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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
13 November 2014 (13.11.2014)

WIPO | PCT

(10) International Publication Number
WO 2014/182310 Al

(51) International Patent Classification:
*H04L 1/18* (2006.01)

(21) International Application Number:
PCT/US20 13/040524

(22) International Filing Date:
10 May 2013 (10.05.2013)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P.** [US/US]; Hewlett-Packard Development Company, L.P., 11445 Compaq Center Drive West, Houston, Texas 77070 (US).

(72) Inventors: **HSU, Meichun;** 1501 Page Mill Road, Palo Alto, California 94304 (US). **CHEN, Qiming;** 1501 Page Mill Road, Palo Alto, California 94304 (US). **CASTELLANOS, Maria Guadalupe;** 1501 Page Mill Road, Palo Alto, California 94304 (US).

(74) Agents: **FERGUSON, Christopher W.** et al; Hewlett-Packard Company, Intellectual Property Administration, 3404 East Harmony Road, Mail Stop 35, Fort Collins, Colorado 80528 (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, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, 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, 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, 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, ML, MR, NE, SN, TD, TG).

**Declarations under Rule 4.17:**

— *as to the identity of the inventor (Rule 4.17(i))*

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

**Published:**

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

(54) Title: TUPLE RECOVERY

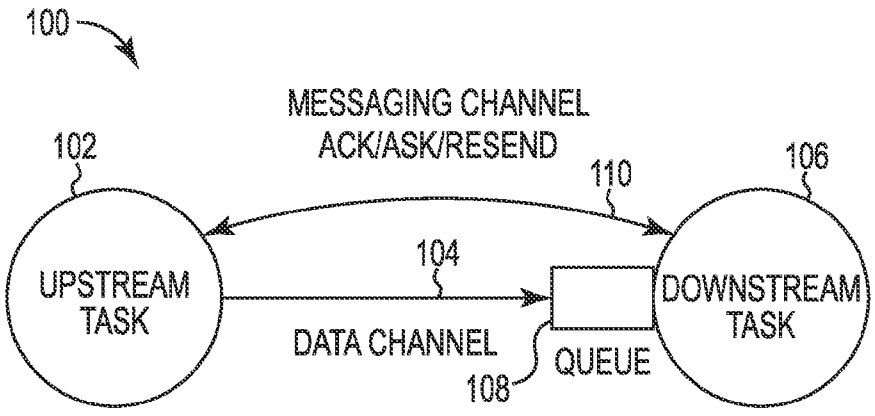


Fig. 1

(57) Abstract: A technique for recovering tuples can include sending or receiving a request to resend a tuple.

WO 2014/182310 A1

# TUPLE RECOVERY

## Background

**[0001]** Data can be sent and/or received as a data stream. A data stream can include a continuous stream of data that can be sent tuple by tuple. A data stream of tuples can be processed in a particular order.

## Brief Description of the Drawings

**[0002]** Figure 1 is an illustration of an example of a system for tuple recovery according to the present disclosure.

**[0003]** Figure 2 is an illustration of an example of a flow chart for tuple recovery according to the present disclosure.

**[0004]** Figure 3 is an illustration of an example of a flow chart for tuple recovery according to the present disclosure.

**[0005]** Figure **4** is an illustration of an example of a processing resource and memory resource for tuple recovery according to the present disclosure.

**[0006]** Figure 5 is a flow chart of an example of a method for tuple recovery according to the present disclosure.

**[0007]** Figure 6 is a flow chart of an example of a method for tuple recovery according to the present disclosure.

## Detailed Description

[0008] A data system can include a continuous stream of data (e.g., a streaming system). An example streaming system can include a distributed streaming system which can perform parallel processing (e.g., perform processing of portions of a data stream simultaneously). The sequence and/or order of communicating the data stream from a particular source to a particular destination (e.g., the dataflow) can be represented as a graph structure having nodes and edges. A node can include an electronic device and/or computer readable instructions that are capable of sending, receiving and/or forwarding a data stream over a streaming system. In some examples, an electronic device can include a plurality of nodes.

[0009] The data stream can include a sequence of data events (e.g., tuples). The tuples can be sent in a particular order from one node to another. Each node can include a task (e.g., an instance) that receives and sends tuples. A data stream can include an operation. An operation can include computer readable instructions to perform a particular function. The execution of an operation can be divided into a number of intervals (e.g., states). An operation can include multiple tasks running in parallel to perform a function.

[0010] The task can monitor the flow of tuples in and out of the task at the node. A task can be upstream or downstream depending on the flow of the data. For example, a first task (e.g., an upstream task) can receive a first tuple. The first task (e.g., upstream task, source task) can process the first tuple and send the first tuple to a second task (e.g., a downstream task, target task). The second task (e.g., downstream task, target task) can then become an upstream task to a third task that is downstream from the second task. A task can include a cycle wherein each cycle includes a number of operations. The operations can include receiving a tuple, processing the received tuple, updating an execution state, and emitting a resulting tuple to a downstream task.

**[001 1]** A task can process a tuple in a data stream tuple by tuple. A tuple may need to be processed by a task once. Tuples can be processed sequentially and a tuple prior in a sequence may need to be processed by a task before a subsequent tuple can be sent to the task. A task can save a record of the tuples received and sent. Some data streams can treat the whole streaming process as a single operation. This approach can experience the loss of intermediate results if a failure occurs.

**[0012]** Some data streams can have a downstream task send an acknowledgment to an upstream task that a tuple has been received. In such data streams, a subsequent tuple may not be sent until the acknowledgment is received. If a first tuple is never received, the second tuple may not be sent. If the acknowledgment is never received, the second tuple may not be sent. The acknowledgment approach can create latencies in the data stream.

