

MANAGING LINEAGE INFORMATION

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority to U.S. Application Serial No. 62/026,228, filed on July 18, 2014.

5

BACKGROUND

This description relates to managing parameter sets.

In data processing systems it is often desirable for certain types of users to have access to reports of a lineage of data as it passes through the systems. Very generally, among a number of uses, such “data lineage” reports can be used to reduce risk, verify compliance obligations, streamline business processes, and safeguard data. It is important that data lineage reports are both correct and complete.

10

SUMMARY

In one aspect, in general, managing sets of parameter values the lineage information that reflects relationships among instances of generic computer programs that were instantiated using those sets of parameter values, enables generation of more accurate and complete data lineage reports.

15

In another aspect, in general, a method for managing lineage information includes: receiving lineage information representing one or more lineage relationships among two or more data processing programs and two or more logical datasets; receiving one or more runtime artifacts, each runtime artifact including information related to a previous execution of a data processing program of the two or more data processing programs; and analyzing the one or more runtime artifacts and the lineage information to determine one or more candidate modifications to the lineage information.

20

Aspects can include one or more of the following features.

25

The one or more candidate modifications include a candidate modification that adds a new indirect lineage relationship between a data processing program of the two or more data processing programs and a logical dataset of the two or more logical datasets.

30

The one or more candidate modifications include a first candidate modification that adds a new direct lineage relationship between a data processing

program of the two or more data processing programs and a logical dataset of the two or more logical datasets.

Analyzing the runtime artifacts and the lineage information includes analyzing logs of previous executions of the two or more data processing programs to determine
5 physical datasets read from or written to by the two or more data processing programs.

Analyzing the runtime artifacts and the lineage information further includes identifying two distinct logical datasets of the two or more logical datasets that are represented in the lineage information and are associated with the same physical
10 dataset.

The first candidate modification includes creation of the new lineage relationship between the two distinct logical datasets.

The first candidate modification includes creation of the new lineage relationship including merging the two distinct logical datasets into a new combined
15 logical dataset.

Each data processing program of the two or more data processing programs is an instance of a generic data processing program instantiated according to a set of one or more parameter values.

Analyzing the one or more runtime artifacts and the lineage information
20 includes: analyzing one or more logs of previous executions of a first data processing program of the two or more data processing programs to determine a first parameter set used in a first instantiation of the first data processing program according to a first set of one or more parameter values, selecting at least some parameters from the first parameter set, and determining that the first instantiation of the first data processing
25 program is not represented in the lineage information based on a generic version of the first data processing program and the at least some parameters.

Selecting at least some parameters from the first parameter set includes selecting parameters based on information received from a user.

Selecting at least some parameters from the first parameter set includes
30 selecting parameters based on one or more predefined rules.

A first rule of the one or more predefined rules specifies that parameters with parameter values in the form of a date are excluded from the selected parameters.

A first rule of the one or more predefined rules specifies that a parameter with a parameter value that is transformed in the logic of a generic data processing program is included in the selected parameters.

The one or more candidate modifications to the lineage information includes a
5 first candidate modification that adds a new lineage relationship between the first data processing program of the two or more data processing programs and a logical dataset of the two or more logical datasets.

In another aspect, in general, software for managing lineage information is stored in a non-transitory form on a computer-readable medium, the software
10 including instructions for causing a computing system to: receive lineage information representing one or more lineage relationships among two or more data processing programs and two or more logical datasets; receive one or more runtime artifacts, each runtime artifact including information related to a previous execution of a data processing program of the two or more data processing programs; and analyze the one
15 or more runtime artifacts and the lineage information to determine one or more candidate modifications to the lineage information.

In another aspect, in general, a computing system for managing lineage information includes: an input device or port configured to receive lineage information representing one or more lineage relationships among two or more data
20 processing programs and two or more logical datasets and one or more runtime artifacts, each runtime artifact including information related to a previous execution of a data processing program of the two or more data processing programs; and at least one processor configured to analyze the one or more runtime artifacts and the lineage information to determine one or more candidate modifications to the lineage
25 information.

In another aspect, in general, a computing system for managing lineage information including: means for receiving lineage information representing one or more lineage relationships among two or more data processing programs and two or more logical datasets and one or more runtime artifacts, each runtime artifact
30 including information related to a previous execution of a data processing program of the two or more data processing programs; and means for analyzing the one or more runtime artifacts and the lineage information to determine one or more candidate modifications to the lineage information.

Aspects can include one or more of the following advantages.

By discovering parameter sets using the approaches described herein and using the discovered parameter sets to augment an existing set of parameter sets, data lineage reports generated using the augmented set of existing parameter sets more accurately represent the true data lineage of a data processing system. In particular, portions of the data lineage for the data processing system that would have been previously overlooked are included in the data lineage report.

In some examples, the results of the parameter set discovery approaches can also be used to augment the log entries of executions of instances of the computer program (i.e., augmenting the log entries with information about discovered parameter sets). The augmented log entry can advantageously be used to verify that logical connections between computer programs and/or datasets correspond to physical connections. The results of this verification ensure that the data lineage presented to a user shows the correct lineage relationships among computer programs and their inputs and outputs.

Other features and advantages of the invention will become apparent from the following description, and from the claims.

DESCRIPTION OF DRAWINGS

FIG. 1 is a block diagram of a system for discovery of parameter sets.

FIG. 2 is a dataflow graph including sub-graphs and their associated parameter sets.

FIG. 3 is a run time configuration of the dataflow graph of FIG. 2.

FIG. 4 is a static analysis configuration of the dataflow graph of FIG. 2.

FIG. 5 is a flow chart of a method for discovery of parameter sets.

FIG. 6 is a first half of an exemplary operation of the method for discovery of parameter sets.

FIG. 7 is a second half of an exemplary operation of the method for discovery of parameter sets.

FIG. 8 is an example dataflow graph including a first sub-graph and second sub-graph.

FIG. 9 shows the dataflow graph of FIG. 8 with its logical datasets resolved to physical datasets.

FIG. 10 shows a data lineage report for the dataflow graph of FIG. 8.

FIG. 11 is an example dataflow graph including a first sub-graph and a second sub-graph and having duplicate logical datasets.

FIG. 12 shows the example dataflow graph of FIG. 11 with its logical datasets resolved to physical datasets.

5 FIG. 13 shows a data lineage report, including a data lineage break, for the dataflow graph of FIG. 11.

FIG. 14 shows a first technique for mitigating the effects of a data lineage break in a data lineage report.

FIG. 15 shows a second technique for mitigating the effects of a data lineage break in a data lineage report.

FIG. 16 shows a third technique for mitigating the effects of a data lineage break in a data lineage report.

