

DEVICE AND METHOD FOR DEPACKAGING DATA PACKAGED ACCORDING TO DIFFERENT PACKAGING STACKS

TECHNICAL FIELD

[0001] Various aspects of this disclosure relate to devices and methods for depackaging data packaged according to different packaging stacks.

BACKGROUND

[0002] In a data distribution environment data may be provided by different data sources which may package information according to different packaging stacks, e.g. data formats, compression schemes and encoding schemes. Depackaging messages to extract information requires a lot of computational effort and may lead to a bottleneck, e.g. in a data server responsible for data extraction, e.g. to make the data accessible to clients. This may in particular be in case of the usage of different packaging stacks by different data sources, because, while it may be possible to accelerate information extraction for a specific packaging stack, it may not be possible to provide computational resources which allow fast extraction of information for every packaging stack that is used.

[0003] Accordingly, approaches which allow fast handling of packaged message are desirable.

SUMMARY

[0004] Various embodiments concern a method for depackaging data packaged according to different packaging stacks, comprising receiving messages packaged according to different packaging stacks, wherein each message contains a sequence of packaged information elements, for each message, processing the packaged information elements in sequence, comprising, for each packaged information element that is processed, extract a corresponding information element by de-packaging the packaged information element from the message according to the packaging stack according to which the message is packaged, checking whether the extracted information element has a predefined filtering value and stopping extracting information elements from the sequence of packaged information elements if the extracted information element has the predefined filtering value.

[0005] According to one embodiment, the method further comprises filtering out the message when the information element has the predefined value.

[0006] According to one embodiment, filtering out the message comprises discarding the message.

[0007] According to one embodiment, the method comprises receiving the messages from different data sources, wherein each data source packages data into messages using a respective packaging stack, wherein the data sources use different packaging stacks.

[0008] According to one embodiment, packaging stacks being different comprises the packaging stacks differing in at least one of an encoding scheme, a data format, a compression scheme and a communication protocol according to which messages are packaged.

[0009] According to one embodiment, checking whether the extracted information element has the predefined filtering value comprises converting the information element into a predefined form and checking whether the extracted information element, in the predefined form, has the predefined filtering value.

[0010] According to one embodiment, the predefined filtering value is a value indicating an error.

[0011] According to one embodiment, the method comprises checking whether the extracted information element has any value of a plurality of predefined filtering values and stopping extracting information elements from the sequence of packaged information elements if the extracted information element has any value of the plurality of predefined filtering values.

[0012] According to one embodiment, the method comprises receiving a specification of the packaging stacks.

[0013] According to one embodiment, extracting the information element corresponding to the packaged information element comprises at least one of unwrapping, decompression and decoding of the packaged information element.

[0014] According to one embodiment, the messages are messages of a sequence of messages and the method comprises processing the messages in sequence, wherein processing a message comprises the processing the packaged information elements in sequence, and, if the extracted information element has the predefined filtering value for a

packaged information element of a message, proceeding with processing the next message in the sequence.

[0015] According to one embodiment, processing the messages comprises pre-filtering the messages.

[0016] According to one embodiment, pre-filtering a message comprises filtering out the message if it fulfils a pre-filtering criterion.

[0017] According to one embodiment, a server computer system is provided comprising a radio interface, a memory interface and a processing unit configured to perform the method for depackaging data packaged according to different packaging stacks described above.

[0018] According to one embodiment, a computer program element is provided comprising program instructions, which, when executed by one or more processors, cause the one or more processors to perform the method for depackaging data packaged according to different packaging stacks described above.

[0019] According to one embodiment, a computer-readable medium is provided comprising program instructions, which, when executed by one or more processors, cause the one or more processors to perform the method for depackaging data packaged according to different packaging stacks described above. The computer-readable medium may be a non-transitory computer-readable medium.

BRIEF DESCRIPTION OF THE DRAWINGS

[0020] The invention will be better understood with reference to the detailed description when considered in conjunction with the non-limiting examples and the accompanying drawings, in which:

- FIG. 1 shows a data processing system.
- FIG. 2 shows a data server according to an embodiment.
- FIG. 3 shows a flow diagram illustrating a processing of data according to an embodiment.
- FIG. 4 illustrates the filtering of messages of a message stream.
- FIG. 5 shows a comparison of a traversal of a message according to different traversal schemes.
- FIG. 6 shows a flow diagram illustrating an operation of the data server of FIG. 2 for processing a message.

