

STORING AND SEARCHING FOR DATA IN DATA STORES

Technical Field

[0001] The present invention relates to measures to store and search for data in a data store. Such data may be machine learning system model data, i.e. data for a machine learning model used by a machine learning system. Examples of such data include, but are not limited to, parameter data and/or state data for the machine learning model. The machine learning system may be a real-time transaction processing system.

Background

[0002] Digital payments have exploded over the last twenty years, with more than three-quarters of global payments using some form of payment card or electronic wallet. Point of sale systems are progressively becoming digital rather than cash-based. Put simply, global systems of commerce are now heavily reliant on electronic data processing platforms. This presents many engineering challenges that are primarily hidden from a lay user. For example, digital transactions need to be completed in real-time, i.e. with a minimal level of delay experienced by computer devices at the point of purchase. Digital transactions also need to be secure and resistant to attack and exploitation. The processing of digital transactions is also constrained by the historic development of global electronic systems for payments. For example, much infrastructure is still configured around models that were designed for mainframe architectures in use over 50 years ago.

[0003] As digital transactions increase, new security risks also become apparent. Digital transactions present new opportunities for fraud and malicious activity. In 2015, it was estimated that 7% of digital transactions were fraudulent, and that figure has only increased with the transition of more economic activity online. Fraud losses are growing.

[0004] While risks like fraud are an economic issue for companies involved in commerce, the implementation of technical systems for processing transactions is an engineering challenge. Traditionally, banks, merchants and card issuers developed “paper” rules or procedures that were manually implemented by clerks to flag or block certain transactions. As transactions became digital, one approach to building technical systems for processing transactions has been to supply computer engineers with these sets of developed criteria and to ask the computer engineers to implement them using digital representations of the transactions, i.e. convert the hand-written rules into coded logic statements that may be applied to electronic transaction data. This traditional approach has run into several problems as digital transaction volumes have grown. First, any applied processing needs to take place at “real-time”, e.g. with millisecond latencies. Second,

many thousands of transactions need to be processed every second (e.g., a common “load” may be 1000-2000 per second), with load varying unexpectedly over time (e.g., a launch of a new product or a set of tickets can easily increase an average load level by several multiples). Third, the digital storage systems of transaction processors and banks are often siloed or partitioned for security reasons, yet digital transactions often involve an interconnected web of merchant systems. Fourthly, large scale analysis of actual reported fraud and predicted fraud is now possible. This shows that traditional approaches to fraud detection are found wanting; accuracy is low and false positives are high. This then has a physical effect on digital transaction processing, more genuine point-of-sale and online purchases are declined and those seeking to exploit the new digital systems often get away with it.

[0005] In the last few years, a more machine learning approach has been taken to the processing of transaction data. As machine learning models mature in academia, engineers have begun to attempt to apply them to the processing of transaction data. However, this again runs into problems. Even if engineers are provided with an academic or theoretical machine learning model and asked to implement it, this is not straightforward. For example, the problems of large-scale transaction processing systems come into play. Machine learning models do not have the luxury of unlimited inference time as in the laboratory. This means that it is simply not practical to implement certain models in a real-time setting, or that they need significant adaptation to allow real-time processing in the volume levels experienced by real-world servers. Moreover, engineers need to contend with the problem of implementing machine learning models on data that is siloed or partitioned based on access security, and in situations where the velocity of data updates is extreme. The problems faced by engineers building transaction processing systems may thus be seen as being akin to those faced by network or database engineers; machine learning models need to be applied but meeting system throughput and query response time constraints set by the processing infrastructure. There are no easy solutions to these problems. Indeed, the fact that many transaction processing systems are confidential, proprietary, and based on old technologies means that engineers do not have the body of knowledge developed in these neighbouring fields and often face challenges that are unique to the field of transaction processing. Moreover, the field of large-scale practical machine learning is still young, and there are few established design patterns or textbooks that engineers can rely on.

[0006] As indicated above, engineers building transaction processing systems may face problems akin to those faced by database engineers. Examples of such database-related problems include, but are not limited to, creation, configuration, management, maintenance, use, structure, optimisation, security and organisation. However, the demands of databases for use in transaction

processing systems, especially those implementing machine learning systems and providing real-time processing, can be significantly different from database demands in other technical fields. For example, such transaction processing systems may need to provide one or more of: (i) real-time database reading capabilities, (ii) high availability for “online” transaction processing capabilities with limited or no downtime for database management, (iii) the ability to handle significant amounts of state data that can build up quickly, (iv) high stability and reliability, (v) data security, and (vi) the ability to handle access from multiple different clients.

[0007] Copy-on-write snapshots are a staple feature of most existing database systems. In addition, in existing database systems, a new set of data can be copied on top of an existing set of data. However, the latter scales linearly with the size of the new set of data. Additionally, in existing database systems, large modifications cannot be done atomically, or concurrently with other clients accessing the same set of data.

Summary

[0008] Aspects of the present invention are set out in the appended independent claims. Certain variations of the invention are then set out in the appended dependent claims. Further aspects, variations and examples are presented in the detailed description below.

Brief Description of the Drawings

[0009] Examples of the invention will now be described, by way of example only, with reference to the accompanying drawings, in which:

[0010] FIGS. 1A to 1C are schematic diagrams showing different example electronic infrastructures for transaction processing.

[0011] FIGS. 2A and 2B are schematic diagrams showing different examples of data storage systems for use by a machine learning transaction processing system.

[0012] FIGS. 3A and 3B are schematic diagrams showing different examples of transaction data.

[0013] FIG. 4 is a schematic diagram showing example components of a machine learning transaction processing system.

[0014] FIGS. 5A and 5B are sequence diagrams showing an example set of processes performed by different computing entities on transaction data.

[0015] FIGS. 6A to 6E are state graphs representing example data stores.

[0016] FIG. 7 is a state graph representing another example data store.

[0017] FIGS. 8A and 8B are schematic diagrams showing different examples of data and information stored in a data store.

[0018] FIG. 9 is a state graph representing another example data store.

[0019] FIG. 10 is a state graph representing another example data store.

[0020] FIG. 11 is a state graph representing another example data store.

[0021] FIGS. 12A and 12B are state graphs representing other example data stores.

[0022] FIGS. 13A and 13B are state graphs representing other example data stores.

[0023] FIG. 14 is a state graph representing another example data store.

[0024] FIG. 15 is a state graph representing another example data store.

[0025] FIGS. 16A to 16E are graphs representing example data queues.

[0026] FIGS. 17A and 17B are graphs representing other example data queues.

[0027] FIG. 18 is a state graph representing another example data store.

Detailed Description

Introduction

[0028] Certain examples described herein relate to measures for use in a machine learning system for use in transaction processing. In certain examples, a machine learning system is applied in real-time, high-volume transaction processing pipelines to provide an indication of whether a transaction or entity matches previously observed and/or predicted patterns of activity or actions, e.g. an indication of whether a transaction or entity is “normal” or “anomalous”. The term “behavioural” is used herein to refer to this pattern of activity or actions. The indication may comprise a scalar value normalised within a predefined range (e.g., 0 to 1) that is then useable to prevent fraud and other misuse of payment systems. The machine learning systems may apply machine learning models that are updated as more transaction data is obtained, e.g. that are constantly trained based on new data, so as to reduce false positives and maintain accuracy of the output metric. Examples described herein relate to improved handling of such data, for example in terms of searching for and storing such data in one or more data stores. The present examples may be particularly useful for preventing fraud in cases where the physical presence of a payment card cannot be ascertained (e.g., online transactions referred to as “card-not-present”) or for commercial transactions where high-value transactions may be routine and where it may be difficult to classify patterns of behaviour as “unexpected”. As such, the present examples facilitate the processing of transactions as these transactions to being primarily “online”, i.e. conducted digitally over one or more public communications networks.

[0029] Certain examples described herein allow machine learning models to be tailored to be specific to certain entities, such as account holders and merchants. For example, the machine learning models may model entity-specific patterns of behaviour as opposed to general group or aggregate behaviour that results in poor accuracy. The measures and machine learning systems described herein are able to provide dynamically updating machine learning models despite large transaction flows and/or despite the need for segregation of different data sources. Again,

examples described herein provide improved measures for reading and storing machine learning model data to be used in such machine learning models.

[0030] The present examples may be applied to a wide variety of digital transactions, including, but not limited to, card payments, so-called “wire” transfers, peer-to-peer payments, Bankers' Automated Clearing System (BACS) payments, and Automated Clearing House (ACH) payments. The output of the machine learning system may be used to prevent a wide variety of fraudulent and criminal behaviour such as card fraud, application fraud, payment fraud, merchant fraud, gaming fraud and money laundering.

[0031] The present example machine learning systems, e.g. as configured and/or as trained according to FIGS. 1A to 18 below, allow for fast inference that can be easily parallelised to provide second or sub-second processing latencies and to manage large processing volumes (e.g., billions of transactions a year).

[0032] More specifically, examples described herein enable multiple diverging sets of data to be stored in a data store (such as a database), based on a common historical set of data, without the historical data being duplicated. Examples enable one set of data to be replaced logically with another set of data, without all data from one set being copied on top of the data in another set. Examples enable making a new set of data accessible to a client as a single atomic operation, without requiring the data to be copied within a data store. Where such data comprises machine learning model data, machine learning system processing latencies can be reduced, and throughput can be increased. As will be explained in more detail below, various measures are provided which make most data changes visible as part of a single atomic change to “state id” metadata. For example, a large set of data can be copied into a data set that is separate from a “live” data set used by running processes, as opposed to being copied into the live data set. The separate data set can have a state id that is different from, and independent of, a live state id used by the live data set. The state id of the separate data set can be made visible atomically to existing processes by (i) creating a new child data set of the live data set with the child data set becoming a new live data set and there being a parent link between the new live data set and the previous live data set, (ii) establishing an overlay relationship (which may also be referred to herein as an “overlay link”, an “overlay association” or the like) between the separate data set and the new live data set (i.e. the newly created child data set) through a state metadata change and, and (iii) notifying clients of the state metadata change. Clients can then read data from the amalgam of the live and other data sets, rather than requiring a manual copy operation to be performed before the data can be read from a single combined data set. Maintaining separate data sets in this manner also reduces the risk of

inadvertent and undesired changes to the live data set, which could negatively affect transaction processing for example.

Certain Term Definitions

[0033] The term “data” is used in different contexts herein to refer to digital information, such as, but not limited to, that represented by known bit structures within one or more programming languages. In use, data may refer to digital information that is stored as bit sequences within computer memory. Certain machine learning models may operate on structured arrays of data of a predefined bit format. Using terms of the art, these may be referred to as multidimensional arrays or “tensors”. It should be noted that for machine learning methods multidimensional arrays, e.g., with a defined extent in multiple dimensions, may be “flattened” so as to be represented (e.g., within memory) as a sequence or vector of values stored according to the predefined format (e.g., *n*-bit integer or floating point number, signed or unsigned). Hence, the term “tensor” as used herein covers multidimensional arrays with one or more dimensions (e.g., vectors, matrixes, volumetric arrays etc). Data may, however, take other forms.

[0034] The term “structured numeric representation” is used to refer to numeric data in a structured form, such as an array of one or more dimensions that stores numeric values with a common data type, such as integers or float values. A structured numeric representation may comprise a tensor (as used within machine learning terminology). A structured numeric representation is typically stored as a set of indexed and/or consecutive memory locations, e.g. a one-dimensional array of 64-bit floats may be represented in computer memory as a consecutive sequence of 64-bit memory locations in a 64-bit computing system.

[0035] The term “transaction data” is used herein to refer to electronic data that is associated with a transaction. A transaction comprises a series of communications between different electronic systems to implement a payment or exchange. In general, transaction data may comprise data indicating events (e.g., actions undertaken in time) that relate to, and may be informative for, transaction processing. Transaction data may comprise structured, unstructured and semi-structured data. In certain cases, transaction data may be used broadly to refer to actions taken with respect to one or more electronic devices. Transaction data may take a variety of forms depending on the precise implementation. However, different data types and formats may be converted by pre or post processing as appropriate.

[0036] The term “interface” is used herein to refer to any physical and/or logical interface that allows for one or more of data input and data output. An interface may be implemented by a network interface adapted to send and/or receive data, or by retrieving data from one or more memory locations, as implemented by a processor executing a set of instructions. An interface may

also comprise physical (network) couplings over which data is received, such as hardware to allow for wired or wireless communications over a particular medium. An interface may comprise an application programming interface and/or a method call or return. For example, in a software implementation, an interface may comprise passing data and/or memory references to a function initiated via a method call, where the function comprises computer program code that is executed by one or more processors; in a hardware implementation, an interface may comprise a wired interconnect between different chips, chipsets or portions of chips. In the drawings, an interface may be indicated by a boundary of a processing block that has an inward and/or outward arrow representing a data transfer.

[0037] The terms “component” and “module” are used interchangeably to refer to either a hardware structure that has a specific function (e.g., in the form of mapping input data to output data) or a combination of general hardware and specific software (e.g., specific computer program code that is executed on one or more general purpose processors). A component or module may be implemented as a specific packaged chipset, for example, an Application Specific Integrated Circuit (ASIC) or a programmed Field Programmable Gate Array (FPGA), and/or as a software object, class, class instance, script, code portion or the like, as executed in use by a processor.

[0038] The term “machine learning model” is used herein to refer to at least a hardware-executed implementation of a machine learning model or function. Known models within the field of machine learning include logistic regression models, Naïve Bayes models, Random Forests, Support Vector Machines and artificial neural networks. Implementations of classifiers may be provided within one or more machine learning programming libraries including, but not limited to, scikit-learn, TensorFlow, and PyTorch.

