

Method and System for Adaptively Executing a Plurality of Tasks

FIELD OF INVENTION

[1] The present disclosure relates broadly, but not exclusively, to methods and systems for adaptively executing a plurality of tasks.

BACKGROUND

[2] One of the ways of implementing risk management for a platform offering various services and/or products for sale is to maintain data points related to, for example, users who are using the platform. Data points can be relied upon for preventing fraud by attackers who use different payment instruments like lost or stolen cards for illicit earnings.

[3] Various types of data points can be used in fraud detection and prevention. For example, aggregates can be computed based on raw data relating to transactions or other events occurring over a given time period, and these aggregates can be stored for use later. The number of transactions performed by a user in the given time period (e.g. the last 30 days) can be one such kind of aggregate. There is another category of data points that is typically not time-dependent, and which can be fetched from other services in a platform, for example a Know Your Customer (KYC) level of a user.

[4] These data points may be used in machine learning (ML) models to detect anomalies and decline potential fraudulent transactions. They may also be used in rules to set hard limits on the usage of various payment instruments that are available for a platform to reduce financial loss. Rules may be of the format 'decline a transaction if the user has done transactions with 50 different merchants in the last 1 week'. The data point in this rule may be the number of unique merchants that a user has transacted within the last 1 week. This is an example of an aggregate used in a rule.

[5] There are various limitations in the existing approaches for obtaining data points for fraud detection and prevention. Most of these approaches adopt, at best, simple

parallel execution of data point retrieval and processing.. Moreover, they do not optimise efficiency of execution and can have latency and network I/O bottlenecks.

[6] A need, therefore, exists to provide methods and systems that seek to overcome or at least minimize the above-mentioned challenges.

SUMMARY

[7] According to a first aspect of the present disclosure, there is provided a method for adaptively executing a plurality of tasks, comprising: defining, by one or more processors, a schema representing a plurality of tasks, each task comprising a data retrieval and/or data evaluation operation, the schema comprising task information indicating how to execute each of the plurality of tasks; generating, by the one or more processors, a graph representation of the plurality of tasks based on the schema, the graph representation comprising a plurality of nodes, each node of the plurality of nodes corresponding to one of the plurality of tasks; executing, by the one or more processors, the plurality of tasks based on the graph representation and the task information; generating a matrix based on the graph representation, the matrix indicating, for each node of the plurality of nodes, whether there is a dependency on another node of the plurality of nodes; generating a frequency map based on the matrix, the frequency map indicating a total number of dependencies for each node of the plurality of nodes; and determining an execution order for the plurality of tasks based on the frequency map.

[8] According to a second aspect of the present disclosure, there is provided a system for adaptively executing a plurality of tasks, comprising: at least one processor; and at least one memory including computer program code; the at least one memory and the computer program code configured to, with the at least one processor, cause the system at least to: define, by one or more processors, a schema representing a plurality of tasks, each task comprising a data retrieval and/or data evaluation operation, the schema comprising task information indicating how to execute each of the plurality of tasks; generate, by the one or more processors, a graph representation of the plurality of tasks based on the schema, the graph representation comprising a plurality of nodes, each node of the plurality of nodes corresponding to one of the plurality of tasks; execute, by the one or more processors, the plurality of tasks based on the graph representation and the task information; generate a matrix based on the graph representation, the matrix indicating, for each node of the plurality of nodes, whether there is a dependency on another node of the plurality of nodes; generate a frequency map based on the matrix, the frequency map indicating a total number of dependencies for each node of the plurality of nodes; and determine an execution order for the plurality of tasks based on the frequency map.

BRIEF DESCRIPTION OF THE DRAWINGS

[9] Embodiments and implementations are provided by way of example only, and will be better understood to one of ordinary skill in the art from the following written description, read in conjunction with the drawings, in which:

[10] Fig. 1 illustrates a system for adaptively executing a plurality of tasks according to various embodiments of the present disclosure.

[11] Fig. 2 is a schematic diagram of a data processing server, according to various embodiments of the present disclosure.

[12] Fig. 3 is an overview of a process for executing a plurality of tasks according to an example.

[13] Fig. 4A depicts an overview of a process for adaptively executing a plurality of tasks according to various embodiments.

[14] Figs. 4B and 4C depict an example illustration of a schema for generating a graph according to various embodiments.

[15] Fig. 5 depicts a graph illustrating a dependency relationship according to the schema of Figs. 4B and 4C.

[16] Figs. 6A – 6G depict example illustrations of various graphs for adaptively executing a plurality of tasks according to various embodiments.

[17] Fig. 7 illustrates an example flow diagram of a method for adaptively executing a plurality of tasks according to various embodiments.

[18] Fig. 8A is a schematic block diagram of a general purpose computer system upon which the data processing server of Fig. 2 can be practiced.

[19] Fig. 8B is a schematic block diagram of a general purpose computer system upon which a combined transaction processing and data processing server of Fig. 1 can be practiced.

[20] Fig. 9 shows an example of a computing device to realize the transaction processing server shown in Fig. 1.

[21] Fig. 10 shows an example of a computing device to realize the data processing server shown in Fig. 1.

[22] Fig. 11 shows an example of a computing device to realize a combined transaction processing and data processing server shown in Fig. 1.

[23] Skilled artisans will appreciate that elements in the figures are illustrated for simplicity and clarity and have not necessarily been depicted to scale. For example, the dimensions of some of the elements in the illustrations, block diagrams or flowcharts may be exaggerated in respect to other elements to help to improve understanding of the present embodiments.

DETAILED DESCRIPTION

[24] A platform refers to a system of networked computer devices that facilitates exchanges between two or more interdependent groups, for example between a user (of a product or service) and a provider (of the product or service), each of which has a respective account registered with the platform. For example, a platform may offer a service offered by a provider such as a ride, delivery, online shopping, insurance, and other similar services to a requestor. The user can typically access the platform via a website, an application, or other similar methods.

[25] A schema refers to a framework or plan for structuring data, and defines how data may be organized within a database. In the present disclosure, a schema may be used for generating a graph for adaptively executing a plurality of tasks. The graph may comprise a plurality of nodes. Each node may be connected to one or more other nodes by an edge or link. A node may be connected from an upstream position to another node

in downstream position. In this case, the node at the downstream position may be termed a child node, while the node at the upstream position may be termed a parent node.

[26] Each node may be representative of a task, such as for example retrieving data from a data source (e.g. a data point (DP) node), processing the retrieved data utilizing machine learning (ML) models (e.g. a model node), evaluation of rules that may be set based on the processed data (e.g. a rule evaluation node), and other similar tasks. The data that is retrieved by executing a data point node may relate to users of a platform. A ML model may process the retrieved data (e.g. by execution of a model node). Computing resources such as processing threads that are used by a ML model for the processing may be termed as a worker pool. Assuming that there are a plurality of ML model nodes in a system, each of the plurality of ML model nodes may utilize its own worker pool or a shared worker pool for processing the retrieved data. Further, a rule evaluation node may utilize the processed data to, for example, evaluate pre-defined static rules. For example, a rule may be a hard limit on the usage of various payment instruments by a user of a platform to reduce financial loss. A rule may be of the format 'decline a transaction if the user has done transactions with 50 different merchants in the last 1 week'. The data point for evaluating this rule may be the number of unique merchants that a user has transacted within the last 1 week.

[27] An example of a graph is shown in Fig. 3. In this graph 300, each of DP nodes DP2 302, DP3 304, DP4 306, DP5 308, DP6 310, DP7 312, DP8 314, DP9 316, DP10 318 and DP11 320 represent a data point, each of model nodes Model-1 322, Model-2 324 and Model-3 326 represents a machine learning model that takes one or more of the data points DP1-DP11 as input and generates one or more outputs based on that input, and rule evaluation node 328 represents a set of rules that can be applied to one or more of the data points DP1-DP11 and/or one or more of the outputs of the machine learning models Model-1, Model-2, Model-3.

[28] In graph 300, there are 2 DP2 nodes 302 and 2 DP3 nodes 304 because both DP2 302 and DP3 304 are required for model evaluations by both Model-1 322 and Model-2 324. In the implementation of the graph 300, there is no way to figure out if, for example, DP2 302 and DP3 304 are being repeated across 2 different model evaluations e.g. Model-1 322 and Model-2 324, which would mean that worker pool resources are being wasted on duplicate work for evaluating a data point that has previously been evaluated. A

possible solution is for all results from the nodes to be cached, such that the cached results can be used to check if a result of the same data point already exists or not. However, this may not work because each model evaluation and rule evaluation may be using its worker pool, to evaluate data points individually. For example, if there are four worker pools in the above scenario, there is no guarantee that the worker pool of Model-1 322 will finish fetching DP2 302 and DP3 304 before the worker pool of Model-2 324 or vice versa. Similarly, as DP11 320, DP10 318, Model-1 322 and Model-2 324 are needed for rule evaluation node 328 and Model-3 326, it is possible to end up having similar problems here. If this problem is extended to thousands of data points, it becomes very evident that this method of fetching data points is not scalable.