**[0013]** In contrast, examples of the present disclosure can include sending a tuple from an upstream node to a downstream node. The state of a node (e.g., what tuples have been sent and/or received) can be recorded (e.g., checkpointed) when a tuple is sent and/or received. Maintaining a record of the last sent and/or received tuple can allow for a recovery at smaller intervals (e.g., a recovery of intermediate results without recovery of the whole streaming process) in the case of a node failure. In addition, the tuple can be sent from an upstream node to a downstream node without waiting for acknowledgment that a sequentially previous tuple has been received at the downstream node. For example, a first tuple can be sent from an upstream node to a downstream node. A second tuple can be sent from the upstream node to the downstream without the upstream node receiving an acknowledgment that the first tuple was received at the downstream node.

**[0014]** The streaming of data can avoid latencies in the system by sending data without first receiving an acknowledgment to proceed. Since failures are less frequent, acknowledgments are not necessary for each sent tuple. When a failure does occur, a process of asking an upstream node to resend a missing tuple can avoid loss of tuples that were not initially received.

**[0015]** Figure 1 is an illustration of an example of a system 100 for tuple recovery according to the present disclosure. The system 100 can include an upstream task 102 (e.g., an upstream node) sending data (e.g., a tuple) over a data channel 104 to a downstream task 106 (e.g., a downstream node). A data channel

104 can include a communication link between an upstream node and a downstream node. The data channel 104 can be used to send tuples from the data stream in sequential order. The downstream task 106 can include a queue 108 (e.g., a buffer, a data storage) to store data received from the upstream task 102.

**[0016]** The downstream task 106 can send a message over a messaging channel 110 to the upstream task 102. A messaging channel 110 can include a communication link between an upstream node and a downstream node. The messaging channel 110 can be used to send a request to resend a tuple, to send an acknowledgment that a tuple was received, and to resend the requested tuple. The messaging channel can be used to send a tuple that is out of order based on the request to resend the tuple. The message can include a request ("ASK") to resend missing data (e.g., a tuple). The upstream task 102 can resend the requested missing data over the messaging channel 110 to the downstream task 106. When a downstream task 106 receives data over a data channel 104 and/or a messaging channel 110, the downstream task can send an acknowledgment ("ACK") over the messaging channel 110 indicating the data was received.

**[0017]** Data sent over a messaging channel 110 can be tracked logically by using the messaging channel as a virtual channel. A virtual channel can allow a task to be identified by a task alias (e.g., a task identifier). The task alias can be used in reasoning, tracking, and communicating the channel information logically.

**[0018]** The data (e.g., tuples) can be sent from the upstream task 102 to the downstream task 106 over the data channel 104 in a sequential order. For example, a first tuple can be sent first and a second tuple can be sent second. The data (e.g., tuples) can include message identifiers. The message identifiers can include sequence information. The sequence information can indicate an order of the data (e.g., the tuples). For example, a first message identifier associated with a first tuple can indicate that the first tuple should be sent first and/or processed first. In addition, a second message identifier associated with a second tuple can indicate that the second tuple should be sent second and/or processed second.

**[0019]** A sequence of data (e.g., tuples) can indicate missing data when data is out of sequence. For example, a sequentially first tuple can be sent from an upstream task 102 (e.g., associated with an upstream node) to a downstream task 106 (e.g., associated with a downstream node). In some examples, a task can be associated with a node. In some examples, a number of tasks can be associated

with a node. A sequentially third tuple can be sent from the upstream task 102 to the downstream task 106. If the downstream task 106 did not receive a sequentially second tuple after the first tuple and before the third tuple, a missing tuple can be identified. An identified missing tuple can be requested over a messaging channel 110 by sending a request from a downstream task to an upstream task to resend the missing tuple.

**[0020]** Figure 2 is an illustration of an example of a flow chart 201 for tuple recovery according to the present disclosure. A flow chart 201 is an example of how a task can process (e.g., perform an execution loop and/or operation for) a cycle of a tuple. A task can de-queue tuple input, at 220 (e.g., as illustrated by queue 108 in Figure 1). A sequential order of the tuple can be checked, at 222. If the tuple is a duplicate tuple (e.g., the information in the received tuple has already been sent in a previous tuple), the tuple is not processed (e.g., ignored) and an acknowledgment ("ACK") can be sent over a messaging channel (e.g., 110 in Figure 1) to an upstream task, at 226. The acknowledgment can indicate to an upstream task (e.g., 102 in Figure 1) that the upstream task can release data associated with the duplicate tuple from the queue.

**[0021]** If the tuple sent to a downstream task is out of order 228 (e.g., an additional tuple should have been sent before the received tuple and is therefore missing), the downstream task can send a request (an "ASK") and the re-sent tuple can be processed, at 230. The processing of the re-sent tuple can include going through the operation of execution (e.g., flowchart 201) by returning (e.g., dotted arrow) to the first operation of de-queuing the input (e.g., at 220). For a re-sent tuple or a tuple that was sent in sequential order, an input channel (e.g., the channel that sent the tuple) can be recorded, at 232. An input channel (e.g., a data channel) can include the channel that the tuple was received through. The recorded input channel can allow the downstream task to determine what upstream task the tuple came from. A sequence number associated with the received tuple can also be recorded, at 232. A sequence number can include an indication of where in a sequence of tuples a tuple should be sent and received in a dataflow of tuples. A sequence identifier associated with the tuple can indicate the sequence number. A sequence number can indicate where in the dataflow stream a tuple should be sent. For example, a sequence number can indicate a tuple should be sent third in a flow of tuples.

**[0022]** The received tuple can be processed and output data based on the input tuple can be derived, at 234. Output data can include tuples sent from a task based on the tuples received. For example, a received tuple can indicate additional tuples and/or data to send along with the received tuple. The output data can reflect that indication. The output data can include the same data that was received. An output channel for the derived output can be determined (e.g., reasoned), at 236. An output channel can include a channel that the output data is sent through to a further downstream task. The output channel can be recorded and a sequence number associated with the derived output, at 238. A state of the task (e.g., current status of input tuples and output tuples) can be determined at a checkpoint 240. For example, a state of a task can include a number of tuples that were received and a number of tuples that are to be sent further downstream. At 242, an acknowledgment that a received tuple has been processed can be sent to an upstream task. The derived output can be emitted, at 244. An output tuple (e.g., derived output) can be stored until an acknowledgment is received indicating the emitted output has been received (e.g., acknowledged (ACKed)) at a downstream task.