FIG. 17 shows a fourth technique for mitigating the effects of a data lineage break in a data lineage report.

15 FIG. 18 shows a fifth technique for mitigating the effects of a data lineage break in a data lineage report.

FIG. 19 shows a sixth technique for mitigating the effects of a data lineage break in a data lineage report.

DESCRIPTION

20 FIG. 1 shows an example of a data processing system 100 in which the parameter set discovery techniques described herein can be used. The system includes a development environment 118 which is, in some implementations, a system for developing applications as dataflow graphs 117 that include vertices (representing data processing components or datasets) connected by directed links (representing flows of work elements, i.e., data) between the vertices. For example, such an environment is described in more detail in U.S. Publication No. 2007/0011668, titled "Managing Parameters for Graph-Based Applications," incorporated herein by reference. A system for executing such graph-based computations is described in U.S. Patent 5,966,072, titled "EXECUTING
30 COMPUTATIONS EXPRESSED AS GRAPHS," incorporated herein by reference. The dataflow graphs 117 made in accordance with this system provide methods for getting information into and out of individual processes represented by graph components, for moving information between the processes, and for defining a

running order for the processes. This system includes algorithms that choose interprocess communication methods from any available methods (for example, communication paths according to the links of the graph can use TCP/IP or UNIX domain sockets, or use shared memory to pass data between the processes). The
5 dataflow graphs 117 created by the developer 120 using the development environment 118 can be stored in a data storage system 116 accessible to the development environment 118 for subsequent use by other modules of the system 100.

An execution environment 104 includes a parameter resolution module 106 and an execution module 112. The execution environment 104 may be hosted, for
10 example, on one or more general-purpose computers under the control of a suitable operating system, such as a version of the UNIX operating system. For example, the execution environment 104 can include a multiple-node parallel computing environment including a configuration of computer systems using multiple central processing units (CPUs) or processor cores, either local (e.g., multiprocessor systems
15 such as symmetric multi-processing (SMP) computers), or locally distributed (e.g., multiple processors coupled as clusters or massively parallel processing (MPP) systems, or remote, or remotely distributed (e.g., multiple processors coupled via a local area network (LAN) and/or wide-area network (WAN)), or any combination thereof.

20 The parameter resolution module 106 receives a specification of the dataflow graphs 117 from the data storage system 116 and resolves parameters for the dataflow graphs 117 (as is described in greater detail below) to prepare the dataflow graph(s) 117 for execution by the execution module 112. The execution module 112 receives the prepared dataflow graphs 117 from the parameter resolution module 106 and uses
25 them to process data from a data source 102 and generate output data 114. The output data 114 may be stored back in the data source 102 or in the data storage system 116 accessible to the execution environment 104, or otherwise used. In general, the data source 102 may include one or more sources of data such as storage devices or connections to online data streams, each of which may store or provide data in any of
30 a variety of formats (e.g., database tables, spreadsheet files, flat text files, or a native format used by a mainframe).

Storage devices providing the data source 102 may be local to the execution environment 104, for example, being stored on a storage medium connected to a computer hosting the execution environment 104 (e.g., hard drive 108), or may be

remote to the execution environment 104, for example, being hosted on a remote system (e.g., mainframe 110) in communication with a computer hosting the execution environment 104, over a remote connection (e.g., provided by a cloud computing infrastructure).

5 The system 100 also includes a metadata environment module 119, which is accessible to enterprise users 121 (e.g., data architects or business users). The metadata environment module 119 includes a data lineage module 115, which processes the dataflow graphs 117 (or metadata that characterizes them and the input and output datasets they reference) to generate a data lineage for the dataflow graphs
10 117. The enterprise user 121 can view the data lineage for reasons such as verification of the dataflow graphs 117 and compliance checking. Data lineage information about a particular data item (e.g., a dataset, or field within a dataset) is based on dependency relationships that arise from processing that is performed by a data processing system, and the term “data lineage” as used herein, generally refers to
15 the set that includes other related data items and the processing entities that consume or generate those data items. A data lineage report (also called a data lineage diagram) may include a graphical representation of the data lineage in the form of a graph with nodes representing the data items and processing entities, and links representing the dependency relationships among them. Some systems capable of
20 generating and displaying data lineage reports are able to automatically present an end-to-end data lineage from ultimate sources of data at an upstream end to the final data produced at a downstream end. Nodes on a path upstream from a particular data item are sometimes called “dependencies” for that data item, and nodes on a path downstream from a particular data item are sometimes called “impacts” for that data
25 item. While “data lineage” is sometimes used to refer only to the upstream dependencies, as used herein, “data lineage” may refer to either or both upstream dependencies and/or downstream impacts as appropriate to the specific context.

1 Dataflow Graph Overview

Referring to FIG. 2, an example of a dataflow graph 217 generated using the
30 development environment 118 of FIG. 1 includes a first sub-graph 202 named gather.mp and a second sub-graph 204 named process.mp.

The first sub-graph 202 receives a first logical dataset DS1 206 and a second logical dataset DS2 208 as input, processes the data from the first and second logical

datasets 206, 208 and writes a result of the processing into a third logical dataset DS3 210. The second sub-graph 204 receives a fourth logical dataset DS4 212 (which happens to point to the same physical file as the third logical dataset 210) as input, processes the data from the fourth logical dataset 212, and writes the result of the
5 processing to a table 214.

Each of the four logical datasets 206, 208, 210, 212 is associated with a parameterized path which, at run time, resolves to a path to a physical file. In particular, the first logical dataset 206 is identified using the parameterized path */\${FEED}/inv_\${DATE}.dat*, the second logical dataset 208 is identified using the
10 parameterized path */\${FEED}/cust_\${DATE}.dat*, the third logical dataset 210 is identified using the parameterized path */trans_\${DATE}.dat*, and the fourth logical dataset 212 is identified using the parameterized path */trans_\${DATE}.dat*.

The first sub-graph 202 receives two parameters, P1 = FEED and P2 = DATE as arguments and, as is described in greater detail below, uses the parameters to
15 resolve the paths to the respective physical locations of the first logical dataset 206, the second logical dataset 208, and the third logical dataset 210 by replacing the FEED and DATE placeholders in the parameterized paths with the values of the received FEED and DATE parameters. Additionally, the first sub-graph 202 includes a “static analysis” value for the DATE parameter. As is described in greater detail
20 below, the static analysis value for the DATE parameter is a placeholder value which is used as the parameter value during static analysis of the dataflow graph 217 (i.e., when the data lineage of the dataflow graph 217 is determined).

