In addition, 'Typ1@DynamicTypeCartridge' refers to a first dynamic data type configured to store a first data value corresponding to the first attribute of the first data stream. In one example, the 'Typ1@DynamicTypeCartridge' may be used to store data values corresponding to the first attribute 'c2' in the first input data stream S1. As an example, the first attribute 'c2' in the first input data stream S1 may refer to a car reservation attribute that stores a car reservation identification number from a user accessing the first application (e.g., the car rental reservation application) .

**[0092]** Similarly, 'Typ2@DynamicTypeCartridge' refers to a second dynamic data type configured to store a second data value corresponding to the second attribute of the second data stream. In one example, the 'Typ2@DynamicTypeCartridge' may be used to store data values corresponding to the second attribute 'c3' in the second input data stream S2. As an example, the second attribute 'c3' in the second input data stream S2 may refer to a flight reservation attribute that stores a flight reservation identification number for the user accessing the second application (e.g., the flight reservation application).

**[0093]** For the above example, it may be observed that there is one attribute (e.g., 'c2') for S1 of type, 'Typ1@DynamicTypeCartridge' and one attribute (e.g., 'c3') for S2 of type 'Typ2@DynamicTypeCartridge.' 'Typ1@DynamicTypeCartridge' may include one public field for each attribute of stream S1 of the same datatype. Similarly, 'Typ2@DynamicTypeCartridge' may include one public field for each attribute of stream S2. It may be noted that the Typ1 and Typ2 are internal types and may not be visible to the end user. For the example considered, Typ1 may include public fields 'c1' and 'c2' while Typ2 may include public fields 'c1' and 'c3.'

**[0094]** Based on the homogeneous schema thus generated, in one embodiment, the combined data stream generation module 120 may be configured to generate a combined data stream. In one example, the combined data stream may be generated by performing a UNION of the first input data stream and the second input data stream, wherein the first input data stream and the second input data stream are represented by the homogenous schema S. In one example, a CQL UNION ALL sub-query may be applied to the first data stream S1 and the second data stream S2 to combine the result sets of the streams S1 and S2 based on the homogeneous schema. The generation of a homogeneous schema enables each CQL SELECT statement within the UNION ALL sub-query to have the same number of fields and similar data types in the results sets. Accordingly, the result sets of the streams S1 and S2 may be combined using the SELECT statements in the UNION ALL sub-query based on the homogeneous schema. The manner in which a first data stream and a second data stream may be combined using a UNION ALL sub-query to generate a combined data stream is discussed in FIG. 4.

**[0095]** In some embodiments, the pattern detection module 122 may then be configured to detect a pattern 132 based on the combined data stream. In one embodiment, the pattern detection module may be configured to output a pattern match by applying a MATCH_RECOGNIZE clause to the combined data stream. As an example, a detected pattern may include the arrival of a first event (e.g., a car reservation by a user) in the first input data stream S1 followed by the arrival of a second event (e.g., a flight ticket issuance to the user) in the second input data stream S2. The manner in which the pattern detection module may be configured to perform pattern recognition in a CQL query that identifies multiple input data streams is discussed in FIG. 3.

**[0096]** FIG. 2 illustrates a simplified block diagram with which features for the detection of patterns across multiple input data streams may be described. As shown, FIG. 2 describes at least one implementation of a CQL Engine and/or CQ Service 200 for managing multiple input data streams. In some embodiments, the modules 116, 118, 120 and 122 shown in FIG. 1 may be executed by the CQL Engine and/or a CQ Service 200. The CQL Engine and/or CQ Service 200 may initially receive information from an input source 204. In one example, the input source 204 may include the data source computers 110 that receive incoming continuous input data streams that include a stream of data or events related to one or more applications. In certain embodiments, when a query (e.g., a CQL query) is identified or received that includes a first input data stream S1 206 and a second input data stream S2 208, the CQL engine 200 may parse the query to detect a pattern across the streams S1 and S2. In one embodiment, the CQL engine and/or CQ Service 200 may execute the CQL query by first performing a UNION of the first input data stream and the second input data stream, wherein the first input data stream and the second input data stream are represented by a homogenous schema S. In one embodiment, the homogeneous schema S may be generated by the dynamic data type generation module 118 as discussed in FIG. 1.

**[0097]** The input data streams S1 and S2 may then be combined to generate a combined data stream based on the homogenous schema. In one embodiment, the combined data stream may be generated by the combined data stream module 120. The pattern detection module 122 may then process a continuous query (e.g., a CQL query) over the combined data stream to detect a pattern across the input data streams. In one embodiment, the pattern detection module may be configured to output a pattern match by applying a MATCH_RECOGNIZE clause to the combined data stream. In some examples, the CQL Engine and/or CQ Service 200 may then store the detected pattern in an output destination 212, such as for example, in the databases 112 shown in FIG. 1.

**[0098]** FIG. 3 is an exemplary illustration of performing pattern recognition using a CQL query that identifies multiple input data streams, in accordance with one embodiment of the present disclosure. In one example, the query 300 is a continuous (e.g., a CQL) query that specifies a pattern to be matched across a first input data stream S1 and a second input data stream S2. In the example shown in FIG. 3, the streams S1 and S2 are specified in a comma separated list surrounded by parenthesis and provided as an input to the MATCH_RECOGNIZE clause. As a result, the FROM clause of the query 300 may be conveniently expressed using the following expression: (S1, S2) MATCH_RECOGNIZE. In some examples, each stream in the list may be identified as a base stream or a view evaluating to a stream or a sub-query evaluating to a stream.

**[0099]** When streams S1 and S2 are specified using a comma separated list as discussed above, each of the participating streams may need to be identified with a distinct alias. So, for example, if a MATCH_RECOGNIZE clause is applied over {S, S} then the FROM clause may need to be specified as (S, S AS S1) MATCH_RECOGNIZE. That is, an optional AS clause may need to be introduced to alias the second stream S to ensure distinct alias names. It may be noted that if no explicit AS clause is specified, the stream name may be considered as the alias name, in other examples.

**[00100]** In some embodiments, the expression of the streams S1 and S2 as a comma separated list may evaluate to a UNION of the input data streams S1 and S2, wherein the streams S1 and S2 are defined by a homogeneous schema. In one embodiment, and as discussed in detail in FIG. 4, a combined data stream may be generated based upon the execution of the UNION ALL sub-query and the combined data stream may be provided as input to the MATCH_RECOGNIZE clause. The MATCH_RECOGNIZE may then be utilized to detect a pattern in the combined data stream.

**[00101]** As discussed above, using the MATCH_RECOGNIZE clause, a user can define conditions on the attributes of incoming events and identify conditions for pattern matching by using identifiers called correlation variables. The pattern recognition functionality allows a user to define conditions on the attributes of incoming events or tuples and to identify these conditions by using string names called correlation variables.

**[00102]** In the query 300, 'A' is a correlation variable. The pattern to be matched is specified as a regular expression over this correlation variable and it determines the sequence or order in which conditions should be satisfied by different incoming events to be recognized as a valid match. In one example, a sequence of consecutive events in the input data streams S1 and S2 satisfying these conditions constitutes a pattern. In one embodiment, the output of a MATCH_RECOGNIZE query is a stream.

**[00103]** FIG. 4 is an exemplary illustration of performing pattern recognition using a CQL query that identifies multiple input data streams, in accordance with another embodiment of the present disclosure. In one embodiment, the query 400 shown in FIG. 4 depicts an internal representation of the FROM clause of the query 300. In one example, the FROM clause may be represented as a UNION ALL sub-query of the streams S1 and S2, wherein the streams S1 and S2 are represented by a homogenous schema.

**[00104]** For the example shown in FIG. 4, a homogenous schema representing stream S1 and S2 may be generated as follows:

Homogeneous Schema S: ($streamName char, c1 int, ELEMENT_TIME int, S1 Typ1@DynamicTypeCartridge, S2 Typ2@DynamicTypeCartridgeCartridge)

**[00105]** Here, 'Typ1@DynamicTypeCartridge' represents a first dynamic data type generated for an attribute of stream S1 that is not present in stream S2. Similarly, 'Typ2@DynamicTypeCartridge' represents a second dynamic data type generated for an attribute of stream S2 that is not present in stream S1. For the example shown in FIG. 4, the 'Typ1@DynamicTypeCartridge' represents a first dynamic data type for a first attribute 'c2' in stream S1 that is not present in stream S2 and the 'Typ2@DynamicTypeCartridge' represents a second dynamic data type for a second attribute 'c3' in stream S2 that is not present in stream S1.

**[00106]** In one embodiment, the generation of the dynamic data types and the homogenous schema discussed above may enable individual stream specific attributes of the streams S1 and S2 to be accessed as follows. At runtime, for an event received from stream S1, the 'Typ1@DynamicTypeCartridge' dynamic data type may be configured to represent the first attribute 'c2' from stream S1 that is not in S2 using the expression: 'Typ1@DynamicTypeCartridge(c1, c2) AS S1', while the 'Typ2@DynamicTypeCartridge' dynamic data type may be configured to represent the second attribute 'c3' from stream S2 that is not in S1 using the expression: 'Typ2@DynamicTypeCartridge( ) AS S2.' In one example, the 'Typ2@DynamicTypeCartridge' may store a NULL value for the second attribute 'c3' of stream S2 for the received event.

**[00107]** Similarly, for an event received from stream S2, the 'Typ1@DynamicTypeCartridge' dynamic data type may be configured to represent the first attribute ('c2') from stream S1 that is not in S2 as 'Typ1@DynamicTypeCartridge( ) AS S1' and store a NULL value for the attribute 'c2.' Additionally, the 'Typ2@DynamicTypeCartridge' may be configured to represent the second attribute 'c3' from stream S2 that is not in S1 using the expression: 'Typ2@DynamicTypeCartridge(c1, c3) AS S2.' In one embodiment, using the dynamic data type representations discussed above, the attribute 'c2' may be accessed in a CQL query as S1.c2 and the attribute 'c3' may be accessed in the CQL query as S2.c3.

**[00108]** Additionally, in the example shown in FIG. 4, the attributes 'c1' and 'ELEMENT_TIME' (timestamp value) may be identified as the common attributes. In some examples, the common attribute 'c1' may be accessed using the homogeneous schema either as A.c1 or A.S1.c1. In some examples, a PARTITION BY clause may also be used to access the common attribute.

**[00109]** It may be appreciated that the generation of a homogeneous schema and the dynamic data types discussed above enables each CQL SELECT statement within the UNION ALL query to have the same number of fields and similar data types in the results sets. Additionally, the dynamic data types enable the assignment of 'NULL' values to attributes that do not belong to a specific stream. In one embodiment, the result sets of the streams S1 and S2 may then be combined to generate a combined data stream based on the homogeneous schema by applying a CQL UNION ALL sub-query to combine the result sets of the streams S1 and S2. The manner in which a combined data stream may be generated may be explained using the following non-limiting example.

**[00110]** As an example, consider that the first input data stream S1 receives the following events and associated timestamps:

```
...
(<timestamp 1000>, <C1 1, C2 5>)
(<timestamp 2000>, <C1 1,C2 10>)
(<timestamp 3000>, <C1 1 C2 15>)
...
```

**[00111]** Similarly, consider that the second input data stream S2 receives the following events and associated timestamps:

```
...
```

(<timestamp 1000>, <C1 1, C3 5>)
(<timestamp 2000>, <C1 1,C3 10>)
(<timestamp 3000>, <C1 1 C3 15>)
...

**[00112]** The result sets of the streams S1 and S2 shown above may be combined to generate a combined data stream as shown below. In one example, the combined data stream includes the result sets of the streams S1 and S2 that may be combined using the SELECT statements in the UNION ALL sub-query based on the homogeneous schema as discussed above. In one example, the combined data stream may be identified by the following homogenous schema ($streamName, c1, S1.c2, S2,c3) and may include the following events and associated timestamps:

...
(<timestamp 1000>, <S1, C1 1, C2 5, NULL>)
(<timestamp 1000>, <S2, C1 1, C2 NULL, C3 5>)
(<timestamp 2000>, <S1, C1 1, C2 10, C3 NULL>)
(<timestamp 2000>, <S2, C1 1, C2 NULL, C3 10>)
(<timestamp 3000>, <S1, C1 1, C2 1, C3 NULL>)
(<timestamp 3000>, <S2, C1 1, C2 NULL, C3 15>)
...

**[00113]** FIGS. 5 and 6 illustrate example flow diagrams showing respective processes 500 and 600 for implementing the detection of patterns across multiple input data streams, described herein. These processes 500 and 600 are illustrated as logical flow diagrams, each operation of which represents a sequence of operations that can be implemented in hardware, computer instructions, or a combination thereof. In the context of computer instructions, the operations represent computer-executable instructions stored on one or more computer-readable storage media that, when executed by one or more processors, perform the recited operations. Generally, computer-executable instructions include routines, programs, objects, components, data structures and the like that perform particular functions or implement particular data types. The order in which the operations are described is not intended to be construed as a limitation, and any number of the described operations can be combined in any order and/or in parallel to implement the processes.