[0039] The term “map” is used herein to refer to the transformation or conversion of a first set of data values to a second set of data values. The two sets of data values may be arrays of different sizes, with an output array being of lower dimensionality than an input array. The input and output arrays may have common or different data types. In certain examples, the mapping is a one-way mapping to a scalar value.

[0040] The term “data store” is used herein to refer to a repository for storing data. An example of a data store is a database. However, data stores may take different forms, for example depending on implementation details. Another example type of data store is a file storage system.

[0041] The term “data set” is used herein to refer to a collection of data. A data set may be empty, for example on its creation.

[0042] The term “data item” is used herein to refer to information being searched for in a data store. A data item may uniquely identify a record in a data store. The terms “key” and “search key”

may be used herein interchangeably with the term “data item”. Data may be stored in association with a data item in a data store.

[0043] The term “data element” is used herein to relate to the whole or part of given data stored in the data store. Data may comprise one or more data elements in this respect.

[0044] The term “metadata” is used herein to relate to information that provides information about other data. Metadata may be stored in the same data store as state data or may be stored in a different data store.

Example Transaction Processing System

[0045] FIGS. 1A to 1C show a set of example transaction processing systems 100, 102, 104. These example transaction processing systems are described to provide context for the inventions discussed herein but should not be seen as limiting; the configuration of any one implementation may differ based on the specific requirements of that implementation. However, the described example transaction processing systems allow those skilled in the art to identify certain high-level technical features that are relevant for the description below. The three example transaction processing systems 100, 102, 104 show different areas where variation may occur.

[0046] FIGS. 1A to 1C show a set of client devices 110 that are configured to initiate a transaction. In this example, the set of client devices 110 comprise a smartphone 110-A, a computer 110-B, a point-of-sale (POS) system 110-C and a portable merchant device 110-D. These client devices 110 provide a set of non-exhaustive examples. Generally, any electronic device or set of devices may be used to undertake a transaction. In one case, the transaction comprises a purchase or payment. For example, the purchase or payment may be an online or mobile purchase or payment made by way of the smartphone 110-A or the computer 110-B, or may be a purchase or payment made at a merchant premises, such as via the POS system 110-C or the portable merchant device 110-D. The purchase or payment may be for goods and/or services.

[0047] In FIGS. 1A to 1C, the client devices 110 are communicatively coupled to one or more computer networks 120. The client devices 110 may be communicatively coupled in a variety of ways, including by one or more wired and/or wireless networks including telecommunications networks. In preferred examples, all communications across the one or more computer networks are secured, e.g. using Transport Layer Security (TLS) protocols. In FIG. 1A, two computer networks are shown 120-A and 120-B. These may be separate networks or different portions of a common network. The first computer network 120-A communicatively couples the client devices 110 to a merchant server 130. The merchant server 130 may execute a computer process that implements a process flow for the transaction. For example, the merchant server 130 may be a back-end server that handles transaction requests received from the POS system 110-C or the

portable merchant device 110-D or may be used by an online merchant to implement a website where purchases may be made. It will be appreciated that the examples of FIGS. 1A to 1C are necessary simplifications of actual architectures; for example, there may be several interacting server devices that implement an online merchant, including separate server devices for providing HyperText Markup Language (HTML) pages detailing a product and/or service and for handling a payment process.

[0048] In FIG. 1A, the merchant server 130 is communicatively coupled to a further set of back-end server devices to process the transaction. In FIG. 1A, the merchant server 130 is communicatively coupled to a payment processor server 140 via a second network 120-B. The payment processor server 140 is communicatively coupled to a first data storage device 142 storing transaction data 146 and a second data storage device 144 storing ancillary data 148. The transaction data 146 may comprise batches of transaction data relating to different transactions that are undertaken over a period of time. The ancillary data 148 may comprise data associated with the transactions, such as records storing merchant and/or end user data. In FIG. 1A, the payment processor server 140 is communicatively coupled to a machine learning server 150 via the second network 120-B. The machine learning server 150 implements a machine learning system 160 for the processing of transaction data. The machine learning system 160 is arranged to receive input data 162 and to map this to output data 164 that is used by the payment processor server 140 to process a particular transaction, such as one arising from the client devices 110. In one case, the machine learning system 160 receives at least transaction data associated with the particular transaction and provides an alert or numeric output that is used by the payment processor server 140 to determine whether the transaction is to be authorised (i.e., approved) or declined. As such, the output of the machine learning system 160 may comprise a label, alert or other indication of fraud, or general malicious or anomalous activity. The output may comprise a probabilistic indication, such as a score or probability. In one case, the output data 164 may comprise a scalar numeric value. The input data 162 may further comprise data derived from one or more of the transaction data 146 and the ancillary data 148. In one case, the output data 164 indicates a level of deviation from a specific expected pattern of behaviour based on past observations or measurements. For example, this may indicate fraud or criminal behaviour as this often differs significantly from observed patterns of behaviour, especially on a large scale. The output data 164 may form a behavioural measure. The expected pattern of behaviour may be defined, either explicitly or implicitly, based on observed interactions between different entities within the transaction process flow, such as end users or customers, merchants (including point-of-sale and back-end locations or entities where these may differ), and banks.

[0049] The machine learning system 160 may be implemented as part of a transaction processing pipeline. An example transaction processing pipeline is described later with respect to FIGS. 5A and 5B. A transaction processing pipeline may comprise electronic communications between the client devices 110, merchant server 130, payment processor server 140 and machine learning server 150. Other server devices may also be involved, such as banking servers that provide authorisation from an issuing bank. In certain cases, client devices 110 may directly communicate with the payment processor server 140. In use, a transaction processing pipeline typically needs to be completed within one or two hundred milliseconds. In general, sub-second processing times may be deemed real-time (e.g., human beings typically perceive events on a timespan of 400ms). Furthermore, 100-200ms may be the desired maximum latency of the full round-trip-time for transaction processing; within this timespan, the time allotted for the machine learning system 160 may be a small fraction of this full amount, such as 10ms (i.e., less than 5-10% of the target processing time), as most of the time may be reserved for other operations in the transaction processing flow. This presents a technical constraint for the implementation of the machine learning system 160. Furthermore, in real-world implementations, average processing volumes may be on the order of 1000-2000 a second. This means that most “off-the-shelf” machine learning systems are not suitable to implement machine learning system 160. It further means that most machine learning approaches described in academic papers cannot be implemented within the aforementioned transaction processing pipeline without non-obvious adaptations. There is also a problem that anomalies are, by their very nature, rare events and so accurate machine learning systems are difficult to train.

[0050] FIG. 1B shows a variation 102 of the example transaction processing system 100 of FIG. 1A. In this variation 102, the machine learning system 160 is implemented within the payment processor computer infrastructure, e.g. executed by the payment processor server 140 and/or executed on a locally coupled server within the same local network as the payment processor server 140. The variation 102 of FIG. 1B may be preferred for larger payment processors as it allows faster response times, greater control, and improved security. However, functionally, the transaction processing pipeline may be similar to that of FIG. 1A. For example, in the example of FIG. 1A, the machine learning system 160 may be initiated by a secure external application programming interface (API) call, such as a Representation State Transfer (REST) API call using Hypertext Transfer Protocol Secure (HTTPS), while in FIG. 1B, the machine learning system 160 may be initiated by an internal API call, but where a common end API may handle both requests (e.g., the REST HTTPS API may provide an external wrapper for the internal API).

[0051] FIG. 1C shows another variation 104 of the example transaction processing system 100 of FIG. 1A. In this variation 104, the machine learning system 160 is communicatively coupled to local data storage devices 170. For example, data storage devices 170 may be on the same local network as machine learning server 150 or may comprise a local storage network accessible to the machine learning server 150. In this case, there are a plurality of local data storage devices 170-A to 170-N, where each data storage device stores partitioned ancillary data 172. The partitioned ancillary data 172 may comprise parameters for one or more machine learning models. In one case, the ancillary data 172 may comprise a state for machine learning models, where the state may relate to one or more specific entities such as a user or merchant. The partitioning of the ancillary data 172 may need to be applied to meet security requirements set by a third party, such as the payment processor, one or more banks and/or one or more merchants. In use, the machine learning system 160 accesses the ancillary data 172-A to 172-N via the plurality of local data storage devices 170-A to 170-N based on the input data 162. For example, the input data 162 may be received by way of an API request from a particular source and/or may comprise data that identifies that a particular partition is to be used to handle the API request. More details of different storage systems that may be applied to meet security requirements are set out in FIGS. 2A and 2B.

Example Data Storage Configurations

[0052] FIGS. 2A and 2B show two example data storage configurations 200 and 202 that may be used by an example machine learning system 210 for the processing of transaction data. The examples of FIGS. 2A and 2B are two non-limiting examples that show different options available for implementations, and particular configurations may be selected according to individual circumstances. The machine learning system 210 may comprise an implementation of the machine learning system 160 described in the previous examples of FIGS. 1A to 1C. The examples of FIGS. 2A and 2B allow for the processing of transaction data that is secured using heterogeneous cryptographic parameters, e.g. for the machine learning system 210 to securely process transaction data for heterogeneous entities. It will be appreciated that the configurations of FIGS. 2A and 2B may not be used if the machine learning system 160 is implemented for a single set of secure transaction and ancillary data, e.g. within an internal transaction processing system or as a hosted system for use by a single payment processor.

[0053] FIG. 2A shows a machine learning system 210 communicatively coupled to a data bus 220. The data bus 220 may comprise an internal data bus of the machine learning server 150 or may form part of storage area network. The data bus 220 communicatively couples the machine learning system 210 to a plurality of data storage devices 230, 232. The data storage devices 230, 232 may comprise any known data storage device such as magnetic hard disks and solid-state

devices. Although data storage devices 230, 232 are shown as different devices in FIG. 2A they may alternatively form different physical areas or portions of storage within a common data storage device. In FIG. 2A, the plurality of data storage devices 230, 232 store historical transaction data 240 and ancillary data 242. In FIG. 2A, a first set of data storage devices 230 store historical transaction data 240 and a second set of data storage devices 232 store ancillary data 242. Ancillary data 242 may comprise one or more of model parameters for a set of machine learning models (such as trained parameters for a neural network architecture and/or configuration parameters for a random forest model) and state data for those models. In one case, the different sets of historical transaction data 240-A to N and ancillary data 242-A to N are associated with different entities that securely and collectively use services provided by the machine learning system 210, e.g. these may represent data for different banks that need to be kept separate as part of the conditions of providing machine learning services to those entities.

[0054] FIG. 2B shows another way different sets of historical transaction data 240-A to N and ancillary data 242-A to N may be stored. In FIG. 2B the machine learning system 210 is communicatively coupled, via data transfer channel 250, to at least one data storage device 260. The data transfer channel 250 may comprise a local storage bus, local storage area network, and/or remote secure storage coupling (e.g., as overlaid over insecure networks such as the Internet). In FIG. 2B, a secure logical storage layer 270 is provided using the physical data storage device 260. The secure logical storage layer 270 may be a virtualized system that appears as separate physical storage devices to the machine learning system 210 while actually being implemented independently upon the at least one data storage device 260. The logical storage layer 270 may provide separate encrypted partitions 280 for data relating to groups of entities (e.g., relating to different issuing banks etc.) and the different sets of historical transaction data 240-A to N and ancillary data 242-A to N may be stored in the corresponding partitions 280-A to N. In certain cases, entities may be dynamically created as transactions are received for processing based on data stored by one or more of the server systems shown in FIGS. 1A to 1C.

Example Transaction Data

[0055] FIGS. 3A and 3B show examples of transaction data that may be processed by a machine learning system such as 160 or 210. FIG. 3A shows how transaction data may comprise a set of records 300, where each record has a timestamp and comprises a plurality of transaction fields. The records 300 may be time-ordered or may be strictly ordered in another manner. In some examples, transaction data may, optionally, be grouped and/or filtered based on the timestamp. For example, FIG. 3A shows a partition of transaction data into current transaction data 310 that is associated with a current transaction and “older” or historical transaction data 320 that is within

a predefined time range of the current transaction. The time range may be set as a hyperparameter of any machine learning system. Alternatively, the “older” or historical transaction data 320 may be set as a certain number of transactions. Mixtures of the two approaches are also possible.

[0056] FIG. 3B shows how transaction data 330 for a particular transaction may be stored in numeric form for processing by one or more machine learning models. For example, in FIG. 3B, transaction data has at least fields: transaction amount, timestamp (e.g., as a Unix epoch), transaction type (e.g., card payment or direct debit), product description or identifier (i.e., relating to items being purchased), merchant identifier, issuing bank identifier, a set of characters (e.g., Unicode characters within a field of predefined character length), country identifier etc. It should be noted that a wide variety of data types and formats may be received and pre-processed into appropriate numerical representations. In certain cases, originating transaction data, such as that generated by a client device and sent to merchant server 130 is pre-processed to convert alphanumeric data types to numeric data types for the application of the one or more machine learning models. Other fields present in the transaction data can include, but are not limited to, an account number (e.g., a credit card number), a location of where the transaction is occurring, and a manner (e.g., in person, over the phone, on a website) in which the transaction is executed.