[29] As an alternative to the previous approach, all the data points may be combined and evaluated using a single worker pool. However, this still incurs a lot of latency because the worker count is generally not sufficient to give considerable performance gains expected from parallel execution. Even if the number of processing threads in the worker pool is increased, there is still no guarantee of sequential execution between dependent nodes.

[30] An objective to be discussed in the present disclosure is to provide an approach for deciding the execution order of nodes in a graph. For example, the graph 300 of Figure 3 may be restructured to graph 400 of Fig. 4A. In graph 400, all DP2 nodes 302 and DP3 nodes 304 are consolidated into only 1 DP2 node 402 and DP3 node 404 respectively. As is very evident from graph 400, the number of vertices (which is directly equivalent to the number of data point evaluations) has gone down considerably in the new design. More importantly, there is a dependency between different data points which did not exist in graph 300. This advantageously ensures none of the data points which are dependent on the current data point gets evaluated before the current data point's evaluation completes.

[31] The proposed architecture advantageously aims to eliminate all the duplicate data point retrieval calls. This design also eliminates the need to have n-dependent application programming interface (API) calls as code, where each API call uses some data from the output of previous calls. These calls can be modelled in, for example, the graph 400 itself, thus ensuring maximum parallelism when executing the nodes. There is an overall reduction in network I/O operations that these systems need to do. It also reduces the burden on the underlying language's thread scheduler. For example, the scheduler does

not have to frequently park threads and bring them back to execution which incurs additional I/O. This design also advantageously makes a platform application more scalable such that a large number of machine learning models and rule evaluations can run parallelly. Further, the proposed architecture is flexible in that if a latency threshold is exceeded (i.e., the overall process of fetching data points takes more than x ms), it is possible to ignore the rest of the data points that were supposed to be evaluated according to the graph 400 and continue with rule evaluation and model evaluation.

[32] Apart from all the other technical benefits that can be gained out of it, it also ensures minimal execution time of a user's request, which is a key metric in deciding a user's stickiness to a platform. In fraud evaluation systems, there is always a chance of requests timing out and denying a lot of user's transactions if the SLA set for the said system is breached. Too many breaches always result in a user abandoning the task, which directly results in loss of business for the platform.

[33] The data points may be represented by a schema (for example, one that is parsed by a parser of a platform) to arrive at the graph structure of graph 400. An example schema 430 is shown in Figs. 4B and 4C, wherein the data points are modelled as a tree structure. While the schema 430 is in a JavaScript Object Notation (JSON) format, it will be appreciated that other similar type of formats may also be utilized. In the schema 430, each label is parsed and appended to the previously parsed labels. These appended labels are stored as-is, so that they may be utilized at a later step to uniquely identify nodes. Some examples of labels are sender 432, success 434, curr_day 440, amount 446, and other similar labels. When arriving at a filter node e.g. under filters 436, a check is performed to see if the values present in filters have a Node: prefix. In this example, this signifies that a node needs to be created and evaluated to find the value as indicated by the filters. Generally, the actual data source from where a node's value has to be retrieved may be found when we arrive at a corresponding node under the output label e.g. output node. In this example, when arriving at the step of processing a sender.id node which is created based on Node:sender.id label 438, the value required by the node needs to be derived from an input data source because data source 466 for the node is indicated as "Input:UserID" 468. If the filter does not have the Node: prefix then the value is taken as is.

[34] Similarly, when arriving at one of the output nodes, a node can be created from it. The name of the node is derived by appending the labels parsed until that node. There can be three types of output nodes, the first type of which has to make an API call and retrieve the value from a given path in the response. For these kind of output nodes there is always a data source field which stores the mapping. An identifier in the mapping before a colon denotes that it is an API call. An identifier in the mapping before an underscore denotes a configuration that needs to be utilized to make an API call. An identifier in the mapping after an underscore denotes a path in a response from which an actual value is fetched. In the schema 430, API:PaxKYCDetails_sender.kycLevel 464 under label kyc 462 signifies that an API call needs to be made to retrieve the user's Know Your Customer (KYC) details. A response structure may then be parsed according to the path sender.kycLevel and the value is obtained for that node.

[35] The second type of output node gathers all the filters from the JSON structure until an output node is encountered and forms a database request which is supposed to retrieve the relevant data point by filtering over a data set with given filters. While parsing the JSON structure for the second type of node, the parsing process keeps storing all the filters that are encountered before the output node. All filters that are encountered on a level before an output node or on a same level as the output node are passed to the output node for filtering. For example in the schema 430, the filter "from_user_id": "Node:sender.id" 438 on the first level becomes a filter for all the nested outputs in the schema. After that "status": "success" 433 and "start_time": "now/d" 442 gets added as filters to already present "from_user_id": "Node:sender.id" 438, they become filters for an output node which is labelled as amount 446. For this kind of node which is also considered a database query type, all the filters get appended in the where clause with the relevant values. Similarly, for output kyc 462 which is on a same level as "filters" 436, only one filter is needed (e.g. "from_user_id": "Node:sender.id" 438 to find the user id of the user) to make the API call and fetch the KYC level of the user. In a further example in the schema 430, aggregation_column 448 indicates amount 450 which is a number (e.g. meta tag 452 indicating type 454 as number 456), and operator 458 is indicated as a summation (e.g. sum 460). The aggregation_column 448 and operator 458 are parsed to create the following database query, select sum(amount) from transactions where from_user_id = ? and status = 'success' and start_time >= '2022-03-09'. A separate parser may be utilized to translate now/d 442 to 2022-03-09 assuming today's date is 2022-03-09.

[36] The third type of output node has the prefix 'Input:' and signifies the input data that we already have. For example, "id" data source 466 indicates Input:UserID 468 to signify that data for "id" can be retrieved from input data map by parsing the path UserID. Further, meta tags (e.g. meta tag 448) define the data type of the output nodes and validates it. The third type of output node is the same as the first type described above, in terms of having an identifier before a colon which defines what kind of node it is. It does not have an identifier to define configuration which is not needed for this kind of node. In this case, a response path is indicated after the colon which can be used to fetch an actual value.

[37] The graph edges are directed from all the filters (parsed before an output node) to the corresponding output nodes in the schema. Please note that output and filters are reserved keywords for this schema and should not be confused with the other labels.

[38] In the schema 430, it can be seen that sender.id is a dependency for fetching values of sender.success.curr_day.amount and sender.kyc . Once the structure is parsed according to the given logic, it will be converted into a graph structure as shown in graph 500 of Fig. 5 (e.g. sender.id 502 is a dependency for fetching values of sender.success.curr_day.amount 504 and sender.kyc 506).

[39] A user may be any suitable type of entity, which may include a person, a consumer looking to purchase a product or service via a transaction processing server, a seller or merchant looking to sell a product or service via the transaction processing server, a motorcycle driver or pillion rider in a case of the user looking to book or provide a motorcycle ride via the transaction processing server, a car driver or passenger in a case of the user looking to book or provide a car ride via the transaction processing server, and other similar entity. A user who is registered to the transaction processing or data processing server will be called a registered user. A user who is not registered to the transaction processing server or data processing server will be called a non-registered user. The term user will be used to collectively refer to both registered and non-registered users. A user may interchangeably be referred to as a requestor (e.g. a person who requests for a product or service) or a provider (e.g. a person who provides the requested product or service to the requestor).

[40] A data processing server is a server that hosts software application programs for performing data processing in relation to adaptively executing a plurality of tasks. The data

processing server may be implemented as shown in schematic diagram 200 of Fig. 2 for adaptively executing a plurality of tasks.

[41] A transaction processing server is a server that hosts software application programs for processing payment transactions for, for example, purchasing of a good or service by a user. The transaction processing server communicates with any other servers (e.g., a data processing server) concerning processing payment transactions relating to the purchasing of the good or service. For example, data relating to an approved or rejected transaction (e.g. date, time, amount, currency, user name, and other similar data relating to the concerned transaction) may be provided to the data processing server as raw data that may be utilized for processing by data points. The processed data may then be stored or transferred to a database. In an implementation, the transaction processing server may also be in communication with a database directly which will store the data relating to an approved or rejected transaction as raw data, or may also be configured to process the data before doing so. The transaction processing server may use a variety of different protocols and procedures in order to process the payment and/or travel co-ordination requests.