**[0023]** Figure 3 is an illustration of an example of a flow chart 303 for tuple recovery according to the present disclosure. The flow chart 303 can be an example of a process that runs before processing input tuples. A task can perform an initiating operation that can include initiating a static state, at 350. An initiating operation can be performed to use a second node when a first node experiences a failure. A static state can include an indication of how many tuples have been received, processed, and sent along with input channel and output channel data. The static state can determine what tuple the task should receive next and what tuple to send next. A task can check a status of the task, at 352. A status can include that the node is being used to process tuples as a new task for that node. A status can include that the node has experienced a failure and is restoring a previous state of the node. During a first-time initiation, a new dynamic state can be initiated, at 356. For example, a first node can experience a failure. A second node can be used to replace the first node. A first time initiation of the second node can include initiating a new dynamic state for the second node so the second node can process tuples for the task. The new dynamic state can include setting up the node to operate a task and process tuples. The task including a new dynamic state can proceed through an execution loop, at 358 (e.g., the process in Figure 2).

**[0024]** The process illustrated by the flowchart 303 can be used to recover from a failure of a task (e.g., a node). If a status check of the task indicates that the task has recovered from a failure, the process can include recovering a latest dynamic state, at 362. A latest dynamic state can include a state of the task when the task failed. The task can recover a latest dynamic state by restoring the latest state of the task at the time of failure, at 364. The latest state can determine which tuples to re-emit. The latest output tuples can be re-emitted to a downstream task (e.g., a target task of the recovered task), at 366. The process can include sending a request (an "ASK") to an upstream task (e.g., a source task) to resend an input tuple, at 368. The process can proceed to an execution loop, at 358 (e.g., the process illustrated in Figure 2).

**[0025]** Figure 4 is an illustration of an example of a system 405 for tuple recovery accordirig to the present disclosure. The system 405 can include software, hardware, firmware, and/or logic to perform a number of functions.

**[0026]** The system 405 can include any combination of hardware and program instructions configured to recover a tuple in a data stream. The hardware, for example can include a processing resource 470, and/or a memory resource 474 (e.g., computer-readable medium (CRM), machine readable medium (MRM), database, etc.) A processing resource 470, as used herein, can include any number of processors capable of executing instructions stored by a memory resource 474. Processing resource 470 may be integrated in a single device or distributed across devices. The program instructions (e.g., computer-readable instructions (CRI)) can include instructions stored on the memory resource 474 and executable by the processing resource 470 to implement a desired function (e.g., determine a plurality of attributes for a plurality of tuples, etc.).

**[0027]** The memory resource 474 can be in communication with a processing resource 470. The memory resource 474 can be in communication with the processing resource 470 via a communication path 472. The communication path 472 can be local or remote to a machine (e.g., a computing device) associated with the processing resource 470. Examples of a local communication path 472 can include an electronic bus internal to a machine (e.g., a computing device) where the memory resource 474 is one of volatile, non-volatile, fixed, and/or removable storage medium in communication with the processing resource 470 via the electronic bus.

**[0028]** The communication path 472 can be such that the memory resource 474 is remote from the processing resource (e.g., 470), such as in a network connection between the memory resource 474 and the processing resource (e.g., 470). That is, the communication path 472 can be a network connection. Examples of such a network connection can include a local area network (LAN), wide area network (WAN), personal area network (PAN), and the Internet, among others. In such examples, the memory resource 474 can be associated with a first computing device and the processing resource 470 can be associated with a second computing device (e.g., a Java® server).

**[0029]** A memory resource 474, as used herein, can include any number of memory components capable of storing instructions that can be executed by processing resource 470. Such memory resource 474 can be a non-transitory CRM. Memory resource 474 may be integrated in a single device or distributed across devices. Further, memory resource 474 may be fully or partially integrated in the same device as processing resource 470 or it may be separate but accessible to that device and processing resource 470. Thus, it is noted that the system 405 may be implemented on a user and/or a client device, on a server device and/or a collection of server devices, and/or on a combination of the user device and the server device and/or devices.

**[0030]** The processing resource 470 can be in communication 472 with a memory resource 474 storing a set of CRI executable by the processing resource 470, as described herein. The CRI can also be stored in remote memory managed by a server and represent an installation package that can be downloaded, installed, and executed. The system 405 can include memory resource 474, and the processing resource 470 can be coupled to the memory resource 474.

**[0031]** Processing resource 470 can execute CRI that can be stored on an internal or external memory resource 474. The processing resource 470 can execute CRI to perform various functions, including the functions described with respect to Figures 1-3, and 5-6. For example, the processing resource 470 can execute CRI to send a request to resend a tuple.

**[0032]** The CRI can include a number of modules 476, 478, 480. The number of modules 476, 478, 480 can include CRI that when executed by the processing resource 470 can perform a number of functions. In a number of examples, the number of modules 476, 478, 480 can include logic. As used herein, "logic" is an

alternative or additional processing resource to execute the actions and/or functions, etc., described herein, which includes hardware (e.g., various forms of transistor logic, application specific integrated circuits (ASICs), etc.), as opposed to computer executable instructions (e.g., software, firmware, etc.) stored in memory and executable by a processor.

**[0033]** The number of modules 476, 478, 480 can be sub-modules of other modules. For example, the receiving module 476 and the determining module 478 can be sub-modules and/or contained within the same computing device. In another example, the number of modules 476, 478, 480 can comprise individual modules at separate and distinct locations (e.g., CRM, etc.).

**[0034]** An receiving module 476 can include CRI that when executed by the processing resource 470 can provide a number of receiving functions. The receiving module 476 can receive a first tuple and a second tuple at a downstream node.