Similarly, the second sub-graph 104 receives a single parameter P1 = DATE and uses it to resolve the path to the physical location of the fourth logical dataset 212
25 by replacing the DATE placeholder in the parameterized path for the fourth logical dataset 212 with the value of the received DATE parameter. Additionally, the second sub-graph 204 includes a “static analysis” value for the DATE parameter. As is described in greater detail below, the static analysis value for the DATE parameter is a placeholder value which is used as the parameter value during static analysis of the
30 dataflow graph 217 (i.e., when the data lineage of the dataflow graph 217 is determined).

Since the operation of the dataflow graph 217 and its sub-graphs depends on the parameters that it receives, the dataflow graph and its sub-graphs are sometimes referred to “generic” dataflow graphs or “generic” computer programs.

1.1 Parameters

In general, the parameters described above can be designated as either “design time” parameters or “run time” parameters. In addition to being used for path resolution as described above, design time parameters affect the logical operation of their associated dataflow graph. In contrast, run time parameters are supplied to the graph on a job-by-job basis and do not affect the logical operation of the graph. In some examples, the logical operation of a dataflow graph refers to both the functionality of the graph and the logical datasets utilized by the graph.

In FIG. 2, the FEED parameter is a design time parameter which affects the logical operation of the gather.mp sub-graph. For example, for one value of the FEED parameter, a sort component 216 in the first sub-graph 202 may sort the data that it receives in ascending order while another, different value of the FEED parameter may cause the sort component 216 to sort the data in descending order. In some examples, a dataflow graph which includes design time parameters is referred to as a “generic graph” since its logical operation changes based on the supplied value of the design time parameter.

The DATE parameter is a run time parameter which has no effect on the logical operation of the sub-graph 202 and is supplied on a job-by-job basis.

1.2 Parameter Sets

In some examples, commonly used sets of parameters for dataflow graphs are stored as “parameter sets” (sometimes referred to as “psets”) which can be saved to disk and easily re-used. For example, in FIG. 2, the first sub-graph 202 has three psets associated with it, PSET_mexico 218, PSET_canada 220, and PSET_usa 222. PSET_mexico 218 includes a commonly used FEED parameter value “mexico” and a commonly used DATE parameter value “today()” which is a function that returns today’s date. PSET_canada 220 includes a commonly used FEED parameter value “canada” and the commonly used DATE parameter value “today().” PSET_usa 222 includes a commonly used FEED parameter value “usa” and the commonly used DATE parameter value “today().”

Similarly, the second sub-graph 204 has a single pset associated with it, PSET 223. PSET 223 includes the commonly used DATE parameter value “today()” which is a function that returns today’s date.

2 Parameter resolution module

In some examples, prior to the dataflow graph 117 being executed by the execution module 112, the parameter resolution module 106 of FIG. 1 identifies one or more psets associated with the dataflow graph 117 (and its associated sub-graphs 202, 204) and determines a number of unique design time parameters in the one or more psets. For each unique design time parameter for a given dataflow graph, the parameter resolution module 106 instantiates a separate executable instance of the dataflow graph. For example, referring to FIG. 3, for the dataflow graph 217 of FIG. 2, three instances of the first sub-graph 202, `gather.mp` are instantiated (PSET_mexico->`gather.mp` 202a, PSET_canada-> `gather.mp` 202b, PSET_usa->`gather.mp` 202c), each instance configured according to a different one of the three unique feed parameters in the psets of FIG. 2: `mexico`, `canada`, and `usa`. Since the second sub-graph 204 is associated with only a single pset 223 that does not include any design time parameters, only a single instance (`process.mp` 204a) of the second sub-graph 204 is instantiated at execution time.

Once the appropriate instances of the sub-graphs 202, 204 are instantiated by the parameter resolution module 106, the parameter resolution module 106 replaces the parameter value placeholders in the parameterized paths for the datasets with actual parameter values from the psets, resolving the paths to the physical locations of the datasets. For example, for the PSET_mexico->`gather.mp` instance 202a of the first sub-graph 202, the path for the first dataset 206 is resolved to `/mexico/inv_031014` since the FEED parameter value is 'mexico' and the DATE parameter value is '031014.'

Once the parameter resolution module 106 has instantiated the dataflow graph 217 including its sub-graphs 202, 204 and has resolved the the physical paths to the dataflow graph's 217 datasets, the dataflow graph 217 is prepared for execution by the execution module 112. During execution, the three instances 202a, 202b, 202c of the first sub-graph 202 read data from their respective input datasets, process the data, and store the processed data in the `/trans_031014.dat` physical file. Since the input dataset (e.g., DS4 212) for the instance 204a of the second sub-graph 202 resolves to the same physical file as the output dataset of the first sub-graph, the `/trans_031014.dat` physical file is read by the instance of `process.mp` and then processed and stored in the table 214.

3 Data lineage module

Referring to FIG. 4, in some examples, rather than executing the dataflow graph 217, the data architect or business user 121 of FIG. 1 may have a need to inspect the lineage of data as it passes through the dataflow graph 217. To do so, the data lineage module 115 of FIG. 1 is configured to analyze the dataflow graph 217 to generate a data lineage report for presentation to the data architect or business user 121.

In some examples, as a first step in determining the data lineage for the dataflow graph 217, data lineage module 115 identifies the individual sub-graphs 202, 204 of the dataflow graph 217. For each of the identified sub-graphs 202, 204, the data lineage module 115 identifies one or more psets 218, 220, 222, 223 associated with the sub-graph 202, 204 and then determines a number of unique design time parameters in the one or more psets 218, 220, 222, 223 for the sub-graph 202, 204. For each unique design time parameter, the parameter resolution module instantiates a separate instance of the sub-graph 202, 204.

In some examples, the data lineage module 115 operates under an assumption that the actual physical files and the data which they store are irrelevant to data lineage analysis. For this reason, any run time parameter values which are used to resolve the physical locations of the datasets are unnecessary and can be replaced with placeholder values. As is noted above, for each run time parameter associated with a sub-graph, a corresponding placeholder, static analysis parameter value is included in the sub-graph. For example, in FIG. 2, since both dataflow graphs 202, 204 include the DATE run time parameter, they also both include a placeholder, static analysis parameter value of 'MMDDYY.'

When the data lineage module 115 analyzes the dataflow graph 217 to determine the data lineage, all instances of the DATE parameter in the dataflow graph are replaced with the 'MMDDYY,' placeholder value, creating temporary dataset objects 452 as is shown in FIG. 4. The interconnections between the various sub-graph instances and the temporary dataset objects are then identified, and presented to the data architect or business user as the data lineage. For example, an analysis of the instances 202a, 202b, 202c of the first sub-graph 202 indicates that all of the instances of the first sub-graph 202 write data to a dataset that is represented by the */trans_MMDDYY.dat* dataset object. The analysis then indicates that the instance

204a of the second dataflow graph 204 reads from the dataset that is represented by the */trans_MMDDYY.dat* dataset object. Based on this information, the data lineage for the dataflow graph 217 indicates that the outputs of the instances 202a, 202b, 202c of the first sub-graph 202 are fed into the input of the instance 204a of the second sub-graph 204.