**[00114]** Additionally, some, any, or all of the processes may be performed under the control of one or more computer systems configured with executable instructions and may be implemented as code (e.g., executable instructions, one or more computer programs, or one or more applications) executing collectively on one or more processors, by hardware, or combinations thereof. As noted above, the code may be stored on a computer-readable storage medium, for example, in the form of a computer program comprising a plurality of instructions executable by one or more processors. The computer-readable storage medium may be non-transitory.

**[00115]** FIG. 5 is a high level flowchart depicting a process for detecting patterns across multiple input data streams, in accordance with one embodiment of the present disclosure. In some examples, the one or more service provider computers 106 (e.g., utilizing at least the input data stream module 116, the dynamic data type generation module 118, the combined data stream generation module 120 or the pattern detection module 122) shown in at least FIG. 1 (and others) may perform the process 500 of FIG. 5. The process 500 may begin at 502 by including identifying and/or receiving a first input data stream and a second input data stream.

**[00116]** At 504, the process 500 may include generating a homogeneous schema to represent the first input data stream and the second input data stream. The process by which a homogeneous schema may be generated is discussed in detail in FIG. 6.

**[00117]** At 506, the process 500 may include combining the first input data stream and the second input data stream to generate a combined data stream based on the homogeneous schema. In one embodiment, generating a combined data stream may include selecting a first set of tuples (via a first CQL SELECT statement) from the first input data stream, selecting a second set of tuples (via a first CQL SELECT statement) from the second input stream and processing a sub-query (e.g., a UNION all sub-query) over the first set of tuples and the second set of tuples to generate the combined data stream. As discussed above, the generation of a homogeneous schema enables each CQL SELECT statement within the UNION ALL sub-query to have the same number of fields and similar data types in the results sets. Accordingly, the result sets of the streams S1 and S2 may be combined using the SELECT statements in the UNION ALL sub-query based on the homogeneous schema to generate the combined data stream.

**[00118]** At 508, the process 500 may include processing a continuous query (e.g., a CQL query) over the combined data stream to identify a pattern. In one embodiment, the combined data stream generated based upon the execution of the UNION ALL sub-query may be provided as an input to the MATCH_RECOGNIZE clause. As discussed in FIG. 3, the MATCH_RECOGNIZE clause may be used to process a CQL query over the combined data stream to detect a pattern in the combined data stream.

**[00119]** FIG. 6 is a high level flowchart depicting a process for generating a homogenous schema, in accordance with one embodiment of the present disclosure. In one embodiment, the process 600 describes more details of performing the process 504 discussed in FIG. 5.

**[00120]** At 602, the process 600 may include identifying a first attribute of the first input data stream.

**[00121]** At 604, the process may include identifying a second attribute of the second input data stream. In one embodiment, the first attribute may be identified as an attribute that is not present in the second input data stream and the second attribute may be identified as an attribute that is not present in the first input data stream.

**[00122]** At 606, the process 600 may include generating a first dynamic data type to represent the first attribute of the first input data stream. At 608, the process 600 may include generating a second dynamic data type to represent the second attribute of the second input data stream.

**[00123]** At 610, the process 600 may include identifying a common attribute. In one embodiment, the common attribute may be identified as an attribute that is present in the first input data stream and the second input data stream.

**[00124]** At 612, the process 600 may include generating a homogeneous schema. In one embodiment, and as described in FIG. 2 in detail, the homogenous schema may include a representation of one or more attributes of the first input data stream and the second input data stream, the representation including at least the common attribute, the first dynamic data type and the second dynamic data type .

**[00125]** FIG. 7 depicts a simplified high level diagram of an event processing system 700 that may incorporate an embodiment of the present disclosure. Event processing system 700 may comprise one or more event sources (704, 706, 708), an event processing server (EPS) 702 that is configured to provide an environment for processing event streams, and one or more event sinks (710, 712). The event sources generate event streams that are received by EPS 702. EPS 702 may receive one or more event streams from one or more event sources. For example, as shown in FIG. 7, EPS 702 receives an input event stream 714 from event source 704, a second input event stream 716 from event source 706, and a third event stream 718 from event source 708. One or more event processing applications (720, 722, and 724) may be deployed on and be executed by EPS 702. An event processing application executed by EPS 702 may be configured to listen to one or more input event streams, process the events received via the one or more event streams based upon processing logic that selects one or more events from the input event streams as notable events. The notable events may then be sent to one or more event sinks (710, 712) in the form of one or more output event streams. For example, in FIG. 7, EPS 702 outputs an output event stream 726 to event sink 710, and a second output event stream 728 to event sink 712. In certain embodiments, event sources, event processing applications, and event sinks are decoupled from each other such that one can add or remove any of these components without causing changes to the other components.

**[00126]** In one embodiment, EPS 702 may be implemented as a Java server comprising a lightweight Java application container, such as one based upon Equinox OSGi, with shared services. In some embodiments, EPS 702 may support ultra-high throughput and microsecond latency for processing events, for example, by using JRockit Real Time. EPS 702 may also provide a development platform (e.g., a complete real time end-to-end Java Event-Driven Architecture (EDA) development platform) including tools (e.g., Oracle CEP Visualizer and Oracle CEP IDE) for developing event processing applications.

**[00127]** An event processing application is configured to listen to one or more input event streams, execute logic (e.g., a query) for selecting one or more notable events from the one or more input event streams, and output the selected notable events to one or more event sources via one or more output event streams. FIG. 7 provides a drilldown for one such event processing application 720. As shown in FIG. 7, event processing application 720 is configured to listen to input event stream 718, execute a query via the CQL engine/CQ service 200 comprising logic for selecting one or more notable events from input event stream 718, and output the selected notable events via output event stream 728 to event sink 712. Examples of event sources include, without limitation, an adapter (e.g., JMS, HTTP, and file), a channel, a processor, a table, a cache, and the like. Examples of event sinks include, without limitation, an adapter (e.g., JMS, HTTP, and file), a channel, a processor, a cache, and the like.

**[00128]** Although event processing application 720 in FIG. 7 is shown as listening to one input stream and outputting selected events via one output stream, this is not intended to be limiting. In alternative embodiments, an event processing application may be configured to listen to multiple input streams received from one or more event sources, select events from the monitored streams, and output the selected events via one or more output event streams to one or more event sinks. The same query can be associated with more than one event sink and with different types of event sinks.

**[00129]** Due to its unbounded nature, the amount of data that is received via an event stream is generally very large. Consequently, it is generally impractical and undesirable to store or archive all the data for querying purposes. The processing of event streams requires processing of the events in real time as the events are received by EPS 702 without having to store all the received events data. Accordingly, EPS 702 provides a special querying mechanism that enables processing of events to be performed as the events are received by EPS 702 without having to store all the received events.

**[00130]** Event-driven applications are rule-driven and these rules may be expressed in the form of continuous queries that are used to process input streams. A continuous query may comprise instructions (e.g., business logic) that identify the processing to be performed for received events including what events are to be selected as notable events and output as results of the query processing. Continuous queries may be persisted to a data store and used for processing input streams of events and generating output streams of events. Continuous queries typically perform filtering and aggregation functions to discover and extract notable events from the input event streams. As a result, the number of outbound events in an output event stream is generally much lower than the number of events in the input event stream from which the events are selected.

**[00131]** Unlike a SQL query that is run once on a finite data set, a continuous query that has been registered by an application with EPS 702 for a particular event stream may be executed each time that an event is received in that event stream. As part of the continuous query execution, EPS 702 evaluates the received event based upon instructions specified by the continuous query to determine whether one or more events are to be selected as notable events, and output as a result of the continuous query execution.

**[00132]** The continuous query may be programmed using different languages. In certain embodiments, continuous queries may be configured using the CQL provided by Oracle Corporation and used by Oracle's Complex Events Processing (CEP) product offerings. Oracle's CQL is a declarative language that can be used to program queries (referred to as CQL queries) that can be executed against event streams. In certain embodiments, CQL is based upon SQL with added constructs that support processing of streaming events data.

**[00133]** In one embodiment, an event processing application may be composed of the following component types:

**[00134]** (1) One or more adapters that interface directly to the input and output stream and relation sources and sinks. Adapters are configured to understand the input and output stream protocol, and are responsible for converting the event data into a normalized form that can be queried by an application processor. Adapters may forward the normalized event data into channels or output streams and relation sinks. Event adapters may be defined for a variety of data sources and sinks.

(2) One or more channels that act as event processing endpoints. Among other things, channels are responsible for queuing event data until the event processing agent can act upon it.

(3) One or more application processors (or event processing agents) are configured to consume normalized event data from a channel, process it using queries to select notable events, and forward (or copy) the selected notable events to an output channel.

(4) One or more beans are configured to listen to the output channel, and are triggered by the insertion of a new event into the output channel. In some embodiments, this user code is a plain-old-Java-object (POJO). The user application can make use of a set of external services, such as JMS, Web services, and file writers, to forward the generated events to external event sinks.

(5) Event beans may be registered to listen to the output channel, and are triggered by the insertion of a new event into the output channel. In some embodiments, this user code may use the Oracle CEP event bean API so that the bean can be managed by Oracle CEP.

**[00135]** In one embodiment, an event adapter provides event data to an input channel. The input channel is connected to a CQL processor associated with one or more CQL queries that operate on the events offered by the input channel. The CQL processor is connected to an output channel to which query results are written.

**[00136]** In some embodiments, an assembly file may be provided for an event processing application describing the various components of the event processing application, how the components are connected together, event types processed by the application. Separate files may be provided for specifying the continuous query or business logic for selection of events.

**[00137]** It should be appreciated that system 700 depicted in FIG. 7 may have other components than those depicted in FIG. 7. Further, the embodiment shown in FIG. 7 is only one example of a system that may incorporate an embodiment of the present disclosure. In some other embodiments, system 700 may have more or fewer components than shown in FIG. 7, may combine two or more components, or may have a different configuration or arrangement of components. System 700 can be of various types including a personal computer, a portable device (e.g., a mobile telephone or device), a workstation, a network computer, a mainframe, a kiosk, a server, or any other data processing system. In some other embodiments, system 700 may be configured as a distributed system where one or more components of system 700 are distributed across one or more networks in the cloud.

**[00138]** The one or more of the components depicted in FIG. 7 may be implemented in software, in hardware, or combinations thereof. In some embodiments, the software may be stored in memory (e.g., a non-transitory computer-readable medium), on a memory device, or some other physical memory and may be executed by one or more processing units (e.g., one or more processors, one or more processor cores, one or more GPUs, etc.).

**[00139]** Systems depicted in some of the figures may be provided in various configurations. In some embodiments, the systems may be configured as a distributed system where one or more components of the system are distributed across one or more networks in a cloud computing system.

**[00140]** FIG. 8 depicts a simplified diagram of a distributed system 800 for implementing one of the embodiments. In the illustrated embodiment, distributed system 800 includes one or more client computing devices 802, 804, 806, and 808, which are configured to execute and operate a client application such as a web browser, proprietary client (e.g., Oracle Forms), or the like over one or more network(s) 810. Server 812 may be communicatively coupled with remote client computing devices 802, 804, 806, and 808 via network 810.

**[00141]** In various embodiments, server 812 may be adapted to run one or more services or software applications provided by one or more of the components of the system. The services or software applications can include nonvirtual and virtual environments. Virtual environments can include those used for virtual events, tradeshows, simulators, classrooms, shopping exchanges, and enterprises, whether two- or three-dimensional (3D) representations, page-based logical environments, or otherwise. In some embodiments, these services may be offered as web-based or cloud services or under a Software as a Service (SaaS) model to the users of client computing devices 802, 804, 806, and/or 808. Users operating client computing devices 802, 804, 806, and/or 808 may in turn utilize one or more client applications to interact with server 812 to utilize the services provided by these components.

**[00142]** In the configuration depicted in the figure, the software components 818, 820 and 822 of system 800 are shown as being implemented on server 812. In other embodiments, one or more of the components of system 800 and/or the services provided by these components may also be implemented by one or more of the client computing devices 802, 804, 806, and/or 808. Users operating the client computing devices may then utilize one or more client applications to use the services provided by these components. These components may be implemented in hardware, firmware, software, or combinations thereof. It should be appreciated that various different system configurations are possible, which may be different from distributed system 800. The embodiment shown in the figure is thus one example of a distributed system for implementing an embodiment system and is not intended to be limiting.

