(19) LE GOUVERNEMENT
DU GRAND-DUCHÉ DE LUXEMBOURG
Ministère de l'Économie

(11) N° de publication : **LU101632**

(12) **BREVET D'INVENTION** **B1**

(21) N° de dépôt: LU101632

(22) Date de dépôt: 07/02/2020

(51) Int. Cl.:
G06F 11/30, G06Q 10/00

(30) Priorité:

(43) Date de mise à disposition du public: 09/08/2021

(47) Date de délivrance: 09/08/2021

(73) Titulaire(s):
Microsoft Technology Licensing LLC – Redmond, WA 98052-6399 (États-Unis)

(72) Inventeur(s):
ZHOU Minxiao – États-Unis, LI Yanglei – États-Unis, ALCANTARA Travis – États-Unis

(74) Mandataire(s):
CMS Luxembourg 2.0 –
1637 Luxembourg (Luxembourg)

(54) **COMPUTER PERFORMANCE DEFECT DETECTION BASED ON ENERGY CONSUMPTION TELEMETRY.**

(57) Detecting a performance defect at an electronic computing platform resulting from a configuration change within the computing platform. A first distribution of first telemetry data is obtained from a first plurality of instances of the computing platform, and a second distribution of second telemetry data is obtained from a second plurality of instances of the computing platform. The first telemetry data corresponds to a pre-change configuration, and the second telemetry data corresponds to a post-change configuration. The telemetry data includes indicates energy consumption by component(s) at corresponding instances of the computing platform. Result(s) are computed using the first and second telemetry data as input. The result(s) characterize differences between the first and second distributions. The result(s) are input to a trained machine learning model to obtain a prediction of whether the differences between the first and second distributions indicate that a performance defect was introduced by the configuration change.

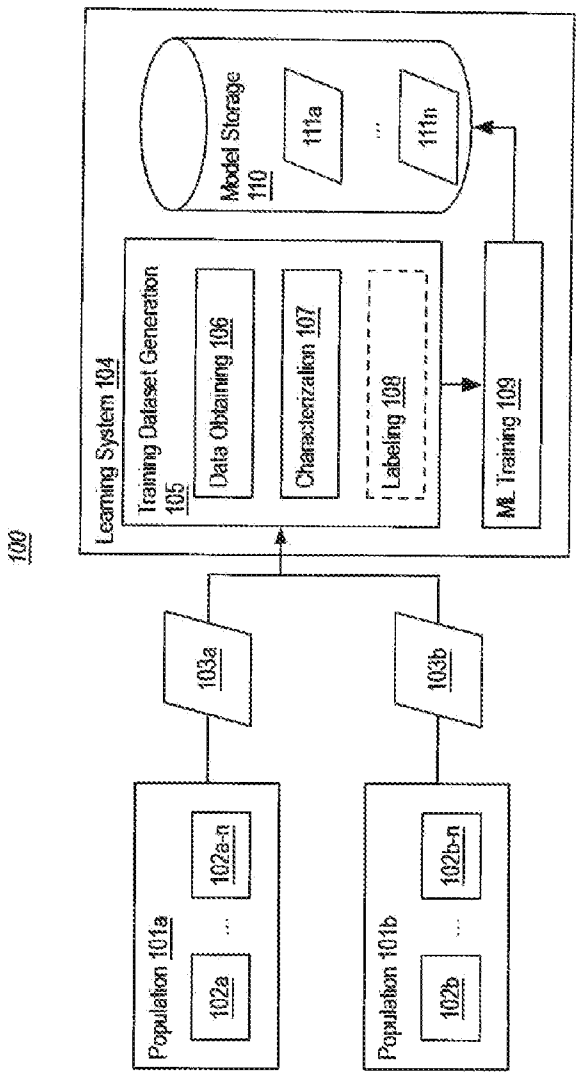


FIG. 1

# COMPUTER PERFORMANCE DEFECT DETECTION BASED ON ENERGY CONSUMPTION TELEMETRY

## TECHNICAL FIELD

**[001]** The present disclosure relates to systems, methods, and devices for identifying performance defects at computer systems based on telemetry data indicating energy consumption at those computer systems.