4 Logical pset Discovery and Creation Method

In some examples, a given dataflow graph is executed using an execution command which receives parameter values as arguments supplied to the execution command rather than from a previously stored pset. Since the method described above determines data lineage using only stored psets, psets associated with the parameter values originating from arguments supplied to the execution command for an execution of the dataflow graph are not represented in the data lineage. This can result in an incomplete or incorrect data lineage being provided to an enterprise architect or an auditor.

FIG. 5 is a flowchart illustrating a method for augmenting a repository of existing logical parameter sets (psets) for a dataflow graph with logical psets created based on sets of parameters identified in a log associated with executions of instances of the dataflow graph. In some examples, the method of described in FIG. 5 is implemented by the data lineage module 115 of FIG. 1.

4.1 Graph Parameters

Initially one example of a dataflow graph (e.g., the first sub-graph 202 of FIG. 1) includes two parameters (P_1 and P_2), each of which can be designated as either a “design time” parameter or a “run time” parameter. As is noted above, a design time parameter is a parameter which affects the logical operation of the graph (e.g., can alter a transformation performed by the graph) while a run time parameter is a parameter which changes on a job-by-job basis (e.g., a date) and does not affect the logical operation of the graph.

4.2 Parameter Classification

The graph 202 is provided to a parameter classification step 424 which analyzes the parameters of the graph 202 to generate parameter classification result 426. In the parameter classification result 426, each parameter is classified as either a design time parameter or a run time parameter. In the exemplary case illustrated in

the flow chart, P_1 is classified as a design time parameter and P_2 is classified as a run time parameter.

In some examples, the parameters for a dataflow graph are pre-classified (e.g., by a user) as being either design time or run time parameters. In other examples (e.g.,
5 for legacy dataflow graphs), the parameters for the dataflow graph are not pre-classified as being either design time or run time parameters. In such cases, the parameter classification step 424 may assume that all parameters are design time parameters. In a later re-classification step, if it is determined that a given parameter has a large (e.g., above a given threshold) number of unique values in a collection of
10 log entries (e.g., the job log data store described below), then the given parameter may be re-classified as a run time parameter. Alternatively, re-classification can be based on data lineage sensitivity analysis. In particular, if a parameter can take on a variety of different values without altering the data lineage internal to the dataflow graph (i.e., impacts or dependencies of datasets or components within the dataflow graph), then
15 the parameter can be classified as a run time parameter. For example, if the associated record formats or other characteristics of a dataset in a graph (e.g., DS1, DS2, DS3 in FIG. 3) are not affected by the various values of a parameter, then that parameter is re-classified as a run time parameters. Variations of this data lineage sensitivity analysis can be used, such as a more comprehensive data lineage
20 sensitivity analysis that includes resolving all internal impacts and dependencies, and a more limited data lineage sensitivity analysis that includes resolving just impacts and dependencies associated with dataset record formats.

In some examples (e.g., for legacy dataflow graphs), a parameter may include both design time and run time portions. For example, a filename parameter
25 “/mexico/inv_031014.dat” may be a hybrid parameter in that it includes a design time portion (i.e., “mexico”) and a run time portion (i.e., “031014”). In such examples, a user can supply a regular expression or some other type of string parsing rules which are used by the parameter classification step 424 to extract and classify the respective design time and run time parameters from the hybrid parameter.

30 4.3 Job Log Data Store

The method utilizes a job log data store 428 including a number of job log entries 429, each including information associated with executions of instances of the dataflow graph 202. Among other information, at least some of the job log entries

include a record of an execution command which was used to instantiate the dataflow graph 202. The execution command for a given job log entry includes a graph name and parameter values which were supplied as arguments to the execution command. In general, at least some of the job log entries in the job log data store 428 instantiate
5 the dataflow graph without accessing any parameter sets but instead receive parameter values as arguments supplied to the execution command.

4.4 Processing Loop

The job log data store 428 and the parameter classification result 426 are provided to a processing loop 430 which, for each job log entry 429 in the job log data
10 store 428, generates a new logical pset for the graph execution command, determines whether the new logical pset already exists in a repository of existing logical psets 448, and adds the new logical pset to the repository 448 if it does not already exist.

4.4.1 Initial Command Line Logical pset Construction

Within the processing loop 430, the parameter classification result 426 and a
15 job log entry, J_n 432 from the job log data store 428 are provided to a logical pset construction step 434 which analyzes the job log entry 432 according to the parameter classification result 426 to generate a logical pset 436. In doing so, the logical pset construction step 434 analyzes the graph execution command included in the job log entry 432 to extract the parameter values that are included as arguments to the graph
20 execution command. The logical pset construction step 434 also extracts a project scope included in the job log entry 432. In some examples, the project scope includes an indication of the project that the dataflow graph is executing in, an indication of internal parameters for the dataflow graph, and an indication of environmental settings, global variables and configuration variables used to by the dataflow graph.

25 The logical pset construction step 434 automatically includes the extracted project scope in the logical pset 436. The logical pset construction step 434 then matches each extracted parameter value with a corresponding parameter in the parameter classification result 426. If the logical pset construction step 434 determines that an extracted parameter value corresponds to a design time parameter
30 in the parameter classification result 426, then the logical pset construction step 434 includes the value of the extracted design time parameter in the logical pset 436. If the logical pset construction step 434 determines than an extracted parameter value

corresponds to a run time parameter in the parameter classification result 426, then the extracted parameter value is not included in the logical pset 436.

4.4.2 pset Signature String Computation

The logical pset 436 is provided to a pset signature string computation step 5 442 which computes a logical pset signature string 444 based on the project scope and the parameter values in the logical pset 436. In some examples, the pset signature string 444 is computed by serializing the project scope for the logical pset 436, name/value pairs of the parameters in the logical pset 436, and a prototype of the dataflow graph associated with the logical pset 436. In other examples, the pset 10 signature string 444 is computed by applying a hash function or some other data mapping algorithm to the logical pset 436.

4.4.3 pset Signature String Search

The pset signature string 444 is provided to a pset signature search step 446 along with the pset signature strings of all existing logical psets in the repository of 15 existing logical psets 448. For each of the existing logical psets, the pset signature string of the existing logical pset is compared to the pset signature string 444. If the pset signature string 444 matches at least one of the pset signature strings of the existing logical psets, then nothing needs to be done since a logical pset for the execution command instantiation of the graph 432 already exists in the repository of 20 existing logical psets 448.