**[00143]** Client computing devices 802, 804, 806, and/or 808 may be portable handheld devices (e.g., an iPhone®, cellular telephone, an iPad®, computing tablet, a personal digital assistant (PDA)) or wearable devices (e.g., a Google Glass® head mounted display), running software such as Microsoft Windows Mobile®, and/or a variety of mobile operating systems such as iOS, Windows Phone, Android, BlackBerry 10, Palm OS, and the like, and being Internet, e-mail, short message service (SMS), Blackberry®, or other communication protocol enabled. The client computing devices can be general purpose personal computers including, by way of example, personal computers and/or laptop computers running various versions of Microsoft Windows®, Apple Macintosh®, and/or Linux operating systems. The client computing devices can be workstation computers running any of a variety of commercially-available UNIX® or UNIX-like operating systems, including without limitation the variety of GNU/Linux operating systems, such as for example, Google Chrome OS. Alternatively, or in addition, client computing devices 802, 804, 806, and 808 may be any other electronic device, such as a thin-client computer, an Internet-enabled gaming system (e.g., a Microsoft Xbox gaming console with or without a Kinect® gesture input device), and/or a personal messaging device, capable of communicating over network(s) 810.

**[00144]** Although exemplary distributed system 800 is shown with four client computing devices, any number of client computing devices may be supported. Other devices, such as devices with sensors, etc., may interact with server 812.

**[00145]** Network(s) 810 in distributed system 800 may be any type of network familiar to those skilled in the art that can support data communications using any of a variety of commercially-available protocols, including without limitation TCP/IP (transmission control protocol/Internet protocol), SNA (systems network architecture), IPX (Internet packet exchange), AppleTalk, and the like. Merely by way of example, network(s) 810 can be a local area network (LAN), such as one based on Ethernet, Token-Ring and/or the like. Network(s) 810 can be a wide-area network and the Internet. It can include a virtual network, including without limitation a virtual private network (VPN), an intranet, an extranet, a public switched telephone network (PSTN), an infra-red network, a wireless network (e.g., a network operating under any of the Institute of Electrical and Electronics (IEEE) 802.11 suite of protocols, Bluetooth®, and/or any other wireless protocol); and/or any combination of these and/or other networks.

**[00146]** Server 812 may be composed of one or more general purpose computers, specialized server computers (including, by way of example, PC (personal computer) servers, UNIX® servers, mid-range servers, mainframe computers, rack-mounted servers, etc.), server farms, server clusters, or any other appropriate arrangement and/or combination. Server 812 can include one or more virtual machines running virtual operating systems, or other computing architectures involving virtualization. One or more flexible pools of logical storage devices can be virtualized to maintain virtual storage devices for the server. Virtual networks can be controlled by server 812 using software defined networking. In various embodiments, server 812 may be adapted to run one or more services or software applications described in the foregoing disclosure. For example, server 812 may correspond to a server for performing processing described above according to an embodiment of the present disclosure.

**[00147]** Server 812 may run an operating system including any of those discussed above, as well as any commercially available server operating system. Server 812 may also run any of a variety of additional server applications and/or mid-tier applications, including HTTP (hypertext transport protocol) servers, FTP (file transfer protocol) servers, CGI (common gateway interface) servers, JAVA® servers, database servers, and the like. Exemplary database servers include without limitation those commercially available from Oracle, Microsoft, Sybase, IBM (International Business Machines), and the like.

**[00148]** In some implementations, server 812 may include one or more applications to analyze and consolidate data feeds and/or event updates received from users of client computing devices 802, 804, 806, and 808. As an example, data feeds and/or event updates may include, but are not limited to, Twitter® feeds, Facebook® updates or real-time updates received from one or more third party information sources and continuous data streams, which may include real-time events related to sensor data applications, financial tickers, network performance measuring tools (e.g., network monitoring and traffic management applications), clickstream analysis tools, automobile traffic monitoring, and the like. Server 812 may also include one or more applications to display the data feeds and/or real-time events via one or more display devices of client computing devices 802, 804, 806, and 808.

**[00149]** Distributed system 800 may also include one or more databases 814 and 816. Databases 814 and 816 may reside in a variety of locations. By way of example, one or more of databases 814 and 816 may reside on a non-transitory storage medium local to (and/or resident in) server 812. Alternatively, databases 814 and 816 may be remote from server 812 and in communication with server 812 via a network-based or dedicated connection. In one set of embodiments, databases 814 and 816 may reside in a storage-area network (SAN). Similarly, any necessary files for performing the functions attributed to server 812 may be stored locally on server 812 and/or remotely, as appropriate. In one set of embodiments, databases 814 and 816 may include relational databases, such as databases provided by Oracle, that are adapted to store, update, and retrieve data in response to SQL-formatted commands.

**[00150]** FIG. 9 is a simplified block diagram of one or more components of a system environment 900 by which services provided by one or more components of an embodiment system may be offered as cloud services, in accordance with an embodiment of the present disclosure. In the illustrated embodiment, system environment 900 includes one or more client computing devices 904, 906, and 908 that may be used by users to interact with a cloud infrastructure system 902 that provides cloud services. The client computing devices may be configured to operate a client application such as a web browser, a proprietary client application (e.g., Oracle Forms), or some other application, which may be used by a user of the client computing device to interact with cloud infrastructure system 902 to use services provided by cloud infrastructure system 902.

**[00151]** It should be appreciated that cloud infrastructure system 902 depicted in the figure may have other components than those depicted. Further, the embodiment shown in the figure is only one example of a cloud infrastructure system that may incorporate an embodiment of the invention. In some other embodiments, cloud infrastructure system 902 may have more or fewer components than shown in the figure, may combine two or more components, or may have a different configuration or arrangement of components.

**[00152]** Client computing devices 904, 906, and 908 may be devices similar to those described above for 802, 804, 806, and 808.

**[00153]** Although exemplary system environment 900 is shown with three client computing devices, any number of client computing devices may be supported. Other devices such as devices with sensors, etc. may interact with cloud infrastructure system 902.

**[00154]** Network(s) 910 may facilitate communications and exchange of data between clients 904, 906, and 908 and cloud infrastructure system 902. Each network may be any type of network familiar to those skilled in the art that can support data communications using any of a variety of commercially-available protocols, including those described above for network(s) 810.

**[00155]** Cloud infrastructure system 902 may comprise one or more computers and/or servers that may include those described above for server 812.

**[00156]** In certain embodiments, services provided by the cloud infrastructure system may include a host of services that are made available to users of the cloud infrastructure system on demand, such as online data storage and backup solutions, Web-based e-mail services, hosted office suites and document collaboration services, database processing, managed technical support services, and the like. Services provided by the cloud infrastructure system can dynamically scale to meet the needs of its users. A specific instantiation of a service provided by cloud infrastructure system is referred to herein as a "service instance." In general, any service made available to a user via a communication network, such as the Internet, from a cloud service provider's system is referred to as a "cloud service." Typically, in a public cloud environment, servers and systems that make up the cloud service provider's system are different from the customer's own on-premises servers and systems. For example, a cloud service provider's system may host an application, and a user may, via a communication network such as the Internet, on demand, order and use the application.

**[00157]** In some examples, a service in a computer network cloud infrastructure may include protected computer network access to storage, a hosted database, a hosted web server, a software application, or other service provided by a cloud vendor to a user, or as otherwise known in the art. For example, a service can include password-protected access to remote storage on the cloud through the Internet. As another example, a service can include a web service-based hosted relational database and a script-language middleware engine for private use by a networked developer. As another example, a service can include access to an email software application hosted on a cloud vendor's web site.

**[00158]** In certain embodiments, cloud infrastructure system 902 may include a suite of applications, middleware, and database service offerings that are delivered to a customer in a self-service, subscription-based, elastically scalable, reliable, highly available, and secure manner. An example of such a cloud infrastructure system is the Oracle Public Cloud provided by the present assignee.

**[00159]** 'Big data' can be hosted and/or manipulated by the infrastructure system on many levels and at different scales. Extremely large data sets can be stored and manipulated by analysts and researchers to visualize large amounts of data, detect trends, and/or otherwise interact with the data. Tens, hundreds, or thousands of processors linked in parallel can act upon such data in order to present it or simulate external forces on the data or what it represents. These data sets can involve structured data, such as that organized in a database or otherwise according to a structured model, and/or unstructured data (e.g., emails, images, data blobs (binary large objects), web pages, complex event processing). By leveraging an ability of an embodiment to relatively quickly focus more (or fewer) computing resources upon an objective, the cloud infrastructure system may be better available to carry out tasks on large data sets based on demand from a business, government agency, research organization, private individual, group of like-minded individuals or organizations, or other entity.

**[00160]** In various embodiments, cloud infrastructure system 902 may be adapted to automatically provision, manage and track a customer's subscription to services offered by cloud infrastructure system 902. Cloud infrastructure system 902 may provide the cloud services via different deployment models. For example, services may be provided under a public cloud model in which cloud infrastructure system 902 is owned by an organization selling cloud services (e.g., owned by Oracle) and the services are made available to the general public or different industry enterprises. As another example, services may be provided under a private cloud model in which cloud infrastructure system 902 is operated solely for a single organization and may provide services for one or more entities within the organization. The cloud services may also be provided under a community cloud model in which cloud infrastructure system 902 and the services provided by cloud infrastructure system 902 are shared by several organizations in a related community. The cloud services may also be provided under a hybrid cloud model, which is a combination of two or more different models.

**[00161]** In some embodiments, the services provided by cloud infrastructure system 902 may include one or more services provided under Software as a Service (SaaS) category, Platform as a Service (PaaS) category, Infrastructure as a Service (IaaS) category, or other categories of services including hybrid services. A customer, via a subscription order, may order one or more services provided by cloud infrastructure system 902. Cloud infrastructure system 902 then performs processing to provide the services in the customer's subscription order.

**[00162]** In some embodiments, the services provided by cloud infrastructure system 902 may include, without limitation, application services, platform services and infrastructure services. In some examples, application services may be provided by the cloud infrastructure system via a SaaS platform. The SaaS platform may be configured to provide cloud services that fall under the SaaS category. For example, the SaaS platform may provide capabilities to build and deliver a suite of on-demand applications on an integrated development and deployment platform. The SaaS platform may manage and control the underlying software and infrastructure for providing the SaaS services. By utilizing the services provided by the SaaS platform, customers can utilize applications executing on the cloud infrastructure system. Customers can acquire the application services without the need for customers to purchase separate licenses and support. Various different SaaS services may be provided. Examples include, without limitation, services that provide solutions for sales performance management, enterprise integration, and business flexibility for large organizations.

**[00163]** In some embodiments, platform services may be provided by the cloud infrastructure system via a PaaS platform. The PaaS platform may be configured to provide cloud services that fall under the PaaS category. Examples of platform services may include without limitation services that enable organizations (such as Oracle) to consolidate existing applications on a shared, common architecture, as well as the ability to build new applications that leverage the shared services provided by the platform. The PaaS platform may manage and control the underlying software and infrastructure for providing the PaaS services. Customers can acquire the PaaS services provided by the cloud infrastructure system without the need for customers to purchase separate licenses and support. Examples of platform services include, without limitation, Oracle Java Cloud Service (JCS), Oracle Database Cloud Service (DBCS), and others.

**[00164]** By utilizing the services provided by the PaaS platform, customers can employ programming languages and tools supported by the cloud infrastructure system and also control the deployed services. In some embodiments, platform services provided by the cloud infrastructure system may include database cloud services, middleware cloud services (e.g., Oracle Fusion Middleware services), and Java cloud services. In one embodiment, database cloud services may support shared service deployment models that enable organizations to pool database resources and offer customers a Database as a Service in the form of a database cloud. Middleware cloud services may provide a platform for customers to develop and deploy various business applications, and Java cloud services may provide a platform for customers to deploy Java applications, in the cloud infrastructure system.

**[00165]** Various different infrastructure services may be provided by an IaaS platform in the cloud infrastructure system. The infrastructure services facilitate the management and control of the underlying computing resources, such as storage, networks, and other fundamental computing resources for customers utilizing services provided by the SaaS platform and the PaaS platform.

**[00166]** In certain embodiments, cloud infrastructure system 902 may also include infrastructure resources 930 for providing the resources used to provide various services to customers of the cloud infrastructure system. In one embodiment, infrastructure resources 930 may include pre-integrated and optimized combinations of hardware, such as servers, storage, and networking resources to execute the services provided by the PaaS platform and the SaaS platform.

**[00167]** In some embodiments, resources in cloud infrastructure system 902 may be shared by multiple users and dynamically re-allocated per demand. Additionally, resources may be allocated to users in different time zones. For example, cloud infrastructure system 930 may enable a first set of users in a first time zone to utilize resources of the cloud infrastructure system for a specified number of hours and then enable the re-allocation of the same resources to another set of users located in a different time zone, thereby maximizing the utilization of resources.

**[00168]** In certain embodiments, a number of internal shared services 932 may be provided that are shared by different components or modules of cloud infrastructure system 902 and by the services provided by cloud infrastructure system 902. These internal shared services may include, without limitation, a security and identity service, an integration service, an enterprise repository service, an enterprise manager service, a virus scanning and white list service, a high availability, backup and recovery service, service for enabling cloud support, an email service, a notification service, a file transfer service, and the like.