- FIG. 7 shows a flow diagram illustrating a method for depackaging data packaged according to different packaging stacks.
- FIG. 8 shows a server computer according to an embodiment.

DETAILED DESCRIPTION

[0021] The following detailed description refers to the accompanying drawings that show, by way of illustration, specific details and embodiments in which the disclosure may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the disclosure. Other embodiments may be utilized and structural, and logical changes may be made without departing from the scope of the disclosure. The various embodiments are not necessarily mutually exclusive, as some embodiments can be combined with one or more other embodiments to form new embodiments.

[0022] Embodiments described in the context of one of the devices or methods are analogously valid for the other devices or methods. Similarly, embodiments described in the context of a device are analogously valid for a method, and vice-versa.

[0023] Features that are described in the context of an embodiment may correspondingly be applicable to the same or similar features in the other embodiments. Features that are described in the context of an embodiment may correspondingly be applicable to the other embodiments, even if not explicitly described in these other embodiments. Furthermore, additions and/or combinations and/or alternatives as described for a feature in the context of an embodiment may correspondingly be applicable to the same or similar feature in the other embodiments.

[0024] In the context of various embodiments, the articles “a”, “an” and “the” as used with regard to a feature or element include a reference to one or more of the features or elements.

[0025] As used herein, the term “and/or” includes any and all combinations of one or more of the associated listed items.

[0026] In the following, embodiments will be described in detail.

[0027] FIG. 1 shows a data processing system 100.

[0028] The data processing system 100 includes a plurality of data sources 101. These may be servers hosting any kind of data base, computers coupled with sensors that perform measurements, data input terminals, etc. The data sources 101 are coupled via a

communication network 102 (or also an arrangement of communication networks), e.g. including or consisting of the Internet but possibly also including non-public internal communication networks, to a data sever 103. The data server 103 is provided to provide responses 105 to queries 104 sent by client 106.

[0029] It should be noted that the data server 103 and one or more clients 106 may be implemented by the same computer but may also be implemented by different computers (in particular, the server 103 may be implemented by a cluster of computers, i.e. may be implemented by a server computer system, possibly including multiple computers). In the latter case, the computers (implementing the clients 106 and the server 103) may for example be connected by the communication network 102 or one or more other communication networks. The entity implementing the data server 103 is also referred to as (data) server computer system.

[0030] The data processing system 100 may be seen to implement a data distribution environment (i.e. data is distributed via the data server 103 from the data sources 101 to the clients 106).

[0031] A query 104 of a client 106 may relate to data provided by any one of the data sources 101.

[0032] A query 104 may relate to results of calculations, e.g. performed by the data server 103 on the basis of data provided by one or more of the data sources 103. The data processing system 100 may therefore also be seen to implement a data calculation environment.

[0033] In a big data calculation environment, there are may be various data sources 101 using different systems e.g., real-time data from Kafka MySQL or HBase, and offline data from Hive Spark or Hadoop. Usually, a task pipeline (for example implemented by data server 103) needs to join different results across them to get final insights, e.g. to respond to queries 104.

[0034] Moreover, the data processing system 100 may be at least partially implemented according to a mixed cloud computing infrastructure. In such as case, data that is transported in the data processing system 100 is typically compressed and serialized to different formats by IaaS (Infrastructure as a Service) components e.g., control bits stream in C language, protobuf byte stream in go language, and JSON or XML string stream in the java tech stack.

[0035] Thus, data which is received by the data server 103 in form of messages (e.g. frames or records) is packaged (or wrapped) according to multiple packaging layers, wherein

the packaging stack according to which the data is packaged may be different, e.g. from data source to data source and for example depend on the way of transmission the data from the data sources 101 to the data server, e.g. depending on any intermediate component via which data is transferred from a data source 101 to the data server 103.