**[0035]** A determining module 478 can include CR! that when executed by the processing resource 470 can perform a number of determining functions. The determining module 478 can determine a third tuple was not received that should have been received after the first tuple and before the second tuple. The determining module 478 can determine the third tuple should have been received based on sequence identifiers associated with the first and second tuples, for example.

**[0036]** A sending module 480 can include CRI that when executed by the processing resource 470 can perform a number of sending functions. The sending module 480 can send a request over a messaging channel (e.g., as illustrated by 110 in Figure 1) to resend a missing tuple. The messaging channel can be a different channel than a data channel (e.g., 104 in Figure 1). For instance, the messaging channel can send acknowledgments that a tuple was received. The messaging channel can send requests to resend a tuple that is missing. The messaging channel can send the missing tuple in response to a request to resend the missing tuple.

**[0037]** A memory resource 474, as used herein, can include volatile and/or non-volatile memory. Volatile memory can include memory that depends upon power to store information, such as various types of dynamic random access memory (DRAM), among others. Non-volatile memory can include memory that does not depend upon power to store information.

**[0038]** The memory resource 474 can be integral, or communicatively coupled, to a computing device, in a wired and/or a wireless manner. For example, the memory resource 474 can be an interna! memory, a portable memory, a portable disk, or a memory associated with another computing resource (e.g., enabling CRIs to be transferred and/or executed across a network such as the Internet).

**[0039]** For example, a processing resource 470 can be in communication with a memory resource 474, wherein the memory resource 474 includes a set of instructions and wherein the processing resource 470 is designed to carry out the set of instructions.

**[0040]** Figure 5 is a flow chart illustrating an example of a method 507 for tuple recovery according to the present disclosure. At 582, the method can include sending a first tuple from an upstream node. The first tuple can be a sequentially first tuple. That is, the first tuple can be sent first in an order of tuples.

**[0041]** At 584, the method 507 can include sending a second tuple from the upstream node without receiving an acknowledgment for the first tuple. The second tuple can be a sequentially third tuple. That is, the second tuple should be sent third in an order of tuples. If the second tuple is sent after the first tuple, the third tuple (e.g., a tuple that should be sent second) can be missing. The third tuple can be lost while sending the tuples from an upstream node to a downstream node. The third tuple can be lost due to a failure of the downstream node. For example, a first tuple can be sent to the downstream node. The downstream node can experience a failure while the upstream node is sending a sequentially second tuple. The downstream node can receive a sequentially third tuple after the first tuple and can be missing the sequentially second tuple.

**[0042]** At 586, the method 507 can include receiving a request to resend a tuple for recovering a downstream node. The request to resend a tuple can include sending a request over a messaging channel.

**[0043]** Figure 6 is a flow chart illustrating an example of a method 609 according to the present disclosure. At 688, the method 609 can include sending a first tuple from an upstream node. The first tuple can be a sequentially first tuple. That is, the first tuple can be sent first in an order of tuples. At 690, the method 609 can include sending a second tuple from the upstream node without receiving an acknowledgment for the first tuple.

**[0044]** At 692, the method 609 can include receiving a request using a messaging channel to resend a tuple for a recovering downstream node. The messaging channel can transfer a request to resend a tuple from a downstream node to an upstream node. The messaging channel can transfer an acknowledgment that indicates a downstream node received a sent tuple from an upstream node.

**[0045]** At 694, the method 609 can include identifying a sequence of the tuples sent from the upstream node using sequence identifiers. The sequence identifiers can be recorded during an execution loop (e.g., as illustrated by flow chart 201 in Figure 2). The sequence number associated with the sequence identifier can be recorded when the tuple input is received (e.g., 232 of Figure 2). The sequence number associated with the sequence identifier can be recorded when the tuple output is sent (e.g., 238 in Figure 2).

**[0046]** At 696, the method 609 can include performing a checkpoint to record a state of a task, wherein the task comprises a number of tuples. The checkpoint can consist of a list of objects (e.g., a list of inputs, outputs, etc.). When a check-in is performed, the list is serialized into a byte-array to write to a binary file. When a check-out is performed, the byte-array obtained from reading the file is de-serialized to the list of objects representing the state.

**[0047]** At 698, the method 609 can include emptying a tuple from a buffer when an acknowledgment associated with the tuple is received at the upstream node. The buffer of an upstream node can store a tuple that has already been sent to a downstream node. The buffer can store the tuple in case a request is received at the upstream node to resend the tuple. An acknowledgment that the tuple has been received at the downstream node can indicate that the upstream node buffer can empty the buffer of the received tuple.

**[0048]** in the detailed description of the present disclosure, reference is made to the accompanying drawings that form a part hereof, and in which is shown by way of illustration how examples of the disclosure may be practiced. These examples are described in sufficient detail to enable those of ordinary skill in the art to practice the examples of this disclosure, and it is to be understood that other examples may be used and the process, electrical, and/or structural changes may be made without departing from the scope of the present disclosure.

[0049] The specification examples provide a description of the applications and use of the system and method of the present disclosure. Since many examples can be made without departing from the spirit and scope of the system and method of the present disclosure, this specification sets forth some of the many possible example configurations and implementations.

What is claimed:

1. A method for tuple recovery, comprising:

sending a first tuple from an upstream node;

sending a second tuple from the upstream node without receiving an acknowledgment for the first tuple; and

receiving a request to resend a tuple for a recovering downstream node.

2. The method of claim 1, comprising identifying a sequence of the tuples sent from the upstream node using sequence identifiers.

3. The method of claim 1, wherein receiving the request includes receiving the request using a messaging channel.

4. The method of claim 1, comprising performing a checkpoint to record a state of a task, wherein the task comprises a number of tuples.

5. The method of claim 1, comprising emptying a tuple from a buffer when an acknowledgment associated with the tuple is received at the upstream node.