**[00169]** In certain embodiments, cloud infrastructure system 902 may provide comprehensive management of cloud services (e.g., SaaS, PaaS, and IaaS services) in the cloud infrastructure system. In one embodiment, cloud management functionality may include capabilities for provisioning, managing and tracking a customer's subscription received by cloud infrastructure system 902, and the like.

**[00170]** In one embodiment, as depicted in the figure, cloud management functionality may be provided by one or more modules, such as an order management module 920, an order orchestration module 922, an order provisioning module 924, an order management and monitoring module 926, and an identity management module 928. These modules may include or be provided using one or more computers and/or servers, which may be general purpose computers, specialized server computers, server farms, server clusters, or any other appropriate arrangement and/or combination.

**[00171]** In exemplary operation 934, a customer using a client device, such as client device 904, 906 or 908, may interact with cloud infrastructure system 902 by requesting one or more services provided by cloud infrastructure system 902 and placing an order for a subscription for one or more services offered by cloud infrastructure system 902. In certain embodiments, the customer may access a cloud User Interface (UI), cloud UI 912, cloud UI 914 and/or cloud UI 916 and place a subscription order via these UIs. The order information received by cloud infrastructure system 902 in response to the customer placing an order may include information identifying the customer and one or more services offered by the cloud infrastructure system 902 that the customer intends to subscribe to.

**[00172]** After an order has been placed by the customer, the order information is received via the cloud UIs, 912, 914 and/or 916.

**[00173]** At operation 936, the order is stored in order database 918. Order database 918 can be one of several databases operated by cloud infrastructure system 918 and operated in conjunction with other system elements.

**[00174]** At operation 938, the order information is forwarded to an order management module 920. In some instances, order management module 920 may be configured to perform billing and accounting functions related to the order, such as verifying the order, and upon verification, booking the order.

**[00175]** At operation 940, information regarding the order is communicated to an order orchestration module 922. Order orchestration module 922 may utilize the order information to orchestrate the provisioning of services and resources for the order placed by the customer. In some instances, order orchestration module 922 may orchestrate the provisioning of resources to support the subscribed services using the services of order provisioning module 924.

**[00176]** In certain embodiments, order orchestration module 922 enables the management of business processes associated with each order and applies business logic to determine whether an order should proceed to provisioning. At operation 942, upon receiving an order for a new subscription, order orchestration module 922 sends a request to order provisioning module 924 to allocate resources and configure those resources needed to fulfill the subscription order. Order provisioning module 924 enables the allocation of resources for the services ordered by the customer. Order provisioning module 924 provides a level of abstraction between the cloud services provided by cloud infrastructure system 900 and the physical implementation layer that is used to provision the resources for providing the requested services. Order orchestration module 922 may thus be isolated from implementation details, such as whether or not services and resources are actually provisioned on the fly or pre-provisioned and only allocated/assigned upon request.

**[00177]** At operation 944, once the services and resources are provisioned, a notification of the provided service may be sent to customers on client devices 904, 906 and/or 908 by order provisioning module 924 of cloud infrastructure system 902.

**[00178]** At operation 946, the customer's subscription order may be managed and tracked by an order management and monitoring module 926. In some instances, order management and monitoring module 926 may be configured to collect usage statistics for the services in the subscription order, such as the amount of storage used, the amount data transferred, the number of users, and the amount of system up time and system down time.

**[00179]** In certain embodiments, cloud infrastructure system 900 may include an identity management module 928. Identity management module 928 may be configured to provide identity services, such as access management and authorization services in cloud infrastructure system 900. In some embodiments, identity management module 928 may control information about customers who wish to utilize the services provided by cloud infrastructure system 902. Such information can include information that authenticates the identities of such customers and information that describes which actions those customers are authorized to perform relative to various system resources (e.g., files, directories, applications, communication ports, memory segments, etc.) Identity management module 928 may also include the management of descriptive information about each customer and about how and by whom that descriptive information can be accessed and modified.

**[00180]** FIG. 10 illustrates an exemplary computer system 1000, in which various embodiments of the present invention may be implemented. The system 1000 may be used to implement any of the computer systems described above. As shown in the figure, computer system 1000 includes a processing unit 1004 that communicates with a number of peripheral subsystems via a bus subsystem 1002. These peripheral subsystems may include a processing acceleration unit 1006, an I/O subsystem 1008, a storage subsystem 1018 and a communications subsystem 1024. Storage subsystem 1018 includes tangible computer-readable storage media 1022 and a system memory 1010.

**[00181]** Bus subsystem 1002 provides a mechanism for letting the various components and subsystems of computer system 1000 communicate with each other as intended. Although bus subsystem 1002 is shown schematically as a single bus, alternative embodiments of the bus subsystem may utilize multiple buses. Bus subsystem 1002 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. For example, such architectures may include an Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus, which can be implemented as a Mezzanine bus manufactured to the IEEE P1386.1 standard.

**[00182]** Processing unit 1004, which can be implemented as one or more integrated circuits (e.g., a conventional microprocessor or microcontroller), controls the operation of computer system 1000. One or more processors may be included in processing unit 1004. These processors may include single core or multicore processors. In certain embodiments, processing unit 1004 may be implemented as one or more independent processing units 1032 and/or 1034 with single or multicore processors included in each processing unit. In other embodiments, processing unit 1004 may also be implemented as a quad-core processing unit formed by integrating two dual-core processors into a single chip.

**[00183]** In various embodiments, processing unit 1004 can execute a variety of programs in response to program code and can maintain multiple concurrently executing programs or processes. At any given time, some or all of the program code to be executed can be resident in processor(s) 1004 and/or in storage subsystem 1018. Through suitable programming, processor(s) 1004 can provide various functionalities described above. Computer system 1000 may additionally include a processing acceleration unit 1006, which can include a digital signal processor (DSP), a special-purpose processor, and/or the like.

**[00184]** I/O subsystem 1008 may include user interface input devices and user interface output devices. User interface input devices may include a keyboard, pointing devices such as a mouse or trackball, a touchpad or touch screen incorporated into a display, a scroll wheel, a click wheel, a dial, a button, a switch, a keypad, audio input devices with voice command recognition systems, microphones, and other types of input devices. User interface input devices may include, for example, motion sensing and/or gesture recognition devices such as the Microsoft

Kinect® motion sensor that enables users to control and interact with an input device, such as the Microsoft Xbox® 360 game controller, through a natural user interface using gestures and spoken commands. User interface input devices may also include eye gesture recognition devices such as the Google Glass® blink detector that detects eye activity (e.g., 'blinking' while taking pictures and/or making a menu selection) from users and transforms the eye gestures as input into an input device (e.g., Google Glass®). Additionally, user interface input devices may include voice recognition sensing devices that enable users to interact with voice recognition systems (e.g., Siri® navigator), through voice commands.

**[00185]** User interface input devices may also include, without limitation, three dimensional (3D) mice, joysticks or pointing sticks, gamepads and graphic tablets, and audio/visual devices such as speakers, digital cameras, digital camcorders, portable media players, webcams, image scanners, fingerprint scanners, barcode reader 3D scanners, 3D printers, laser rangefinders, and eye gaze tracking devices. Additionally, user interface input devices may include, for example, medical imaging input devices such as computed tomography, magnetic resonance imaging, position emission tomography, medical ultrasonography devices. User interface input devices may also include, for example, audio input devices such as MIDI keyboards, digital musical instruments and the like.

**[00186]** User interface output devices may include a display subsystem, indicator lights, or non-visual displays such as audio output devices, etc. The display subsystem may be a cathode ray tube (CRT), a flat-panel device, such as that using a liquid crystal display (LCD) or plasma display, a projection device, a touch screen, and the like. In general, use of the term "output device" is intended to include all possible types of devices and mechanisms for outputting information from computer system 1000 to a user or other computer. For example, user interface output devices may include, without limitation, a variety of display devices that visually convey text, graphics and audio/video information such as monitors, printers, speakers, headphones, automotive navigation systems, plotters, voice output devices, and modems.

**[00187]** Computer system 1000 may comprise a storage subsystem 1018 that comprises software elements, shown as being currently located within a system memory 1010. System memory 1010 may store program instructions that are loadable and executable on processing unit 1004, as well as data generated during the execution of these programs.

**[00188]** Depending on the configuration and type of computer system 1000, system memory 1010 may be volatile (such as random access memory (RAM)) and/or non-volatile (such as read-only memory (ROM), flash memory, etc.) The RAM typically contains data and/or program modules that are immediately accessible to and/or presently being operated and executed by processing unit 1004. In some implementations, system memory 1010 may include multiple different types of memory, such as static random access memory (SRAM) or dynamic random access memory (DRAM). In some implementations, a basic input/output system (BIOS), containing the basic routines that help to transfer information between elements within computer system 1000, such as during start-up, may typically be stored in the ROM. By way of example, and not limitation, system memory 1010 also illustrates application programs 1012, which may include client applications, Web browsers, mid-tier applications, relational database management systems (RDBMS), etc., program data 1014, and an operating system 1016. By way of example, operating system 1016 may include various versions of Microsoft Windows®, Apple Macintosh®, and/or Linux operating systems, a variety of commercially-available UNIX® or UNIX-like operating systems (including without limitation the variety of GNU/Linux operating systems, the Google Chrome® OS, and the like) and/or mobile operating systems such as iOS, Windows® Phone, Android® OS, BlackBerry® 10 OS, and Palm® OS operating systems.

**[00189]** Storage subsystem 1018 may also provide a tangible computer-readable storage medium for storing the basic programming and data constructs that provide the functionality of some embodiments. Software (programs, code modules, instructions) that when executed by a processor provide the functionality described above may be stored in storage subsystem 1018. These software modules or instructions may be executed by processing unit 1004. Storage subsystem 1018 may also provide a repository for storing data used in accordance with the present invention.

**[00190]** Storage subsystem 1000 may also include a computer-readable storage media reader 1020 that can further be connected to computer-readable storage media 1022. Together and, optionally, in combination with system memory 1010, computer-readable storage media 1022 may comprehensively represent remote, local, fixed, and/or removable storage devices plus storage media for temporarily and/or more permanently containing, storing, transmitting, and retrieving computer-readable information.

**[00191]** Computer-readable storage media 1022 containing code, or portions of code, can also include any appropriate media known or used in the art, including storage media and communication media, such as but not limited to, volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage and/or transmission of information. This can include tangible, non-transitory computer-readable storage media such as RAM, ROM, electronically erasable programmable ROM (EEPROM), flash memory or other memory technology, CD-ROM, digital versatile disk (DVD), or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or other tangible computer readable media. When specified, this can also include nontangible, transitory computer-readable media, such as data signals, data transmissions, or any other medium which can be used to transmit the desired information and which can be accessed by computing system 1000.

**[00192]** By way of example, computer-readable storage media 1022 may include a hard disk drive that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive that reads from or writes to a removable, nonvolatile magnetic disk, and an optical disk drive that reads from or writes to a removable, nonvolatile optical disk such as a CD ROM, DVD, and Blu-Ray® disk, or other optical media. Computer-readable storage media 1022 may include, but is not limited to, Zip® drives, flash memory cards, universal serial bus (USB) flash drives, secure digital (SD) cards, DVD disks, digital video tape, and the like. Computer-readable storage media 1022 may also include, solid-state drives (SSD) based on non-volatile memory such as flash-memory based SSDs, enterprise flash drives, solid state ROM, and the like, SSDs based on volatile memory such as solid state RAM, dynamic RAM, static RAM, DRAM-based SSDs, magnetoresistive RAM (MRAM) SSDs, and hybrid SSDs that use a combination of DRAM and flash memory based SSDs. The disk drives and their associated computer-readable media may provide non-volatile storage of computer-readable instructions, data structures, program modules, and other data for computer system 1000.

**[00193]** Communications subsystem 1024 provides an interface to other computer systems and networks. Communications subsystem 1024 serves as an interface for receiving data from and transmitting data to other systems from computer system 1000. For example, communications subsystem 1024 may enable computer system 1000 to connect to one or more devices via the Internet. In some embodiments communications subsystem 1024 can include radio frequency (RF) transceiver components for accessing wireless voice and/or data networks (e.g., using cellular telephone technology, advanced data network technology, such as 3G, 4G or EDGE (enhanced data rates for global evolution), WiFi (IEEE 802.11 family standards, or other mobile communication technologies, or any combination thereof), global positioning system (GPS) receiver components, and/or other components. In some embodiments communications subsystem 1024 can provide wired network connectivity (e.g., Ethernet) in addition to or instead of a wireless interface.

**[00194]** In some embodiments, communications subsystem 1024 may also receive input communication in the form of structured and/or unstructured data feeds 1026, event streams 1028, event updates 1030, and the like on behalf of one or more users who may use computer system 1000.

**[00195]** By way of example, communications subsystem 1024 may be configured to receive data feeds 1026 in real-time from users of social media networks and/or other communication services such as Twitter® feeds, Facebook® updates, web feeds such as Rich Site Summary (RSS) feeds, and/or real-time updates from one or more third party information sources.