[42] Transactions that may be performed via a transaction processing server include product or service purchases, credit purchases, debit transactions, fund transfers, account withdrawals, etc. Transaction processing servers may be configured to process transactions via cash-substitutes, which may include payment cards, letters of credit, checks, payment accounts, etc.

[43] The transaction processing server is usually managed by a service provider that may be an entity (e.g. a company or organization) which operates to process transaction requests and/or travel co-ordination requests e.g. pair a provider of a travel co-ordination request to a requestor of the travel co-ordination request. The transaction processing server may include one or more computing devices that are used for processing transaction requests and/or travel co-ordination requests.

[44] A transaction account is an account of a user who is registered at a transaction processing server. The user can be a customer, a merchant providing a product for sale on a platform and/or for onboarding the platform, a hail provider (e.g., a driver), or any third parties (e.g., a courier) who want to use the transaction processing server. In certain

circumstances, the transaction account is not required to use the transaction processing server. A transaction account includes details (e.g., name, address, vehicle, face image, etc.) of a user. The transaction processing server manages the transaction.

[45] The above paragraphs introduce certain terminology that may be helpful in understanding the invention and its various embodiments. These should not be considered to be definitions that are limiting on the scope of the claims.

[46] Embodiments will be described, by way of example only, with reference to the drawings. Like reference numerals and characters in the drawings refer to like elements or equivalents.

[47] Some portions of the description which follows are explicitly or implicitly presented in terms of algorithms and functional or symbolic representations of operations on data within a computer memory. These algorithmic descriptions and functional or symbolic representations are the means used by those skilled in the data processing arts to convey most effectively the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities, such as electrical, magnetic or optical signals capable of being stored, transferred, combined, compared, and otherwise manipulated.

[48] Unless specifically stated otherwise, and as apparent from the following, it will be appreciated that throughout the present specification, discussions utilizing terms such as “defining”, “comparing”, “determining”, “calculating”, “retrieving”, “processing”, “storing”, “aggregating”, “identifying”, “executing”, “reducing”, or the like, refer to the action and processes of a computer system, or similar electronic device, that manipulates and transforms data represented as physical quantities within the computer system into other data similarly represented as physical quantities within the computer system or other information storage, transmission or display devices.

[49] In addition, the present specification also implicitly discloses a computer program, in that it would be apparent to the person skilled in the art that the individual steps of the method described herein may be put into effect by computer code. The computer program is not intended to be limited to any particular programming language and implementation

thereof. It will be appreciated that a variety of programming languages and coding thereof may be used to implement the teachings of the disclosure contained herein. Moreover, the computer program is not intended to be limited to any particular control flow. There are many other variants of the computer program, which can use different control flows without departing from the scope of the specification.

[50] Furthermore, one or more of the steps of the computer program may be performed in parallel rather than sequentially. Such a computer program may be stored on any computer readable medium. The computer readable medium may include storage devices such as magnetic or optical disks, memory chips, or other storage devices suitable for interfacing with a computer. The computer readable medium may also include a hard-wired medium such as exemplified in the Internet system, or wireless medium such as exemplified in the GSM mobile telephone system. The computer program when loaded and executed on such a computer effectively results in an apparatus that implements the steps of the preferred method.

[51] Fig. 1 illustrates a block diagram of a system 100 for adaptively executing a plurality of tasks. Further, the system 100 enables a payment transaction for a good or service, and/or a request for a ride between a requestor and a provider.

[52] The system 100 comprises a requestor device 102, a provider device 104, an acquirer server 106, a transaction processing server 108, an issuer server 110, a data processing server 140 and a database 150.

[53] The requestor device 102 is in communication with a provider device 104 via a connection 112. The connection 112 may be wireless (e.g., via NFC communication, Bluetooth, etc.) or over a network (e.g., the Internet). The requestor device 102 is also in communication with the data processing server 140 via a connection 121. The connection 121 may be via a network (e.g., the Internet). The requestor device 102 may also be connected to a cloud that facilitates the system 100 for adaptively executing a plurality of tasks. For example, the requestor device 102 can send a signal or data to the cloud directly via a wireless connection (e.g., via NFC communication, Bluetooth, etc.) or over a network (e.g., the Internet).

[54] The provider device 104 is in communication with the requestor device 102 as described above, usually via the transaction processing server 108. The provider device 104 is, in turn, in communication with an acquirer server 106 via a connection 114. The provider device 104 is also in communication with the data processing server 140 via a connection 123. The connections 114 and 123 may be via a network (e.g., the Internet). The provider device 104 may also be connected to a cloud that facilitates the system 100 for adaptively executing a plurality of tasks. For example, the provider device 104 can send a signal or data to the cloud directly via a wireless connection (e.g., via NFC communication, Bluetooth, etc.) or over a network (e.g., the Internet).

[55] The acquirer server 106, in turn, is in communication with the transaction processing server 108 via a connection 116. The transaction processing server 108, in turn, is in communication with an issuer server 110 via a connection 118. The connections 116 and 118 may be via a network (e.g., the Internet).

[56] The transaction processing server 108 is further in communication with the data processing server 140 via a connection 120. The connection 120 may be over a network (e.g., a local area network, a wide area network, the Internet, etc.). In one arrangement, the transaction processing server 108 and the data processing server 140 are combined and the connection 120 may be an interconnected bus.

[57] The data processing server 140, in turn, is in communication with the reference databases 150A and 150B via respective connection 122. The connection 122 may be a network (e.g., the Internet). The data processing server 140 may also be connected to a cloud that facilitates the system 100 for adaptively executing a plurality of tasks. For example, the data processing server 140 can send a signal or data to the cloud directly via a wireless connection (e.g., via NFC communication, Bluetooth, etc.) or over a network (e.g., the Internet).

[58] The database 150 may comprise data relating users, transactions, products, services, and other similar data, for example relating to a platform. The data may be raw or aggregated data. The database 150 may be combined with the data processing server 140. In an example, the database 150 may be managed by an external entity and the data processing server 140 is a server that, based on a schema comprising information indicating how to execute each of a plurality of tasks, executes the plurality of tasks based

on the information. The information may further indicate data to be retrieved and a data source for a task of the plurality of tasks, and executing the task by the data processing server 140 may further comprise retrieving the indicated data from the data source. The indicated data may be raw data or aggregated data. The data source may be the database 150, the data processing server 140, the transaction processing server 108, or other similar data source. In an implementation, the database 150 may store the schema for which the data processing server 140 utilizes for executing the plurality of tasks. In an implementation, there may be more than one database, in which the data processing server 140 may be configured to determine which database to use based on the information. In this case, these databases are collectively referred to herein as databases 150.

[59] Alternatively, one or more modules may store the raw data or aggregated data instead of the database 150, wherein the module may be integrated as part of the data processing server 140 or external from the data processing server 140.

[60] In the illustrative embodiment, each of the devices 102, 104, and the servers 106, 108, 110, 140, and/or database 150 provides an interface to enable communication with other connected devices 102, 104 and/or servers 106, 108, 110, 140, and/or database 150. Such communication is facilitated by an application programming interface (“API”). Such APIs may be part of a user interface that may include graphical user interfaces (GUIs), Web-based interfaces, programmatic interfaces such as application programming interfaces (APIs) and/or sets of remote procedure calls (RPCs) corresponding to interface elements, messaging interfaces in which the interface elements correspond to messages of a communication protocol, and/or suitable combinations thereof. For example, it is possible for at least one of the requestor device 102 and the provider device 104 to send data (e.g. relating to a user account, product, transaction, or other similar matter) in response to an enquiry shown on the GUI running on the respective API.

[61] Use of the term ‘server’ herein can mean a single computing device or a plurality of interconnected computing devices which operate together to perform a particular function. That is, the server may be contained within a single hardware unit or be distributed among several or many different hardware units.

[62] The data processing server 140 is associated with an entity (e.g. a company or organization or moderator of the service). In one arrangement, the data processing server 140 is owned and operated by the entity operating the transaction processing server 108. In such an arrangement, the data processing server 140 may be implemented as a part (e.g., a computer program module, a computing device, etc.) of the transaction processing server 108.

[63] The transaction processing server 108 may also be configured to manage the registration of users. A registered user has a transaction account (see the discussion above) which includes details of the user. The registration step is called on-boarding. A user may use either the requestor device 102 or the provider device 104 to perform on-boarding to the transaction processing server 108.