6. A system for tuple recovery, the system comprising:

a processing resource;

a memory resource coupled to the processing resource to implement:

a receiving module to receive a first tuple and a second tuple at a downstream node;

a determining module to determine a third tuple was not received that should have been received after the first tuple and before the second tuple; and

a sending module to send a request over a messaging channel to resend the third tuple.

7. The system of claim 6, comprising receiving the third tuple over the messaging channel.

8. The system of claim 6, comprising a checkpointing module to checkpoint a state of a task comprising sequence information associated with the received first and second tuples.

9. The system of claim 6, comprising an acknowledging module to send an acknowledgment of a received duplicate tuple of an already received tuple at the downstream node, wherein the downstream node does not process the duplicate tuple.

10. The system of claim 6, wherein the third tuple was not received due to a failure of the downstream node.

11. A non-transitory computer-readable medium storing a set of instructions executable by a processing resource to cause a computer to:

check a status of a recovered downstream node;

restore a state of the recovered downstream node;

send a request over a messaging channel to send a tuple; and

receive the tuple over the messaging channel.

12. The medium of claim 11, comprising instructions executable by a processing resource to cause a computer to receive a sequentially second tuple over a data channel without the downstream node sending an acknowledgment indicating the downstream node has received a sequentially first tuple.

13. The medium of claim 11, comprising instructions executable by a processing resource to cause a computer to emit output tuples.

14. The medium of claim 11, comprising instructions executable by a processing resource to cause a computer to record a sequence number of the tuple.

15. The medium of claim 11, comprising instructions executable by a processing resource to cause a computer to store data in a buffer associated with sent tuples until an acknowledgment is received indicating the sent tuples were received.

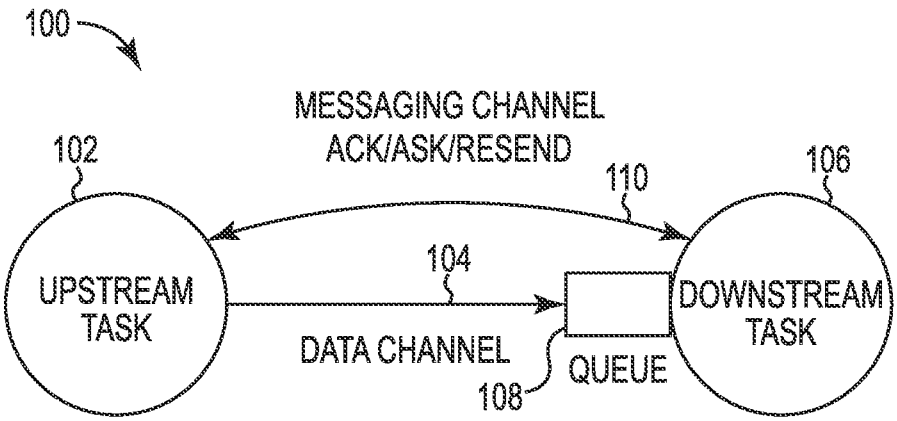
100
MESSAGING CHANNEL
ACK/ASK/RESEND
102
110
106
104
UPSTREAM TASK
DOWNSTREAM TASK
DATA CHANNEL
QUEUE
108

Fig. 1

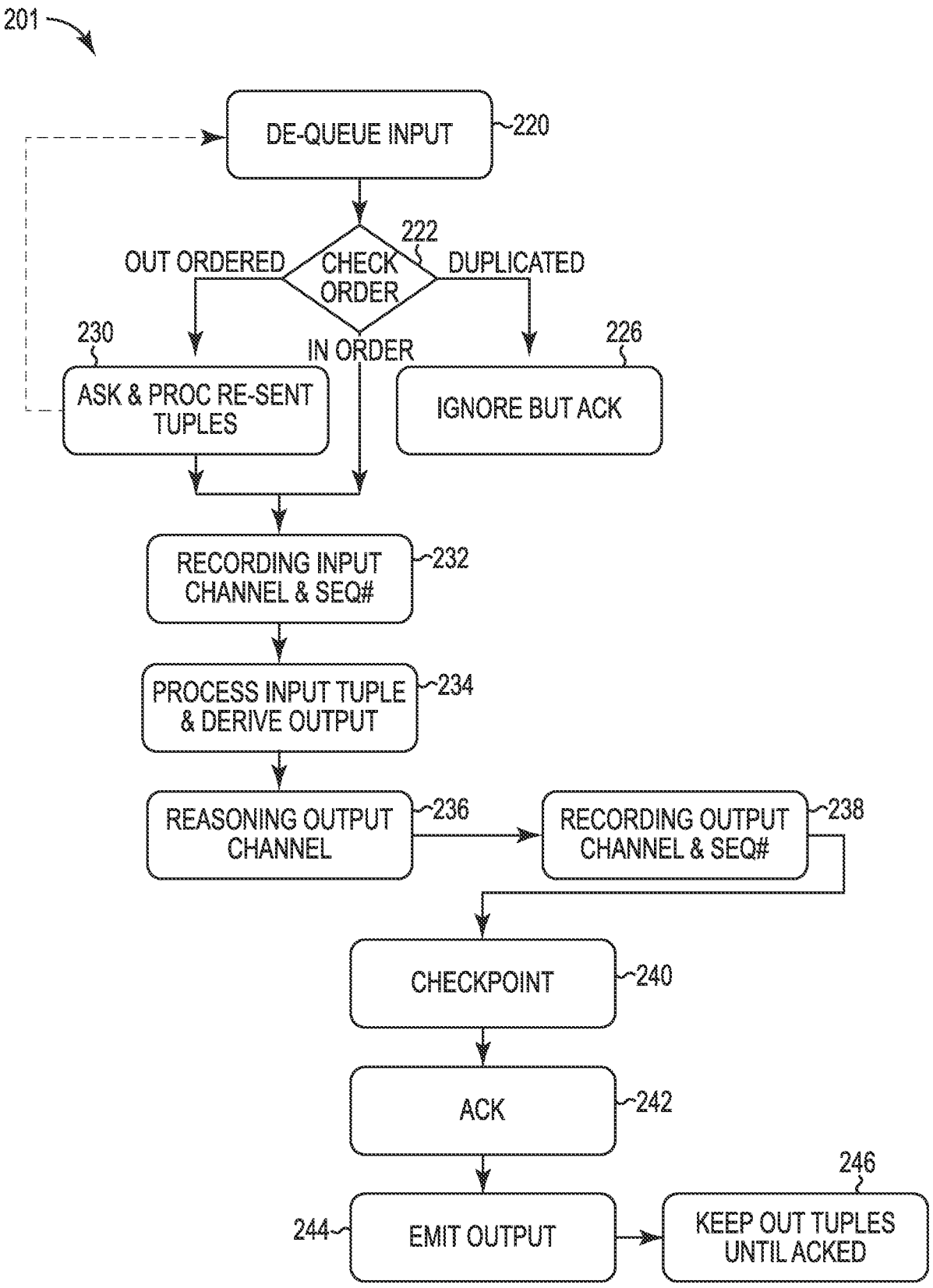
201
DE-QUEUE INPUT
220
222
CHECK ORDER
OUT ORDERED
DUPLICATED
IN ORDER
230
ASK & PROC RE-SENT TUPLES
226
IGNORE BUT ACK
RECORDING INPUT CHANNEL & SEQ#
232
PROCESS INPUT TUPLE & DERIVE OUTPUT
234
REASONING OUTPUT CHANNEL
236
RECORDING OUTPUT CHANNEL & SEQ#
238
CHECKPOINT
240
ACK
242
246
244
EMIT OUTPUT
KEEP OUT TUPLES UNTIL ACKED