In some examples, the pset signature strings of all existing logical psets in the repository of existing logical psets 448 are stored along side the existing logical psets in the repository 448. In other examples, the signature strings for the existing logical psets are computed on the fly and on an as-needed basis.

25 4.4.4 Addition of New Logical pset

Otherwise, if none of the signature strings of the existing logical psets matches the pset signature string 444, the logical pset 436 and its signature string 444 are added as a new logical pset to the repository of existing logical psets 448 by a new logical pset addition step 450.

4.5 Example

Referring to FIGs. 6 and 7, an exemplary operation of the logical pset discovery and creation method of FIG. 4 as applied to the first sub-graph 202 of FIG. 2 is presented. The first sub-graph 202 of FIG. 2 includes two parameters, P1 = FEED and P2 = DATE. The first sub-graph 202 is provided to the parameter classification step 424 where the parameters are classified as either “design time” or “run time” parameters, generating a parameter classification result 426. The parameter classification result 426 indicates that the P1 (FEED) parameter is a design time parameter and the P2 (DATE) parameter is a run time parameter.

The parameter classification result 426 and the job log data store 428 are provided to the logical pset construction step 434. In the example of FIG. 6, the job log data store 428 includes four job log entries that include information associated with executions of instances of the first sub-graph 202 (i.e., gather.mp). Each job log entry includes an execution command which received values for the DATE and FEED parameters as arguments.

The logical pset construction step 434 creates a different logical pset 436 for each of the job log entries in the job log data store 428. Since the P1 (FEED) parameter is a design time parameter, its value (e.g., mexico, usa, canada, or hong kong), which was supplied as an argument to the execution command, is included for each of the of the logical psets 436. Since the P2 (DATE) parameter is a run time parameter, its value, which was supplied as an argument to the execution command, is not included in the logical psets 436. Each of the logical psets 436 includes the project scope for its corresponding instance of the first sub-graph 202.

Referring to FIG. 7, the logical psets 436 are provided to a pset signature string computation step 442 which computes a different logical pset signature string 444 for each of the logical psets 436.

The logical pset signature strings 444 and a set of logical pset signature strings for the existing psets 447 in the repository of existing psets 448 are provided to a search step 446. As was the case in FIG. 2, there are three existing psets associated with the first sub-graph 202: one for the mexico FEED parameter, one for the usa FEED parameter, and one for the canada FEED parameter. Thus, the set of logical pset signature strings 444 for the existing psets 447 includes a string for each of the existing psets associated with the first sub-graph 202.

The search step 446 searches for the presence of each of the logical pset signature strings 444 in the set of logical pset signature strings for the existing psets 447. In this example, the result generated by the search step 446 is that the only logical pset signature string not included in the set of logical pset signatures strings
5 for the existing psets 447 is the logical pset signature string associated with the logical pset with a FEED parameter value of 'hong kong.'

The result of the search step 446 and the logical pset 436 that includes the 'hong kong' feed parameter are provided to a logical pset addition step 450 which adds the logical pset which includes the FEED parameter of 'hong kong,' and its
10 corresponding logical pset signature string 444 to the repository of existing logical psets 448.

By adding the new logical pset to the repository, a 'hong kong' instance of the first sub-graph 202, which would have been overlooked in previous data lineage results, will be represented in the data lineage results.

15 It is noted that while the static analysis values for run time parameters are described as being stored in the dataflow graphs themselves in the above examples, in some examples, the static analysis values for run time parameters can be maintained in one or more psets associated with the dataflow graphs.

In some examples, certain design time parameter values are derived from
20 sources (e.g., from a database) that are not necessarily present at static analysis time. However, in some examples, the job log entries stored in the job log data store include values for all parameters that were resolved for that particular job. At static analysis time, the stored parameter values can be used in place of the parameter values derived from sources that are not present at static analysis time.

25 In some examples, the job log entries in the job log data store include all resolved parameters for a dataflow graph, a log of all files read and written by the dataflow graph, and performance tracking information. In some examples, the job log entries in the job log data store are augmented with any logical parameter sets that are discovered by the method of FIG. 4. In some examples, augmenting job log entries in
30 the job log data store with discovered logical parameter sets includes forming an association between the job log entries and the discovered logical parameter sets. The augmented job log entries in the job log data store can be leveraged to provide various forms of information to a data architect or business user. In some examples, the augmented job log entries can be analyzed to ensure that dataflow graphs that are

logically connected are also physically connected. In some examples, the augmented job log entries can be analyzed to determine which logical dataset instances a physical dataset corresponds to. In some examples, the augmented job log entries can be analyzed to identify datasets that have the same physical file name but are associated
5 with different static analysis parameters. In such examples, the inconsistency can be presented to the user for manual repair or can be automatically repaired. In some examples, the data lineage report can include an indication of the inconsistency and whether or not it has been automatically repaired.

In some examples, the augmented job log entries can be used by the data
10 lineage module to filter data lineage reports by frequency and/or recency. For example, the metadata environment module may maintain a number of dataflow graphs and psets that are no longer executed by the execution module. Such dataflow graphs and psets may be left in place just in case it is needed at a later time. However, the unexecuted dataflow graphs and psets can cause unnecessary clutter in
15 data lineage reports. To reduce the clutter, the augmented job log entries can be analyzed to determine which dataflow graphs and/or psets are infrequently used and/or have not been recently used. Based on this frequency and recency information, infrequently and non-recently executed dataflow graphs and psets (e.g., a dataflow graph that hasn't run in the past year) can be filtered out of a data lineage report prior
20 to presentation to an enterprise user.

In some examples, a logical pset for a given dataflow graph (e.g., a pset including FEED = USA) may exist, but one or more jobs that invoke the dataflow graph do so by directly supplying parameter values to the dataflow graph instead of utilizing the existing pset. In such cases, an association maintained between jobs and
25 the logical psets that were accessed by the jobs (e.g., via signatures associated with the jobs) can be used to group job log entries based on their associated logical psets. Based on the grouping, any jobs that are instantiated by invoking a graph directly instead of utilizing an existing pset can be identified as being related to the logical pset and its parameters.

30 In some examples, each job log entry for a dataflow graph includes, among other information, a list of all resolved parameter values for the execution of the dataflow graph that is associated with the job log entry. Once a number of job log entries have accumulated, the resolved parameter values included in the job log entries can be compared to identify the various "design time instances" of the

dataflow graph. For example, certain resolved parameters in the job log entries may be represented by only a few values in all of the job log entries, while certain other resolved parameters may be represented by many different values in all of the job log entries. Those resolved parameters that are represented by only a few values in the
5 job log entries are likely “design time” parameters and the other resolved parameters that are represented by many different values in the job log entries are likely “run time parameters.” Any instances of the dataflow graph that share a unique combination of “design time parameters” are grouped together and are considered to all be a “design time instance” of the dataflow graph. The data lineage module includes the different
10 design time instances of the dataflow graph in the data lineage report.