**[00196]** Additionally, communications subsystem 1024 may also be configured to receive data in the form of continuous data streams, which may include event streams 1028 of real-time events and/or event updates 1030, that may be continuous or unbounded in nature with no explicit end. Examples of applications that generate continuous data may include, for example, sensor data applications, financial tickers, network performance measuring tools (e.g. network monitoring and traffic management applications), clickstream analysis tools, automobile traffic monitoring, and the like.

**[00197]** Communications subsystem 1024 may also be configured to output the structured and/or unstructured data feeds 1026, event streams 1028, event updates 1030, and the like to one or more databases that may be in communication with one or more streaming data source computers coupled to computer system 1000.

**[00198]** Computer system 1000 can be one of various types, including a handheld portable device (e.g., an iPhone® cellular phone, an iPad® computing tablet, a PDA), a wearable device (e.g., a Google Glass® head mounted display), a PC, a workstation, a mainframe, a kiosk, a server rack, or any other data processing system.

**[00199]** Due to the ever-changing nature of computers and networks, the description of computer system 1000 depicted in the figure is intended only as a specific example. Many other configurations having more or fewer components than the system depicted in the figure are possible. For example, customized hardware might also be used and/or particular elements might be implemented in hardware, firmware, software (including applets), or a combination. Further, connection to other computing devices, such as network input/output devices, may be employed. Based on the disclosure and teachings provided herein, a person of ordinary skill in the art will appreciate other ways and/or methods to implement the various embodiments. In other words, in some embodiments, configurations of computer system 1000, as a whole, may include: means for receiving a plurality of input data streams comprising at least a first input data stream and a second input data stream; means for generating a first dynamic data type for the first input data stream; means for generating a second dynamic data type for the second input data stream; means for combining the first input data stream and the second input data stream to generate a combined data stream based at least in part on the first dynamic data type and the second dynamic data type; and means for processing, by the computer system, a continuous query over the combined data stream to detect a pattern.

**[00200]** FIG. 11 illustrates a simplified block diagram of an exemplary service provider device, in which various embodiments of the present invention may be implemented. The blocks of the service provider device 1100 may be implemented by hardware, software, or a combination of hardware and software to carry out the principles of the invention. It is understood by persons of skill in the art that the blocks described in FIG. 11 may be combined or separated into sub-blocks to implement the principles of the invention as described above.

Therefore, the description herein may support any possible combination or separation or further definition of the functional blocks described herein.

**[00201]** As shown in FIG. 11, the service provider device 1100 may comprise an input data stream receiving unit 1101, a first dynamic data type generation unit 1103, a second dynamic data type generation unit 1105, a combined data stream generation unit 1107 and a pattern detection unit 1109. The input data stream receiving unit 1101 may configured to receive a plurality of input data streams comprising at least a first input data stream and a second input data stream. The first dynamic data type generation unit 1103 may be configured to generate a first dynamic data type for the first input data stream. The second dynamic data type generation unit 1105 may be configured to generate a second dynamic data type for the second input data stream. The combined data stream generation unit 1107 may be configured to combine the first input data stream and the second input data stream to generate a combined data stream based at least in part on the first dynamic data type and the second dynamic data type. The pattern detection unit 1109 may be configured to process a continuous query over the combined data stream to detect a pattern. In one embodiment, each unit may be implemented as a processor that performs corresponding processes by reading computer program instructions.

**[00202]** In one embodiment, the first dynamic data type generation unit 1103 may be further configured to identify a first attribute of the first input data stream as not being present in the second data stream; and generate the first dynamic data type for the first attribute.

**[00203]** In one embodiment, the first dynamic data type may be configured to store a first data value corresponding to the first attribute of the first input data stream.

**[00204]** In one embodiment, the second dynamic data type generation unit 1105 may be further configured to identify a second attribute of the second input data stream as not being present in first data stream; and generate the second dynamic data type for the second attribute, the second dynamic data type configured to store a second data value corresponding to the second attribute of the second input data stream.

**[00205]** In one embodiment, the service provider device 1100 may further comprise a common attribute identification unit 1111 and a homogeneous schema generation unit 1113. The common attribute identification unit 1111 may be configured to identify a common attribute, the common attribute identified as being present in the first input data stream and being present in the second input data stream. The homogeneous schema generation unit 1113 may be configured to generate a homogeneous schema, the homogeneous schema including a representation of one or more attributes of the first input data stream and the second input data stream, the representation including at least the common attribute, the first dynamic data type and the second dynamic data type.

**[00206]** In one embodiment, the homogeneous schema may comprise at least one of a stream name identifier attribute, a first timestamp attribute associated with the first input data stream or a second timestamp attribute associated with the second input data stream.

**[00207]** In one embodiment, the combined data stream generation unit 1107 may be further configured to select a first set of tuples from the first data stream, the first input data stream identified by the homogeneous schema, select a second set of tuples from the second input data stream, the second input data stream identified by the homogeneous schema, and process a sub-query over the first set of tuples and the second set of tuples to generate the combined data stream.

**[00208]** In one embodiment, the pattern may be detected based at least in part on analyzing the combined data stream, wherein the pattern identifies a first event in the first input data stream followed by a second event in the second input data stream.

**[00209]** The blocks described in Figure 11 may be combined or separated into sub-blocks to implement the principles of the invention as described above. For example, the first dynamic data type generation unit 1103 and the second dynamic data type generation unit 1105 may be integrated as a dynamic data type generation unit. The integrated dynamic data type generation unit can also incorporate the common attribute identification unit 1111 and the homogeneous schema generation unit 1113.

**[00210]** Those skilled in the art can understand that, the service provider device 1100 may be an exemplary implementation of the service provider computer 106 described in the previous exemplary embodiments. Those skilled in the art can also understand that, the service provider device 1100 may be modified as desired to perform any of operations or the combination thereof according to the principle of the invention.

**[00211]** In the foregoing specification, aspects of the invention are described with reference to specific embodiments thereof, but those skilled in the art will recognize that the invention is not limited thereto. Various features and aspects of the above-described invention may be used individually or jointly. Further, embodiments can be utilized in any number of environments and applications beyond those described herein without departing from the broader spirit and scope of the specification. The specification and drawings are, accordingly, to be regarded as illustrative rather than restrictive.

## CLAIMS

WHAT IS CLAIMED IS:

1. A computer-implemented method, comprising:.

receiving, by a computer system configured with computer-executable instructions, a plurality of input data streams comprising at least a first input data stream and a second input data stream;

generating, by the computer system, a first dynamic data type for the first input data stream;

generating, by the computer system, a second dynamic data type for the second input data stream;

combining, by the computer system, the first input data stream and the second input data stream to generate a combined data stream based at least in part on the first dynamic data type and the second dynamic data type; and

processing, by the computer system, a continuous query over the combined data stream to detect a pattern.

2. The computer-implemented method of claim 1, wherein generating the first dynamic data type comprises:

identifying a first attribute of the first input data stream as not being present in the second data stream; and

generating the first dynamic data type for the first attribute.

3. The computer-implemented method of claim 2, wherein the first dynamic data type is configured to store a first data value corresponding to the first attribute of the first input data stream.

4. The computer-implemented method of any one of claims 1-3, wherein generating the second dynamic data type comprises:

identifying a second attribute of the second input data stream as not being present in first data stream; and generating the second dynamic data type for the second attribute, the second dynamic data type configured to store a second data value corresponding to the second attribute of the second input data stream.

5. The computer-implemented method of any one of claims 1-4, further comprising:

identifying a common attribute, the common attribute identified as being present in the first input data stream and being present in the second input data stream; and

generating a homogeneous schema, the homogeneous schema including a representation of one or more attributes of the first input data stream and the second input data stream, the representation including at least the common attribute, the first dynamic data type and the second dynamic data type.

6. The computer-implemented method of claim 5, wherein the homogeneous schema comprises at least one of a stream name identifier attribute, a first timestamp attribute associated with the first input data stream or a second timestamp attribute associated with the second input data stream.

7. The computer-implemented method of claim 5 or 6, wherein combining the first input data stream and the second input data stream further comprises:

selecting a first set of tuples from the first data stream, the first input data stream identified by the homogeneous schema;

selecting a second set of tuples from the second input data stream, the second input data stream identified by the homogeneous schema; and

processing a sub-query over the first set of tuples and the second set of tuples to generate the combined data stream.

8. The computer-implemented method of any one of claims 1-7, wherein the pattern is detected based at least in part on analyzing the combined data stream, wherein the pattern identifies a first event in the first input data stream followed by a second event in the second input data stream.

9. A system, comprising:

a memory storing a plurality of instructions; and

a processor configured to access the memory, wherein the processor is further configured to execute the plurality of instructions to at least:

receive a continuous query identifying a first input data stream and a second input data stream;

identify a first dynamic data type for a first attribute of the first input data stream;

identify a second dynamic data type for a second attribute of the second input data stream;

generate a combined data stream based at least in part on the first dynamic data type identified in the first input data stream and the second dynamic data type identified in the second input data stream; and

execute the continuous query over the combined data stream to detect a pattern.

10. The system of claim 9, wherein the at least one processor is configured to execute the computer-executable instructions to identify the first attribute of the first input data stream as not being present in the second data stream.

11. The system of claim 9 or 10, wherein the at least one processor is configured to execute the computer-executable instructions to identify the second attribute of the second input data stream as not being present in first data stream.

12. The system of any one of claims 9-11, wherein the at least one processor is further configured to execute the computer-executable instructions to:

identify a common attribute, the common attribute identified as being present in the first input data stream and the second input data stream; and

generate a homogeneous schema, the homogeneous schema comprising a representation of at least the common attribute, the first attribute, the first dynamic data type, the second attribute and the second dynamic data type.

13. The system of claim 12, wherein the homogeneous schema further comprises at least one of a stream name identifier attribute, a first timestamp attribute associated with the first input data stream or a second timestamp attribute associated with the second input data stream.

14. The system of claim 13, wherein the at least one processor is further configured to execute the computer-executable instructions to generate a combined data stream by executing instructions to:

select a first set of tuples from the first data stream, the first input data stream identified by the homogeneous schema;

select a second set of tuples from the second input data stream, the second input data stream identified by the homogeneous schema; and

process a sub-query over the first set of tuples and the second set of tuples to generate the combined data stream.

15. The system of any one of claims 9-14, wherein the pattern is detected based at least in part on analyzing the combined data stream, wherein the pattern identifies a first event in the first input data stream followed by a second event in the second input data stream.

16. One or more non-transitory computer-readable media storing computer executable instructions executable by one or more processors, the computer-executable instructions comprising:

instructions that cause the one or more processors to receive a plurality of input data streams comprising at least a first input data stream and a second input data stream;

instructions that cause the one or more processors to generate a first dynamic data type for the first input data stream;

instructions that cause the one or more processors to generate a second dynamic data type for the second input data stream;

instructions that cause the one or more processors to combine the first input data stream and the second input data stream to generate a combined data stream based at least in part on the first dynamic data type and the second dynamic data type; and instructions that cause the one or more processors to process a continuous query over the combined data stream to detect a pattern.

17. The computer-readable media of claim 16, wherein the instructions to generate the first dynamic data type further comprise instructions to:

identify a first attribute of the first input data stream as not being present in the second data stream; and

generate the first dynamic data type for the first attribute, the first dynamic data type configured to store a first data value corresponding to the first attribute of the first input data stream.

18. The computer-readable media of claim 16 or 17, wherein the instructions to generate the second dynamic data type further comprise instructions that cause the one or more processors to:

identify a second attribute of the second input data stream as not being present in first data stream; and

generate the second dynamic data type for the second attribute, the second dynamic data type configured to store a second data value corresponding to the second attribute of the second input data stream.

19. The computer-readable media of any one of claims 16-18, wherein the computer-executable instructions further comprise:

instructions that cause the one or more processors to identify a common attribute, the common attribute identified as being present in the first input data stream and being present in the second input data stream;

instructions that cause the one or more processors to generate a homogeneous schema, the homogeneous schema including a representation of one or more attributes of the first input data stream and the second input data stream, the representation including at least the common attribute, the first dynamic data type and the second dynamic data type; and

instructions that cause the one or more processors to generate the combined data stream based at least in part on the homogeneous schema.

20. The computer-readable media of any one of claims 16-19, wherein the computer-executable instructions further comprise instructions that cause the one or more processors to detect the pattern based at least in part on analyzing the combined data stream, wherein the pattern identifies a first event in the first input data stream followed by a second event in the second input data stream.