Fig. 2

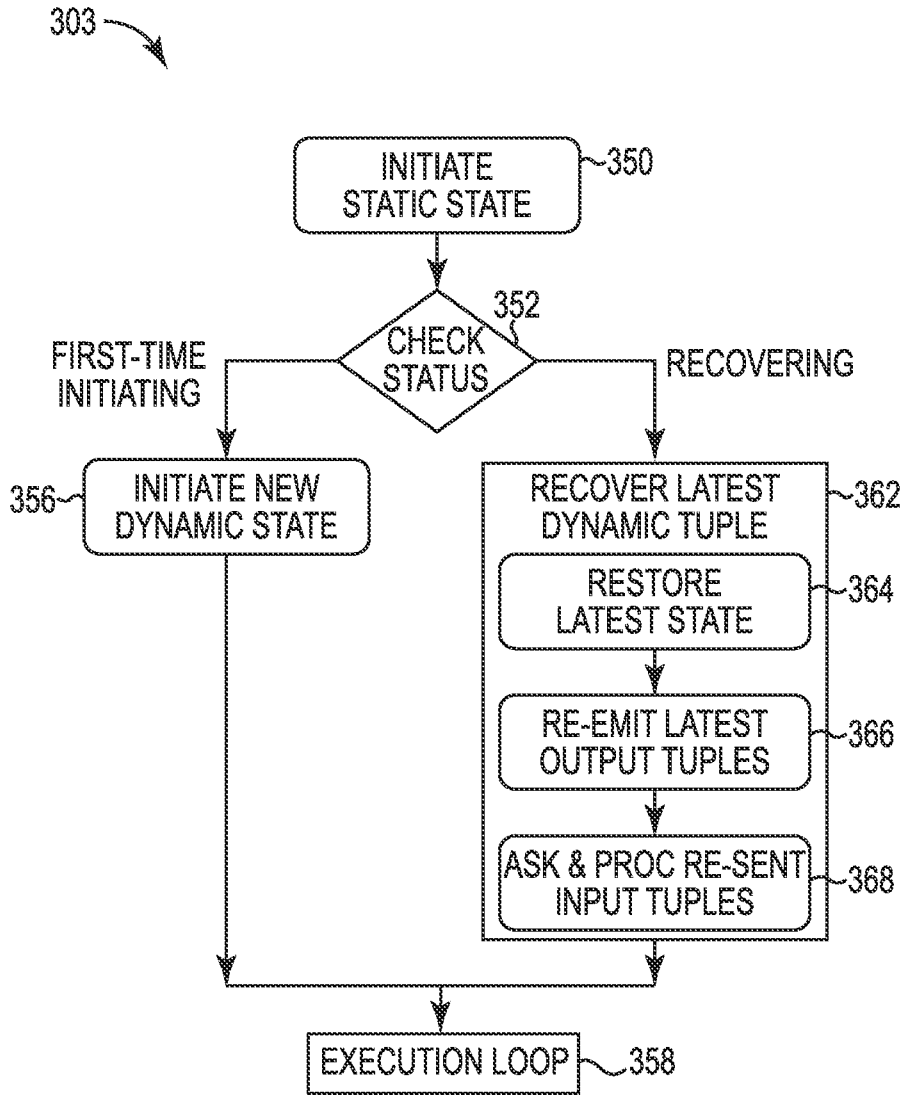
303
INITIATE STATIC STATE
350
CHECK STATUS
352
FIRST-TIME INITIATING
RECOVERING
356
INITIATE NEW DYNAMIC STATE
RECOVER LATEST DYNAMIC TUPLE
362
RESTORE LATEST STATE
364
RE-EMIT LATEST OUTPUT TUPLES
366
ASK & PROC RE-SENT INPUT TUPLES
368
EXECUTION LOOP
358