[0036] Without unpacking a message generated according such a multi-layer packaging, the original data (i.e. the data elements, also referred to as information elements, before packaging) which represents the real-world information (i.e. useful data such as sensor data etc.) cannot be reached. Thus, without unpacking a message, the data server 103 cannot decide how to deal with the message.

[0037] However, unpacking a message generated by packaging data according to a packaging stack (e.g. a message generated by a sequence of encodings applied to original data) to get to the original data involves a high amount of processing, including processing of information (such as header fields) which is not needed for the decision how to proceed with the message. This causes a high computational burden for the data server 103 in data processing procedures with a high amount of data (and a high number of messages).

[0038] In view of the above, according to various embodiments, in order to reach the original information (i.e. original data) as fast as possible, the data server 103 uses Depth-First Search to reach the original information first for each data element (e.g. each data field of the data structure (e.g. array) of the original data) and then makes an early decision on how to deal with the remainder of the message currently in the processing pipeline.

[0039] The data server 103 may save overhead for schema (i.e. packaging stack) registration and discovery, taking advantage of known formats and in-house encoding to reduce the processing and system complexity.

[0040] FIG. 2 shows a data server 200 according to an embodiment.

[0041] The data server 200 for example corresponds to the data server 103. As explained with reference to FIG.1, the data server 200 receives data in form of packaged messages from various data sources. The data may be data from a data base 201, (text) documents 202, data of data streams 203 etc. The various data are packaged by multiple layers 204 to 212 of packaging stacks which may be different for the data sources 101 and the data 201 to 203. A layer 204 to 212 may for example include an encoding of the layer's input data, putting the layer's input data into a certain format or data structure, encapsulating the layer's input data

according to a certain communication protocol, compressing the layer's input data according to a certain compression scheme etc.

[0042] The data server 200 comprises a message processor 213, a control interface 214 for receiving configuration parameters to control the processing performed by the message processor 213 and a schema storage 215 for storing schemas describing packaging stacks.

[0043] The data server 200 further comprises a job scheduler 216, an aggregate monitor 217, a log 218. These components may be used to ensure that the message processor 213 operates in an automatic and connective manner.

[0044] The message processor 213 comprises a DFS (depth first search) unpacker core 219, a value converter 220 and/or mapper 221, and a filter 222.

[0045] As will be described in more detail below, the DFS unpacker core 219 unpacks (in other words: unwraps) onion-like highly packaged message provided by the heterogeneous data sources and the type converter 220 or field mapper 221 convert the original data elements (i.e. the unpacked elements of the original data) message to a uniform form, and the filter filters the message depending on the original data elements, wherein the filters considers the original data elements one-by-one in the sequence as they are unpacked by the DFS unpacker core 219.

[0046] Optionally, the message processor 213 may further comprises a preconditional filter 223 (or transformation) and a post transformer (or filter) 224 which allows customizing or changing the processing pipeline and procedure implemented by the message processor 213.

[0047] The output of the message processor 213 is a sequence of reconstructed data elements 225 for the currently processed message which may be stored by the data server 200 in a memory 226 (e.g. buffer) and output, e.g. in response to a client request 104 or for further processing by a processing entity 227 (which may or may not be part of the data server 200). In particular, a DFS-based message processing method may be cascaded to include messages or data flow of other heterogeneous systems 228.

[0048] If, in the processing, one of the reconstructed data elements 225 meets a filtering criterion implemented by the filter 222, the message processor 213 stops depackaging of the currently processed message, i.e. stops reconstructing data elements 225 from the currently processed message. The filtering criterion may for example be that the reconstructed data element indicates an error.

[0049] A filter criterion may for example specify that a message should be filtered out

- if the message contains/does not contain a certain keyword;
- if the message contains/does not contain a word similar to a certain keyword (e.g. ignoring whether it is included in upper case or lower case) ;
- if the message contains/does not contain a string matching a predefined regular expression.

[0050] FIG. 3 shows a flow diagram 300 illustrating the processing of data according to an embodiment.

[0051] The left hand flow 301 refers to the packaging process which is performed by the data sources 101 and any intermediate components between the data sources 101 and the data server 103 in accordance with the packaging layers 204 to 212.

[0052] The right hand flow 302 refers to the unpackaging performed by the data server 103, 200, in particular the processing performed by the DFS unpacker core 219.