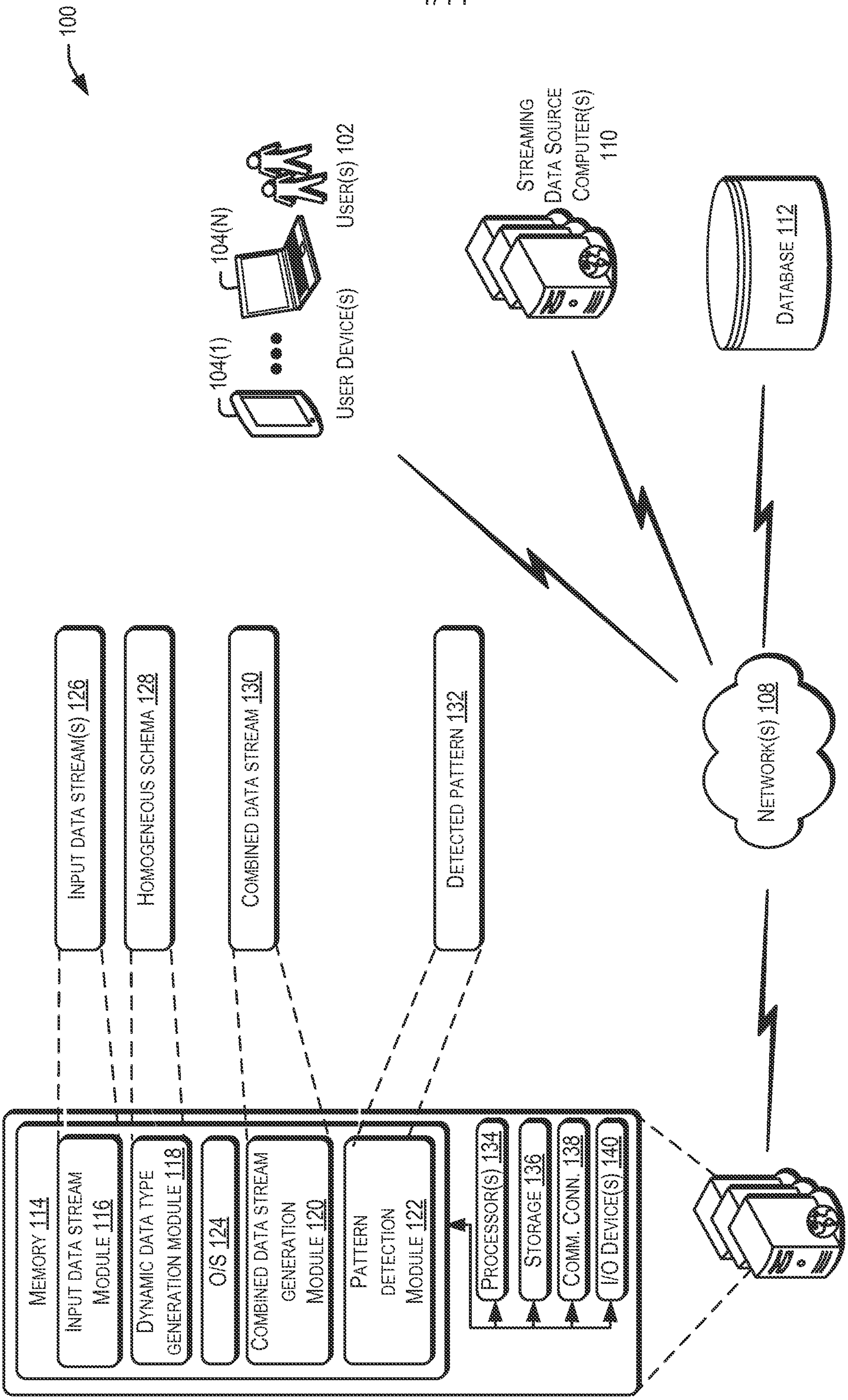
100
User Device(s)
104(1)
104(N)
User(s) 102
Streaming Data Source Computer(s) 110
Database 112
Input Data Stream(s) 126
Homogeneous Schema 128
Combined Data Stream 130
Detected Pattern 132
Network(s) 108
Memory 114
Input Data Stream Module 116
Dynamic Data Type Generation Module 118
O/S 124
Combined Data Stream Generation Module 120
Pattern Detection Module 122
Processor(s) 134
Storage 136
Comm. Conn. 138
I/O Device(s) 140
Service Provider Computer(s) 106

FIG. 1

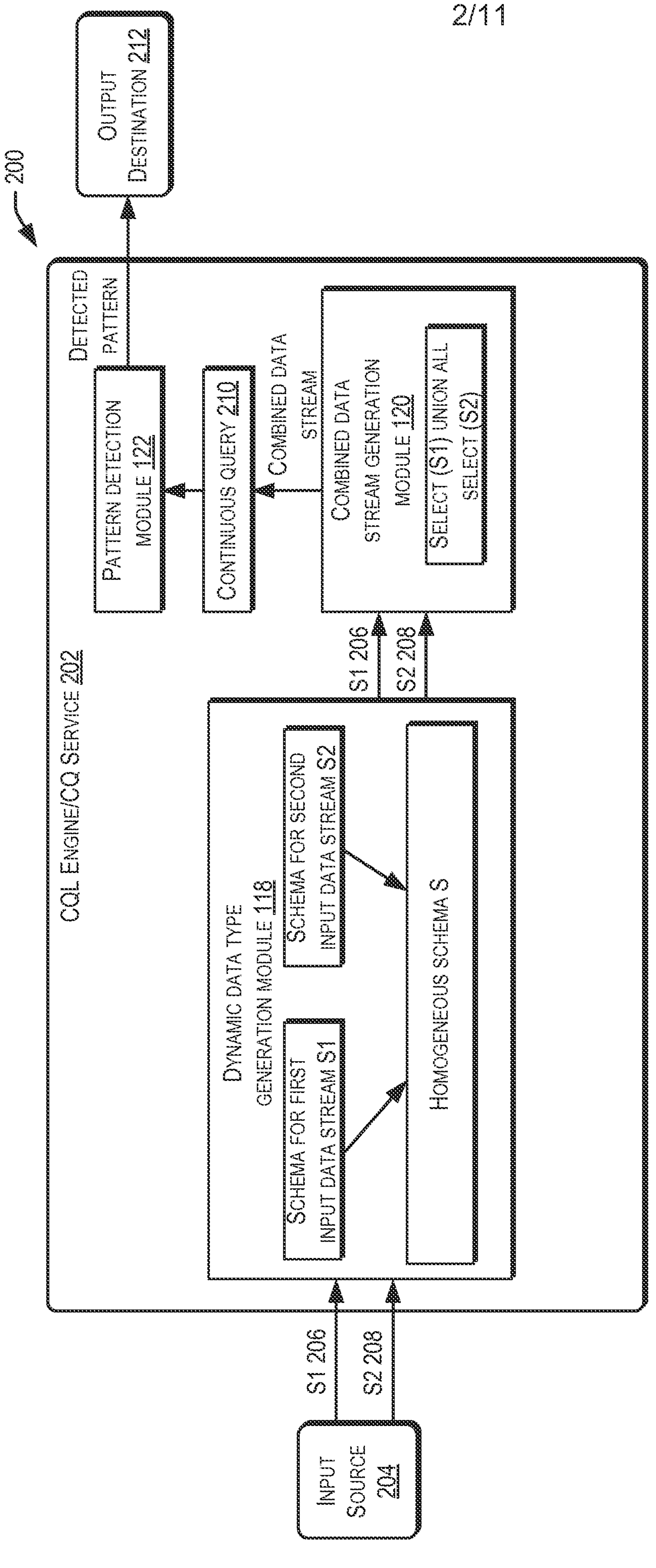
200
OUTPUT DESTINATION 212
CQL ENGINE/CQ SERVICE 202
DETECTED PATTERN
PATTERN DETECTION MODULE 122
CONTINUOUS QUERY 210
COMBINED DATA STREAM
COMBINED DATA STREAM GENERATION MODULE 120
SELECT (S1) UNION ALL SELECT (S2)
S1 206
S2 208
DYNAMIC DATA TYPE GENERATION MODULE 118
SCHEMA FOR FIRST INPUT DATA STREAM S1
SCHEMA FOR SECOND INPUT DATA STREAM S2
HOMOGENEOUS SCHEMA S
S1 206
S2 208
INPUT SOURCE 204

FIG. 2

QUERY 300

```
SELECT T.*
    FROM (S1, S2)
        MATCH_RECOGNIZE
        (
            MEASURES
              $streamName as name
                 A.c1 as Ac1,
                 A.S1.c2 as Ac2,
                 A.S2.c3 as Ac3,
            PATTERN (A)
            DEFINE
            A AS (1 = 1)
        ) as T
```

FIG. 3

QUERY 400

```
SELECT T.*
  FROM

        (select "S1" as $streamName, c1, ELEMENT_TIME,
Type1@DynamicTypeCartridge(c1, c2) AS S1, Type2@DynamicTypeCartridge( )
                          AS S2 from S1
                            UNION ALL
select "S2" as $streamName, c1, ELEMENT_TIME, Type1@DynamicTypeCartridge(
) as S1, Type2@DynamicTypeCartridge(c1, c3) AS S2 from S2)  AS _SQ_ALIAS


      MATCH_RECOGNIZE
          (
          MEASURES
            $streamName as name
           A.c1 as Ac1,
           A.S1.c2 as Ac2,
           A.S2.c3 as Ac3,
           PATTERN (A)
          DEFINE
          A AS (1 = 1)
          ) as T
```

FIG. 4

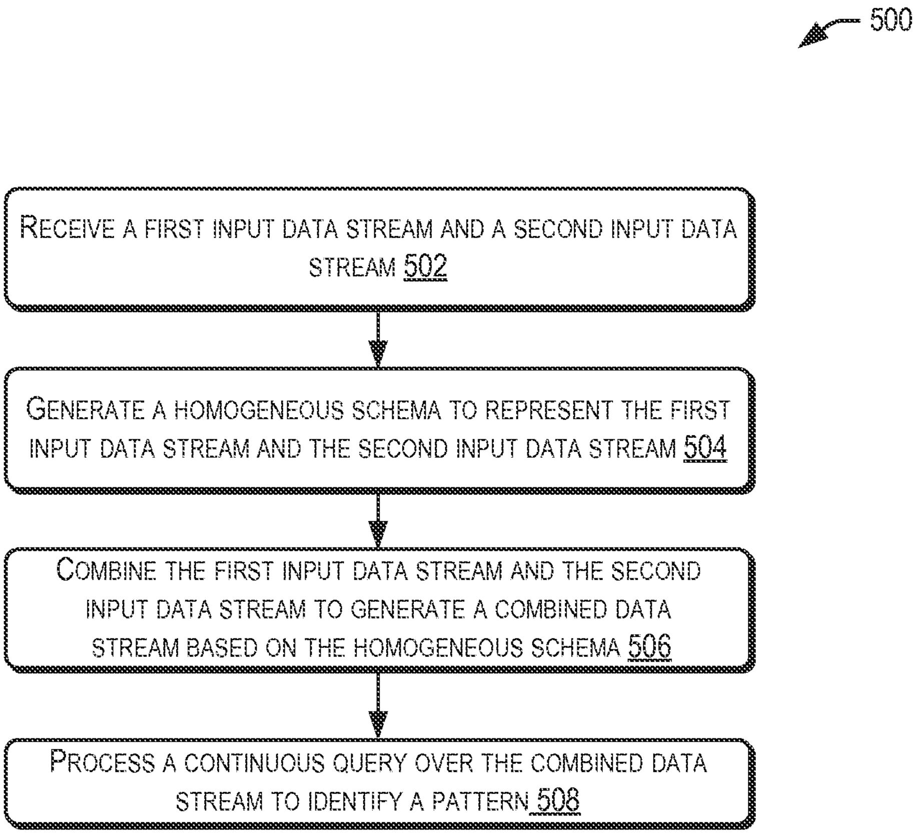
500
RECEIVE A FIRST INPUT DATA STREAM AND A SECOND INPUT DATA STREAM 502
GENERATE A HOMOGENEOUS SCHEMA TO REPRESENT THE FIRST INPUT DATA STREAM AND THE SECOND INPUT DATA STREAM 504
COMBINE THE FIRST INPUT DATA STREAM AND THE SECOND INPUT DATA STREAM TO GENERATE A COMBINED DATA STREAM BASED ON THE HOMOGENEOUS SCHEMA 506
PROCESS A CONTINUOUS QUERY OVER THE COMBINED DATA STREAM TO IDENTIFY A PATTERN 508