## BACKGROUND

**[002]** A computer system rarely runs the software and firmware originally installed thereon throughout its entire lifecycle. Instead, in today's connected world, most computer systems frequently receive software and firmware updates. For example, it is common for a computer system's operating system and application software to receive software updates (e.g., with updated software code that includes security and/or feature updates, configuration changes, etc.). Additionally, is also common for a computer system's hardware to receive firmware updates (e.g., with updated firmware code that addresses performance and/or security issues, configuration changes, etc.). Each of these updates represents a change in the configuration of the computer system that has the potential to cause unwanted performance defects at the computer system. Conventionally, these performance defects have been identified using techniques such as manual detection, rule-based detection, or lab testing. Manual detection involves a human user observing that a performance defect has occurred after an update. Rule-based detection identifies a performance defect based on determining that some metric observed at the computer system has exceeded some pre-defined threshold for the metric. Lab testing involves running one or more pre-defined test scenarios at the computer system, and determining if each test scenario succeeds or fails.

**[003]** Each of these performance defect detection techniques relies at least somewhat on chance (e.g., that a user both perceives and then recognizes the defect, that a defect actually manifests when running a testing scenario, etc.) and/or requires *a priori* definition of thresholds and/or testing scenarios that are actually able to detect the performance defect. As such, conventional performance defect detection techniques are often resource-intensive (e.g., in terms of human time, and/or in terms of computer resources used to monitor thresholds and run test scenarios), and imprecise (e.g., due to human failings, and/or due to the difficulty in anticipating proper thresholds/test scenarios for surfacing performance defects).

## BRIEF SUMMARY

**[004]** At least some embodiments described herein detect performance defects in a computing platform based on analyzing power consumption data obtained from a population of a plurality of instances of the computing platform. In particular, after a configuration change—such as a software or firmware update—in the computing platform, the embodiments described herein obtain a post-change distribution of power consumption telemetry data from the population of instances of the computing platform. The embodiments herein then compare this post-change distribution of power consumption telemetry data with a pre-change distribution data of power consumption telemetry data generated by that population (or from a similar population) prior to the configuration change. The comparison computes one or more characterizations of differences (e.g., shifts) in the distribution of power consumption telemetry data found in the pre-change distribution as compared to the distribution of power consumption telemetry data found in the post-change distribution. Embodiments provide these computed characterization(s) to a trained machine learning (ML) model, which has been trained by training dataset(s) comprising previously-computed characterizations of differences in prior distributions of power consumption telemetry data obtained by the population (or from a similar population). Based on providing the new computed characterization(s) to the trained ML model, embodiments obtain a prediction as to whether or not the differences between the post-change distribution and the pre-change distribution are indicative that the configuration change introduced a performance defect in the computing platform.

**[005]** In some embodiments, methods, systems, and computer program products detect a performance defect at an electronic computing platform resulting from a configuration change within the computing platform. A first distribution of first telemetry data is obtained from a first plurality of instances of the computing platform. The first telemetry data corresponds to a pre-change configuration, and includes data corresponding to each of the first plurality of instances of the computing platform and indicating energy consumption by at least one component at the corresponding instance of the computing platform. A second distribution of second telemetry data is also obtained from a second plurality of instances of the computing platform. The second telemetry data corresponds to a post-change configuration, and includes data corresponding to each of the second plurality of instances of the computing platform and indicating energy consumption by the at least one component at the corresponding instance of the computing platform. One or more results are computed using at least a portion of the first telemetry data and at least a portion of the second telemetry data as input. The one or more results characterize one or more differences between the first distribution and the second distribution. The one or more results are input

to a trained ML model to obtain a prediction of whether the one or more differences between the first distribution and the second distribution indicate that a performance defect was introduced by the configuration change.