5 Duplicate Logical Dataset Discovery and Mitigation Method

5.1 Overview

In general, input and output datasets (e.g., databases or tables of data) for a given dataflow graph are specified as logical datasets in the dataflow graph. In some
15 examples, each logical dataset is associated with an identifier such as a logical file name.

Before the dataflow graph is executed, it is prepared for execution including resolving each logical dataset to a corresponding physical dataset (e.g., a file on disk). In some examples, each physical dataset is associated with an identifier such as a
20 physical file name (e.g., “*summary.dat*”). The parameter resolution process is able to successfully resolve a logical dataset to its corresponding physical dataset even if the logical file name of the logical dataset differs from the physical file name of the corresponding physical dataset.

When a data lineage report is determined for a dataflow graph including two
25 or more sub-graphs, the lineage relationships between the sub-graphs are at least in part determined according to the logical file names of the input and output logical datasets of the two or more sub-graphs. For this reason, the correctness of the lineage relationships requires that any input and output logical datasets of the two or more sub-graphs that refer to a given physical dataset share the same logical file name.
30 Indeed, if a first sub-graph writes to a given physical dataset and a second sub-graph subsequently reads from the given physical dataset, but the logical file names of the output logical dataset of the first sub-graph and the input logical dataset of the second

sub-graph do not match, no lineage relationship will be identified between the two sub-graphs. In some examples, two logical datasets that resolve to the same physical dataset but have non-matching logical file names are referred to as “duplicate logical datasets.”

5 As is described in detail below, duplicate logical datasets in a dataflow graph can be identified and presented to a user. The user can then choose to address the duplicate logical datasets in a number of ways.

5.2 Example without Duplicate Logical Datasets

Referring to FIG. 8 an example of a dataflow graph 817 generated using the
10 development environment 118 of FIG. 1 includes a first sub-graph 802 named gather.mp and a second sub-graph 804 named process.mp.

The first sub-graph 802 receives a first logical dataset D_{L1} 806 with a logical file name “Acct_1.dat” and a second logical dataset D_{L2} 808 with a logical file name “Acct_2.dat” as input. The first sub-graph 802 processes the data from the first and
15 second logical datasets 806, 808 and writes a result of the processing into a third logical dataset D_{L3} 810 with a logical file name “Acct_summ.dat.” The second sub-graph 804 receives the third logical dataset D_{L3} 810 with a logical file name “Acct_summ.dat” as input, processes the data from the third logical dataset 810, and writes the result of the processing to a table 814. Note that both the third logical
20 dataset 810, which is used by both the first sub-graph 802 and the second sub-graph 804 has the same logical file name in both of the sub-graphs 802, 804.

Referring to FIG. 9, when the dataflow graph 817 is resolved prior to execution, the logical datasets are resolved to their corresponding physical datasets. For example, the first logical dataset 806 is resolved to a first physical dataset, D_{P1}
25 814 with a physical file name “Acct_1.dat,” the second logical dataset 808 is resolved to a second physical dataset, D_{P2} 816 with a physical file name “Acct_2.dat,” the third logical dataset 810 is resolved to a third physical dataset, D_{P3} 818 with a physical file name “summary.dat.”

Referring to FIG. 10, a data lineage report 1017 for the dataflow graph
30 includes the first sub-graph 1002, the second sub-graph 1004, the first logical dataset 1006, the second logical dataset 1008, and the third logical dataset 1010. The data lineage report 1017 also includes a first lineage relationship 1018 between the first logical dataset 1006 and an input of the first sub-graph 1002, a second lineage

relationship 1020 between the second logical dataset 1008 and an input of the first sub-graph 1002, a third lineage relationship 1022 between an output of the first sub-graph 1002 and the third logical dataset 1010, and a fourth lineage relationship 1024 between the third logical dataset 1010 and the second sub-graph 1004. Note that data
 5 lineage report 1017 is correct in this case since the same logical dataset (i.e., the third logical dataset, D_{L3} 810) with the same logical file name (i.e., “*Acct_summ.dat*”) is present at the output of the first sub-graph 802 and at the input of the second sub-graph 804.

5.3 Example with Duplicate Logical Datasets

10 Referring to FIG. 11 another example of a dataflow graph 1117 generated using the development environment 118 of FIG. 1 includes a first sub-graph 1102 named gather.mp and a second sub-graph 1104 named process.mp.

The first sub-graph 1102 receives a first logical dataset D_{L1} 1106 with a logical file name “*Acct_1.dat*” and a second logical dataset D_{L2} 1108 with a logical
 15 file name “*Acct_2.dat*” as input. The first sub-graph 1102 processes the data from the first and second logical datasets 1106, 1108 and writes a result of the processing into a third logical dataset D_{L3} 1110 with a logical file name “*Acct_summ.dat*.” The second sub-graph 1104 receives, as in put a fourth logical dataset D_{L4} 1111 with a logical file name “*Acct-summ.dat*” as input, processes the data from the fourth logical dataset
 20 1111, and writes the result of the processing to a table 814. Note that the logical file name for the third logical dataset 1110 (i.e., “*Acct_summ.dat*”) differs from the logical file name for the fourth logical dataset 1111 (i.e., “*Acct-summ.dat*”).

Referring to FIG. 12, when the dataflow graph 1117 is resolved prior to execution, the logical datasets are resolved to their corresponding physical datasets.
 25 For example, the first logical dataset 1106 is resolved to a first physical dataset, D_{P1} 1114 with a physical file name “*Acct_1.dat*,” the second logical dataset 1108 is resolved to a second physical dataset, D_{P2} 1116 with a physical file name “*Acct_2.dat*,” and both the third logical dataset 1110 and the fourth logical dataset 1111 are resolved to a third physical dataset, D_{P3} 1218 with a physical file name
 30 “*summary.dat*.” Note that the third logical dataset 1110 and the fourth logical dataset 1111 are duplicate logical datasets since they each point to the same physical dataset (i.e., the third physical dataset 1218).

Referring to FIG. 13, a data lineage report 1317 for the dataflow graph includes the first sub-graph 1102, the second sub-graph 1104, the first logical dataset 1106, the second logical dataset 1108, the third logical dataset 1110, and the fourth logical dataset 1111. The data lineage report 1317 also includes a first lineage relationship 1318 between the first logical dataset 1106 and an input of the first sub-graph 1102, a second lineage relationship 1320 between the second logical dataset 1108 and an input of the first sub-graph 1102, a third lineage relationship 1322 between an output of the first sub-graph 1102 and the third logical dataset 1110, and a fourth lineage relationship 1324 between the fourth logical dataset 1111 and the second sub-graph 1104.