[64] It may not be necessary to have a transaction account at the transaction processing server 108 to access the functionalities of the transaction processing server 108. However, there are functions that are available to a registered user. These additional functions will be discussed below.

[65] The on-boarding process for a user is performed by the user through one of the requestor device 102 or the provider device 104. In one arrangement, the user downloads an app (which includes the API to interact with the transaction processing server 108) to the requestor device 102 or the provider device 104. In another arrangement, the user accesses a website (which includes the API to interact with the transaction processing server 108) on the requestor device 102 or the provider device 104. The user is then able to interact with the data processing server 140. The user may be a requestor or a provider associated with the requestor device 102 or the provider device 104, respectively.

[66] Details of the registration may include, for example, name of the user, address of the user, emergency contact, blood type or other healthcare information, next-of-kin contact, permissions to retrieve data and information from the requestor device 102 and/or the provider device 104 for product identification purposes, such as permission to use a camera of the requestor device 102 and/or the provider device 104 to take a picture of the user for identification purposes. Alternatively, another mobile device may be selected instead of the requestor device 102 and/or the provider device 104 for retrieving the data. Once on-boarded, the user would have a transaction account that stores all the details.

[67] The requestor device 102 is associated with a customer (or requestor) who is a party to a transaction that occurs between the requestor device 102 and the provider device 104, or between the requestor device 102 and the data processing server 140. The requestor device 102 may be a computing device such as a desktop computer, an interactive voice response (IVR) system, a smartphone, a laptop computer, a personal digital assistant computer (PDA), a mobile computer, a tablet computer, and the like.

[68] The requestor device 102 includes transaction credentials (e.g., a payment account) of a requestor to enable the requestor device 102 to be a party to a payment transaction. If the requestor has a transaction account, the transaction account may also be included (i.e., stored) in the requestor device 102. For example, a mobile device (which is a requestor device 102) may have the transaction account of the customer stored in the mobile device.

[69] In one example arrangement, the requestor device 102 is a computing device in a watch or similar wearable and is fitted with a wireless communications interface (e.g., a NFC interface). The requestor device 102 can then electronically communicate with the provider device 104 regarding a transaction request. The customer uses the watch or similar wearable to initiate the transaction request by pressing a button on the watch or wearable.

[70] The provider device 104 is associated with a provider who is also a party to the transaction request that occurs between the requestor device 102 and the provider device 104. The provider device 104 may be a computing device such as a desktop computer, an interactive voice response (IVR) system, a smartphone, a laptop computer, a personal digital assistant computer (PDA), a mobile computer, a tablet computer, and the like.

[71] Hereinafter, the term “provider” refers to a service provider and any third party associated with providing a product or service for purchase, or a travel or ride or delivery service via the provider device 104. Therefore, the transaction account of a provider refers to both the transaction account of a provider and the transaction account of a third party (e.g., a travel co-ordinator or merchant) associated with the provider.

[72] If the provider has a transaction account, the transaction account may also be included (i.e., stored) in the provider device 104. For example, a mobile device (which is a provider device 104) may have the transaction account of the provider stored in the mobile device.

[73] In one example arrangement, the provider device 104 is a computing device in a watch or similar wearable and is fitted with a wireless communications interface (e.g., a NFC interface). The provider device 104 can then electronically communicate with the requestor to initiate the transaction request by pressing a button on the watch or wearable.

[74] The acquirer server 106 is associated with an acquirer who may be an entity (e.g. a company or organization) which issues (e.g. establishes, manages, administers) a payment account (e.g. a financial bank account) of a merchant. Examples of the acquirer include a bank and/or other financial institution. As discussed above, the acquirer server 106 may include one or more computing devices that are used to establish communication with another server (e.g., the transaction processing server 108) by exchanging messages with and/or passing information to the other server. The acquirer server 106 forwards the payment transaction relating to a transaction request to the transaction processing server 108.

[75] The transaction processing server 108 is configured to process processes relating to a transaction account by, for example, forwarding data and information associated with the transaction to the other servers in the system 100 such as the data processing server 140. In an example, the transaction processing server 108 may transmit data relating to an approved or rejected transaction (e.g. date, time, amount, currency, user name, and other similar data relating to the concerned transaction) to the data processing server 140. In a case where processing data in relation to a request for data is a service provided by the platform, the transaction processing server 108 may communicate with the data processing server 140 to facilitate payment for the data processing service after data relating to a request for data is retrieved and provided to the requestor. The transaction processing server 108 may use a variety of different protocols and procedures in order to process the payment and/or travel co-ordination requests.

[76] The issuer server 110 is associated with an issuer and may include one or more computing devices that are used to perform a payment transaction. The issuer may be an

entity (e.g. a company or organization) which issues (e.g. establishes, manages, administers) a transaction credential or a payment account (e.g. a financial bank account) associated with the owner of the requestor device 102. As discussed above, the issuer server 110 may include one or more computing devices that are used to establish communication with another server (e.g., the transaction processing server 108) by exchanging messages with and/or passing information to the other server.

[77] The database 150 is a database or server associated with an entity (e.g. a company or organization) which manages (e.g. establishes, administers) data relating to users, transactions, products, services, and other similar data, for example relating to the entity. In an arrangement, the database 150 may store raw or aggregated data relating to users of a platform, such as relating to user details, historical transactions, statistics relating to a user's transaction and activities, and other similar data that may be retrieved by a DP node, and processed by the ML model node, which may then be used to set up or evaluate rules by a rule evaluation node. In an arrangement, the database 150 may store a schema based on which the data processing server 140 may utilize for adaptively executing a plurality of tasks.

[78] Advantageously, the system 100 aims to eliminate all duplicate data point retrieval calls and enable maximum parallelism when executing a plurality of tasks, making a platform application more scalable such that a large number of machine learning models and rule evaluations can run parallelly. In one example, an implementation of the system 100 executed topup transactions with topup latency reduced by 30 ms, approximately 1/3 fewer queries for Aerospike-based aggregates, and about 8 fewer queries for timescale-based aggregates per topup request. It will be appreciated that requests which involve a greater number of duplicated data points may have even greater improvements in latency and efficiency.

[79] Fig. 2 illustrates a schematic diagram of the data processing server 140 according to various embodiments. The data processing server 140 may comprise a data module 260 configured to receive data and information from the requestor device 102, provider device 104, transaction processing server 108, database 150, a cloud and other sources of information to facilitate adaptively executing a plurality of tasks. The data module 260 may be further configured to send information relating to a completed task to the requestor

device 102, the provider device 104, the transaction processing server 108, or other destinations where the information is required.

[80] The data processing server 140 may comprise a sequence module 262 that is configured to define, by one or more processors, a schema representing a plurality of tasks, each task comprising a data retrieval and/or data evaluation operation, the schema comprising task information indicating how to execute each of the plurality of tasks; and generating, by the one or more processors, a graph representation of the plurality of tasks based on the schema, the graph representation comprising a plurality of nodes, each node of the plurality of nodes corresponding to one of the plurality of tasks. The sequence module 262 may be further configured to determine an execution order for the plurality of tasks by: generating a matrix based on the graph representation, the matrix indicating, for each node of the plurality of nodes, whether there is a dependency on another node of the plurality of nodes; generating a frequency map based on the matrix, the frequency map indicating a total number of dependencies for each node of the plurality of nodes; and determining an execution order for the plurality of tasks based on the frequency map. Determining the execution order may further comprise identifying a node with zero dependencies from the frequency map, and adding the identified node to an execution queue. The sequence module 262 may be further configured to reduce the total number of dependencies for each node that is dependent on the identified node by one in the frequency map after the identified node is executed, and remove the identified node from the execution queue.

[81] The sequence module 262 may be further configured to define a plurality of tasks based on a schema, the schema comprising information indicating how to execute each of the plurality of tasks.

[82] In an implementation, the information further may indicate a node to be generated for each of the plurality of tasks to form a plurality of nodes in a graph, such as shown in the graph 400 of Fig. 4A. Defining the plurality of tasks may further comprise generating a node for each of the plurality of tasks based on the information, each of the plurality of nodes representing a corresponding task to be executed. Executing the plurality of tasks may be further based on a sequence of the plurality of nodes in the graph. The execution of the plurality of tasks based on the sequence is further explained in Figs. 6A – 6G.

[83] In an implementation, defining the plurality of tasks may further comprise determining one or more of the plurality of tasks that can only be executed after a first task has been executed, a counter for each of the one or more tasks, and identifying a second task from the one or more tasks to be executed based on a number indicated in each counter.