**[006]** As will be appreciated in view of the disclosure herein, the performance defect detection techniques described herein provide distinct technical advantages over prior techniques. For example, these techniques consume fewer computing resources than prior techniques, since they operate on relatively easy to generate telemetry data, rather than based on monitoring specific thresholds at individual computer systems and running test scenarios at individual computer systems. In addition, these techniques can detect many performance defects that would be difficult to anticipate and develop appropriate thresholds and/or test scenarios to detect. Further, since these techniques operate on telemetry data from a potentially large number of computer systems, they can detect performance defects that may be difficult to reproduce on an individual testing system.

**[007]** This summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[008]** In order to describe the manner in which the above-recited and other advantages and features of the invention can be obtained, a more particular description of the invention briefly described above will be rendered by reference to specific embodiments thereof which are illustrated in the appended drawings. Understanding that these drawings depict only typical embodiments of the invention and are not therefore to be considered to be limiting of its scope, the invention will be described and explained with additional specificity and detail through the use of the accompanying drawings in which:

**[009]** Figure 1 illustrates an example computing environment that facilitates training an ML model based on data characterizing differences in power consumption by two populations of instances of a particular computing platform;

**[010]** Figure 2 illustrates an example of data distributions based on telemetry data received from two populations of a computing platform;

**[011]** Figure 3 illustrates a flow chart of an example method for training an ML model based on data characterizing differences in power consumption by two populations of instances of a particular computing platform;

**[012]** Figure 4 illustrates an example computing environment that uses a trained ML model to identify a performance defect within a computing platform;

**[013]** Figure 5 illustrates a flow chart of an example method for detecting a performance defect at an electronic computing platform resulting from a configuration change within the computing platform; and

**[014]** Figure 6 illustrates a flow chart of an example method for calculating an averaged z-score delta.

## DETAILED DESCRIPTION

**[015]** Figure 1 illustrates an example computing environment 100 that facilitates training an ML model based on data characterizing differences in power consumption by two populations of instances of a particular computing platform. As shown, computing environment 100 represents a plurality of populations 101 (i.e., population 101a and population 101b), each made up of a plurality of device instances 102 (i.e., device instances 102a to 102a-n in population 101a and device instances 102b to 102b-n in population 101b) of a particular computing platform. In computing environment 100, population 101a represents device instances 102a at a first time/state, while population 101b comprises device instances 102b at a later second time/state. In differing embodiments, device instances 102a and device instances 102b could be the same devices (i.e., the same computing devices at both the first and second times), could be different devices (i.e., a first set of computing devices at the first time and different second set of computing devices at the second time), or there could be some overlap between device instances 102a and device instances 102b.

**[016]** As used herein, a "computing platform," refers to a hardware platform of which there can be a plurality of separate physical instances. In embodiments, a "computing platform" refers to a particular computing device model or device model family. In some examples, a "computing platform" refers to a smartphone model or model family, a tablet model or model family, a laptop computer model or model family, a desktop computer model or model family, etc. In embodiments, a device model refers to a particular hardware configuration (e.g., the Stock Keeping Unit (SKU) "Surface_Pro_X_1876", referring to the MICROSOFT SURFACE PRO X with a particular hardware build configuration), while a device model family refers to different, but related, hardware configurations (e.g., "MICROSOFT SURFACE PRO X", referring to any SURFACE PRO X regardless of the hardware build configuration). Thus, in some embodiments the device instances 102 within a given population 101 have substantially identical hardware configurations (e.g., the same model number or identifier), while in other embodiments the device instances 102 within a given population 101 have different, but related, hardware configurations (e.g., the same model family number or identifier). In embodiments, the device instances within a given population have a similar software configuration, such as the same operating system version/build. Thus,

in embodiments each of device instances 102a in population 101a operates the same operating system version (e.g., MICRSOFT WINDOWS 10 version 1903). As discussed, population 101a represents device instances 102a at a first time, while population 101b comprises device instances 102b at a later second time. Thus, in this example, each of device instances 102b in population 101b could operate the same operating system version as devices instances 102a (e.g., MICRSOFT WINDOWS 10 version 1903), or they could alternatively operate some newer operating system version (e.g., MICRSOFT WINDOWS 10 version 1909).