Example Machine Learning System

[0057] FIG. 4 shows one example 400 of a machine learning system 402 that may be used to process transaction data. Machine learning system 402 may implement one or more of machine learning systems 160 and 210. The machine learning system 402 receives input data 410. The form of the input data 410 may depend on which machine learning model is being applied by the machine learning system 402. In a case where the machine learning system 402 is configured to perform fraud or anomaly detection in relation to a transaction, e.g. a transaction in progress as described above, the input data 410 may comprise transaction data such as 330 (i.e., data forming part of a data package for the transaction) as well as data derived from historical transaction data (such as 300 in FIG. 3A) and/or data derived from ancillary data (such as 148 in FIGS. 1A to 1C or 242 in FIGS. 2A and 2B). The ancillary data may comprise secondary data linked to one or more entities identified in the primary data associated with the transaction. For example, if transaction data for a transaction in progress identifies a user, merchant and one or more banks associated with the transaction (such as an issuing bank for the user and a merchant bank), such as via unique identifiers present in the transaction data, then the ancillary data may comprise data relating to these transaction entities. The ancillary data may also comprise data derived from records of activity, such as interaction logs and/or authentication records. In one case, the ancillary data is stored in one or more static data records and is retrieved from these records based on the

received transaction data. Additionally, or alternatively, the ancillary data may comprise machine learning model parameters that are retrieved based on the contents of the transaction data. For example, machine learning models may have parameters that are specific to one or more of the user, merchant and issuing bank, and these parameters may be retrieved based on which of these is identified in the transaction data. For example, one or more of users, merchants, and issuing banks may have corresponding embeddings, which may comprise retrievable or mappable tensor representations for said entities. For example, each user or merchant may have a tensor representation (e.g., a floating-point vector of size 128-1024) that may either be retrieved from a database or other data storage or may be generated by an embedding layer, e.g. based on a user or merchant index.

[0058] The input data 410 is received at an input data interface 412. The input data interface 412 may comprise an API interface, such as an internal or external API interface as described above. In one case, the payment processor server 140 as shown in FIGS. 1A to 1C makes a request to this interface, where the request payload contains the transaction data. The API interface may be defined to be agnostic as to the form of the transaction data or its source. The input data interface 412 is communicatively coupled to a machine learning model platform 414. In one case, a request made to the input data interface 412 triggers the execution of the machine learning model platform 414 using the transaction data supplied to the interface. The machine learning model platform 414 is configured as an execution environment for the application of one or more machine learning models to the input data 410. In one case, the machine learning model platform 414 is arranged as an execution wrapper for a plurality of different selectable machine learning models. For example, a machine learning model may be defined using a model definition language (e.g., similar to, or using, markup languages such as eXtended Markup Language - XML). Model definition languages may include (amongst others, independently or in combination), SQL, TensorFlow, Caffe, Thinc and PyTorch. In one case, the model definition language comprises computer program code that is executable to implement one or more of training and inference of a defined machine learning model. The machine learning models may, for example, comprise, amongst others, artificial neural network architectures, ensemble models, regression models, decision trees such as random forests, graph models, and Bayesian networks. The machine learning model platform 414 may define common (i.e., shared) input and output definitions such that different machine learning models are applied in a common (i.e., shared) manner.

[0059] In the present example, the machine learning model platform 414 is configured to provide at least a single scalar output 416. This may be normalised within a predefined range, such as 0 to 1. When normalised, the scalar output 416 may be seen as a probability that a transaction

associated with the input data 410 is fraudulent or anomalous. In this case, a value of “0” may represent a transaction that matches normal patterns of activity for one or more of a user, merchant and issuing bank, whereas a value of “1” may indicate that the transaction is fraudulent or anomalous, i.e. does not match expected patterns of activity (although those skilled in the art will be aware that the normalised range may differ, such as be inverted or within different bounds, and have the same functional effect). It should be noted that although a range of values may be defined as 0 to 1, output values may not be uniformly distributed within this range, for example, a value of “0.2” may be a common output for a “normal” event and a value of “0.8” may be seen as being over a threshold for a typical “anomalous” or fraudulent event. The machine learning model implemented by the machine learning platform 414 may thus implement a form of mapping between high-dimensionality input data (e.g., the transaction data and any retrieve ancillary data) and a single value output. In one case, for example, the machine learning platform 414 may be configured to receive input data for the machine learning model in a numeric format, wherein each defined machine learning model is configured to map input data defined in the same manner. The exact machine learning model that is applied by the machine learning model platform 414, and the parameters for that model, may be determined based on configuration data. The configuration data may be contained within, and/or identified using, the input data 410 and/or may be set based on one or more configuration files that are parsed by the machine learning platform 414.

[0060] In certain cases, the machine learning model platform 414 may provide additional outputs depending on the context. In certain implementations, the machine learning model platform 414 may be configured to return a “reason code” capturing a human-friendly explanation of a machine learning model’s output in terms of suspicious input attributes. For example, the machine learning model platform 414 may indicate which of one or more input elements or units within an input representation influenced the model output, e.g. a combination of an “amount” channel being above a learnt threshold and a set of “merchant” elements or units (such as an embedding or index) being outside a given cluster. In cases, where the machine learning model platform 414 implements a decision tree, these additional outputs may comprise a route through the decision tree or an aggregate feature importance based on an ensemble of trees. For neural network architectures, this may comprise layer output activations and/or layer filters with positive activations.

[0061] In FIG. 4, certain implementations may comprise an optional alert system 418 that receives the scalar output 416. In other implementations, the scalar output 416 may be passed directly to an output data interface 420 without post processing. In this latter case, the scalar output 416 may be packaged into a response to an original request to the input data interface 412. In both cases, output data 422 derived from the scalar output 416 is provided as an output of the machine

learning system 402. The output data 422 is returned to allow final processing of the transaction data. For example, the output data 422 may be returned to the payment processor server 140 and used as the basis of a decision to approve or decline the transaction. Depending on implementation requirements, in one case, the alert system 418 may process the scalar output 416 to return a binary value indicating whether the transaction should be approved or declined (e.g., “1” equals decline). In certain cases, a decision may be made by applying a threshold to the scalar output 416. This threshold may be context dependent. In certain cases, the alert system 418 and/or the output data interface 420 may also receive additional inputs, such as explanation data (e.g., the “reason code” discussed above) and/or the original input data. The output data interface 420 may generate an output data package for output data 422 that combines these inputs with the scalar output 416 (e.g., at least for logging and/or later review). Similar, an alert generated by the alert system 418 may include and/or be additionally based on the aforementioned additional inputs, e.g. in addition to the scalar output 416.

[0062] In a preferred implementation, the machine learning system 402 is used in an “online” mode to process a high volume of transactions within a narrowly defined time range. For example, in normal processing conditions the machine learning system 402 may process requests within 7-12ms and be able to manage 1000-2000 requests a second (these being median constraints from real-world operating conditions). However, the machine learning system 402 may also be used in an “offline” mode, e.g. by providing a selected historical transaction to the input data interface 412. In an offline mode, input data may be passed to the input data interfaces in batches (i.e., groups). The machine learning system 402 may also be able to implement machine learning models that provide a scalar output for an entity as well as, or instead of, a transaction. For example, the machine learning system 402 may receive a request associated with an identified user (e.g., a card or payment account holder) or an identified merchant and be arranged to provide a scalar output 416 indicating a likelihood that the user or merchant is fraudulent, malicious, or anomalous (i.e., a general threat or risk). For example, this may form part of a continuous or periodic monitoring process, or a one-off request (e.g., as part of an application for a service). The provision of a scalar output for a particular entity may be based on a set of transaction data up to and including a last approved transaction within a sequence of transaction data (e.g., transaction data for an entity similar to that should in FIG. 3A).

Example Transaction Process Flow

[0063] FIGS. 5A and 5B show two possible example transaction process flows 500 and 550. These process flows may take place in the context of the example transaction process systems 100, 102, 104 shown in FIGS. 1A to 1C as well as other systems. The process flows 500 and 550 are

provided as one example of a context in which a machine learning transaction processing system may be applied, however not all transaction process flows will necessarily follow the processes shown in FIGS. 5A and 5B and process flows may change between implementations, systems and over time. The example transaction process flows 500 and 550 reflect two possible cases: a first case represented by transaction process flow 500 where a transaction is approved, and a second case represented by transaction process flow 550 where a transaction is declined. Each transaction process flow 500, 550 involves the same set of five interacting systems and devices: a POS or user device 502, a merchant system 504, a payment processor (PP) system 506, a machine learning (ML) system 508 and an issuing bank system 510. The POS or user device 502 may comprise one of the client devices 110, the merchant system 504 may comprise the merchant server 130, the payment processor system 506 may comprise the payment processor server 140, and the machine learning system 508 may comprise an implementation of the machine learning system 160, 210 and/or 402. The issuing bank system 510 may comprise one or more server devices implementing transaction functions on behalf of an issuing bank. The five interacting systems and devices 502 to 510 may be communicatively coupled by one or more internal or external communication channels, such as networks 120. In certain cases, certain ones of these systems may be combined, e.g. an issuing bank may also act as a payment processor and so systems 506 and 510 may be implemented with a common system. In other cases, a similar process flow may be performed specifically for a merchant (e.g., without involving a payment processor or issuing bank). In this case, the machine learning system 508 may communicate directly with the merchant system 504. In these variations, a general functional transaction process flow may remain similar to that described below.

[0064] The transaction process flow in both FIGS. 5A and 5B comprises a number of common (i.e., shared) processes 512 to 528. At block 512, the POS or user device 502 initiates a transaction. For a POS device, this may comprise a cashier using a front-end device to attempt to take an electronic payment; for a user device 502 this may comprise a user making an online purchase (e.g., clicking “complete” within an online basket) using a credit or debit card, or an online payment account. At block 514, the payment details are received as electronic data by the merchant system 504. At block 516, the transaction is processed by the merchant system 504 and a request is made to the payment processor system 506 to authorise the payment. At block 518, the payment processor system 506 receives the request from the merchant system 504. The request may be made over a proprietary communications channel or as a secure request over public networks (e.g., an HTTPS request over the Internet). The payment processor system 506 then makes a request to the machine learning system 508 for a score or probability for use in processing the transaction.

Block 518 may additionally comprise retrieving ancillary data to combine with the transaction data that is sent as part of the request to the machine learning system 508. In other cases, the machine learning system 508 may have access to data storage devices that store ancillary data (e.g., similar to the configurations of FIGS. 2A and 2B) and so retrieve this data as part of internal operations (e.g., based on identifiers provided within the transaction data and/or as defined as part of an implemented machine learning model).

[0065] Block 520 shows a model initialisation operation that occurs prior to any requests from the payment processor system 506. For example, the model initialisation operation may comprise loading a defined machine learning model and parameters that instantiate the defined machine learning model. At block 522, the machine learning system 508 receives the request from the payment processor system 506 (e.g., via a data input interface such as 412 in FIG. 4). At block 522, the machine learning system 508 may perform any defined pre-processing prior to application of the machine learning model initialised at block 520. For example, in the case that the transaction data still retains character data, such as a merchant identified by a character string or a character transaction description, this may be converted into suitable structured numeric data (e.g., by converting string categorical data to an identifier via a look-up operation or other mapping, and/or by mapping characters or groups of characters to vector embeddings). Then at block 524 the machine learning system 506 applies the instantiated machine learning model, supplying the model with input data derived from the received request. This may comprise applying the machine learning model platform 414 as described with reference to FIG. 4. At block 526, a scalar output is generated by the instantiated machine learning model. This may be processed to determine an “approve” or “decline” binary decision at the machine learning system 508 or, in a preferred case, is returned to the payment processor system 506 as a response to the request made at block 518.

[0066] At block 528, the output of the machine learning system 508 is received by the payment processor system 506 and is used to approve or decline the transaction. FIG. 5A shows a process where the transaction is approved based on the output of the machine learning system 508; FIG. 5B shows a process where the transaction is declined based on the output of the machine learning system 508. In FIG. 5A, at block 528, the transaction is approved. Then at block 530, a request is made to the issuing bank system 532. At block 534, the issuing bank system 532 approves or declines the request. For example, the issuing bank system 532 may approve the request if an end user or card holder has sufficient funds and approval to cover the transaction cost. In certain cases, the issuing bank system 532 may apply a second level of security; however, this may not be required if the issuing bank relies on the anomaly detection performed by the payment processor using the machine learning system 508. At block 536, the authorisation from the issuing bank

system 510 is returned to the payment processor system 506, which in turn sends a response to the merchant system 504 at block 538, and the merchant system 504 in turn responds to the POS or user device 502 at block 540. If the issuing bank system 510 approves the transaction at block 534, then the transaction may be completed, and a positive response returned via the merchant system 504 to the POS or user device 502. The end user may experience this as an “authorised” message on screen of the POS or user device 502. The merchant system 504 may then complete the purchase (e.g., initiate internal processing to fulfil the purchase).

[0067] At a later point in time, one or more of the merchant system 504 and the machine learning system 508 may save data relating to the transaction, e.g. as part of transaction data 146, 240 or 300 in the previous examples. This is shown at dashed blocks 542 and 544. The transaction data may be saved along with one or more of the output of the machine learning system 508 (e.g., the scalar fraud or anomaly probability) and a final result of the transaction (e.g., whether it was approved or declined). The saved data may be stored for use as training data for the machine learning models implemented by the machine learning system 508. The saved data may also be accessed as part of future iterations of block 524, e.g. may form part of future ancillary data. In certain cases, a final result or outcome of the transaction may not be known at the time of the transaction. For example, a transaction may only be labelled as anomalous via later review by an analyst and/or automated system, or based on feedback from a user (e.g., when the user reports fraud or indicates that a payment card or account was compromised from a certain date). In these cases, ground truth labels for the purposes of training the machine learning system 508 may be collected over time following the transaction itself.