Note that the data lineage report 1317 is incorrect in this case since two different logical datasets (i.e., the third logical dataset 1110 and the fourth logical dataset 1111) with different logical file names refer to the same physical dataset (i.e., the third physical dataset 1218). In particular, the third logical dataset, D_{L3} 1110 with the logical file name “*Acct_summ.dat*” is present at the output of the first sub-graph 1102 and the fourth logical dataset 1111 with the logical file name “*Acct-summ.dat*” is present at the input of the second sub-graph 1104. The data lineage report 1317 represents the third logical dataset 1110 and the fourth logical dataset 1111 as separate datasets without any lineage relationship with one another. As such, the data lineage report 1317 incorrectly includes a break in the data lineage between the third logical dataset 1110 and the fourth logical dataset 1111.

5.4 Duplicate Logical Dataset Discovery

In some examples, duplicate logical datasets in a dataflow graph can be discovered by analyzing runtime artifacts (e.g., the job logs 429 of in FIG. 5) generated by executions of the dataflow graph. In particular, each time the dataflow graph is executed, a job log is generated.

The job log includes information associated with the execution of the dataflow graph including the graph instance name and, for each dataset component in a graph, the physical datasets it accessed and the type of access (read or write). Graph instances can be examined to determine logical dataset names for each dataset component. By matching on the graph instance and the dataset component name, the system is able to map logical dataset names to physical dataset names.

To identify duplicate logical datasets, the job logs are analyzed to identify any logical to physical dataset mappings in which the first logical dataset of the mapping differs from the second logical dataset of the mapping. Any logical to physical dataset mappings in which the first logical dataset and the second logical dataset
5 differ are classified as duplicate logical datasets.

The identified duplicate logical datasets are either presented to a user who decides whether or not to correct the duplicate logical datasets or are automatically mitigated.

5.4.1 Example of Duplicate Logical Dataset Discovery

10 Referring again to FIG. 12, when the resolved dataflow graph 1117 executes, a job log for the dataflow graph execution is generated. The job log includes a single logical to physical dataset mapping corresponding to the flow between the first sub-graph 1102 and the second sub-graph 1104. The logical to physical dataset mapping includes an identifier for the third logical dataset, D_{L3} 1110 at the output of the first
15 sub-graph 1104, an identifier for the fourth logical dataset, D_{L4} 1111 at the input of the second sub-graph 1106, and an identifier for the third physical dataset 1218.

Since the third logical dataset 1110 and the fourth logical dataset 1111 are distinct logical datasets (e.g., logical datasets with different logical file names) that point to the same physical dataset (i.e., the third physical dataset 1218), the third
20 logical dataset 1110 and the fourth logical dataset 1111 are classified as duplicate logical datasets.

Note that while the simple example described above includes the identification of a single pair of duplicate logical datasets from a single job log, in an actual implementation of a data processing system that includes the above duplicate logical
25 dataset discovery approaches, a number of pairs of duplicate logical datasets may be identified using a number of job logs.

5.5 Duplicate Logical Dataset Mitigation

As is noted above, duplicate logical datasets may result in breaks in data lineage reports. Once the duplicate logical datasets are identified, a number of
30 different approaches can be taken to eliminate the duplicate logical datasets or to mitigate their effects on data lineage reports. In some examples, the identified duplicate logical datasets are presented to a user in, for example, a spreadsheet form.

The user can then edit the dataflow graphs that include the duplicate logical datasets to eliminate the duplicate logical datasets (e.g., by ensuring that, in a given dataflow graph, a given physical dataset is referred to only by a single logical dataset). In other examples, the user can mark a pair of duplicate logical datasets as being equivalent.

5 In this way, the user is not required to make any changes to the dataflow graphs. In yet other examples, pairs of duplicate logical datasets can be automatically marked as being equivalent.

When a pair of duplicate logical datasets are marked as being equivalent, there are a number of approaches to displaying the equivalency in a data lineage report. In one approach, the physical dataset that the pair of duplicate logical datasets refers to is shown connected to the duplicate logical datasets in the data lineage report. For example, referring to FIG. 14, the third physical dataset, D_{P3} 1218 is included in the data lineage report 1317. Both the third logical dataset, D_{L3} 1110 and the fourth logical dataset, D_{L4} 1111 are shown as being connected to the third physical dataset 15 1218 by lineage relationships 1450 and 1452.

In another approach, the logical datasets of the pair of duplicate logical datasets are shown connected to one another in the data lineage report by a lineage relationship. For example, referring to FIG. 15, the third logical dataset, D_{L3} 1110 is shown as being connected to the fourth logical dataset, D_{L4} 1111 by a lineage 20 relationship 1550 in the data lineage report 1317.

In another approach, the pair of duplicate logical datasets is represented by a combined logical dataset in the data lineage report. For example, referring to FIG. 16, the pair of duplicate logical datasets is represented by a combined logical dataset, D_{LR} 1654 in the data lineage report 1317.

25 In another approach, one logical dataset of the pair of duplicate logical datasets is chosen to represent the pair of duplicate logical datasets in the data lineage report. For example, referring to FIG. 17, the fourth logical dataset, D_{L4} 1111 represents the pair of duplicate logical datasets in the data lineage report 1317.

In another approach, the pair of duplicate logical datasets and a combined logical dataset representation of the pair of duplicate logical datasets are included in 30 the data lineage report. A unique configuration of lineage relationships between pair of duplicate logical datasets and the combined logical dataset representation is shown in the data lineage graph. For example, referring to FIG. 18, the data lineage report 1317 includes a combined logical dataset representation of the pair of duplicate

logical datasets, D_{LR} , 1854, the third logical dataset, D_{L3} 1110, and the fourth logical dataset, D_{L4} 1111. The combined logical dataset 1854 is shown as having direct lineage relationships with the first sub-graph 1102 and the second sub-graph 1104. The combined logical dataset 1845 is also shown as having an indirect lineage relationship with the first sub-graph 1102 via the third logical dataset 1110 and as having an indirect lineage relationship with the second sub-graph 1104 via the fourth logical dataset 1111.

In another approach, the logical datasets of the pair of duplicate logical datasets are included in the data lineage report. A unique configuration of lineage relationships between the logical datasets of the pair of duplicate logical datasets is shown in the data lineage graph. For example, referring to FIG. 19, the data lineage report 1317 includes the third logical dataset, D_{L3} 1110 and the fourth logical dataset, D_{L4} 1111. The fourth logical dataset 1111 is shown as having direct lineage relationships with the first sub-graph 1102 and the second sub-graph 1104. The third logical dataset, D_{L3} 1110 is shown as having a direct lineage relationship with the first sub-graph 1102 and as having an indirect lineage relationship with the second sub-graph 1104 via the fourth logical dataset 1111.