**[017]** In computing environment 100, the populations 101 provide telemetry data 103 (i.e., telemetry data 103a corresponding to population 101a, and telemetry data 103b corresponding to population 101b) to a learning system 104. Although not expressly shown, the populations 101 and the learning system 104 are connected via one or more communications channels, such as one or more networks, and the individual device instances 102 within those populations 101 provide the telemetry data 103 over those communications channel(s). Thus, the telemetry data 103 includes one or more individual data points from each of those device instances 102. In embodiments, the telemetry data 103 includes one or more data points such as measurements indicating energy consumption by at least one component at each device instance 102, and/or indicating energy consumption by each device instance 102 generally. Further examples of this telemetry data 103 are provided later.

**[018]** In general, the learning system 104 receives telemetry data 103a from population 101a that was sensed or generated by device instances 102a at one time, and receives telemetry data 103b from population 101b that was sensed or generated by device instances 102b at a later time. The learning system 104 uses this telemetry data 103 to create training datasets for training ML models 111 (i.e., ML model 111a to ML model 111n). These ML models 111 that are then usable later (e.g., by a prediction system 404, see Figure 4) to determine if subsequent telemetry data received from populations 101 (or similar populations of the same computing platform) indicates that a performance defect (e.g., a regression in hardware performance or a regression in software performance) was introduced by a subsequent configuration change in the computing platform, such as by a software or firmware update.

**[019]** As shown, the learning system 104 includes a training dataset generation component 105, which receives telemetry data 103a/103b from populations 101a/101b (e.g., using a data obtaining component 106). Based on this obtained telemetry data 103, the training dataset generation component 105 utilizes a characterization component 107 to characterize shifts in this telemetry data 103. For example, as mentioned, the telemetry data 103 for a population 101 can include data points, from each device instance 102 in the population 101, indicating energy consumption by at least one component of the

device instance 102, and/or indicating energy consumption by the device instance 102 generally. In some embodiments, these data points include relatively course data such as an average energy consumption rate at the device (or by a hardware component at the device), an amount of energy consumed at the device (or by a hardware component at the device), an average hardware component (e.g., processor, graphics processor, communications interface, etc.) utilization at the device, a total hardware component (e.g., processor, graphics processor, communications interface, etc.) utilization at the device, etc. In other embodiments, these data points include somewhat more granular data such as an average energy consumption caused at the device when executing a subject process, an amount of energy consumed at the device when executing the subject process, an amount of hardware component (e.g., processor, graphics processor, communications interface, etc.) utilization caused at the device when executing the subject process, etc. In some cases the telemetry data 103 is measured by an operating system of a device and sent to the learning system 104 over a communications channel.

**[020]** Based on these data points, the characterization component 107 creates data distributions for each population 101. In embodiments, the characterization component 107 creates these data distributions based on specified metrics (e.g., received by the data obtaining component 106), such as metrics identifying particular types of data points in the telemetry data 103 that are of interest. As an example, the telemetry data 103 for a device could include one or more different types of power data (e.g., power consumption rate, power consumption amount, hardware component utilization, etc.) for one or more hardware or software components at the device, and/or for the device generally. In embodiments, metrics are used to narrow down the particular type(s) of power data that are of interest, and for which component(s) and/or for the device generally. Thus, metrics can be used to focus individually on different aspects of operation of the subject computing platform.