[84] In an implementation, identifying the second task may further comprise reducing the number indicated in each counter of the one or more tasks by one after the first task is executed; and identifying the second task when the counter for the identified second task is zero.

[85] In an implementation, wherein the information further indicates an identifier for each of the plurality of tasks, determining the one or more tasks further comprises identifying a match with the first task from a database (e.g. the database 150), the database comprising a plurality of tasks each indicating an identifier, and further indicating, for each task of the plurality of tasks, one or more tasks that can only be executed after each respective task of the plurality of tasks has been executed, the identified match indicating a same identifier as the first task; and determining the one or more tasks that corresponds to the identified match.

[86] In an implementation, wherein the information further indicates an identifier for each of the plurality of tasks, determining a counter further comprises identifying a match with each of the one or more tasks from a database, the database comprising a plurality of tasks each indicating an identifier, and further indicating a counter for each of the plurality of tasks, each identified match indicating a same identifier as a corresponding task of the one or more tasks; and determining the counter that corresponds to each identified match of the one or more tasks.

[87] In an implementation, the sequence module 262 may be further configured to identify a match with the first task from the database, the identified match indicating a same identifier as the first task; and removing the identified match from the database after the first task is executed.

[88] In an implementation, the sequence module 262 may be further configured to reduce a number indicated in each counter of the one or more identified matches in the

database by one after the first task is executed; and identify the second task from the one or more identified matches when the counter for the identified second task is zero.

[89] In an implementation, the sequence module 262 may be further configured to identify a plurality of tasks whose counter is zero, and execute the plurality of tasks in parallel with one another.

[90] The data processing server 140 may also comprise a data point module 264 that is configured for executing, by the one or more processors, the plurality of tasks based on the graph representation and the task information. Two or more tasks of the plurality of tasks may be executed in parallel by one or more processors. The task information may further indicate data to be retrieved and a data source for a data retrieval operation of the plurality of tasks, wherein executing the task further comprises retrieving the indicated data from the data source. The data point module 264 may be further configured to execute a task corresponding to the identified node. The data point module 264 may be further configured to execute a task for a DP node. In an implementation, wherein the information further indicates data to be retrieved and a data source for a task of the plurality of tasks, executing the task further comprises retrieving the indicated data from the data source.

[91] The data processing server 140 may also comprise a machine learning module 266 that is configured for processing data relating to a task for a ML model node. The data processing server 140 may also comprise a rule evaluation module 268 that is configured for evaluating rules based on the data from, for example, one or more DP nodes and/or one or more ML model nodes.

[92] In an implementation, the plurality of tasks are executed by the data point module 264, machine learning module 266 and rule evaluation module 268 based on the information indicated in the schema.

[93] Each of the data module 260, sequence module 262, data point module 264, machine learning module 266 and rule evaluation module 268 may further be in communication with a processing module (not shown) of the data processing server 140, for example for coordination of respective tasks and functions during the process. The data module 260 may be further configured to communicate with and store data and

information for each of the processing module, sequence module 262, data point module 264, machine learning module 266 and rule evaluation module 268. Alternatively, all the tasks and functions required for adaptively executing a plurality of tasks may be performed by a single processor of the data processing server 140.

[94] Figs. 6A – 6G illustrate how a plurality of tasks may be executed according to the present disclosure. Matrix 600 of Fig. 6A assigns an identifier to each of nodes 402 – 428 in graph 400 of Fig. 4A. For example, the following identifiers are assigned: start node = 1, DP2 402 = 2, DP3 404 = 3, DP4 406 = 4, DP5 408 = 5, DP6 410 = 6, DP7 412 = 7, DP8 414 = 8, DP9 416 = 9, DP10 418 = 10, DP11 420 = 11, model-1 422 = 12, model-2 424 = 13, model-3 416 = 14, rule evaluation node 428 = 15. These identifiers may be used in an algorithm to evaluate the data points, for example based on the schema 430.

[95] In existing graph-based algorithms, some shortcomings are observed. For example, when utilizing existing algorithms based on topological sorting for the above matrix 600, the resulting order of execution may be 1 8 10 7 11 6 5 4 9 3 2 13 12 14 15. If the nodes are executed parallelly, there might be a scenario in which DP10 418 (e.g. identifier 10) starts executing parallelly with DP8 414 (e.g. identifier 8). There is no way to ensure dependency between nodes when using such a topological sort. Further, when utilizing existing algorithms based on breadth-first search (e.g. level order traversal) for the matrix 600, the resulting order of execution may be 1 | 2 3 4 5 6 7 8 | 10 11 9 12 13 | 14 15, in which identifier 1 is executed in a first level, identifiers 2 3 4 5 6 7 8 are executed in a second level, identifiers 10 11 9 12 13 are executed in a third level, and identifiers 14 15 are executed in a fourth level. Even though there are levels introduced with this approach, it still does not maintain dependencies among nodes. For example, in the above order, DP9 416 (e.g. identifier 9) would be evaluated before model-1 422 (e.g. identifier 12).

[96] In contrast, the present proposed algorithm may comprise three data structures. In a first data structure, a boolean adjacency matrix where each node u of the graph 400 is represented as rows and columns. An entry $u-v$ is marked as 1 if node v has a dependency on node u for its evaluation. This adjacency matrix may be called adj_mat e.g. matrix 602 as shown in Fig. 6B. For example, as DP2 402 only has 1 dependency e.g. on start node 401, entry 604 is marked with a value of '1' to indicate this dependency on start node 401 (e.g. with identifier 1), while the remaining entries in the column corresponding to DP2 402

(e.g. with identifier 2) are each marked with a value of '0'. In a second data structure, a map of nodes with a key representing each respective node id and a value representing a count of nodes that each node is dependent on may be utilized. This map may be called `freq_map` e.g. map 606 of Fig. 6C. The map 606 can be easily constructed by counting the number of times the value '1' occurs in a corresponding column in matrix 602, and then indicating the count in a corresponding entry in map 606. For example, as rule evaluation node 428 with identifier 15 has 5 dependencies (e.g. model-3 node 426, model-1 node 422, model-2 node 424, DP11 420 and DP10 418 as seen in graph 400 of Fig. 4A), a value '5' is indicated in entry 610 of map 606 for identifier 15 (see reference 608). In a third data structure, a worker pool implementation may be utilized that ensure the nodes are executed in an execution queue, and ensures parallel execution of the nodes. This queue may be called `execution_queue`.

[97] The proposed algorithm may be implemented, for example by the sequence module 262 of the data processing server 140, based on the following steps:

1. Start with a dummy node (e.g. start node with identifier 1) and push it into the `execution_queue`.
2. At the end of execution of the dummy node, the respective count for all columns for which value is set in the row denoted by the dummy node (which has just finished executing) in the `adj_mat` is reduced by 1 and the `freq_map` is updated accordingly (e.g. by recounting based on the updated `adj_mat`).
3. As soon as the value for any node in `adj_mat` becomes 0 after step 2, the node/s with value '0' are added to the `execution_queue`.
4. The new node/s that are added to `execution_queue` are then directly picked up for execution by a worker pool for the respective new node/s.
5. Steps 2-4 are then repeated until the end of the graph is reached (e.g. until rule evaluation node 428 in graph 400 is executed).

[98] For example, at a start of execution of step 1, a first data structure (e.g. `adj_mat`) may be constructed as shown in matrix 602, a second data structure (e.g. `freq_map`) may be constructed as shown in map 606, and a third data structure (e.g. `execution_queue`) may be constructed as follows: `<start> / 1 / <end>` to indicate that the node with identifier 1 (e.g. start node 401 of graph 400) is to be executed.

[99] Once start node 401 is executed, the sequence module 262 checks row 1 of matrix 602 (e.g. the row corresponding to identifier 1), and determines that nodes with identifiers 2,3,4,5,6,7 and 8 are marked (e.g. with a value '1'). Therefore, the sequence module 262 reduces the value indicated in the row 1 for the identifiers 2,3,4,5,6,7 and 8 by 1 in matrix 602. Map 606 is also updated accordingly (e.g. by the sequence module 262) as shown in updated map 612 which now indicates that the value for each of identifiers 2,3,4,5,6,7 and 8 is now '0'. As soon as the values are reduced to '0' for the aforementioned nodes as shown in map 612, these nodes are then added to the execution_queue, such that it becomes:

`<start> 2 / 3 / 4 / 5 / 6 / 7 / 8 <end>`

Start node 401 will be removed from the execution_queue because it has finished executing.