Fig. 3

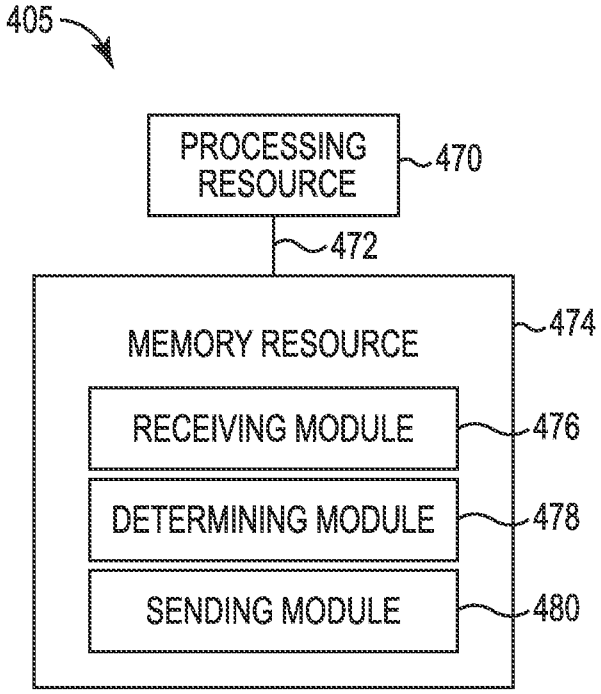
405
PROCESSING RESOURCE
470
472
MEMORY RESOURCE
474
RECEIVING MODULE
476
DETERMINING MODULE
478
SENDING MODULE
480

Fig. 4

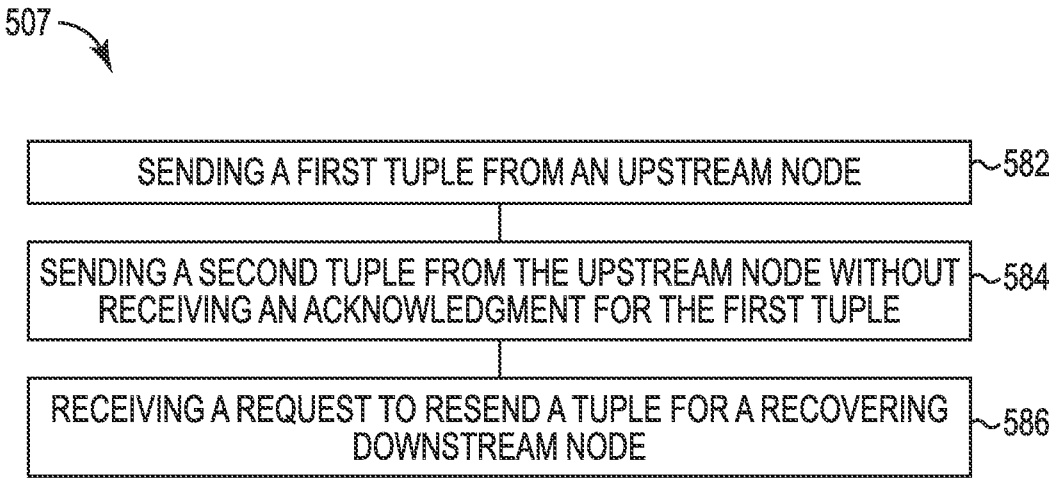
507
SENDING A FIRST TUPLE FROM AN UPSTREAM NODE
582
SENDING A SECOND TUPLE FROM THE UPSTREAM NODE WITHOUT RECEIVING AN ACKNOWLEDGMENT FOR THE FIRST TUPLE
584
RECEIVING A REQUEST TO RESEND A TUPLE FOR A RECOVERING DOWNSTREAM NODE
586

Fig. 5

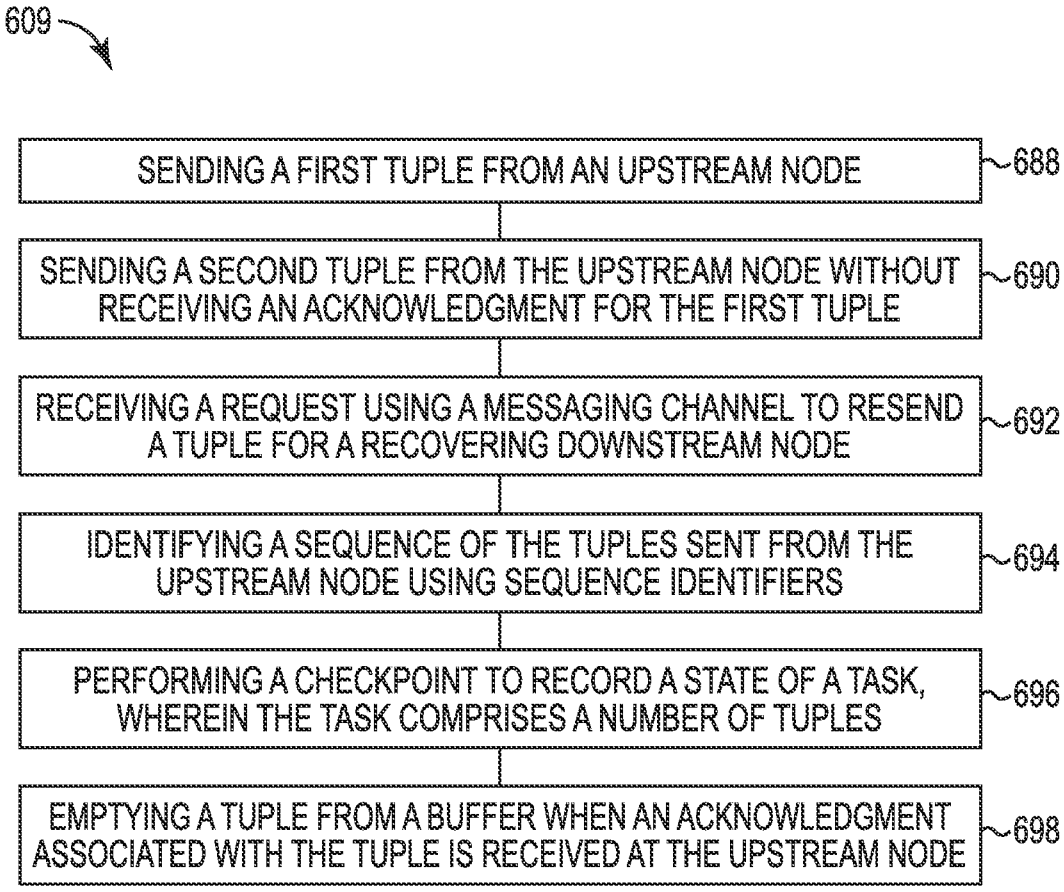
609
SENDING A FIRST TUPLE FROM AN UPSTREAM NODE
688
SENDING A SECOND TUPLE FROM THE UPSTREAM NODE WITHOUT RECEIVING AN ACKNOWLEDGMENT FOR THE FIRST TUPLE
690
RECEIVING A REQUEST USING A MESSAGING CHANNEL TO RESEND A TUPLE FOR A RECOVERING DOWNSTREAM NODE
692
IDENTIFYING A SEQUENCE OF THE TUPLES SENT FROM THE UPSTREAM NODE USING SEQUENCE IDENTIFIERS
694
PERFORMING A CHECKPOINT TO RECORD A STATE OF A TASK, WHEREIN THE TASK COMPRISES A NUMBER OF TUPLES
696
EMPTYING A TUPLE FROM A BUFFER WHEN AN ACKNOWLEDGMENT ASSOCIATED WITH THE TUPLE IS RECEIVED AT THE UPSTREAM NODE
698