**[021]** Figure 2 illustrates and example 200 that includes two data distributions 201a and 201b (collectively, distributions 201) based on telemetry data received from two populations of a computing platform. For example, distribution 201a could correspond to telemetry data 103a and distribution 201b and could correspond to telemetry data 103b. In the distributions 201, the X-axis represents different values reported for a given data point in the telemetry data 103 (e.g., an energy consumption rate caused by a subject software component), and the Y-axis represents a number of device instances 102 which reported those values in the telemetry data 103. Thus, data distribution 201a shows a plot 202a derived from telemetry data 103a, while data distribution 201a shows a plot 202b derived from telemetry data 103b. From plots 202a/202b (collectively plots 202), it is clear that population 101a and population 101b reported somewhat different patterns for the given data point (e.g., energy consumption rate caused by

the subject software component). As will be appreciated, the differences between plots 202a and 202b might be due to normal variances in the way in which devices instances 102 were being used over time and/or by different users. However, the differences between plots 202a and 202b might alternatively be due to some configuration change in population 101b as compared to population 101a. In particular, since populations 101a and 101b represent devices instances 102/102b at different times, device instances 102b might have undergone a configuration change—such as a software or firmware update—when compared to device instances 102a. Thus, it is possible that device instances 102b are in a "post-change" configuration, while device instances 102a are in a "pre-change" configuration. As a result, if that configuration change caused a performance defect within device instances 102b that affected energy usage at those device instances 102b, telemetry data 103b could have also been affected by that configuration change (i.e., as compared to telemetry data 103a). Thus, the differences between plots 202a and 202b might be due to that performance defect.

**[022]** In order to quantify/characterize any differences between data distribution 201a and data distribution 201b (i.e., shifts in data), the characterization component 107 generates one or more statistical results from those data distributions 201. For example, in embodiments the characterization component 107 uses data distributions 201 as input to calculate statistical results that include one or more of (i) one or more probability values (*p*-values) or (ii) one or more z-scores. In embodiments, a *p*-value indicates the probability of the truth of a null hypothesis that the inputs used to generate the *p*-value are drawn from the same distribution. Thus, in the context of distributions 201, an example null hypotheses is that, even though the particular data points in the distributions differ (e.g., due to differences how the individual device instances 102 were being used when telemetry data 103 was generated), those data points were generated by device instances of the same computing platform. In embodiments, if the calculated *p*-value(s) for two distributions 201 is below a threshold (e.g., $p < .05$), then this null hypothesis may not actually be true for those distributions 201, which could indicate that there was a configuration difference between device instances 102a and device instances 102b that introduced performance defect in device instances 102b. In embodiments, *p*-values are computed using at least one of the two-sample Kolmogorov-Smirnov test (K-S test) or Welch's *t*-test (Welch test), which tests are known and understood by those of ordinary skill in the art.

**[023]** As will be understood by one of ordinary skill in the art, a z-score (or standard score) is a signed fractional number of standard deviations by which the value of an observation or data point is above the mean value of what is being observed or measured, and is calculated by subtracting the population mean from an individual raw score and then dividing the difference by the population standard deviation. In

embodiments, the characterization component 107 calculates a z-score for each distribution 201 at one or more predefined percentiles. For example, example 200 illustrates that, in embodiments, the characterization component 107 calculates a z-score for each of distributions 201a and 201b at the 10th percentile 203a (i.e., P-10), at the 50th percentile 203b (i.e., P-50), and/or at the 90th percentile 203c (i.e., P-90). In an example, to compute the P-10 z-score for distribution 201a, the characterization component 107 calculates the distribution's mean value and the distribution's standard distribution, and identifies the distribution's value at the 10th percentile 203a (i.e., point 204a). Then, the characterization component 107 subtracts the calculated mean value from the identified value at the 10th percentile 203a, and divides the result by the calculated standard deviation. After computing z-scores at each identified percentile for both distributions 201, the characterization component 107 identifies the differences (deltas) between the z-scores at corresponding percentiles. In one example, the characterization component 107 determines the delta between the 10th percentile z-scores of distribution 201a and distribution 201b, the delta between the 50th percentiles z-scores of distribution 201a and distribution 201b, and the delta between the 90th percentiles z-scores of distribution 201a and distribution 201b. A magnitude of the size of these deltas can be an indication of an amount of data shift between distribution 201a and distribution 201b, with larger data shifts indicating that there could have been a configuration difference between device instances 102a and device instances 102b that introduced performance defect in device instances 102b.