[100] For the next step, it is assumed that only one node executes at a time in the worker pool for simplicity. In the next step, after the node with identifier 2 (e.g. DP2 402) is executed (e.g. by the data point module 264 of the data processing server 140), the sequence module 262 checks row 2 of matrix 602 (e.g. the row corresponding to identifier 2) and determines that nodes with identifiers 12 and 13 are marked. Therefore, the sequence module 262 reduces the value indicated in the row 2 for the identifiers 12 and 13 by 1 in matrix 602. Map 612 is also updated accordingly (e.g. by the sequence module 262) as shown in updated map 614 which now indicates that the value for each of identifiers 12 and 13 is now reduced by 1. For example, the value indicated for identifier 12 is reduced from '3' to '2', and the value indicated for identifier 13 is reduced from '4' to '3'. Since the values indicated for identifiers 12 and 13 do not become 0 in the freq_map, the nodes corresponding to these identifiers will not be added to the execution_queue. Node 2 will be removed from the execution_queue because it has finished executing, and the execution_queue is now updated (e.g. by the sequence module 262) to:

`<start> 3 / 4 / 5 / 6 / 7 / 8 <end>`

[101] In the next step, after the node with identifier 3 (e.g. DP3 404) is executed (e.g. by the data point module 264 of the data processing server 140), the sequence module 262 checks row 3 of matrix 602 (e.g. the row corresponding to identifier 3) and determines that nodes with identifiers 12 and 13 are marked. Therefore, the sequence module 262 reduces the value indicated in the row 3 for the identifiers 12 and 13 by 1 in matrix 602. Map 614 is also updated accordingly (e.g. by the sequence module 262) as shown in updated map

616 which now indicates that the value for each of identifiers 12 and 13 is now reduced by 1. For example, the value indicated for identifier 12 is reduced from '2' to '1', and the value indicated for identifier 13 is reduced from '3' to '2'. Since the values indicated for identifiers 12 and 13 do not become 0 in the freq_map, the nodes corresponding to these identifiers will not be added to the execution_queue. Node 3 will be removed from the execution_queue because it has finished executing, and the execution_queue is now updated (e.g. by the sequence module 262) to:

<start> 4 / 5 / 6 / 7 / 8 <end>

[102] In the next step, after the node with identifier 4 (e.g. DP4 406) is executed (e.g. by the data point module 264 of the data processing server 140), the sequence module 262 checks row 4 of matrix 602 (e.g. the row corresponding to identifier 4) and determines that nodes with identifier 9 is marked. Therefore, the sequence module 262 reduces the value indicated in the row 4 for the identifier 9 by 1 in matrix 602. Map 616 is also updated accordingly (e.g. by the sequence module 262) as shown in updated map 618 which now indicates that the value for identifier 9 is now reduced by 1. For example, the value indicated for identifier 9 is reduced from '1' to '0'. Since the value indicated for identifier 9 is now 0 in the freq_map, the node corresponding to this identifier (e.g. DP9 416) will be added to the execution_queue. Node 4 will be removed from the execution_queue because it has finished executing, and the execution_queue is now updated (e.g. by the sequence module 262) to:

<start> 5 / 6 / 7 / 8 / 9 <end>

[103] This process continues until the length of freq_map is reduced to 0 e.g. all nodes are executed. When the process arrives at rows 12, 13 and 14 of matrix 602 (e.g. corresponding to identifiers 12, 13 and 14 for model-1 node 422, model-2 node 424 and model-3 node 426 respectively), each of the corresponding model-1 node 422, model-2 node 424 and model-3 node 426 may be executed by the machine learning module 266 of the data processing server 140. Further, when the process arrives at the last row 15 of matrix 602 (e.g. corresponding to identifier 15 for rule evaluation node 428), the corresponding rule evaluation node 428 may be executed by the rule evaluation module 268 of the data processing server 140.

[104] Fig. 7 illustrates an example flow diagram of a method for adaptively executing a plurality of tasks according to various embodiments. In a step 702, a schema representing

a plurality of tasks is defined, each task comprising a data retrieval and/or data evaluation operation, the schema comprising task information indicating how to execute each of the plurality of tasks. In a step 704, a graph representation of the plurality of tasks is generated based on the schema, the graph representation comprising a plurality of nodes, each node of the plurality of nodes corresponding to one of the plurality of tasks. In a step 706, the plurality of tasks is executed based on the graph representation and the task information.

[105] Fig. 8A depict a general-purpose computer system 1400, upon which the data processing server 140 described can be practiced. The computer system 1400 includes a computer module 1401. An external Modulator-Demodulator (Modem) transceiver device 1416 may be used by the computer module 1401 for communicating to and from a communications network 1420 via a connection 1421. The communications network 1420 may be a wide-area network (WAN), such as the Internet, a cellular telecommunications network, or a private WAN. Where the connection 1421 is a telephone line, the modem 1416 may be a traditional “dial-up” modem. Alternatively, where the connection 1421 is a high capacity (e.g., cable) connection, the modem 1416 may be a broadband modem. A wireless modem may also be used for wireless connection to the communications network 1420.

[106] The computer module 1401 typically includes at least one processor unit 1405, and a memory unit 1406. For example, the memory unit 1406 may have semiconductor random access memory (RAM) and semiconductor read only memory (ROM). The computer module 1401 also includes an interface 1408 for the external modem 1416. In some implementations, the modem 1416 may be incorporated within the computer module 1401, for example within the interface 1408. The computer module 1401 also has a local network interface 1411, which permits coupling of the computer system 1400 via a connection 1423 to a local-area communications network 1422, known as a Local Area Network (LAN). As illustrated in Fig. 8A, the local communications network 1422 may also couple to the wide network 1420 via a connection 1424, which would typically include a so-called “firewall” device or device of similar functionality. The local network interface 1411 may comprise an Ethernet circuit card, a Bluetooth® wireless arrangement or an IEEE 802.11 wireless arrangement; however, numerous other types of interfaces may be practiced for the interface 1411.

[107] The I/O interfaces 1408 may afford either or both of serial and parallel connectivity, the former typically being implemented according to the Universal Serial Bus (USB) standards and having corresponding USB connectors (not illustrated). Storage devices 1409 are provided and typically include a hard disk drive (HDD) 1410. Other storage devices such as a floppy disk drive and a magnetic tape drive (not illustrated) may also be used. An optical disk drive 1412 is typically provided to act as a non-volatile source of data. Portable memory devices, such optical disks, USB-RAM, portable, external hard drives, and floppy disks, for example, may be used as appropriate sources of data to the system 1400.

[108] The components 1405 to 1412 of the computer module 1401 typically communicate via an interconnected bus 1404 and in a manner that results in a conventional mode of operation of the computer system 1400 known to those in the relevant art. For example, the processor 1405 is coupled to the system bus 1404 using a connection 1418. Likewise, the memory 1406 and optical disk drive 1412 are coupled to the system bus 1404 by connections 1419. Examples of computers on which the described arrangements can be practised include IBM-PC's and compatibles, Sun Sparcstations, Apple or like computer systems.

[109] The method 700, where performed by the data processing server 140 may be implemented using the computer system 1400. The processes may be implemented as one or more software application programs 1433 executable within the computer system 1400. In particular, the sub-processes 400, 500, and 600 are effected by instructions in the software 1433 that are carried out within the computer system 1400. The software instructions may be formed as one or more code modules, each for performing one or more particular tasks. The software may also be divided into two separate parts, in which a first part and the corresponding code modules performs the methods and a second part and the corresponding code modules manage a user interface between the first part and the user.

[110] The software may be stored in a computer readable medium, including the storage devices described below, for example. The software is loaded into the computer system 1400 from the computer readable medium, and then executed by the computer system 1400. A computer readable medium having such software or computer program recorded on the computer readable medium is a computer program product. The use of

the computer program product in the computer system 1400 preferably effects an advantageous apparatus for a data processing server 140.

[111] The software 1433 is typically stored in the HDD 1410 or the memory 1406. The software is loaded into the computer system 1400 from a computer readable medium, and executed by the computer system 1400. Thus, for example, the software 1433 may be stored on an optically readable disk storage medium (e.g., CD-ROM) 1425 that is read by the optical disk drive 1412. A computer readable medium having such software or computer program recorded on it is a computer program product. The use of the computer program product in the computer system 1400 preferably effects an apparatus for a data processing server 140.