Fig. 6

**INTERNATIONAL SEARCH REPORT**

International application No.
**PCT/US2013/040524**

**A. CLASSIFICATION OF SUBJECT MATTER**

**H04L l/18(2006.01)i**

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)
H04L 1/18; G06F 15/16; G06F 11/07; H04B 7/216; H04L 12/28; G01R 31/08; G06F 9/45; H04J 3/24

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & Keywords: acknowledgment, sequence, recovery, tuple, checkpoint, buffer

**C. DOCUMENTS CONSIDERED TO BE RELEVANT**

| Category * | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | US 2010-0161824 Al (PASCAL VIGER et al.) 24 June 2010<br>See paragraphs 14, 48; claims 1, 8; and figure 1. | 1 |
| Y | | 2-15 |
| Y | US 7835273 B2 (JIN-HYUN SIN) 16 November 2010<br>See column 2, line 4; column 5, lines 61-63; column 6, lines 2-4; claim *V*, and figure 4. | 6-10 |
| Y | US 2010-0293532 Al (HENRIQUE ANDRADE et al.) 18 November 2010<br>See abstract; paragraph 27; claims 10, 19; and figures 2A-2B. | 4,8,11-15 |
| Y | US 2003-0202500 Al (SANG-HYUCK HA et al.) 30 October 2003<br>See paragraphs 15, 49-52; claim 1; and figure 2. | 2-3,5,14-15 |
| Y | US 7965698 B2 (JAE-GYU JUNG) 21 June 2011<br>See abstract; column 5, lines 20-29; claim *V*, and figures 3-4. | 9 |

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

| Date of the actual completion of the international search | Date of mailing of the international search report |
|---|---|
| 11 February 2014 (11.02.2014) | **12 February 2014 (12.02.2014)** |


Name and mailing address of the ISA/KR
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City, 302-701, Republic of Korea
Facsimile No. +82-42-472-7140

Authorized officer
KANG, Hee Gok
Telephone No. +82-42-481-8264

**INTERNATIONAL SEARCH REPORT**

Information on patent family members

International application No.
**PCT/US2013/040524**

| Patent document cited in search report | Publication date | Patent family member(s) | Publication date |
|---|---|---|---|
| US 2010-0161824 Al | 24/06/2010 | FR 2939994 Al | 18/06/2010 |
| | | FR 2939994 Bl | 17/12/2010 |
| US 7835273 B2 | 16/11/2010 | CN 1700629 A | 23/11/2005 |
| | | CN 1700629 C | 16/04/2008 |
| | | KR 10-0533686 Bl | 05/12/2005 |
| | | KR 10-2005-0111194 A | 24/11/2005 |
| | | US 2005-0259577 Al | 24/11/2005 |
| US 2010-0293532 Al | 18/11/2010 | None | |
| US 2003-0202500 Al | 30/10/2003 | AU 2003-224464 Al | 10/11/2003 |
| | | AU 2003-224464 B2 | 25/08/2005 |
| | | BR 0304554 A | 19/10/2004 |
| | | CA 2452268 Al | 06/11/2003 |
| | | CA 2452268 C | 12/07/2011 |
| | | CN 1288872 C | 06/12/2006 |
| | | CN 1537372 A | 13/10/2004 |
| | | CN 1537372 C | 06/12/2006 |
| | | DE 60328148 Dl | 13/08/2009 |
| | | EP 1357695 A2 | 29/10/2003 |
| | | EP 1357695 A3 | 16/05/2007 |
| | | EP 1357695 Bl | 01/07/2009 |
| | | JP 03967355 B2 | 29/08/2007 |
| | | JP 2005-523669 A | 04/08/2005 |
| | | KR 10-0547892 Bl | 31/01/2006 |
| | | KR 10-2003-0084735 A | 01/11/2003 |
| | | RU 2003137005 A | 27/05/2005 |
| | | RU 2267225 C2 | 27/12/2005 |
| | | US 7447968 B2 | 04/11/2008 |
| | | WO 03-092213 Al | 06/11/2003 |
| US 7965698 B2 | 21/06/2011 | CN 1822532 A | 23/08/2006 |
| | | DE 602006000721 Dl | 30/04/2008 |
| | | EP 1694010 Al | 23/08/2006 |
| | | EP 1694010 Bl | 19/03/2008 |
| | | JP 04256395 B2 | 22/04/2009 |
| | | JP 2006-229955 A | 31/08/2006 |
| | | KR 10-0597425 Bl | 29/06/2006 |
| | | US 2006-0184664 Al | 17/08/2006 |