**[024]** It was mentioned previously that, in embodiments, the data obtaining component 106 obtains metrics, such as metrics for selecting the data used to create distributions 201. In embodiments, these metrics additionally, or alternatively, include metrics for characterizing the differences between data distributions 201. For example, metrics could indicate which statistical results to compute (e.g., whether to use the K-S test, the Welch test, and/or z-score), any parameters for computing those results (e.g., at which percentile(s) to compute z-scores), and/or metrics for interpreting those results (e.g., thresholds for rejecting the null hypothesis, the magnitude of deltas between z-scores to consider significant, etc.).

**[025]** The inventors have noted that use of *p*-value and/or z-score results, while helpful, may not always be a reliable indication of whether or not there was a performance defect introduced by a configuration difference between device instances 102a and device instances 102b. For instance, in some cases, the statistical results indicate there is an abnormality between data distribution 201a and data distribution 201b when there was, in fact, no configuration difference between device instances 102a and device instances 102b. Alternatively, in some cases, the statistical results indicate there is no abnormality between data distribution 201a and data distribution 201b when there was, in fact, a performance defect

that was introduced by a configuration difference between device instances 102a and device instances 102b.

**[026]** Thus, in embodiments, data generated by the characterization component 107 is utilized as one or more training data sets that are fed to an ML training component 109 which creates one or more ML models 111 (i.e., model 111a to model 111n) that characterize these results. In embodiments, the training dataset generation component 105 operates repeatedly on different telemetry data 103 inputs to generate these training datasets. Thus, in an example, the data obtaining component 106 receives multiple unique sets of telemetry data 103 at different times and, for each set of telemetry data 103, the characterization component 107 produces a corresponding set of data distributions 201 and uses those data distributions 201 as inputs for computing *p*-value and/or z-score results as the basis of training data sets.

**[027]** In some embodiments, the ML training component 109 operates using supervised ML, which uses labeled training data sets as input. Accordingly, the training dataset generation component 105 is shown as potentially including a labeling component 108. In embodiments, the labeling component 108 tags different *p*-value and/or z-score results computed from a given set of telemetry data 103 with an indication of those results' reliability and/or with an indication as to whether those results imply a performance defect. In embodiments, the labeling component 108 operates based on user input. For example, the labeling component 108 could obtain user input from subject matter experts, such as developers or engineers who experience with recognizing whether or not differences in data distributions, including *p*-value and/or z-score results computed from those distributions, are indicative of performance defects. In additional, or alternative, embodiments the labeling component 108 operates based on knowledge associated with the input telemetry data 103. For example, at least some telemetry data 103 received from populations 101 could be specially crafted or selected to produce *p*-value and/or z-score results corresponding to no performance defect, and/or at least some additional telemetry data 103 received from populations 101 could be specially crafted or selected to produce *p*-value and/or z-score results corresponding to a performance defect, and the labeling component 108 could label the results computed from this telemetry data 103 based on knowledge of how the telemetry data 103 had been crafted/selected. To illustrate, one set of telemetry data 103 could be from two populations 101a and 101b that are known to have no configuration changes, and the labeling component 108 could use this knowledge to label one or more results from the characterization component 107 identifying a performance defect from this telemetry data 103 as being inaccurate and/or to label one or more results identifying no performance defect from this telemetry data 103 as being accurate. In another example,

another set of telemetry data 103 could be from two populations 101a and 101b in which it is known that population 101b has a performance defect as compared to population 101a, and the labeling component 108 could use this knowledge to label one or more results from the characterization component 107 identifying a performance defect from this telemetry data 103 as being accurate and/or to label one or more results identifying no performance defect from this telemetry data 103 as being inaccurate.