[0053] In the packaging, original data 303 is packaged according to a packaging stack, e.g. encoded 304, compressed 305 and wrapped 306 according to various packaging layers, i.e. formats, protocols and/or schemas.

[0054] Further, the resulting packaged data is serialized 307 and transmitted in the form of a message 308 via a communication network 309, e.g. corresponding to communication network 102, to the data server 103, which performs deserialization in 310 and unpackaging, e.g. unwrapping 311, decompressing 312 and decoding 313 in line with the packaging layers that were used in the packaging 301 to extract the original information.

[0055] The data server 200 performs the depackaging 302 in a DFS manner. This means that the original data 303, which includes a plurality of data elements 314, is reconstructed data element by data element and, as explained above, the data server 200 stops reconstruction of data elements for the current message 308 (and for example discards the message 308) if when it detects that it has reconstructed a data element which meets a filtering criterion.

[0056] FIG. 4 illustrates the filtering of messages of a message stream 403.

[0057] A first diagram 401 illustrates the processing when messages are completely unpacked before checking the filtering criterion and a second diagram 402 illustrates the processing when messages are unpacked in a DFS fashion including that data elements are sequentially unpacked and the filtering criterion is checked for each data element.

[0058] In FIG. 4 each block (i.e. rectangle) represents the unpacking of a data element.

[0059] According to the processing of the first diagram 401, each message is completely unpacked (here represented by the processing of eight blocks) and then, the message is possibly filtered out.

[0060] According to the processing of the second diagram 402, the processing of a message may stop much earlier. In the examples shown, the processing of one message is stopped after six blocks and the processing of two messages is stopped after three blocks because the data elements unpacked at that time have met the filtering criterion.

[0061] Compared to the processing according to the first diagram 401, the DFS-based processing according to the second diagram 402 allows achieving a complexity reduction from $O(k \times n)$ to $O(n)$ or to $O(\log n)$, for each message, where k = message layer or parse rounds of the packaging stack, and n = the number of data elements (atoms or fields) of the message.

[0062] For this, it should be noted that possibly, a message can be filtered out before all k layers are traversed for a certain information element, as it is illustrated in FIG. 5.

[0063] FIG. 5 shows a comparison of a traversal of a message according to different traversal schemes: complete (i.e. “brute force”), DFS and DFS including stopping traversal of the message if a filter criterion is met for the current information element. The information elements are denoted by $f1$ to $f5$ in the example of FIG. 5.

[0064] For a set of sequence of messages a reduction of $O(k \times n \times N)$ to $O(n \times N)$ or $O(\log n \times N')$, may be achieved where N = is the number of messages, N' = is the count of messages that are not filtered out.

[0065] The “log n ” occurs because of depth first search and pruning (i.e. filtering out) during recursive parsing in best cases.

[0066] FIG. 6 shows a flow diagram 600 illustrating an operation of the data server 200 of FIG. 2 for processing a message.

[0067] In 601, information about the packaging stack and thus the unpacking schema for unpacking the message is determined, e.g. from the storage 215.

[0068] In 602, configuration information like rules to transform data elements and one or more filtering criteria are input by the control interface 214.

[0069] In 603, the preconditional filter checks one or more pre-filtering criteria and, if one of them applies, filters out the message. The Pre-filtering criteria could for example be a

reception error (which can be detected without unpackaging the message such as a CRC error or the message not having a predefined size).

[0070] In 604, the DFS unpacker core 219 unpacks a data element from the message.

[0071] In 605, the type converter 220 converts the type of the data element and/or the filed mapper 221 performs a mapping the data element of the data element (e.g. in accordance with a reference format, value range, character encoding, etc.).

[0072] In 606, the filter 222 checks whether the data element (in each transformed or mapped form) fulfils a filtering criterion (or any of multiple filtering criteria).

[0073] If the data element does not fulfil the filtering criterion (or does not fulfil any one of multiple filtering criteria), the DFS unpacker core continues with the next data element of the message in 604 (i.e. the process returns to 604 for the next data element).