[0068] Turning now to the alternative process flow of FIG. 5B, in this case one or more of the machine learning system 508 and the payment processor system 506 declines the transaction based on the output of the machine learning system 508. For example, a transaction may be declined if the scalar output of the machine learning system 508 is above a retrieved threshold. At block 552, the payment processor system 506 issues a response to the merchant system 504, which is received at block 554. At block 554, the merchant system 504 undertakes steps to prevent the transaction from completing and returns an appropriate response to the POS or user device 502. This response is received at block 556 and an end user or customer may be informed that their payment has been declined, e.g. via a “Declined” message on screen. The end user or customer may be prompted to use a different payment method. Although not shown in FIG. 5B, in certain cases, the issuing bank system 510 may be informed that a transaction relating to a particular account holder has been declined. The issuing bank system 510 may be informed as part of the process shown in FIG. 5B or may be informed as part of a periodic (e.g., daily) update. Although, the transaction may not

become part of transaction data 146, 240 or 300 (as it was not approved), it may still be logged by at least the machine learning system 508 as indicated by block 544. For example, as for FIG. 5A, the transaction data may be saved along with the output of the machine learning system 508 (e.g., the scalar fraud or anomaly probability) and a final result of the transaction (e.g., that it was declined).

Example Data Store Structure

[0069] As explained above, example machine learning systems described herein store data in, and retrieve data from, one or more data stores in one or more data storage devices. Examples that will now be described relate to how such data stores are created, maintained and used, and the structure and content of such data stores. Such examples address or at least ameliorate specific database-related problems in the context of machine learning systems, and in particular machine learning systems that facilitate real-time transaction processing. More specifically, examples that will now be described relate to a “layered state” or “contextual layered state” data store configuration in which multiple different “layers” of state data (also referred to herein as “data”) are stored in the data store(s). Each layer comprises one or more state data sets (also referred to herein as “data sets” or “states”), which store state data. Such layered state configuration may be used in various different types of state implementation, i.e. implementations in which state is to be stored and retrieved, for example for use by the example machine learning systems described herein. In examples, the data stored in the data store comprises historical transaction data and/or ancillary data such as described above. However, the techniques described herein may be applied to other types of data.

[0070] Layered state allows an “effective state” to be constructed dynamically using state data from several different state data sources, such as different state data sets, without having to modify individual state data one-by-one. In effect, layered state amalgamates state data from multiple different state data sets, rather than requiring the state data to be in a single state data set. This can be especially, but not exclusively, effective where large amounts of state data exist in distinct data sets and where combining the separate data sets into a single data set in a single procedure would involve significant amounts of resources, such as time. Such data sets may be separate for various reasons, for example to allow different machine learning models to be tested, to provide a data maturation environment, etc. However, the state data in those separate data sets may nevertheless be related and the example machine learning systems described herein may benefit from using state data from the separate data sets, as opposed to being constrained to using a single data set, for example in terms of providing more accurate results. Some example implementations described herein may be used for database-based state types. Other example implementations are described

for data queues, where the data stored is ordered in time rather than being random-access as in a database. One such example implementation is on Apache Kafka (which may be referred to herein simply as “Kafka”). However, such example implementations may take other forms.

[0071] Layered state, as described herein, uses various concepts, including parents and overlays, which will now be described. In general terms, a parent data set of a child data set is a data set that is created before the child data set is created and to which the child data set is linked through parent metadata. In general terms, an overlay data set having an overlay relationship with a given data set is a data set to which the given data set is linked through overlay metadata. As will be explained in more detail below, such an overlay data set may not modify data in the given data set as such, but may modify data from one or more ancestors of the given data set. Such an overlay data set nevertheless has an overlay relationship with the given data set.

[0072] FIG. 6A shows a state graph (which may be referred to herein simply as a “graph”) 600 representing an example data store. The example data store comprises a first data set 611, which stores state data. The first data set 611 is identified using identification data in the form of a state id “state1”. Although the example data store represented in the graph 600 includes only one data set, as will become apparent below, the state id is used in example state implementations to separate out different sets of state in the same collection, i.e. different data sets in the same data store. In this example, “state1” is the “current” state id, which may also be called the “root” or “live” state id. When writing into the data store, all data is written into a specified live data set. In this example, the first data set 611 is a root data set of the data store, as indicated by item 620. In this example, all data is read from the first data set 611 since this is the only data set in the example data store.

[0073] FIG. 6B shows another state graph 602 representing another example data store. The data store represented by the graph 602 may correspond to the data store represented by the graph 600, but at a later point in time. The example graph 602 demonstrates parent state. In this example, the data store comprises first and second data sets 611, 612, the first data set 611 being a parent data set (or simply “a parent”) of the second data set 612, and the second data set 612 being a child data set (or simply “a child”) of the first data set 611. As such, the parent and child data sets 611, 612 have a hierarchical relationship in the state graph 602, as indicated at 630. In this example, the parent and child data sets 611, 612 are identified using identification data in the form of state ids “state1” and “state2” respectively. As before, new data is written into the root (child) data set 612 having state id “state2”. New data is not written into the parent data set 611 having state id “state1”, but (historic) data may be read from the parent data set 611. In examples, a new data set having a new state id starts off as an empty data set. In this specific example, the child data set 612 would

have been allocated a new state id ("state2"), and would have initially been empty, on its creation. As such, a new data set with a new state id will fill with (modified) data as new data gets written into the new data set. The parent link, indicated by item 630, comes into play when reading data from the data store. When performing a read operation, if a data item (which may also be referred to as a "key") being searched for is not found in the current data set, the data item is searched for in the parent data set. In examples, the parent link is via parent metadata stored in the data store. The parent metadata may indicate that one data set is a parent of another data set or that one data set is a child of another data set. Although, in this example, metadata is stored in the data store, metadata is not necessarily stored in the same data store as state data. The metadata may be stored in a different type of data store from one in which state data is stored. For example, state data may be stored in a database and metadata may be stored in files, in an Apache Kafka queue, etc. Alternatively, metadata and state data may be stored in the same type of data store as each other.

[0074] FIG. 6C shows another state graph 604 representing another example data store. The data store represented by the graph 604 may correspond to the data store represented by the graph 602, but at a later point in time. The example graph 604 demonstrates multi-parent state. In this example, the data store comprises a first data set 611, a second data set 612 which is a child of the first data set 611, and a third data set 613 which is a child of the second data set 612 (and hence a grandchild of the first data set 611). The first, second and third data sets 611, 612, 613 are therefore referred to as the grandparent, parent and child data sets 611, 612, 613 respectively. In this example, the grandparent, parent and child data sets 611, 612, 613 are identified using identification data in the form of state ids "state1", "state2" and "state3" respectively. In this example, the child data set 613 is a root of the data store, as indicated by item 620. Parent links 630, 631 exist between the grandparent and parent data sets 611, 612 and the parent and child data sets 612, 613 respectively. First parent metadata may define a link between the grandparent and parent data sets 611, 612 and second parent metadata may define a link between the parent and child data sets 612, 613. This is in contrast to metadata that would only define a link between the grandparent and child data sets 611, 613 which, as will be explained below, would not provide as comprehensive a search of the data set. When performing a read operation, if a data item being searched for is not found in the current data set (in this example the child data set 613), the parent data set 612 is searched for the data item and so on up the state graph to the grandparent data set 611. Effectively, this means that each data set, apart from the root, is a copy-on-write state snapshot at the point a new data set is branched off. The parent data set is no longer modified once a child data set is created from it, i.e. it becomes immutable when a child data set is created from it.

[0075] FIG. 6D shows another state graph 606 representing another example data store. The example graph 606 demonstrates multi-root state. In this example, the data store comprises a first data set 611, a second data set 612 which is a child of the first data set 611, a third data set 613 which is a child of the second data set 612 (and hence a grandchild of the first data set 611), and a fourth data set 614 which is also a child of the second data set 612 (and hence also a grandchild of the first data set 611). The first, second, third and fourth data sets 611, 612, 613, 614 are therefore referred to as the grandparent, parent, first child and second child data sets 611, 612, 613, 614 respectively. In this example, the grandparent, parent, first child and second child data sets 611, 612, 613, 614 are identified using identification data in the form of state ids “state1”, “state2”, “state3” and “state4” respectively. In this example, both child data sets 613, 614 are root data sets of the data store, as indicated by items 620, 621. As such, new data could be written to one or both of the child data sets 613, 614. Specifically, state id “state3” may be used to write data to the first child data set 613 and state id “state4” may be used to write data to the second child data set 614. Similarly, read access to the data store using state id “state3” would initially start at the first child data set 613, whereas read access to the data store using state id “state4” would initially start at the second child data set 614. Different clients may have, and use, different state ids, for example for data security purposes. Parent links 630, 631, 632 exist between the grandparent and parent data sets 611, 612, the parent and first child data sets 612, 613, and the parent and second child data sets 612, 614 respectively. As can be seen in this multi-root example, multiple child state ids, namely “state3” and “state4” can point at a single parent state id, namely “state2”.

[0076] FIG. 6E shows another state graph 608 representing another example data store. The example graph 608 corresponds to the example graph 606 shown in FIG. 6D, but with example state data values shown to demonstrate state divergence. In this example, the parent data set 612 stores data 640 comprising a single data element having an example value of “10” stored against a given data item (not shown in FIG. 6E), the first child data set 613 stores data 641 comprising a single data element having an example value of “15” stored against the same given data item, and the second child data set 614 stores data 642 comprising a single data element having an example value of “13” again stored against the same given data item. The two child data sets 613, 614 both initially start off identical, as they are both empty on creation and both share the same parent data set 612. However, they then slowly diverge over time as different data 641, 642 is written into each root 620, 621 and, hence, to each child data set 613, 614.

[0077] When a data store is searched in accordance with examples, the search is looking for a data item, and data associated with the data item is returned. The data item can take different forms, for example depending on implementation details. However, in one possible form, the data item is

a tuple based on a bankID (identifying a particular bank), an entityType (indicative of a particular type of entity), and an entityId (identifying a particular entity).

[0078] FIG. 7 shows another state graph 700 representing another example data store. The example graph 700 demonstrates an overlay and overlay state.

[0079] In this example, the state graph 700 comprises first and second data sets 711, 712. Data was stored in the second data set 712 in association with a data item “E1” (as indicated by item 740). The state graph 700 also comprises third and fourth data sets 713, 714, which are both children of the second data set 712. Data was stored in the third and fourth data sets 713, 714 in association with a data item “E2”, as indicated by items 741, 742. Unlike in FIG. 6E, values of the data stored are not depicted in FIG. 7. The values of the data stored in each of the third and fourth data sets 713, 714 in association with the data item “E2” may be different from each other. In this example, a fifth data set 715 was created as a child of the third child data set 713. Additionally, an overlay link was established between the fourth and fifth data sets 714, 715, as indicated by item 750. In this example, the overlay link was established through overlay metadata indicating the (overlay) link between the fourth and fifth data sets 714, 715. In this example, data was stored in the fifth data set 715 in association with a data item “E3” (as indicated by item 743). In this example, the overlay metadata was created when the fifth data set 715 was created. As such, the overlay link between the fourth and fifth data sets 714, 715 was established before any data could be written into the fifth data set 715. The overlay metadata could be created at a different time before any data is written into the fifth data set 715. This may, however, increase the risk of data being written into the fifth data set 715 before the overlay link with the fourth data set 714 is established.

[0080] As explained above, a data element represents the whole or part of given data stored in the data store, and data may comprise one or more data elements in this respect. In this example, overlay data element selection metadata indicates that overlay data elements “m1” and “m2” from the fourth data set 714 should be applied to data read from the data store, as indicated by item 750. Overlay data elements “m1” and “m2” may relate to different machine learning models and may comprise different machine learning model data. The overlay data element selection metadata indicates specific data elements that are to be used from the fourth data set 714 as an overlay data set. The fourth data set 714 can include data elements that are not applied via the overlay. For example, the fourth data set 714 could include an additional data element “m3” which is not being overlayed. This could be the case even if another data set in the data store, for example the third data set 713, also comprises a corresponding “m3” data element. In examples, specific data elements from the overlay data set that improve machine learning system operation may be

selected for application, whereas others that may not improve machine learning system operation may be inhibited from being applied.

[0081] As such, in accordance with examples, an overlay data set is used to update data from one data set with data from another data set, without actually copying any data from the one data set to the other data set. This is achieved, in this specific example, by creating metadata for the fifth data set 715 on creation of the fifth data set 715 to indicate that data is, potentially, to be read from the overlay data set, in this example the fourth data set 714, instead of from one or more parent ancestor data sets of the fifth data set 715, such as the first, second or third data sets 711, 712, 713.

[0082] An example sequence of operations to produce the example state graph 700 will now be described. Initially, the first data set 711 having state id “state1” is created empty and with no parent. The second data set 712 having state id “state2” is created as a child of the first data set 711. The second data set 712 is empty on its creation and the first data set 711 becomes immutable on creation of the second data set 712. Data item “E1” is written into the second data set 712, with the second data set 712 being the root state at that time. The third and fourth data sets 713, 714 having state ids “state3” and “state4” respectively are created as children of the second data set 712. The third and fourth data sets 713, 714 are empty on their creation and the second data set 712 becomes immutable on creation of the third and fourth data sets 713, 714. The data item “E2” is written into the third and fourth data sets 713, 714, with the third and fourth data sets 713, 714 both being roots at the time. The fifth data set 715 is created, and overlay metadata linking the fourth and fifth data sets 714, 715 is also created. Overlay data elements “m1” and “m2” have been selected for overlay application onto other data read from the data store. The third and fourth data sets 713, 714 become immutable on creation of the fifth data set 715 and the establishment of the overlay link between the fourth and fifth data sets 714, 715. The data item “E3” is written into the fifth data set 715, with the fifth data set 715 being the only root data set (and, hence, not being immutable) at that time.