FIG. 5

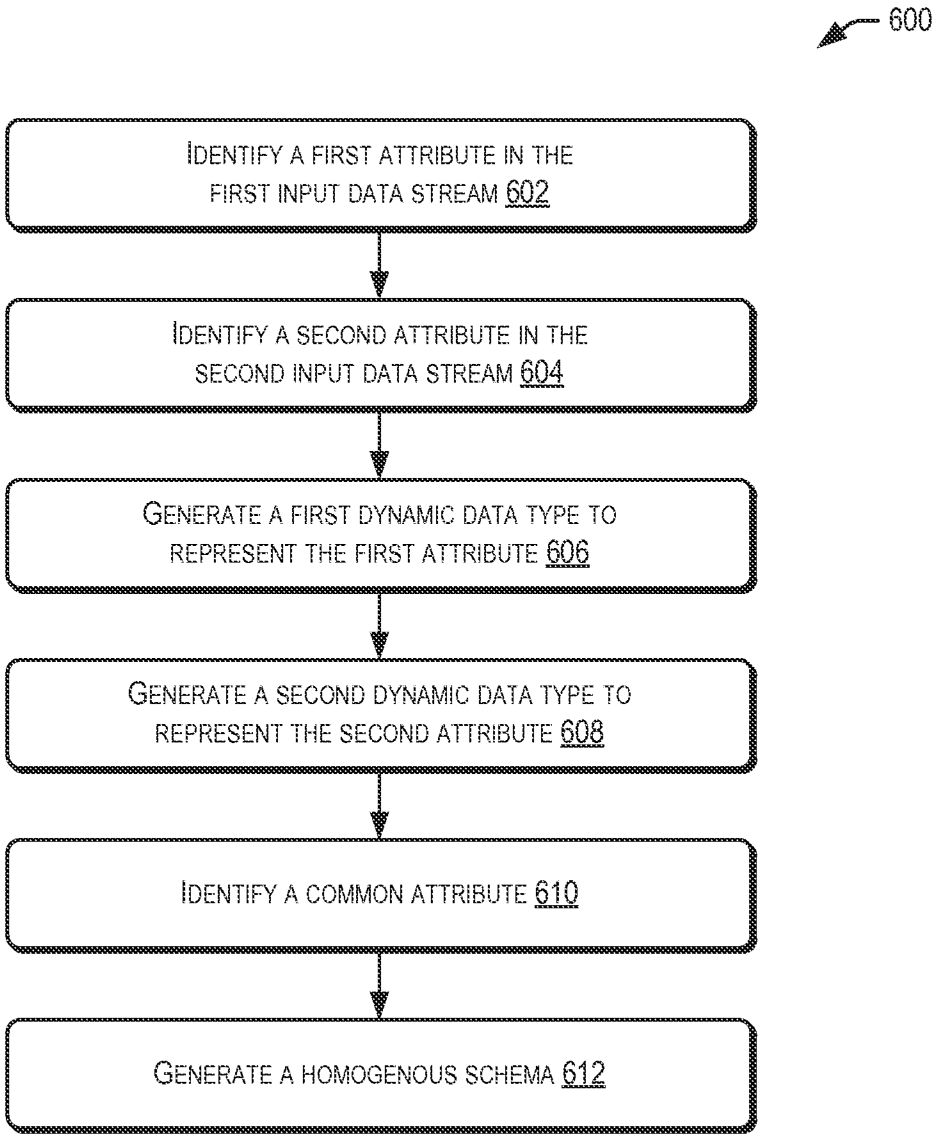
600
IDENTIFY A FIRST ATTRIBUTE IN THE FIRST INPUT DATA STREAM 602
IDENTIFY A SECOND ATTRIBUTE IN THE SECOND INPUT DATA STREAM 604
GENERATE A FIRST DYNAMIC DATA TYPE TO REPRESENT THE FIRST ATTRIBUTE 606
GENERATE A SECOND DYNAMIC DATA TYPE TO REPRESENT THE SECOND ATTRIBUTE 608
IDENTIFY A COMMON ATTRIBUTE 610
GENERATE A HOMOGENOUS SCHEMA 612

FIG. 6

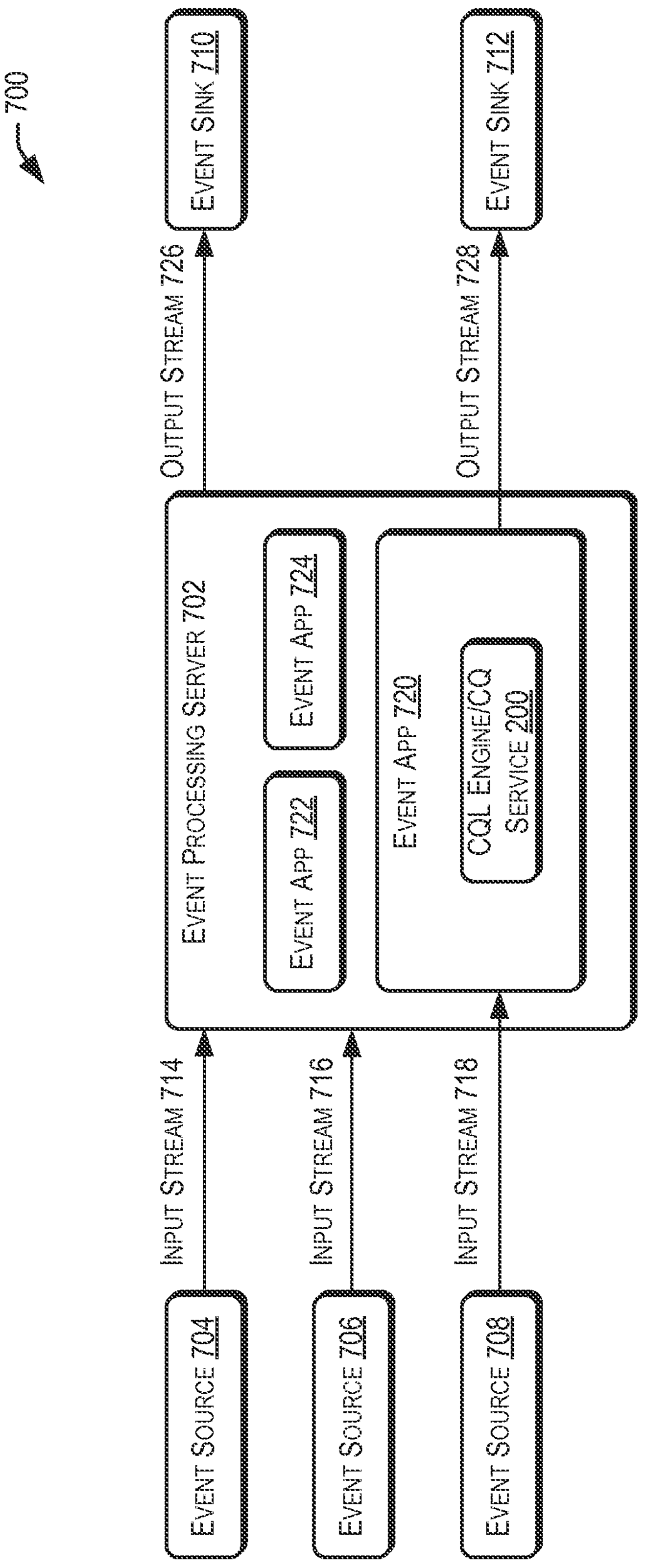
700
Event Source 704
Event Source 706
Event Source 708
Input Stream 714
Input Stream 716
Input Stream 718
Event Processing Server 702
Event App 722
Event App 724
Event App 720
CQL Engine/CQ Service 200
Output Stream 726
Output Stream 728
Event Sink 710
Event Sink 712

FIG. 7

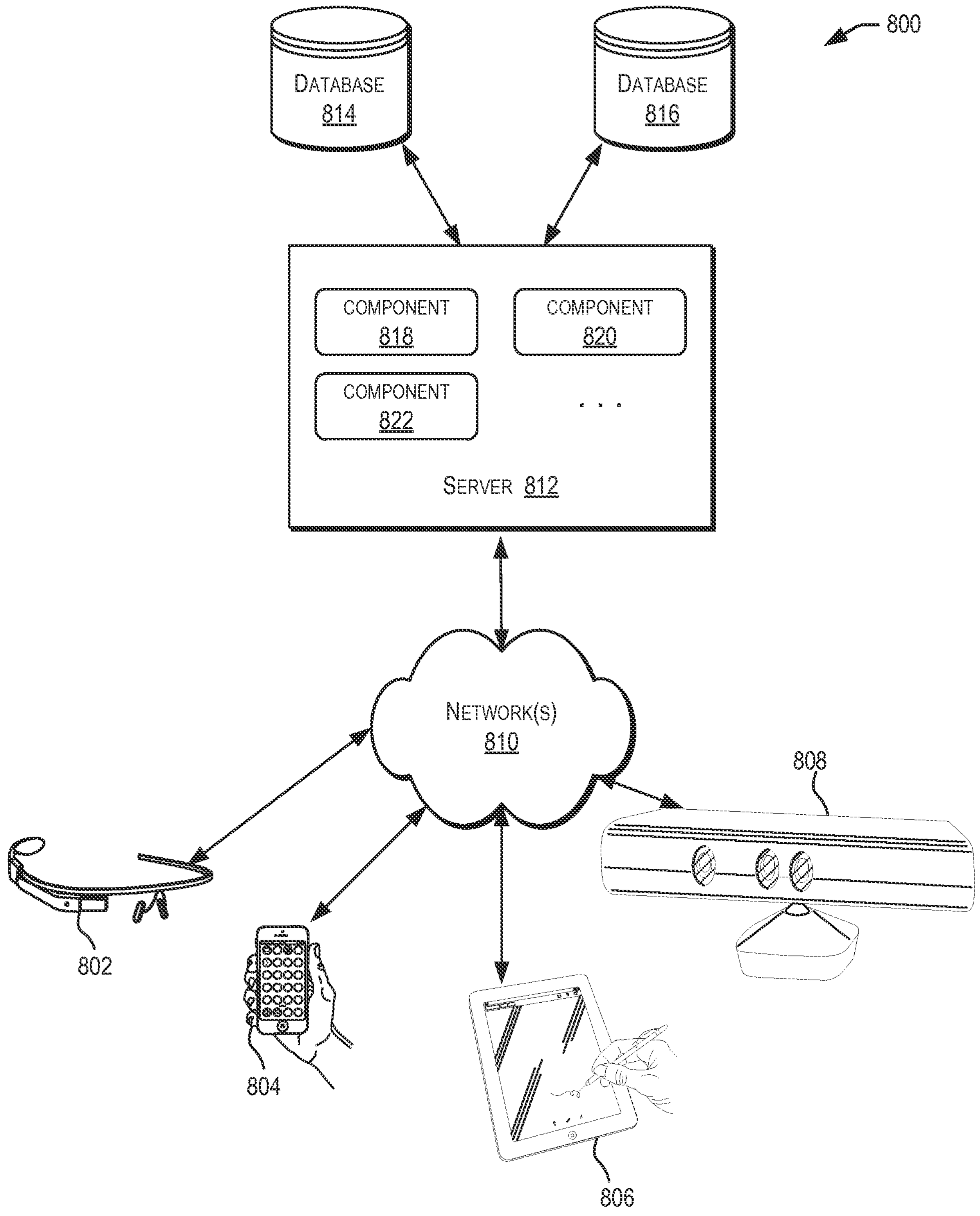
800
Database
814
Database
816
Component
818
Component
820
Component
822
. . .
Server 812
Network(s)
810
808
802
804
806