[0074] If, however, the data element fulfils the filtering criterion (or does not fulfil any one of multiple filtering criteria), the message is filtered out and the pipeline is halted for the message (e.g. a corresponding processing thread is dropped) in 607. The process then ends in 608 (for the current message). When all data elements have been extracted from a message, the process also ends in 608. The process may then continue with the next message (e.g. of a received sequence of messages).

[0075] Erroneous data and runtime errors may be logged in the log 218 and the status of the message as well as aggregate statistics may be reported by the monitor 217.

[0076] The data from the upstream end of the data link (i.e. the data sources 101) is wrapped into the innermost layers according to the packaging stack. Using DFS data elements which indicate whether the message should be filtered out allows filtering the message as quickly as possible (with suitable filtering criteria). It should be noted that since extracting of data elements from a message is stopped when a data element fulfilling a filtering criterion is encountered, the filtering can also be seen as pruning the message, i.e. pruning data elements from the message which should not be extracted (e.g. because an earlier data element indicates that this is not necessary). When the message is regarded as a tree with the leaves being the information elements (and the other nodes corresponding to different stages of packaging of information elements, up to the root corresponding to the packaged message), this can be seen as traversing the tree in a DFS manner and pruning the unprocessed parts of the tree when an information element which fulfils the filtering criterion (i.e. is equal to a filtering value) is extracted.

[0077] As explained with reference to FIG. 4, compared to approaches which unpack whole messages layer by layer, the DFS approach according to various embodiments only needs to process packaged data elements which are not pruned one time by traversing the fields (leaves) in the message one time with DFS according to the packaging stack.

[0078] In summary, according to various embodiments, a method is provided as illustrated in FIG. 7.

[0079] FIG. 7 shows a flow diagram 700 illustrating a method for depackaging data packaged according to different packaging stacks.

[0080] In 701, messages packaged according to different packaging stacks are received, wherein each message contains a sequence of packaged information elements.

[0081] In 702, for each message, the packaged information elements are processed in sequence. This comprises, for each packaged information element that is processed, extracting a corresponding information element by de-packaging the packaged information element from the message according to the packaging stack according to which the message is packaged in 703, checking whether the extracted information element has a predefined filtering value in 704 and stopping extracting information elements from the sequence of packaged information elements if the extracted information element has the predefined filtering value in 705.

[0082] According to various embodiments, in other words, rather than completely extracting a set (e.g. array) of information elements from a message, the information elements are processed in sequence and if one information element has a certain value, the extraction of information elements is stopped. This can be seen as a depth-first-search processing of the message, wherein the root is the packaged message and the leaves are the (unpackaged) information elements.

[0083] Each information element may for example be a field (of an array) or a value (e.g. double, float value), a data word. For example, each information element comprises one byte, two bytes, or four bytes.

[0084] The method of FIG. 7 allows message processing (including data flow processing, e.g. processing of a stream in a sequence of messages) that can improve the efficiency and throughput in a mixed cloud computing environment with heterogeneous data platform (data packaging) stacks. According to various embodiments, Depth-First Search is used to reach the original information first, which means unpacking packaged (e.g. encoded) data fields

sequentially until it is in the form of an atom data type like an understandable numeric or string, and then based on it to make an early decision of how to process the whole message in the data processing pipeline. Thus, the approach of FIG. 7 allows reducing redundant and useless processing, in particular in big data computing procedures.

[0085] The approach of FIG. 7 may in particular be applied as message processing approach in a system where there are many heterogeneous data processing platforms (e.g. data sources), e.g. in a cloud computing environment. It may be used to support big data computing and micro service-oriented architecture like syncing and transforming stream data or batch data from MySQL Hadoop Hive Spark Flink, and micro service's real-time messages queue disposing of like Kafka. It can be also adapted to other multi-platform (packaging) stack data flow processing systems. It can be applied in big data processing platforms or data insight services like PaaS (Platform as a Service) and DaaS (Desktop as a Service), etc.. With the expansion of the data scale and the increase of accumulated data, the approach of FIG. 7 also reducing the calculation effort by a large margin. Accordingly, a high data throughput rate can be maintained by reducing the effect of a data processing bottleneck. This for example allows obtaining results in real-time and/or cost reduction and efficiency enhancement.