[0083] The example data store represented by the state graph 700 may store the following data:

```
{ stateId: state2, key: E1, data: {[m1(E1)2, m2(E1)2, m3(E1)2]} };
```

```
{ stateId: state3, key: E2, data: {[m1(E2)3, m2(E2)3, m3(E2)3]} };
```

```
{ stateId: state4, key: E2, data: {[m1(E2)4, m2(E2)4, m3(E2)4]} };
```

```
{ stateId: state5, key: E3, data: {[m1(E3)5, m2(E3)5, m3(E3)5]} }.
```

[0084] In such example data, “mi(Ej)_k” denotes a data element comprising one more values relating to an i^{th} machine learning model, stored in association with key E_j , in the k^{th} data set.

[0085] The example data store represented by the state graph 700 may store the following metadata:

```

{ stateId: state1 };
{ stateId: state2, parent: state1 };
{ stateId: state3, parent: state2 };
{ stateId: state4, parent: state2 }; and
{ stateId: state5, parent: state3, overlay: { state4, [m1, m2] } }.

```

[0086] In this example, although the third data set 713 is a child of the second data set 712, it does not itself comprise the data item “E1” of the second data set 712. Also, even though an overlay link has been established between the fourth and fifth data sets 714, 715, the data associated with the data item “E2” in the fourth data set 714 has not actually been copied into the fifth data set 715. Instead, the metadata links the second, third, fourth, and fifth data sets 712, 713, 714, 715 such that, as will be explained below, data can be read from the amalgam of these data sets without data having been copied from one such data set to another.

[0087] As such the metadata is separate from, and provides information about, data stored in the data store. In examples, the metadata provides information at the data set level, rather than at the data or data element level. In examples, the metadata is modifiable independently of the other data. Small changes to the metadata, which can be quick to make, can result in significant changes to the effective data that can be read from the data store, without significant amounts of data having to be copied.

[0088] Example (simplified) read logic for reading from the state graph 700 will now be described. Example write logic will also be described below. Although read and write operations may be described together herein, a read does not necessarily imply that a write will happen immediately. The same data item may be read multiple times before a write is performed into the root data set.

[0089] In general terms, the read logic searches for a data item in the current root. If the data item is found in the current root, the data item is returned from the current root. If the data item is not found in the current root, the parent ancestor(s) of the current root are searched for the data item. Any overlays are also searched for the data item. If the data item is found in the parent ancestor(s) of the current root (and/or any overlays(s) of the parent ancestor(s)) and is not also in the overlay(s) linked with the current root, the data item is returned from the parent ancestor(s) of the current root (and/or any overlays(s) of the parent ancestor(s)). If, however, the data item is found in the parent ancestor(s) of the current root (and/or any overlays(s) of the parent ancestor(s)) and the overlay(s) linked with the current root, the overlay(s) take(s) precedence in respect of any data elements (of the data item) indicated by any overlay data element selection metadata of the overlay(s), and any such data elements are returned from the overlay(s) of the current root. If the

data item is not found in the parent ancestor(s) (and/or any overlays(s) of the parent ancestor(s)) of the current root but is in the overlay(s) linked with the current root, the data item is returned from the overlay(s) of the current root, again subject to overlay data element selection metadata.

[0090] An alternative data store configuration and read logic could apply an overlay to other state with at least one timestamp. The overlay could be applied with a first (“upper”) timestamp, “u”, indicating when the overlay was applied to the other state and could, optionally, be applied with a second (“lower”) timestamp, “t” indicating when the overlay was created. Once a data item “E” has been found in a data set, a timestamp “t_E” stored with the data item indicating when the data item was last modified could be examined to determine which overlays should be applied. For all overlays for which $t < t_E \leq u$, in chronological order based on upper timestamp, the data elements “m_n” from each overlay could be applied. Such data elements may be copied over the data read from somewhere in the parent state hierarchy. The resulting data could then be returned to a querying application. However, this may require a single timestamp, reference or point to determine whether or not to apply an overlay. This can become more difficult when the input data is partitioned, as there is no longer a single timestamp to use. However, this alternative read logic does enable overlays to be applied to a data set after the data set has been created, whereas the above-described read logic, in which overlay relationships are established when, or shortly after, a data set is created, may not. This approach may also be effective when the processing system cannot be paused to apply changes at a single, consistent point. This does, however, result in more state ids being created, which can increase the size of the containing data store.

[0091] To read the “E1” data item, the current data set, namely the fifth data set 715, is searched (also referred to as “queried”). “E1” is not found in the fifth data set 715. The third data set 713 is identified via parent metadata and is queried as the parent data set of the fifth data set 715. “E1” is not found in the third data set 713. The second data set 712 is identified via parent metadata and is queried as the parent data set of the third data set 713, and “E1” is found. The fourth data set 714 is identified via overlay metadata and is queried in view of the overlay relationship between the fourth and fifth data sets 714, 715. “E1” is not found in the fourth data set 714. In this example, data associated with “E1” is returned, with the returned data being [m1(E1)₂, m2(E1)₂, m3(E1)₂]. The returned data may be written directly to the root data set, namely the fifth data set 715, in association with the data item “E1”. Subsequent read requests for “E1” would retrieve the returned data from the fifth data set 715, as opposed to retrieving the data from the second data set 712 again. Retrieving the data from the fifth data set 715 reduces the number of read operations compared to retrieving the data from the second data set 712.

[0092] To read the “E2” data item, the current data set, namely the fifth data set 715, is searched. “E2” is not found in the fifth data set 715. The third data set 713 is identified via parent metadata and is queried as the parent data set of the fifth data set 715. “E2” is found in the third data set 713. The fourth data set 714 is identified via overlay metadata (linking the fourth and fifth data sets 714, 715), and the data elements “m1(E2)₄” and “m2(E2)₄” are read from the fourth data set 714. In this example, the data element “m3(E2)₄” is not read from the fourth data set 714 in view of the overlay data element selection metadata. In this example, “m1(E2)₄” and “m2(E2)₄” as read from the fourth data set 714 replace data elements “m1(E2)₃” and “m2(E2)₃” read from the third data set 713, but data element “m3(E2)₃” as read from the third data set 713 is not replaced by data from the fourth data set 714. The data associated with “E2” as modified, namely [m1(E2)₄, m2(E2)₄, m3(E2)₃] is returned. The “E2” data read from the fourth data set 714, namely “m1(E2)₄” and “m2(E2)₄”, or modified such data, i.e. “m1(E2)_{4_mod}” and “m2(E2)_{4_mod}”, is then written directly to the root data set, namely the fifth data set 715, along with “m3(E2)₃” or modified such data, i.e. “m3(E2)_{3_mod}”.

[0093] The data read from the third and/or fourth data set 713, 714 may be modified by the machine learning system before being written back to the root data set 715. In general, a write will be performed after data read from a data store has been modified by the machine learning system. As such, the data written to the root data set may be the data as read or a modified version of the data as read.

[0094] When an overlay relationship is established between one data set and another data set, a new state id is not created for the overlay data set. Instead, the metadata of the new data set, which is created in conjunction with the establishment of the overlay relationship, redirects reads to the overlay data set, where applicable. From that point, the overlay data set effectively becomes an irrevocable part of the content of the root data set and any child of the root data set.

[0095] To read the “E3” data item, the current data set, namely the fifth data set 715, is searched and “E3” is found. In examples, since “E3” is found in the current data set, the search does not proceed to any ancestors of the fifth data set 715 or to the fourth data set 714. Data associated with “E3”, namely [m1(E3)₅, m2(E3)₅, m3(E3)₅], is returned from the fifth data set 715 without modification.

[0096] FIGS. 8A and 8B show how data and metadata may be stored in a data store in accordance with examples.

[0097] In this example, entity 800, which may be referred to as a “record”, represents example information stored in a data store. In this example, entity 800 includes a stateId field 801 having a value “state1”, indicating that the information depicted in entity 800 is associated with a data set

having been allocated state id “state1”. In this example, entity 800 also includes a data item field 802 having a value “E1”, indicating that entity 800 stores data relating to data item “E1”. In this example, entity 800 also includes a data field 803 having data elements [m1, m2, m3]. In this example, “m1”, “m2”, and “m3” each comprise one or more respective values associated with respective machine learning models.

[0098] In this example, entity 810 represents example metadata. In this example, entity 810 includes a stateId field 811 having a value “state4”, indicating that the metadata depicted in entity 810 is associated with a data set having been allocated state id “state4”. In this example, entity 810 also includes a parent field 812 having a value “state2”, indicating that the data set having state id “state2” is a parent of the data set having state id “state3”. In this example, entity 810 also includes an overlay field 813 having a value “state 3, [m1, m2]”, indicating (i) that the data set having state id “state4” is associated with the data set having state id “state3” via an overlay relationship, and (ii) that only data elements “m1” and “m2” should be applied from the overlay data set (having state id “state3”).

[0099] FIG. 9 shows another state graph 900 representing another example data set. The example graph 900 demonstrates unrelated state. In this example, the data store comprises first, second, third, fourth, fifth, sixth and seventh data sets 911, 912, 913, 914, 915, 916, 917. In this example, the data sets 911 to 917 are identified using identification data in the form of state ids “state1” to “state7” respectively. In this example, the third and fourth data sets 913, 914 were created as children of the second data set 912. Subsequently, the fifth data set 915 was created as a child of the third data set 913, along with overlay metadata indicating an overlay link between the fourth and fifth data sets 914, 915. In this example, the sixth data set 916 has an overlay relationship with the seventh data set 917, but is not a child of any other data sets. The seventh data set 917 was created as a child of the fifth data set 915. In the configuration shown in FIG. 9, both the fourth and sixth overlay data sets 914, 916 are immutable.

[0100] Although an overlay data set can share a parent data set with another data set, that is not necessarily the case. An unrelated data set may be used as an overlay. This can be used, for example, to insert new matured model data into an existing system without having to copy all the data from the new matured model into the existing system. In the example graph 900, the sixth data set 916 corresponds to (initially) unrelated state.

[0101] The relevance of parent state relationships to the definition of state can therefore be seen. State relationship metadata for a state id cannot be changed without drastically changing the data accessible in that state, as the effective data associated with that state is the result of an amalgam of state from parent data sets and overlays.

[0102] FIG. 10 shows another state graph 1000 representing another example data set. The example data set represented by the state graph 1000 is similar to the example data set represented by the state graph 900. However, in this example, the root data set 1017 has two overlay links, namely with the fourth and sixth data sets 1014, 1016. Additionally, in this example the sixth data set 1016 is a child of the third data set 1013. In this example, the fifth data set 1015 was created at the same time as the sixth data set 1016, and the seventh data set 1017 was created at the same time as overlay metadata indicating the overlay links with the fourth and sixth data sets 1014, 1016.

[0103] By way of a specific example, suppose: (i) the fourth data set 1014 comprises a data item “E1” having data elements “m14” and “m24” and overlay data element selection metadata indicating that only “m1” should be applied from the fourth data set 1014, (ii) the sixth data set 1016 also comprises a data item “E1” having data elements “m16” and “m26” and overlay data element selection metadata indicating that only “m2” should be applied from the sixth data set 1016, and (iii) the first, second and/or third data set 1011, 1012, 1013 also comprises a data item “E1” having data elements “m1”, “m2” and “m3”. In this specific example, it is assumed that the third data set 1013 comprises a data item “E1” having data elements “m13”, “m23” and “m33” and that the fifth and seventh data sets 1015, 1017 do not comprise a data item “E1”. The read logic would: (i) initially find that data item “E1” is not in the seventh data set 1017, (ii) search for data item “E1” in the parent ancestors of the seventh data set 1017, (iii) find the data item “E1” in the third data set 1013, (iv) identify the overlay link between the fourth and seventh data sets 1014, 1017 based on overlay metadata, search for the data item “E1” in the fourth data set 1014, and identify that only data element “m1” should be applied from the fourth data set 1014, (v) identify the overlay link between the sixth and seventh data sets 1016, 1017 based on overlay metadata, search for the data item “E1” in the sixth data set 1016, and identify that only data element “m2” should be applied from the sixth data set 1016, and (vi) return, for the data item “E1”, the data element “m14” from the fourth data set 1014, the data element “m26” from the sixth data set 1016, and the data element “m33” from the third data set 1013. Data elements “m14_{mod}”, “m26_{mod}”, and “m33_{mod}”, namely “m14”, “m26”, and “m33” as modified, may be written into the seventh data set 1017 in association with the data item “E1”. As explained above, such writing may not occur immediately after such reading.

[0104] In terms of performance, each read into a separate data set and overlay is a separate read operation to the data store. The parent and overlay logic is relatively complex and may be difficult to perform as a single query on the data store. As such, depending on how many data sets are involved, several read operations may be performed before given data is returned. However, all

writes occur into the root data set. As such, the next time that data is read, it will be read directly from the root data set, as a single read. However, when a new data set is created (as an empty data set), all the reads will be from the parent with at least two read operations. In some implementations, the new data set may become populated with the most frequently modified data relatively quickly. Additionally, the number of read operations decreases over time and can tend towards the previous operation rate. Nevertheless, determining that a particular data item is not present in the effective data set (the amalgam of the applicable data sets in the graph) involves reading from all related data sets until the earliest data set has been read. This increases read operations whenever a new data set is created from the current root. To mitigate this, a clean-up operation can be configured to run to consolidate old data sets together to reduce the number of data sets in the hierarchy. This can cap the number of read operations required. The clean-up operation may be run periodically, or otherwise.