FIG. 8

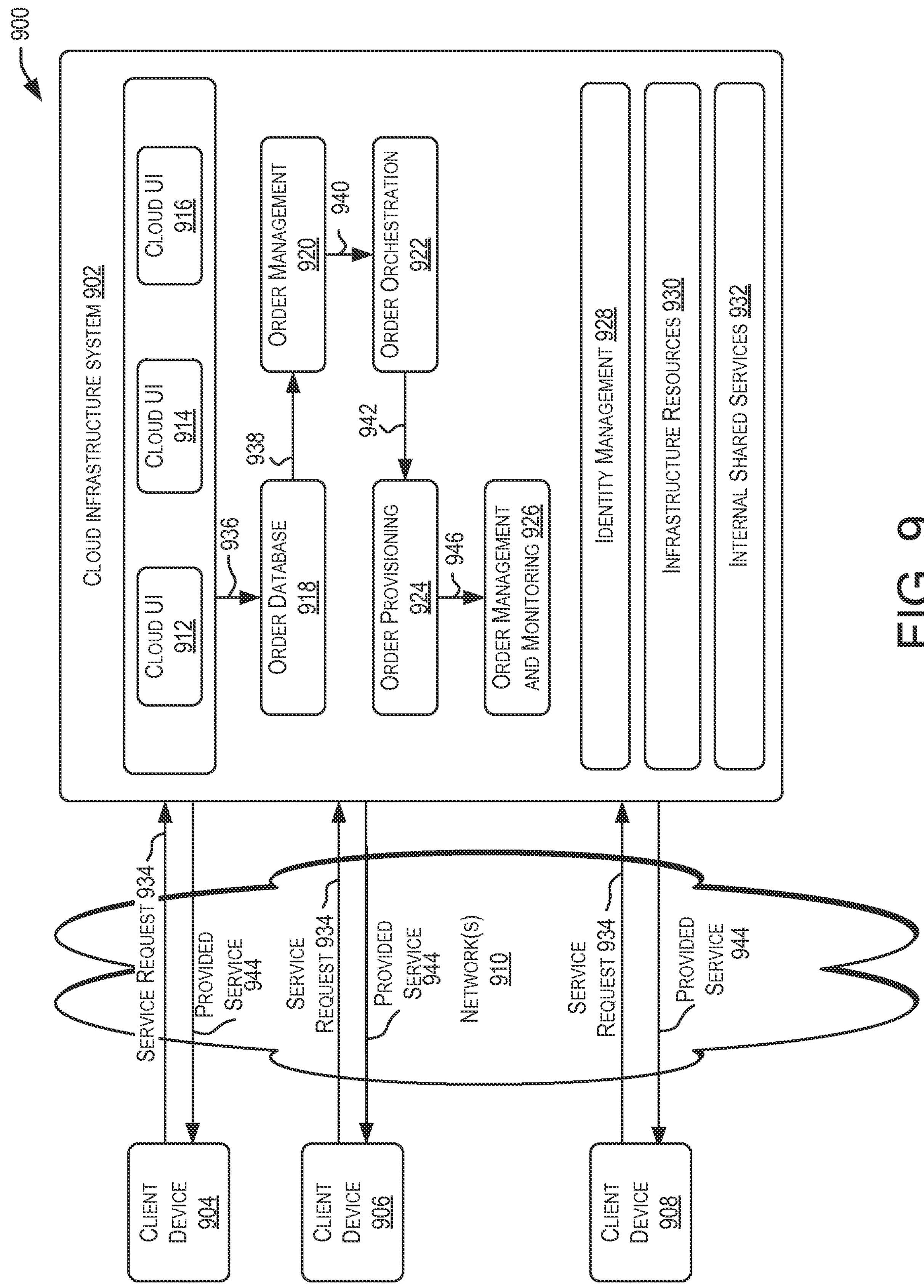
900
Cloud Infrastructure System 902
Cloud UI 912
Cloud UI 914
Cloud UI 916
936
Order Database 918
938
Order Management 920
940
Order Orchestration 922
942
Order Provisioning 924
946
Order Management and Monitoring 926
Identity Management 928
Infrastructure Resources 930
Internal Shared Services 932
Service Request 934
Provided Service 944
Service Request 934
Provided Service 944
Network(s) 910
Service Request 934
Provided Service 944
Client Device 904
Client Device 906
Client Device 908

FIG. 9

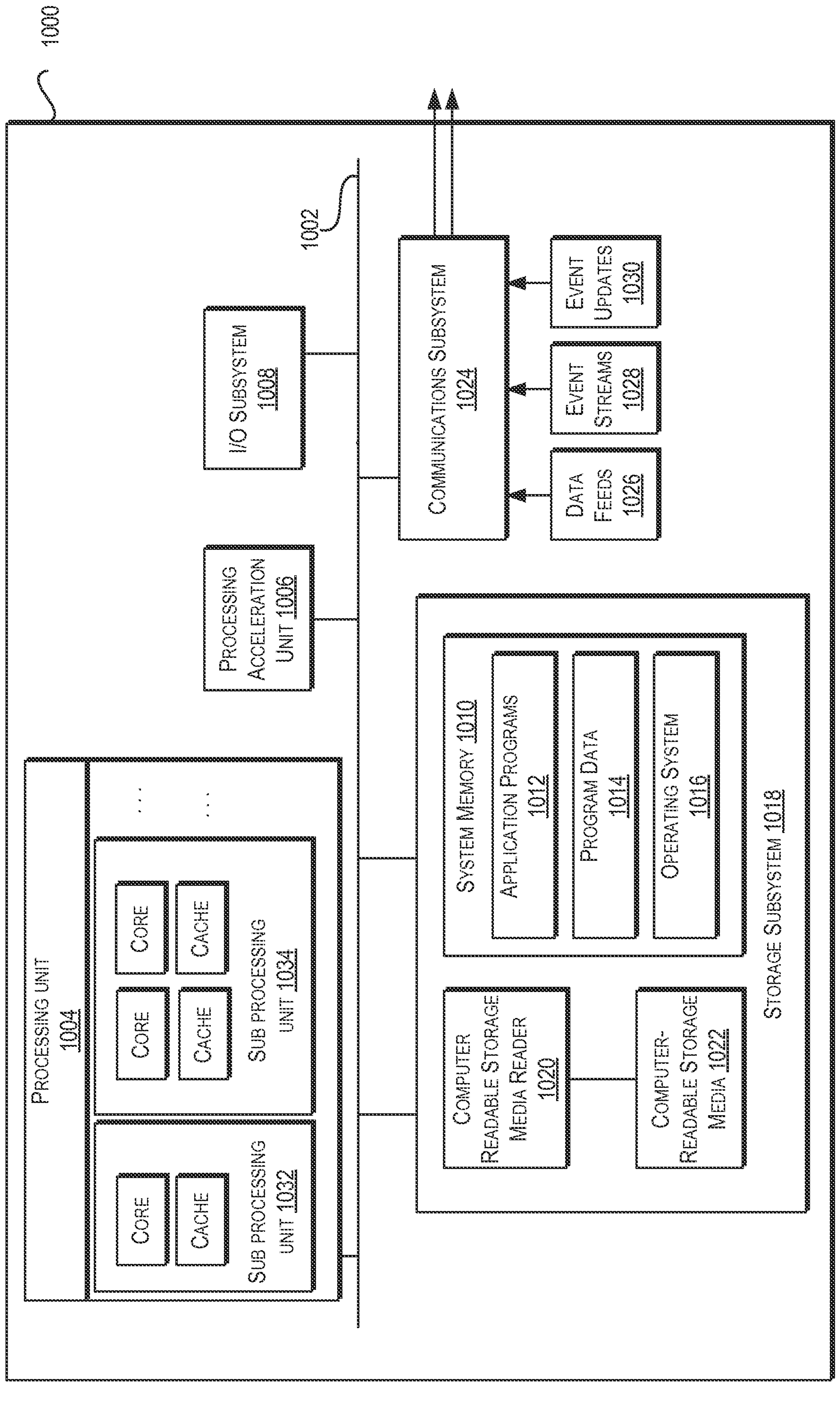
1000
1002
Processing Unit 1004
Sub Processing Unit 1032
Core
Cache
Sub Processing Unit 1034
Core
Core
Cache
Cache
Processing Acceleration Unit 1006
I/O Subsystem 1008
Communications Subsystem 1024
Data Feeds 1026
Event Streams 1028
Event Updates 1030
Storage Subsystem 1018
System Memory 1010
Application Programs 1012
Program Data 1014
Operating System 1016
Computer Readable Storage Media Reader 1020
Computer-Readable Storage Media 1022

FIG. 10

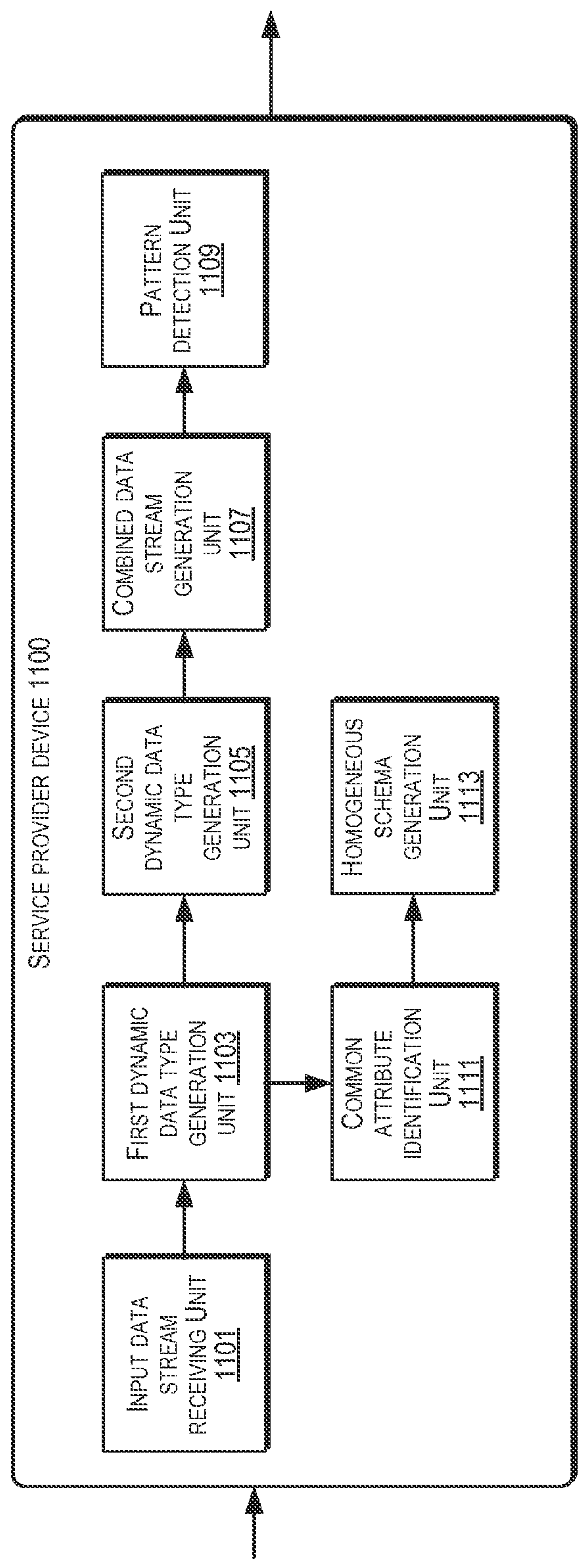
SERVICE PROVIDER DEVICE 1100
INPUT DATA STREAM RECEIVING UNIT 1101
FIRST DYNAMIC DATA TYPE GENERATION UNIT 1103
SECOND DYNAMIC DATA TYPE GENERATION UNIT 1105
COMBINED DATA STREAM GENERATION UNIT 1107
PATTERN DETECTION UNIT 1109
COMMON ATTRIBUTE IDENTIFICATION UNIT 1111
HOMOGENEOUS SCHEMA GENERATION UNIT 1113

FIG. 11

# INTERNATIONAL SEARCH REPORT

| |
|---|
| International application No |
| PCT/US2014/068641 |

**A. CLASSIFICATION OF SUBJECT MATTER**
INV. G06F17/30
ADD.

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)
G06F

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

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

EPO-Internal, WPI Data

**C. DOCUMENTS CONSIDERED TO BE RELEVANT**

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | US 2012/324453 A1 (CHANDRAMOULI BADRISH [US] ET AL) 20 December 2012 (2012-12-20) paragraph [0001] - paragraph [0002] paragraph [0025] paragraph [0032] - paragraph [0039] paragraph [0054] ----- | 1-20 |
| A | CHUCK CRANOR ET AL: "Gigascope", PROCEEDINGS OF THE 2003 ACM SIGMOD INTERNATIONAL CONFERENCE ON ON MANAGEMENT OF DATA , SIGMOD '03, 9 June 2003 (2003-06-09), - 9 June 2003 (2003-06-09), page 647, XP055170688, New York, New York, USA DOI: 10.1145/872836.872838 the whole document ----- -/-- | 1-20 |

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

[X] 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<br>19 February 2015 | Date of mailing of the international search report<br>26/02/2015 |
| Name and mailing address of the ISA/<br>European Patent Office, P.B. 5818 Patentlaan 2<br>NL - 2280 HV Rijswijk<br>Tel. (+31-70) 340-2040,<br>Fax: (+31-70) 340-3016 | Authorized officer<br>de Castro Palomares |

2

International application No
PCT/US2014/068641

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| A | ARVIND ARASU ET AL: "The CQL continuous query language: semantic foundations and query execution", THE VLDB JOURNAL ; THE INTERNATIONAL JOURNAL ON VERY LARGE DATA BASES, SPRINGER, BERLIN, DE, vol. 15, no. 2, 22 July 2005 (2005-07-22), pages 121-142, XP019431176, ISSN: 0949-877X the whole document ----- | 1-20 |
| A | DANIEL J. ABADI ET AL: "Aurora: a new model and architecture for data stream management", THE VLDB JOURNAL THE INTERNATIONAL JOURNAL ON VERY LARGE DATA BASES, vol. 12, no. 2, 16 July 2003 (2003-07-16), pages 120-139, XP055009044, ISSN: 1066-8888, DOI: 10.1007/s00778-003-0095-z page 120 - page 122 page 130 ----- | 1-20 |

2

# INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2014/068641

| Patent document cited in search report | Publication date | Patent family member(s) | Publication date |
|---|---|---|---|
| US 2012324453 A1 | 20-12-2012 | CA 2838966 A1 | 20-12-2012 |
| | | CN 103620584 A | 05-03-2014 |
| | | EP 2721511 A2 | 23-04-2014 |
| | | JP 2014524175 A | 18-09-2014 |
| | | KR 20140038462 A | 28-03-2014 |
| | | US 2012324453 A1 | 20-12-2012 |
| | | WO 2012174023 A2 | 20-12-2012 |