[0086] The method of FIG. 7 is for example carried out by a server computer as illustrated in FIG. 8.

[0087] FIG. 8 shows a server computer system 800 according to an embodiment.

[0088] The server computer 800 includes a communication interface 801 (e.g. configured to receive the messages). The server computer system 800 further includes a processing unit 802 and a memory 803. The memory 803 may be used by the processing unit 802 to store, for example, the messages to be processed. The server computer system 800 is configured to perform the method of FIG. 7.

[0089] The methods described herein may be performed and the various processing or computation units and the devices and computing entities described herein may be implemented by one or more circuits. In an embodiment, a "circuit" may be understood as any kind of a logic implementing entity, which may be hardware, software, firmware, or any combination thereof. Thus, in an embodiment, a "circuit" may be a hard-wired logic circuit or a programmable logic circuit such as a programmable processor, e.g. a microprocessor. A "circuit" may also be software being implemented or executed by a processor, e.g. any kind

of computer program, e.g. a computer program using a virtual machine code. Any other kind of implementation of the respective functions which are described herein may also be understood as a "circuit" in accordance with an alternative embodiment.

[0090] While the disclosure has been particularly shown and described with reference to specific embodiments, it should be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention as defined by the appended claims. The scope of the invention is thus indicated by the appended claims and all changes which come within the meaning and range of equivalency of the claims are therefore intended to be embraced.

CLAIMS

1. A method for depackaging data packaged according to different packaging stacks, comprising:
receiving messages packaged according to different packaging stacks, wherein each message contains a sequence of packaged information elements;
for each message, processing the packaged information elements in sequence, comprising, for each packaged information element that is processed,
 - extracting a corresponding information element by de-packaging the packaged information element from the message according to the packaging stack according to which the message is packaged;
 - checking whether the extracted information element has a predefined filtering value; and
 - stopping extracting information elements from the sequence of packaged information elements if the extracted information element has the predefined filtering value.
2. The method of claim 1, further comprising filtering out the message when the information element has the predefined value.
3. The method of claim 2, wherein filtering out the message comprises discarding the message.
4. The method of any one of claims 1 to 3, comprising receiving the messages from different data sources, wherein each data source packages data into messages using a respective packaging stack, wherein the data sources use different packaging stacks.
5. The method of any one of claims 1 to 4, wherein packaging stacks being different comprises the packaging stacks differing in at least one of an encoding scheme, a data format, a compression scheme and a communication protocol according to which messages are packaged.

6. The method of any one of claims 1 to 5, wherein checking whether the extracted information element has the predefined filtering value comprises converting the information element into a predefined form and checking whether the extracted information element, in the predefined form, has the predefined filtering value.
7. The method of any one of claims 1 to 6, wherein the predefined filtering value is a value indicating an error.
8. The method of any one of claims 1 to 7, comprising checking whether the extracted information element has any value of a plurality of predefined filtering values and stopping extracting information elements from the sequence of packaged information elements if the extracted information element has any value of the plurality of predefined filtering values.
9. The method of any one of claims 1 to 8, comprising receiving a specification of the packaging stacks.
10. The method of any one of claims 1 to 9, wherein extracting the information element corresponding to the packaged information element comprises at least one of unwrapping, decompression and decoding of the packaged information element.
11. The method of any one of claims 1 to 10, wherein the messages are messages of a sequence of messages and the method comprises processing the messages in sequence, wherein processing a message comprises the processing the packaged information elements in sequence, and, if the extracted information element has the predefined filtering value for a packaged information element of a message, proceeding with processing the next message in the sequence.
12. The method of claim 11, wherein processing the messages comprises pre-filtering the messages.

13. The method of claim 12, wherein pre-filtering a message comprises filtering out the message if it fulfils a pre-filtering criterion.
14. A server computer system comprising a radio interface, a memory interface and a processing unit configured to perform the method of any one of claims 1 to 13.
15. A computer program element comprising program instructions, which, when executed by one or more processors, cause the one or more processors to perform the method of any one of claims 1 to 13.
16. A computer-readable medium comprising program instructions, which, when executed by one or more processors, cause the one or more processors to perform the method of any one of claims 1 to 13.