[112] In some instances, the application programs 1433 may be supplied to the user encoded on one or more CD-ROMs 1425 and read via the corresponding drive 1412, or alternatively may be read by the user from the networks 1420 or 1422. Still further, the software can also be loaded into the computer system 1400 from other computer readable media. Computer readable storage media refers to any non-transitory tangible storage medium that provides recorded instructions and/or data to the computer system 1400 for execution and/or processing. Examples of such storage media include floppy disks, magnetic tape, optical disc, a hard disk drive, a ROM or integrated circuit, USB memory, a magneto-optical disk, or a computer readable card such as a PCMCIA card and the like, whether or not such devices are internal or external of the computer module 1401. Examples of transitory or non-tangible computer readable transmission media that may also participate in the provision of software, application programs, instructions and/or data to the computer module 1401 include radio or infra-red transmission channels as well as a network connection to another computer or networked device, and the Internet or Intranets including e-mail transmissions and information recorded on Websites and the like.

[113] The second part of the application programs 1433 and the corresponding code modules mentioned above may be executed to implement one or more graphical user interfaces (GUIs) to be rendered or otherwise represented upon a display. Through manipulation of typically a keyboard and a mouse, a user of the computer system 1400 and the application may manipulate the interface in a functionally adaptable manner to provide controlling commands and/or input to the applications associated with the GUI(s).

Other forms of functionally adaptable user interfaces may also be implemented, such as an audio interface utilizing speech prompts output via loudspeakers and user voice commands input via a microphone.

[114] It is to be understood that the structural context of the computer system 1400 (i.e., the data processing server 140) is presented merely by way of example. Therefore, in some arrangements, one or more features of the computer system 1400 may be omitted. Also, in some arrangements, one or more features of the computer system 1400 may be combined together. Additionally, in some arrangements, one or more features of the computer system 1400 may be split into one or more component parts.

[115] Fig. 9 shows an alternative implementation of the transaction processing server 108 (i.e., the computer system 1300). In the alternative implementation, the transaction processing 108 may be generally described as a physical device comprising at least one processor 802 and at least one memory 804 including computer program codes. The at least one memory 804 and the computer program codes are configured to, with the at least one processor 802, cause the transaction processing server 108 to facilitate the operations described in method 700. The transaction processing server 108 may also include a transaction processing module 806. The memory 804 stores computer program code that the processor 802 compiles to have transaction processing module 806 perform the respective functions.

[116] With reference to Fig. 1, the transaction processing module 806 performs the function of communicating with the requestor device 102 and the provider device 104; and the acquirer server 106 and the issuer server 110 to respectively receive and transmit a transaction, travel request message, or other similar messages. Further, the transaction processing module 806 may provide data and information relating to an approved or rejected transaction (e.g. date, time, amount, currency, user name, and other similar data relating to the concerned transaction) to the data processing server 140 as raw data that may be utilized for processing by data points. The processed data may then be stored or transferred to a database, for example database 150. In an implementation, the transaction processing server may also be in communication with a database directly which will store the data relating to an approved or rejected transaction as raw data, or may also be configured to process the data before doing so.

[117] Fig. 10 shows an alternative implementation of the data processing server 140 (i.e., the computer system 1400). In the alternative implementation, data processing server 140 may be generally described as a physical device comprising at least one processor 902 and at least one memory 904 including computer program codes. The at least one memory 904 and the computer program codes are configured to, with the at least one processor 902, cause the data processing server 140 to perform the operations described in the method 700. The data processing server 140 may also include a data module 906, a sequence module 908, a data point module 910, a machine learning module 912 and a rule evaluation module 914. The memory 904 stores computer program code that the processor 902 compiles to have each of the modules 906 to 914 performs their respective functions.

[118] With reference to Figs. 1 to 7, the sequence module 908 performs the function of defining, by one or more processors, a schema representing a plurality of tasks, each task comprising a data retrieval and/or data evaluation operation, the schema comprising task information indicating how to execute each of the plurality of tasks; and generating, by the one or more processors, a graph representation of the plurality of tasks based on the schema, the graph representation comprising a plurality of nodes, each node of the plurality of nodes corresponding to one of the plurality of tasks. The sequence module 908 may be further configured to determine an execution order for the plurality of tasks by: generating a matrix based on the graph representation, the matrix indicating, for each node of the plurality of nodes, whether there is a dependency on another node of the plurality of nodes; generating a frequency map based on the matrix, the frequency map indicating a total number of dependencies for each node of the plurality of nodes; and determining an execution order for the plurality of tasks based on the frequency map. Determining the execution order may further comprise identifying a node with zero dependencies from the frequency map, and adding the identified node to an execution queue. The sequence module 908 may be further configured to reduce the total number of dependencies for each node that is dependent on the identified node by one in the frequency map after the identified node is executed, and remove the identified node from the execution queue.

[119] The sequence module 908 may be further configured to define a plurality of tasks based on a schema, the schema comprising information indicating how to execute each of the plurality of tasks. The sequence module 908 may be further configured to determine

one or more of the plurality of tasks that can only be executed after a first task has been executed, and a counter for each of the one or more tasks, and identify a second task from the one or more tasks to be executed based on a number indicated in each counter.

[120] With reference to Figs. 1 to 7, the data point module 910 performs the function of executing, by the one or more processors, the plurality of tasks based on the graph representation and the task information. Two or more tasks of the plurality of tasks may be executed in parallel by one or more processors. The task information may further indicate data to be retrieved and a data source for a data retrieval operation of the plurality of tasks, wherein executing the task further comprises retrieving the indicated data from the data source. The data point module 910 may be further configured to execute a task corresponding to the identified node. The data point module 910 may be further configured to execute a task for a DP node. In an implementation, wherein the information further indicates data to be retrieved and a data source for a task of the plurality of tasks, executing the task further comprises retrieving the indicated data from the data source.

[121] With reference to Figs. 1 to 7, the machine learning module 912 performs the function of processing data relating to a task for a ML model node. With reference to Figs. 1 to 7, the rule evaluation module 914 performs the function of evaluating rules based on the data from, for example, one or more DP nodes and/or one or more ML model nodes.

[122] With reference to Figs. 1 to 7, the data module 906 performs the functions of receiving data and information from the requestor device 102, provider device 104, transaction processing server 108, database 150, a cloud and other sources of information to facilitate the method 700. For example, the data module 906 may be configured to receive data and information from the requestor device 102, provider device 104, transaction processing server 108, database 150, a cloud and other sources of information to facilitate adaptively executing a plurality of tasks. For example, the data module 906 may be configured to receive data and information required for adaptively executing a plurality of tasks from the requestor device 102, the provider device 104, transaction processing server 108, database 150, and/or other sources of information. The data module 906 may be further configured to send information relating to a completed task to the requestor device 102, the provider device 104, the transaction processing server 108, or other destinations where the information is required. The data module 906 may be further configured to communicate with and store data and information for each of the

sequence module 908, data point module 910, machine learning module 912 and rule evaluation module 914. Alternatively, all the tasks and functions required for facilitating the method 700 may be performed by a single processor 902 of the data processing server 140, or by one or more processors.

[123] Fig. 8B depicts a general-purpose computer system 1500, upon which a combined transaction processing server 108 and data processing server 140 described can be practiced. The computer system 1500 includes a computer module 1501. An external Modulator-Demodulator (Modem) transceiver device 1516 may be used by the computer module 1501 for communicating to and from a communications network 1520 via a connection 1521. The communications network 1520 may be a wide-area network (WAN), such as the Internet, a cellular telecommunications network, or a private WAN. Where the connection 1521 is a telephone line, the modem 1516 may be a traditional “dial-up” modem. Alternatively, where the connection 1521 is a high capacity (e.g., cable) connection, the modem 1516 may be a broadband modem. A wireless modem may also be used for wireless connection to the communications network 1520.

[124] The computer module 1501 typically includes at least one processor unit 1505, and a memory unit 1506. For example, the memory unit 1506 may have semiconductor random access memory (RAM) and semiconductor read only memory (ROM). The computer module 1501 also includes an interface 1508 for the external modem 1516. In some implementations, the modem 1516 may be incorporated within the computer module 1501, for example within the interface 1508. The computer module 1501 also has a local network interface 1511, which permits coupling of the computer system 1500 via a connection 1523 to a local-area communications network 1522, known as a Local Area Network (LAN). As illustrated in Fig. 8B, the local communications network 1522 may also couple to the wide network 1520 via a connection 1524, which would typically include a so-called “firewall” device or device of similar functionality. The local network interface 1511 may comprise an Ethernet circuit card, a Bluetooth® wireless arrangement or an IEEE 802.11 wireless arrangement; however, numerous other types of interfaces may be practiced for the interface 1511.