[0105] FIG. 11 shows another state graph 1100 representing an example data store. The example graph 1100 demonstrates eligible states.

[0106] The layered state clean-up operation addresses both a potential performance degradation and data store size, caused by an ever-growing number of layered states. As indicated, performance potentially degrades as more state layers are created and linked to one another. Finding a data item can take an increasing amount of time as an increasing number of parent data sets might need to be searched. This is especially true if a data item is not present in any of the data sets layers, as this forces the lookup to visit the entire data set history. The number of queries grows linearly with the number of data sets. This problem increases as data sets are promoted, creating more overlays. When a data item is not found in the main branch of the state tree, it could still be found in an overlay, so overlays are checked as well.

[0107] To manage such potential performance degradation, older data sets can be cleaned up as part of the layered state clean-up operation to reduce the number of queries to be performed when looking for a specific data item. This operation also saves space in the data store. In examples, unlayered data sets are not cleaned up. This does not appreciably affect performance where unlayered data sets are small. In terms of the layered state clean-up operation procedure, data set layers that are eligible for clean-up are collapsed into a single data set layer, referred to herein as the “target state”, thus preventing extra reads. The target state is the lowest common ancestor of the data set layers pointed to by a root with a parent that has been locked (in other words that has become immutable) before a configurable cut-off. Data sets eligible for clean-up are: (i) the strict ancestors of the target data set, (ii) overlays associated with the strict ancestors of the target data set, and (iii) overlays associated with the target data set.

[0108] In the example graph 1100, the second data set 1112 is the lowest common ancestor of the data sets pointed to by a root data set (the fourth and fifth data sets 1114, 1115) with a parent that has been locked before the cut-off 1170 (data set 1111). As such, data set 1112 is the target state. Necessarily, it is not eligible for clean-up itself. Data sets 1113, 1114 and 1115 are descendants of the target data set, data set 1112. Therefore, they are also not eligible for clean-up. Data set 1111 is a strict ancestor of the target data set, data set 1112. Therefore, data set 1111 is eligible for clean-up.

[0109] In this example, the clean-up procedure starts with identifying the target data set. The data sets eligible for clean-up are identified. All of the relevant data from the data sets eligible for clean-up is merged into the target data set. Overlays are applied when relevant so that no data is lost. The parent reference of the target data set is removed, making the target data set the new farthest ancestor of the data sets pointed to by roots. Also, any overlays associated with the target data set are removed. The data sets eligible for clean-up are completely removed.

[0110] FIG. 12A shows another state graph 1200 representing another example data store. The example graph 1200 demonstrates layered state clean-up. The example graph 1200 represents the data store before clean-up is performed.

[0111] In this example, the data store comprises a great-grandparent data set 1211, a grandparent data set 1212, a parent data set 1213, and a child data set 1214. In this example with four serial state layers, data sets 1211 and 1212 are the data sets eligible for clean-up. Data set 1213 is the target data set. The data of data sets 1211, 1212 and 1213 are merged and are written to data set 1213. The order in which the different steps are executed provides consistency of the data throughout the clean-up procedure. To merge data into the target data set, the effective data in the target is determined by reading the data from the target data set as a “virtual root”. For each data item present in the data sets eligible for clean-up and the target data set, the data item is read from the virtual root as per the layered state reading logic described above. The resulting data is then written to the target data set. The target data set is not necessarily locked. In such cases, the target data set is also the live root data set. In this special case, the data the target data set contains is still subject to modifications. In such examples, the clean-up operation yields to concurrent writes. A concurrent write guarantees that the most up-to-date data is written to a target live root data set. In such instances, the merge step for that specific data item is made obsolete and is aborted. Because writes are atomic, there is no risk of getting corrupted data, which would result from a mix of the data written by the clean-up and an external source. In general, if a data item to be moved already exists in a target data set, then the data item in the target data set is not overwritten. This would still happen if the target data set is locked. More specifically, a write is made atomically,

conditional on the data item not existing in the target data set already. A cut-off line is indicated by broken line 1270.

[0112] FIG. 12B shows another state graph 1202 representing another example data store. The example graph 1202 shown in FIG. 12B corresponds to the example graph 1200 shown in FIG. 12A after the clean-up operation has been performed.

[0113] FIG. 13A shows another state graph 1300 representing another example data store. The example graph 1300 demonstrates layered state clean-up using a specific example.

[0114] An example sequence of operations to produce the state graph 1300 before the clean-up is as follows. The first data set 1311 is created empty and with no parent state. The data items “E1” and “E2” are written into the first data set 1311. Second and third data sets 1312, 1313 are created (empty) as children of the first data set 1311, and the first data set 1311 becomes immutable. The data items “E1” and “E4” are written into the third data set 1313. The fourth data set 1314 and overlay metadata linking the third and fourth data sets 1313, 1314 are created, with overlay data elements “m1” and “m2” selected. The second and third data sets 1312, 1313 become immutable. The data items “E2” and “E3” are written into the fourth data set 1314. The fifth data set 1315 is created (empty) as a child of the fourth data set 1314. The fourth data set 1314 becomes immutable.

[0115] In this example, the target data set is the fourth data set 1314, and the data sets eligible for clean-up are the first, second and third data sets 1311, 1312, 1313. For each data item present in the first, second, third and/or fourth data sets 1311, 1312, 1313, 1314, the effective data is read from a virtual root pointing to the fourth data set 1314, and is then written to the fourth data set 1314.

[0116] The simplified clean-up logic is as follows. For merging “E1”, “E1” is not found in the fourth data set 1314. “E1” is found in the first data set 1311, and in the third data set 1313. “E1” as found in the first data set 1311 is used as a base. Data relating to the overlay data elements “m1” and “m2” found in the third data set 1313 is copied over data elements “m1” and “m2” from the first data set 1311, if they exist, and the result is written into the fourth data set 1314. For merging “E2”, “E2” is found in the fourth data set 1314. “E2” as found in the fourth data set 1314 is not modified because it is already in the fourth data set 1314, even though “E2” is also in the first data set 1311. This would be the case even if “E2” were in the second and/or third data set 1312, 1313 as well as or instead of the first data set 1311. For merging “E3”, “E3” is also already in the fourth data set 1314. For merging “E4”, “E4” is not found in the fourth data set 1314. “E4” is also not found in the first or second data sets 1311, 1312. “E4” is found in the third data set 1313. The effective data only contains the data relating to “m1” and “m2”. The effective data is written to the

fourth data set 1314. Once the data items have been merged to the fourth data set 1314, the first, second and third data sets 1311, 1312, 1313 are deleted.

[0117] FIG. 13B shows another state graph 1302 representing another example data store. The example graph 1302 shown in FIG. 13B corresponds to the example graph 1300 shown in FIG. 13A after the clean-up operation has been performed. In effect, the “E1” data in the first data set 1311 has been read, the overlay data elements “m1” and “m2” found in the third data set 1313 have been applied, and the result has been written into the fourth data set 1314. The “E2” and “E3” data in the fourth data set 1314 are unmodified. The “E4” data in the third data set 1313 relating to “m1” and “m2” has been moved to the fourth data set 1314.

[0118] The frequency at which the layered state clean-up operation runs may be set, for example by an operator. The length of history of state snapshots to keep before cleaning them up may also be set, for example by an operator.

[0119] While data is being merged, which can be a relatively long time (e.g. several hours), additional changes may still be made to the data store, as long as any modifications occur after the cut-off for the active clean-up operation. In examples, the layered state clean-up operation always yields to concurrent writes, as they are necessarily providing the most up-to-date data.

[0120] In some examples, data items and/or state metadata store logical and/or wall-clock timestamps indicating when additions and/or modifications were made. Timestamp data can enable identification of when data items were modified. Such information may be used to remove data items that have not been modified recently, i.e. if they were modified more than a threshold time ago. However, this does not affect layering logic. In examples, and in relation to clean-up, a timestamp written back into a consolidated data item may be the maximum timestamp of the timestamps of all the data items merged into the consolidated data item. This may be the case even if the latest data item is from an overlay data set that only has a subset of the final data in the consolidated data item.

[0121] In relation to data deletion, data may be deleted from a layered state. If the data only exists in the current root, then the data could simply be removed from the data store. However, if the data exists in a parent state and/or an overlay state, the data cannot be removed directly. This is because all states apart from the root(s) are immutable. Instead, a marker is inserted into the current root state indicating the data to be regarded as deleted. Then, when subsequent reads find the deletion marker, they stop reading at that point and return no data. However, the old data, which should be regarded as deleted, still exists in the parent and/or overlay states. As such, if a new root is created from a snapshot state, the data that is to be regarded as deleted will still be present in the snapshot state. The deletion marker additionally acts as an indicator not to insert another marker in future

roots should the data item in question be deleted again, provided it is the latest. In examples, the deletion marker is only not inserted if the data item in question only exists in overlays for non-promoted data elements, e.g. models. An overlay can, in effect, undo a deletion made in relation to another data set by overlaying data (from the overlay) over data where the deletion marker was in the other data set.

[0122] When cleaning up data items with a deletion marker, a deletion marker effectively makes the clean-up operation ignore all state for that data item before that point. Suppose, in a first example, a data store includes, for a clean-up operation, a target data set, a parent data set of the target data set and a grandparent data set of the target data set. Suppose also that the parent data set comprises a data item “E1” with a deletion marker and that the grandparent data set also comprises data item “E1” but with a data element “m1” having a value “30” (which may be denoted “E1: {m1: 30}”). After clean-up, the target data set would not have any data for “E1” at all in view of the deletion marker, and the parent and grandparent data sets would be removed from the data store, including any deletion markers in them. Suppose, in a second example, that the data store of the first example included an additional data set between the parent data set and the target data set and that the additional data set also comprised data item “E1” but with data element “m1” having a value “10” (which may be denoted “E1: {m1: 10}”). In the second example, the clean-up operation would put “E1: {m1: 10}” in the target data set, copied from the additional data set, then the additional, parent and grandparent data sets would be removed from the data store.

[0123] It is also possible for an overlay to undo a deletion marker. For example, suppose that a data store comprises a parent data set with a data item “E1” having data elements “m1” with a value “15” and “m2” with a value “20” (which may be denoted “E1: {m1: 15, m2: 20}”). Suppose also that a first child data set of the parent data set was created, was the root of the data store, and comprises a data item “E1” having a deletion marker (which may be denoted “E1: delete”). Suppose further that a second child data set of the parent data set was created and comprises a data item “E1” having a data element “m1” with a value “20” and a data element “m2” with a value “25” (which may be denoted “E1: {m1: 20, m2: 25}”). Suppose additionally that a grandchild data set was created as a child of the first child data set and that an overlay link was established between the second child data set and the grandchild data set with overlay data element selection metadata indicating that only data element “m1” is to be applied from the second child data set. In such an example, and assuming the grandchild data set does not comprise “E1”, the read logic would find the deletion marker for “E1” in the first child data set, and would stop searching ancestors of the first child data set in response. The read logic would then check overlays, identify the overlay link

with the second child data set, and would read “m1” from the second child data set. As such, the returned data would be “E1: {m1: 20}”. The same logic may be used in clean-up operations as described herein. Such logic would also apply if the deletion marker were in the second child data set rather than the first child data set, removing data elements from the data item before returning them.

[0124] FIG. 14 shows another example of a state graph 1400 representing a data store. The example graph 1400 demonstrates recursive overlays. In this example, second and third data sets 1412, 1413 are both children data sets of a first data set 1411. Fourth and fifth data sets 1414, 1415 are children of the second data set 1412. A sixth data set 1416 is created as a child data set of the fifth data set 1415, and an overlay link is established between the fourth and sixth data sets 1414, 1416 on creation of the sixth data set 1416. A seventh data set 1417 is created as a child data set of the third data set 1413, and an overlay link is established between the sixth and seventh data sets 1416, 1417 on creation of the seventh data set 1417. If the sixth data set 1416 is searched for a data item, the first, second, fourth and/or fifth data sets 1411, 1412, 1414, 1415 may be searched in a similar manner to how searching is performed in respect of the seventh data set 1417.

[0125] FIG. 15 shows another example of a state graph 1500 representing a data store. The example data store represented by the state graph 1500 is similar to the example data store represented by the state graph 1400. However, in this example, the root data set 1516 has two overlay links; with the fourth and fifth data sets 1514, 1515.

[0126] Examples described above relate to data stores in the form of databases. Examples will now be described which relate to data queues. In examples, such data queues store feature vectors by machine learning models. However, such data queues may store other data to be output by machine learning models.

[0127] As part of the processing of events, models can output data onto one or more model-specific data queues that can then be read. As part of defining a queue, the model also defines the minimum duration data should be kept in the queue for. In a live system, the data queues may be stored in a persisted message queue. In this specific example, the data queues are stored in Apache Kafka. However, data may be stored in another manner in other examples. More specifically, there are other ways to implement state layers, for example using message queues. The data retention period maps onto the queue retention period for the Apache Kafka topic storing the data. Topics are created as necessary by a controller when a model is uploaded into the engine. However, they are not deleted when a model is deleted. This is because the model may be recreated soon after and would expect the data to still be there. If a model is deleted permanently, then all the data in the topics will time out over time, leaving empty topics in Apache Kafka taking up minimal disk

space. In terms of layered queue state, the data queues follow the state layering principles described above. This is achieved on Apache Kafka by maintaining separate topics for each active queue in each state, and stitching the queues together into one logical queue to read the full contents. With reference, specifically, to Apache Kafka, data cannot be deleted from Apache Kafka queues, so there is no data clean-up operation to be run as such. Instead, queues are removed once they have timed out. A separate operation may be provided to remove any unused empty queues, and generally to remove any unused and unlinked states from a data store or metadata.