Note that, in some examples, the mitigation approaches described above are shown in data lineage reports in dashed lines, bold lines, or in another alternative fashion such that it is clear to a user of the data lineage report that a mitigation approach has been applied to the data lineage report.

It is noted that, while the above duplicate logical dataset discovery and mitigation approaches are described using a scenario where a first component writes to a physical dataset and another component reads from that physical dataset, other scenarios can result in duplicate logical datasets. For example, a pair of duplicate logical datasets can result from two different logical datasets reading from the same physical dataset. Similarly, a pair of duplicate logical datasets can result from two different logical datasets writing to the same physical dataset.

The approaches described above can incorporate features from a variety of other approaches for managing and presenting data lineage information and for managing dataset objects, as described in more detail in U.S. Application Serial No. 12/393,765, filed on February 26, 2009, U.S. Application Serial No. 13/281,039, filed on October 25, 2011, and U.S. Provisional Application Serial No. 62/028,485, filed on July 24, 2014, all of which are incorporated herein by reference.

The approaches described above can be implemented, for example, using a programmable computing system executing suitable software instructions or it can be implemented in suitable hardware such as a field-programmable gate array (FPGA) or in some hybrid form. For example, in a programmed approach the software may

5 include procedures in one or more computer programs that execute on one or more programmed or programmable computing system (which may be of various architectures such as distributed, client/server, or grid) each including at least one processor, at least one data storage system (including volatile and/or non-volatile memory and/or storage elements), at least one user interface (for receiving input using

10 at least one input device or port, and for providing output using at least one output device or port). The software may include one or more modules of a larger program, for example, that provides services related to the design, configuration, and execution of dataflow graphs. The modules of the program (e.g., elements of a dataflow graph) can be implemented as data structures or other organized data conforming to a data

15 model stored in a data repository.

The software may be provided on a tangible, non-transitory medium, such as a CD-ROM or other computer-readable medium (e.g., readable by a general or special purpose computing system or device), or delivered (e.g., encoded in a propagated signal) over a communication medium of a network to a tangible, non-transitory

20 medium of a computing system where it is executed. Some or all of the processing may be performed on a special purpose computer, or using special-purpose hardware, such as coprocessors or field-programmable gate arrays (FPGAs) or dedicated, application-specific integrated circuits (ASICs). The processing may be implemented in a distributed manner in which different parts of the computation specified by the

25 software are performed by different computing elements. Each such computer program is preferably stored on or downloaded to a computer-readable storage medium (e.g., solid state memory or media, or magnetic or optical media) of a storage device accessible by a general or special purpose programmable computer, for configuring and operating the computer when the storage device medium is read by

30 the computer to perform the processing described herein. The inventive system may also be considered to be implemented as a tangible, non-transitory medium, configured with a computer program, where the medium so configured causes a computer to operate in a specific and predefined manner to perform one or more of the processing steps described herein.

A number of embodiments of the invention have been described.

Nevertheless, it is to be understood that the foregoing description is intended to illustrate and not to limit the scope of the invention, which is defined by the scope of the following claims. Accordingly, other embodiments are also within the scope of

5 the following claims. For example, various modifications may be made without departing from the scope of the invention. Additionally, some of the steps described above may be order independent, and thus can be performed in an order different from that described.

Claims

1. A method for managing lineage information, the method including:
receiving lineage information representing one or more lineage relationships among two or more data processing programs and two or more logical datasets, wherein at least one of the logical datasets resolves to a physical dataset at run time of at least one of the data processing programs;
receiving one or more runtime artifacts, each runtime artifact including information related to a previous execution of a data processing program of the two or more data processing programs; and
analyzing the one or more runtime artifacts and the lineage information and determining one or more candidate modifications to the lineage information based on results of the analyzing, wherein at least one candidate modification includes a modification to a representation of at least one of the two or more logical datasets based at least in part on a result of the analyzing, wherein the result is associated with one or more physical datasets;
wherein the one or more candidate modifications include a first candidate modification that adds a new direct lineage relationship between a data processing program of the two or more data processing programs and a logical dataset of the two or more logical datasets;
wherein analyzing the runtime artifacts and the lineage information includes analyzing logs of previous executions of the two or more data processing programs to determine physical datasets read from or written to by the two or more data processing programs; and
wherein analyzing the runtime artifacts and the lineage information further includes identifying two distinct logical datasets of the two or more logical datasets that are represented in the lineage information and are associated with the same physical dataset.

2. The method of claim 1, wherein the one or more candidate modifications include a second candidate modification that adds a new indirect lineage relationship between a data processing program of the two or more data processing programs and a logical dataset of the two or more logical datasets.

3. The method of claim 1, wherein the first candidate modification includes creation of the new direct lineage relationship between the two distinct logical datasets.

4. The method of claim 1, wherein the first candidate modification includes creation of the new direct lineage relationship including merging the two distinct logical datasets into a new combined logical dataset.

5. The method of claim 1, wherein each data processing program of the two or more data processing programs is an instance of a generic data processing program instantiated according to a set of one or more parameter values.

6. The method of claim 5, wherein analyzing the one or more runtime artifacts and the lineage information includes:

analyzing one or more logs of previous executions of a first data processing program of the two or more data processing programs to determine a first parameter set used in a first instantiation of the first data processing program according to a first set of one or more parameter values,
selecting at least some parameters from the first parameter set, and
determining that the first instantiation of the first data processing program is not represented in the lineage information based on a generic version of the first data processing program and the at least some parameters.

7. The method of claim 6, wherein selecting at least some parameters from the first parameter set includes selecting parameters based on information received from a user.

8. The method of claim 6, wherein selecting at least some parameters from the first parameter set includes selecting parameters based on one or more predefined rules.

9. The method of claim 8, wherein a first rule of the one or more predefined rules specifies that parameters with parameter values in the form of a date are excluded from the selected parameters.

10. The method of claim 8, wherein a first rule of the one or more predefined rules specifies that a parameter with a parameter value that is transformed in the logic of a generic data processing program is included in the selected parameters.

11. The method of claim 1, wherein the results of the analyzing include identification of at least one physical dataset to which at least one logical dataset resolved at run time of at least one data processing program.

12. The method of claim 1, further including applying a selected one of the one or more candidate modifications to the lineage information.

13. The method of claim 12, wherein the selected candidate modification is selected based at least in part on user input received after presenting one or more of the candidate modifications.

14. A computer-readable storage medium comprising instructions which, when executed by a computing system, cause the computing system to carry out the method of any one of claims 1-13.

15. A computing system comprising at least one processor configured to perform the method of any one of claims 1-13.