[125] The I/O interfaces 1508 may afford either or both of serial and parallel connectivity, the former typically being implemented according to the Universal Serial Bus (USB) standards and having corresponding USB connectors (not illustrated). Storage

devices 1509 are provided and typically include a hard disk drive (HDD) 1510. Other storage devices such as a floppy disk drive and a magnetic tape drive (not illustrated) may also be used. An optical disk drive 1512 is typically provided to act as a non-volatile source of data. Portable memory devices, such optical disks, USB-RAM, portable, external hard drives, and floppy disks, for example, may be used as appropriate sources of data to the system 1500.

[126] The components 1505 to 1512 of the computer module 1501 typically communicate via an interconnected bus 1504 and in a manner that results in a conventional mode of operation of the computer system 1500 known to those in the relevant art. For example, the processor 1505 is coupled to the system bus 1504 using a connection 1518. Likewise, the memory 1506 and optical disk drive 1512 are coupled to the system bus 1504 by connections 1519. Examples of computers on which the described arrangements can be practised include IBM-PC's and compatibles, Sun Sparcstations, Apple or like computer systems.

[127] The steps of the method 700 performed by the data processing server 140 and facilitated by the transaction processing server 108 may be implemented using the computer system 1500. For example, the steps of the method 700 as performed by the data processing server 140 may be implemented as one or more software application programs 1533 executable within the computer system 1500. In particular, the steps of the method 700 are effected by instructions in the software 1533 that are carried out within the computer system 1500. The software instructions may be formed as one or more code modules, each for performing one or more particular tasks. The software may also be divided into two separate parts, in which a first part and the corresponding code modules performs the steps of the method 700 and a second part and the corresponding code modules manage a user interface between the first part and the user.

[128] The software may be stored in a computer readable medium, including the storage devices described below, for example. The software is loaded into the computer system 1500 from the computer readable medium, and then executed by the computer system 1500. A computer readable medium having such software or computer program recorded on the computer readable medium is a computer program product. The use of the computer program product in the computer system 1500 preferably effects an

advantageous apparatus for a combined transaction processing and data processing server.

[129] The software 1533 is typically stored in the HDD 1510 or the memory 1506. The software is loaded into the computer system 1500 from a computer readable medium, and executed by the computer system 1500. Thus, for example, the software 1533 may be stored on an optically readable disk storage medium (e.g., CD-ROM) 1525 that is read by the optical disk drive 1512. A computer readable medium having such software or computer program recorded on it is a computer program product. The use of the computer program product in the computer system 1500 preferably effects an apparatus for a combined transaction processing and data processing server.

[130] In some instances, the application programs 1533 may be supplied to the user encoded on one or more CD-ROMs 1525 and read via the corresponding drive 1512, or alternatively may be read by the user from the networks 1520 or 1522. Still further, the software can also be loaded into the computer system 1500 from other computer readable media. Computer readable storage media refers to any non-transitory tangible storage medium that provides recorded instructions and/or data to the computer system 1500 for execution and/or processing. Examples of such storage media include floppy disks, magnetic tape, optical disc, a hard disk drive, a ROM or integrated circuit, USB memory, a magneto-optical disk, or a computer readable card such as a PCMCIA card and the like, whether or not such devices are internal or external of the computer module 1501. Examples of transitory or non-tangible computer readable transmission media that may also participate in the provision of software, application programs, instructions and/or data to the computer module 1501 include radio or infra-red transmission channels as well as a network connection to another computer or networked device, and the Internet or Intranets including e-mail transmissions and information recorded on Websites and the like.

[131] The second part of the application programs 1533 and the corresponding code modules mentioned above may be executed to implement one or more graphical user interfaces (GUIs) to be rendered or otherwise represented upon a display. Through manipulation of typically a keyboard and a mouse, a user of the computer system 1500 and the application may manipulate the interface in a functionally adaptable manner to provide controlling commands and/or input to the applications associated with the GUI(s).

Other forms of functionally adaptable user interfaces may also be implemented, such as an audio interface utilizing speech prompts output via loudspeakers and user voice commands input via a microphone.

[132] It is to be understood that the structural context of the computer system 1500 (i.e., combined transaction processing and data processing server 1500) is presented merely by way of example. Therefore, in some arrangements, one or more features of the server 1500 may be omitted. Also, in some arrangements, one or more features of the server 1500 may be combined together. Additionally, in some arrangements, one or more features of the server 1500 may be split into one or more component parts.

[133] Fig. 11 shows an alternative implementation of combined transaction processing and data processing server (i.e., the computer system 1500). In the alternative implementation, the combined transaction processing and data processing server may be generally described as a physical device comprising at least one processor 1002 and at least one memory 904 including computer program codes. The at least one memory 1004 and the computer program codes are configured to, with the at least one processor 1002, cause the combined transaction processing and data processing server to perform the operations described in the steps of the method 700. The combined transaction processing and data processing server may also include a transaction processing module 806, a data module 906, a sequence module 908, a data point module 910, a machine learning module 912 and a rule evaluation module 914. The memory 1004 stores computer program code that the processor 1002 compiles to have each of the modules 806 to 912 performs their respective functions. The transaction processing module 806 performs the same functions as described for the same transaction processing module in Fig. 9. The data module 906, a sequence module 908, a data point module 910, a machine learning module 912 and a rule evaluation module 914 perform the same functions as described for the same corresponding modules in Fig. 10.

[134] It will be appreciated by a person skilled in the art that numerous variations and/or modifications may be made to the present disclosure as shown in the specific embodiments without departing from the scope of the specification as broadly described. The present embodiments are, therefore, to be considered in all respects to be illustrative and not restrictive.

CLAIMS

What is claimed is:

1. A method for adaptively executing a plurality of tasks, comprising:
 - defining, by one or more processors, a schema representing a plurality of tasks, each task comprising a data retrieval and/or data evaluation operation, the schema comprising task information indicating how to execute each of the plurality of tasks;
 - generating, by the one or more processors, a graph representation of the plurality of tasks based on the schema, the graph representation comprising a plurality of nodes, each node of the plurality of nodes corresponding to one of the plurality of tasks;
 - executing, by the one or more processors, the plurality of tasks based on the graph representation and the task information;
 - generating a matrix based on the graph representation, the matrix indicating, for each node of the plurality of nodes, whether there is a dependency on another node of the plurality of nodes;
 - generating a frequency map based on the matrix, the frequency map indicating a total number of dependencies for each node of the plurality of nodes; and
 - determining an execution order for the plurality of tasks based on the frequency map.
2. A method according to claim 1, wherein two or more tasks of the plurality of tasks are executed in parallel by one or more processors.
3. The method of claim 1 or claim 2, wherein the task information further indicates data to be retrieved and a data source for a data retrieval operation of the plurality of tasks, wherein executing the task further comprises retrieving the indicated data from the data source.
4. The method of claim 1, wherein determining the execution order further comprises:
 - identifying a node with zero dependencies from the frequency map; and
 - adding the identified node to an execution queue.
5. The method of claim 4, further comprising:
 - executing a task corresponding to the identified node;

reducing the total number of dependencies for each node that is dependent on the identified node by one in the frequency map; and
 removing the identified node from the execution queue.

6. A system for adaptively executing a plurality of tasks, comprising:
 at least one processor; and
 at least one memory including computer program code;
 the at least one memory and the computer program code configured to, with the at least one processor, cause the system at least to:
 define, by one or more processors, a schema representing a plurality of tasks, each task comprising a data retrieval and/or data evaluation operation, the schema comprising task information indicating how to execute each of the plurality of tasks;
 generate, by the one or more processors, a graph representation of the plurality of tasks based on the schema, the graph representation comprising a plurality of nodes, each node of the plurality of nodes corresponding to one of the plurality of tasks;
 execute, by the one or more processors, the plurality of tasks based on the graph representation and the task information;
 generate a matrix based on the graph representation, the matrix indicating, for each node of the plurality of nodes, whether there is a dependency on another node of the plurality of nodes;
 generate a frequency map based on the matrix, the frequency map indicating a total number of dependencies for each node of the plurality of nodes; and
 determine an execution order for the plurality of tasks based on the frequency map.
7. The system according to claim 6, wherein two or more tasks of the plurality of tasks are executed in parallel by one or more processors.
8. The system of claim 6 or claim 7, wherein the task information further indicates data to be retrieved and a data source for a data retrieval operation of the plurality of tasks, wherein executing the task further comprises retrieving the indicated data from the data source.
9. The system of claim 6, wherein determining the execution order further comprises:
 identifying a node with zero dependencies from the frequency map; and
 adding the identified node to an execution queue.

10. The system of claim 9, further configured to:
 - execute a task corresponding to the identified node;
 - reduce the total number of dependencies for each node that is dependent on the identified node by one in the frequency map; and
 - remove the identified node from the execution queue.