[0128] FIG. 16A shows an example queue 1600 in a single state, represented as a single Apache Kafka topic: “queue1-state1: [topic1: 0]”.

[0129] FIG. 16B shows an example queue 1602 in which, when a child state is created, a new topic is created for that state, and metadata for the queue as a whole is updated to link the queues together. New messages are put onto the end of the most recent topic: “queue1-state2: [topic1: 0, topic2: 23]”. As part of the queue metadata, each topic has an “offset offset” specifying the offset of the topic contents in the overall queue. This offset is added to the offset of the message within each topic to get the message offset within the queue as a whole.

[0130] FIG. 16C shows another example queue 1604. Similar to layered state, there can be multiple child topics based on the same parent topic: “queue1-state2: [topic1: 0, topic2: 23]” and “queue1-state3: [topic1: 0, topic3: 23]”.

[0131] FIG. 16D shows another example queue 1606. When a state gets overlayed, the root pointer changes to point to the overlayed topic directly, and the obsolete topic is deleted: “queue1-state2: [topic1: 0, topic3: 23]”.

[0132] FIG. 16E shows another example queue 1608. There are special cases in all of these state operations so that, when the current topic is empty, the “parent” link of the new topic points directly at the parent of the current topic, bypassing the empty topic entirely. For example: “queue1-state3: [topic1: 0, topic3: 23]”. This is to ensure there are no topics in the middle of a queue which have no data in them.

[0133] Data in a data queue has a different state implementation from other state types. As such, the clean-up operation operates differently on this state type. Owing to the guarantee on data retention the engine provides to model queues, and the way data is stored (Apache Kafka rather than a database), the state is only cleaned up when a model queue is found to be empty. When that is the case, the metadata corresponding to the model queue made obsolete can be deleted, as well as the corresponding topic itself. Message queues become empty when all the messages they contain eventually time out. In a specific example, a model data queue is eligible for clean-up if it fulfils the following conditions: (i) all of its ancestors are empty, itself included, (ii) it is an ancestor

of a queue pointed to by a root, and (ii) it is not directly pointed to by a root. Other conditions may, however, be used in other examples.

[0134] FIG. 17A shows an example model queue 1700 eligible for clean-up. “topic1” is empty; all the items it contained have timed out. “topic1” has no ancestors, so all of its ancestors can be considered to be empty. “topic1” is a strict ancestor of “topic8”, pointed to by a root. Therefore, “topic1” is eligible for clean-up. “topic2” is also empty. All of the ancestors of “topic2” (namely “topic1”) are also empty. “topic2” is a strict ancestor of “topic8”. Therefore, “topic2” is eligible for clean-up. “topic3” is also empty. All of the ancestors of “topic3” (namely “topic1”) are also empty. “topic3” is also an ancestor of a topic (namely “topic5”) pointed to by a root. Therefore, “topic3” is eligible for clean-up. “topic4” is empty. However, “topic4” is not an ancestor of any topic pointed to by a root. Therefore, “topic4” is not eligible for clean-up. “topic5” is not empty. Moreover, “topic5” is pointed to by a root. Therefore, “topic5” is not eligible for clean-up. “topic6” is empty, all of its ancestors (namely “topic1” and “topic2”) are also empty, and “topic6” is a strict ancestor of “topic8”, pointed to by a root. Therefore, “topic6” is eligible for clean-up. “topic7” is not empty. Therefore, “topic7” is not eligible for clean-up. “topic8” is empty. However, one of the ancestors of “topic8” (namely “topic7”) is not empty and “topic8” is directly pointed to by a root. Therefore, “topic8” is not eligible for clean-up. “topic9” is empty. “topic9” has no ancestors, so ancestors of “topic9” are considered to be empty. However, “topic9” is directly pointed to by a root. Therefore, “topic9” is not eligible for clean-up.

[0135] FIG. 17B shows an example model queue graph 1702 after the clean-up procedure has been completed. When a topic is completely blank (in other words, when it never contained any messages), such as “topic4” from the above example, its children will bypass it to directly reference its parent (if any). In this specific example, and in accordance with the example conditions set out above, such blank topics are not cleaned up. However, such topics should remain relatively rare. More generally, in this specific example, topics not reachable from the root are not cleaned up. However, in other examples, empty topics, such as “topic4” can be cleaned up.

[0136] As such, examples described herein provide the capability to have multiple independent sets of data stored in a data store (such as a database), with a hierarchical relationship.

[0137] A specific set of the data is accessed using a state id. In some examples, metadata is stored in the data store with information on the relationship between different state ids. However, as mentioned above, metadata is not necessarily stored in the same data store as state data. Relationships can be a parent relationship, or an overlay relationship. A relationship can also be an ancestor relationship where one state is a strict ancestor of another state (e.g. parent, grandparent, great-grandparent, etc). A single state id can have zero or one parents, and, if it has a

parent, zero to “N” overlays. In examples, the parent and any overlays of a state id are fixed at creation time, i.e. at the time of creation of that state id. A state id can have multiple child state ids. Similarly, a state id can have an overlay link with multiple other state ids. An external client accessing the data store does so using a specific root state id.

[0138] When saving data for a data item “E”, with data elements “m1”, “m2”, “m3”, the data is simply written into the root state, potentially overwriting anything else stored in the root state for “E”, if it exists.

[0139] When reading, the client is looking for a specific data item, “E”. If “E” is present in the root state, data stored in the root state in association with “E” is returned to the client. If not, then the parent state is searched, and so on up the parent hierarchy until “E” is found, or the state at the end (i.e. top) of the tree is searched, and “E” is determined not to be present.

[0140] Additionally, while searching, applicable overlays may be searched. Data indicating the set of data elements that should be applied from the overlay is also stored.

[0141] As an example, assume there is a state id “state1”, a child state id “state2” as a current root, and an overlay state id “ol1” applied to “state2” with overlay data element selection data “m1”, “m2”. When searching for a data item “X”, if “X” is found in “state2”, then it is returned as-is. If “X” is not found in “state2”, “state1” is then searched. If “X” is found in “state1”, then because the current root state is “state2”, “m1” and “m2” from the overlay state are copied over any data read for “X” in “state1”, as it is 'read-through' the parent link to “state2”, and then returned to the client.

[0142] In examples, each overlay has unique overlay data element selection data. In other words there is only one overlay applying a specific data element to a specific state id. However, in other examples, different overlays have different priorities and may have non-unique overlay data element selection data. In such other examples, the highest-priority overlay may be used in preference to a lower-priority overlay in respect of one or more data elements. In some examples in which priorities are used, overlay data element selection data is not used. Overlays are applied to data items as they are read-through each parent state id link, from where the item was found, all the way to the current root, copying the data for all data elements over that read from the relevant parent state id. In examples, the parent and overlays of a state id are fixed when the state id is first defined.

[0143] A clean-up operation can be run concurrently to client reads and writes. This collapses old states together, copying forward any data that does not exist in the target state, e.g. a root state. This uses an atomic insertion operation such that any data saved concurrently in the root state by a running client is not overwritten.

[0144] Techniques described herein may be implemented on top of Mongo, SQL, and Apache Kafka. Different implementation details may be involved.

[0145] The entire contents of a data set may be identified by listing all of the data items present in all data sets in a data store, or at least all data items in the hierarchy from a particular root, and then querying for each data item through root data set to see its effective data from that root.

[0146] FIG. 18 shows another state graph 1800 representing another example data store. Reference will be made to the example state graph 1800 to summarise various features and effects of features described herein.

[0147] Various measures (for example, methods, apparatuses, systems, computer programs, and computer-readable media) are provided to search for data in a data store. The data store comprises a first data set 1811, a second data set 1812, and a third data set 1813. A data item is searched for in the first, second and/or second data sets 1811, 1812, 1813. In response to said searching finding the data item, data stored in association with the data item is returned. If the data item is found in the first data set 1811, said returning comprises returning first data stored in association with the data item in the first data set 1811. If the data item is not found in the first set 1811 and is found in the second and third data sets 1812, 1813, said returning comprises, in preference to returning second data stored in association with the data item in the second data set 1812, returning third data stored in association with the data item in the third data set 1813. The second data set 1812 is identified based on parent metadata. The parent metadata is indicative of the second data set 1812 being a parent of the first data set 1811. The third data set 1813 is identified based on overlay metadata. The overlay metadata is indicative of the third data set 1813 being an overlay of the first data set 1811.

[0148] Various measures (for example, methods, apparatuses, systems, computer programs, and computer-readable media) are also provided to store data in a data store. The data store comprises a first data set 1811, a second data set 1812, and a third data set 1813. The first data set 1811 is created as a child of the second data set 1812. Overlay metadata is created and stored. The overlay metadata is indicative of an overlay relationship between the first data set 1811 and the third data set 1813. The first data set 1811 and the overlay metadata are both created before data is written into the first data set 1811.

[0149] The terms “first”, “second” and “third” as used in relation to the first, second and third data sets 1811, 1812, 1813 do not imply a temporal order in which the data sets are created.

[0150] Such measures use an overlay and overlay metadata to facilitate searching for or retrieving a data item in a data store, where the data item may be in one or multiple of the first, second, and third data sets 1811, 1812, 1813. The first, second, and third data sets 1811, 1812, 1813 are separate

data sets in that their data has not been merged into a single data set, but they are related via the parent and overlay metadata. The amalgam of the first, second, and third data sets 1811, 1812, 1813 in this manner enables effective data to be identified and returned from one or more of the first, second and third data sets 1811, 1812, 1813 without the first, second, and third data sets 1811, 1812, 1813 having been merged. The parent and overlay metadata are therefore data structures controlling how a search operation is performed in the data store.

[0151] Such measures may improve query response times by enabling responses to be constructed using data from different (potentially very large) data sets, as compared to query responses having to be delayed while different (potentially very large) data sets are merged.

[0152] Such measures execute structured queries based on the data item to be searched for. Such queries may comprise or may otherwise identify the data item to be searched for. The data item may uniquely identify data to be retrieved from the data store.

[0153] Such measures may, particularly but not exclusively in combination with other features described herein, provide improved computational efficiency in terms of reducing memory read operations, provide improved security, provide improved scalability, provide improved throughput and/or provide improved data retention.

[0154] The first, second, and third data sets 1811, 1812, 1813 may have different technical properties. For example, the first data set 1811 may be a root data set, and the second and third data sets 1812, 1813 may be non-root data sets. In such examples, data may be written to and read from the first data set 1811, while data may only be read from the second and third data sets 1812, 1813.

[0155] In accordance with examples, live data may be stored separately from other data, such as test data. The test data may be applied to the live data as an overlay, without the test data needing to be copied over the live data in a single copy operation. In the case of a real-time transaction processing, transaction processing may continue based on the live data while test data can be created in parallel and applied where appropriate, without copying-related downtime. The live and test data can be created and written to independently until, for example, the test data is to be applied as an overlay.

[0156] In accordance with examples, copy-on-write historical snapshots may be provided. A child state can be created from an old state. Assuming they are available, the events can be replayed with a different machine learning configuration. The effect the changes would have had, compared to what happened, can then be seen. Those changes can be merged into a live system without having to re-mature all the data again.

[0157] In accordance with examples, separation of data can be provided on a temporal basis. For example, the second (parent) data set 1812 may comprise historical data, whereas the first (child) data set 1811 may comprise newer, for example current, data.

[0158] In some examples, the data item being searched for is associated with an entity in relation to which real-time anomaly detection is being performed. As explained above, measures provided herein can reduce query response times, which may be especially effective in the context of real-time anomaly detection.

[0159] In some examples, the data in the data store comprises parameter data and/or state data for a machine learning model.

[0160] In some examples, if the third data is returned, the third data is caused to be stored in association with the data item in the first data set 1811. The search for the data may initially start in the first data set 1811. In such cases, in future searches for the data item, the third data may be retrieved from the first data set 1811. This can reduce searching and response times compared to searching both the first and third data sets 1811, 1813 for the data item, as an additional read operation to read data from the third data set 1813 may not be needed.

[0161] In some examples, the third data is caused to be stored in association with the data item in the first data set 1811. This can be especially effective in examples in which the data comprises machine learning model data, since machine learning model data can be used in a live data set and machine learning model data can be tested in a test data set.

[0162] In some examples, the third data set 1813 comprises further data. The further data may not be caused to be stored in the first data set 1811 concurrently with the third data item being stored in association with the data item in the first data set 1811. In contrast to techniques in which all data in the third data set 1813 would be concurrently copied into the first data set 1811, such examples enable data to be stored in the data set 1811 on an as-needed basis.

[0163] In some such examples, the further data is stored in association with the data item in the third set 1813. Overlay data element selection metadata may be indicative that the third data is to be stored in association with the data item in the first data set 1811 and that the further data is not to be stored in association with the data item in the first data set 1811. Based on the overlay data element selection data, the further data may be inhibited from being stored in association with the data item in the first data set 1811. Such examples provide a reliable and precisely defined indication of the effective data that is to be constructed in the first data set 1811 at a data element level.

[0164] In other such examples, the further data is stored in association with a further data item in the third data set 1813. Again, in contrast to techniques in which all data in the third data set 1813

would be concurrently copied into the first data set 1811, such examples enable data to be stored in the data set 1811 on an as-needed basis at a data item level.

[0165] In some examples, the first data set 1811 potentially comprises data not comprised in the second data set 1812. As such, the second (parent) data set 1812 may comprise historical data, whereas the first (child) data set 1811 may comprise newer, for example current, data not in the historical data. This differs from other data store configurations in which a child data set comprises a subset of the data of a parent data set.

[0166] In some examples, in response to said searching not finding the data item in the first data set 1811, the data item is searched for in the second and third data sets 1812, 1813. As such, data items can still be searched for even if they are not in the first data set 1811.

[0167] In some examples, the data store further comprises a fourth data set 1814, wherein the fourth data set 1814 has a parent relationship with the second data set 1812. The data item may be searched for in the fourth data set 1814. As such, additional layers of state can be provided in the data store.

[0168] In some examples, such as shown in FIG. 18, the fourth data set 1814 is also a parent of the third data set 1813. As such multiple different potentially diverging data sets can be derived from and share common parent data.

[0169] In other examples, which are not shown in FIG. 18, the fourth data set 1814 is not a parent of the third data set 1813. As such, an independent data set can be used, for example for data maturation. The fifth data set 1815 shown in FIG. 18 is an example of a data set having an overlay relationship with the first data set 1811 and which does not share a common parent ancestor with the first data set 1811.

[0170] In some examples, the fourth data set 1814 became immutable on creation of the second data set 1812 and/or the third data set 1813. As such, a historic record of data can be preserved.

[0171] In some examples, the third data set 1813 became immutable on establishment of the overlay relationship with the first data set 1811. Additionally, new data writes may be limited to a live data set.

[0172] In some examples, a clean-up operation is performed on a given data set in the data store. The clean-up operation may comprise causing data from another data set to be written into the given data set. The clean-up operation may comprise causing metadata indicative of a link between the other data set and the given data set to be removed. The clean-up operation may comprise causing the other data set to be removed from the data store. Such examples can reduce search query times by reducing the number of read operations to return a positive or negative search result.

[0173] In some examples, the first data set 1811 was empty on creation of the first data set 1811. In some examples, the second data set 1812 was empty on creation of the second data set 1812. In some examples, the third data set 1813 was empty on creation of the third data set 1813. In examples, in which the first, second and/or third data set 1811, 1812, 1813 is a child of another data set, the first, second and/or third data set 1811, 1812, 1813 can be created quickly as an empty set, with the amalgam of the first, second and/or third data set 1811, 1812, 1813 and its parent data set still effectively providing access to the (potentially historic) data of the parent data set.

[0174] In some examples, the data store further comprises a data set 1816 that has an overlay relationship with the third data set 1813. The data item may be searched for in the data set 1816 that has the overlay relationship with the third data set 1813. As such, recursive overlays may be searched. In examples, overlay metadata indicative of an overlay relationship between the data set 1816 and the third data set 1813 is created when the third data set 1813 is created. However, more generally, in examples, the third data set 1813 and the overlay metadata are created before data is written into the third data set 1813.

[0175] In some examples, if the data item is not found in the first data set 1811, if the data item is not found in the third data set 1813, and if the data item is found in the second data set 1812, said returning comprises returning the second data stored in association with the data item in the second data set 1812. As such, processing may be performed where a parent comprises a data item that was not present in a live data set or in an overlay data set, for example where the data item only exists in the parent and does not exist in the first data set 1811.

[0176] In some examples, a deletion marker associated with given data stored in the data store is identified. Returning of the given data may be inhibited on the basis of the deletion marker. As such, an effective technique for data being regarded as deleted can be provided, while such data can still be retained with the data store for other purposes.

[0177] In some examples, (i) the first data set 1811 and (ii) the overlay metadata indicative of the overlay relationship between the first data set 1811 and the third data set 1813 are both created before data is written into the first data set 1811. As such, a relatively flexible mechanism whereby data is written to the first data set 1811 after the overlay relationship is established is provided.

[0178] In some examples, the overlay metadata indicative of the overlay relationship between the first data set 1811 and the third data set 1813 is created when the first data set 1811 is created. This reduces the likelihood of data being written into the first data set 1811 before the overlay relationship is established.

[0179] In some examples, the data store comprises another data set 1815 that has an overlay relationship with the first data set 1811, and the method comprises searching for the data item in

the other data set 1815 that has the overlay relationship with the first data set 1811. This can increase the effective data that can be read from the data store while providing efficient reading through the use of overlay metadata.

[0180] Various measures (for example, methods, apparatuses, systems, computer programs, and computer-readable media) are also provided to search for machine learning model data in a database. The database comprises live state 1811, a parent state 1812 and an overlay state 1813. A key is searched for in the live, parent and/or overlay state 1811, 1812, 1813. The key is associated with one or more entities in relation to which real-time anomaly detection is being performed. In response to said searching finding the key, machine learning model data stored in association with the key is returned. The machine learning model data comprises parameter data and/or state data for a machine learning model to be used to perform said real-time anomaly detection. If the key is found in the live state 1811, said returning comprises returning first machine learning model data stored in association with the key in the live state 1811. If the key is not found in the live state 1811 and is found in the parent and overlay states 1812, 1813, said returning comprises returning, in preference to returning second machine learning model data stored in association with the key in the parent state 1812, returning third machine learning model data stored in association with the key in the overlay state 1813. The parent state 1812 is identified based on parent metadata. The parent metadata is indicative of the parent state 1812 being a parent state of the live state 1811. The overlay state 1813 is identified based on overlay metadata. The overlay metadata is indicative of the overlay relationship between the live state 1811 and the overlay state 1813.

[0181] Various measures (for example, methods, apparatuses, systems, computer programs, and computer-readable media) are also provided to search for data in a data store, the data store comprising: a first data set (such as data set 1811), a second data set (such as data set 1812) and a third data set (such as data set 1813). A data item is searched for in the first, second and/or third data sets. In response to said searching finding the data item, data stored in association with the data item is returned. If the data item is found in the first data set, said returning comprises returning first data stored in association with the data item in the first data set. If the data item is not found in the first data set and is found in the second and third data sets, said returning comprises, in preference to returning second data stored in association with the data item in the second data set, returning third data stored in association with the data item in the third data set. The second and third data sets are identified based on metadata. Said returning is based on the third data set having a higher priority than the second data set. Priority information associated with the second and/or third data set may be stored in metadata. The second data set may be a parent of the first data set. The third data set may have an overlay relationship with the first data set. Data

element selection metadata may be indicative of which data element(s) are to be used from the second and/or third data sets.

[0182] In examples described above, a root data set is searched for a data item and, (only) if the data item is not found in the root data set, another data set is then identified using metadata and searched for the data item. In such examples, the other data set is only read if it is to be searched. In other examples, the root data set and the other data set(s) are read before it has been determined whether the other data set(s) need(s) to be searched for the data item.

[0183] Certain examples described herein may be implemented via instructions that are stored within a computer-readable storage medium, such as a non-transitory computer-readable medium. The computer readable medium may comprise one or more of a rotating magnetic disk, a rotating optical disk, a flash random access memory (RAM) chip, and other mechanically moving or solid-state storage media. In use, the instructions are executed by one or more of processors to cause said processor to perform the operations described above. The above embodiments, variations and examples are to be understood as illustrative. Further embodiments, variations and examples are envisaged. Although certain components of each example have been separately described, it is to be understood that functionality described with reference to one example may be suitably implemented in another example, and that certain components may be omitted depending on the implementation. It is to be understood that any feature described in relation to any one example may be used alone, or in combination with other features described, and may also be used in combination with one or more features of any other of the examples, or any combination of any other of the examples. For example, features described with respect to the system components may also be adapted to be performed as part of the described methods. Furthermore, equivalents and modifications not described above may also be employed without departing from the scope of the invention, which is defined in the accompanying claims.

CLAIMS

1. A method of searching for data in a data store, the data store comprising:
a first data set;
a second data set; and
a third data set,
wherein the method comprises:
searching for a data item in the first, second and/or third data sets; and
in response to said searching finding the data item, returning data stored in association with the data item,
wherein, if the data item is found in the first data set, said returning comprises returning first data stored in association with the data item in the first data set,
wherein, if the data item is not found in the first data set and is found in the second and third data sets, said returning comprises, in preference to returning second data stored in association with the data item in the second data set, returning third data stored in association with the data item in the third data set,
wherein the method comprises identifying the second data set based on parent metadata, the parent metadata being indicative of the second data set being a parent of the first data set, and
wherein the method comprises identifying the third data set based on overlay metadata, the overlay metadata being indicative of an overlay relationship between the first data set and the third data set.
2. A method according to claim 1, wherein the data item is associated with an entity in relation to which real-time anomaly detection is being performed.
3. A method according to claim 1 or 2, wherein the data comprises parameter data and/or state data for a machine learning model.
4. A method according to any of claims 1 to 3, wherein, if the third data is returned, the method comprises causing the third data to be stored in association with the data item in the first data set.

5. A method according to any of claims 1 to 4, wherein the third data set comprises further data and wherein the further data is not caused to be stored in the first data set concurrently with the third data item being stored in association with the data item in the first data set.
6. A method according to claim 5, wherein the further data is stored in association with the data item in the third set, wherein overlay data element selection metadata is indicative that the third data is to be stored in association with the data item in the first data set and that the further data is not to be stored in association with the data item in the first data set, and wherein the method comprises inhibiting, based on the overlay data element selection metadata, the further data from being stored in association with the data item in the first data set.
7. A method according to claim 5, wherein the further data is stored in association with a further data item in the third data set.
8. A method according to claim any of claims 1 to 7, wherein the first data set potentially comprises data not comprised in the second data set.
9. A method according to any of claims 1 to 8, wherein, in response to said searching not finding the data item in the first data set, the method comprises searching for the data item in the second and third data sets.
10. A method according to claim 9, wherein the data store further comprises a fourth data set, wherein the fourth data set has a parent relationship with the second data set, and wherein the method comprises searching for the data item in the fourth data set.
11. A method according to claim 10, wherein the fourth data set is also a parent of the third data set.
12. A method according to claim 11, wherein the data store further comprises:
a data set that has an overlay relationship with the third data set, and wherein the method comprises searching for the data item in the data set that has the overlay relationship with the third data set.

13. A method according to claim 10, wherein the fourth data set is not a parent of the third data set.
14. A method according to any of claims 9 to 13, wherein the fourth data set became immutable on creation of the second and/or third data set.
15. A method according to any of claims 1 to 14, wherein the third data set became immutable on establishment of the overlay relationship with the first data set.
16. A method according to any of claims 1 to 15, comprising performing a clean-up operation on a given data set in the data store, wherein the clean-up operation comprises:
 - causing data from another data set to be written into the given data set;
 - causing metadata indicative of a link between the other data set and the given data set to be removed; and
 - causing the other data set to be removed from the data store.
17. A method according to any of claims 1 to 16, wherein the first data set was empty on creation of the first data set, wherein the second data set was empty on creation of the second data set and/or wherein the third data set was empty on creation of the third data set.
18. A method according to any of claims 1 to 17, wherein, if the data item is not found in the first data set, if the data item is not found in the third data set, and if the data item is found in the second data set, said returning comprises returning the second data stored in association with the data item in the second data set.
19. A method according to any of claims 1 to 18, comprising:
 - identifying a deletion marker associated with given data stored in the data store; and
 - inhibiting returning of the given data on the basis of the deletion marker.
20. A method according to any of claims 1 to 19, wherein:
 - the first data set and the overlay metadata indicative of the overlay relationship between the first data set and the third data set are both created before data is written into the first data set;
 - and/or

the overlay metadata indicative of the overlay relationship between the first data set and the third data set is created when the first data set is created.

21. A method according to any of claims 1 to 20, wherein the data store comprises another data set that has an overlay relationship with the first data set, and wherein the method comprises searching for the data item in the other data set that has the overlay relationship with the first data set.

22. A method of storing data in a data store, the data store comprising:
a first data set;
a second data set; and
a third data set,
wherein the method comprises:
creating the first data set as a child of the second data set; and
creating and storing overlay metadata, the overlay metadata being indicative of an overlay relationship between the first data set and the third data set,
wherein the first data set and the overlay metadata are both created before data is written into the first data set.

23. A method of searching for machine learning model data in a database, the database comprising:
live state;
a parent state; and
an overlay state,
wherein the method comprises:
searching for a key in the live, parent and/or overlay state, the key being associated with one or more entities in relation to which real-time anomaly detection is being performed; and
in response to said searching finding the key, returning machine learning model data stored in association with the key, the machine learning model data comprising parameter data and/or state data for a machine learning model to be used to perform said real-time anomaly detection,
wherein, if the key is found in the live state, said returning comprises returning first machine learning model data stored in association with the key in the live state,

wherein, if the key is not found in the live state and is found in the parent and overlay states, said returning comprises, in preference to returning second machine learning model data stored in association with the key in the parent state, returning third machine learning model data stored in association with the key in the overlay state,

wherein the method comprises identifying the parent state based on parent metadata, the parent metadata being indicative of the parent state being a parent state of the live state, and

wherein the method comprises identifying the overlay state based on overlay metadata, the overlay metadata being indicative of an overlay relationship between the live state and the overlay state.

24. A method of searching for data in a data store, the data store comprising:

a first data set;

a second data set; and

a third data set,

wherein the method comprises:

searching for a data item in the first, second and/or third data sets; and

in response to said searching finding the data item, returning data stored in association with the data item,

wherein, if the data item is found in the first data set, said returning comprises returning first data stored in association with the data item in the first data set,

wherein, if the data item is not found in the first data set and is found in the second and third data sets, said returning comprises, in preference to returning second data stored in association with the data item in the second data set, returning third data stored in association with the data item in the third data set,

wherein the method comprises identifying the second and third data sets based on metadata, and

wherein said returning is based on the third data set having a higher priority than the second data set.

25. A system configured to perform a method according to any of claims 1 to 24.