**[028]** In other embodiments, the ML training component 109 operates using unsupervised ML. In these embodiments, the ML training component 109 operates based on finding previously unknown patterns in training data sets that lack pre-existing labels (e.g., data sets generated by the characterization component, and without use of the labeling component 108). In an example, the ML training component 109 utilizes the density-based spatial clustering of applications with noise (DBSCAN). DBSCAN is a density-based clustering non-parametric algorithm—i.e., given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away).

**[029]** As shown, using training data set(s) produced by the training dataset generation component 105 (whether they be labeled or unlabeled), the ML training component 109 produces one or more ML models 111 (i.e., model 111a to model 111n), which are stored within model storage 110. In embodiments, the ML training component 109 utilizes at least one of a logistic regression algorithm, a support vector machine algorithm, a random forest algorithm (e.g., random forest with 100 trees), a k-nearest neighbors algorithm, a Naïve Bayes algorithm (e.g., gaussian Naïve Bayes), or DBSCAN.

**[030]** To demonstrate operation learning system 104, Figure 3 illustrates a flowchart of a method 300 for training an ML model based on labeled data characterizing differences in power consumption by two populations of instances of a particular computing platform. Method 300 will be described with respect to the components and data of computing environment 100, and the data distributions 201 of example 200.

**[031]** As shown, method 300 comprises an act 301 for obtaining data. In some embodiments act 301 includes obtaining telemetry data and, optionally, obtaining one or more metrics. In an example, the data obtaining component 106 receives telemetry data 103a from population 101a and telemetry data 103b from population 101b. In embodiments, the data obtaining component 106 also obtains additional metrics, such as from a user.

**[032]** Method 300 also comprises an act 302 of characterizing differences in the telemetry data. In an example, the characterization component 107 creates a different data distribution 201 for each population 101, such as data distribution 201a for population 101a (i.e., using telemetry data 103a) and

data distribution 201b for population 101b (i.e., using telemetry data 103b). In embodiments, creation of these data distributions 201 is based on received metrics, such as metrics selecting which type(s) of data points in the telemetry data 103 should be considered. Then, the characterization component 107 characterizes the differences between the data distributions 201 by calculating the one or more statistical results, such as by using the K-S test, the Welch test, or z-score. In embodiments, computation of these statistical results is based on received metrics, such as metrics selecting which results to calculate, parameters for those calculations, etc.

**[033]** In some embodiments, method 300 also comprises an act 303 of labeling the characterization(s). In an example, the labeling component 108 labels any statistical results calculated by the characterization component 107 with an indication of those results' reliability and/or with an indication as to whether those results indicate a performance defect. In some but not all examples, operation of the labeling component 108 is guided by user input and/or by knowledge associated with the input telemetry data 103. In some cases knowledge of the presence of a performance defect is available from other sources such as customer feedback.

**[034]** Method 300 also comprises an act 304 of training an ML model. In an example, the training dataset generation component 105 provides one or more training data sets (i.e., comprising the statistical results produced by the characterization component 107 and which, in some embodiments, have been labeled by the labeling component 108), as input to the ML training component 109. Using these training dataset(s), the ML training component 109 utilizes supervised and/or unsupervised ML to train and/or refine one or more ML models 111 that are usable to provide a prediction as to whether or not subsequent telemetry data indicates a performance defect. Notably, method 300 can be repeated any number of times, on any number of sets of telemetry data, with each iteration of method 300 further refining the predictive accuracy of the ML models 111.

**[035]** Embodiments also consume these ML models 111 in order to identify performance defects within a computing platform and, potentially, to identify a root cause of an identified performance defect. For example, Figure 4 illustrates an example computing environment 400 that uses a trained ML model to identify a performance defect within a computing platform. As shown, similar to computing environment 100, computing environment 400 also represents a plurality of populations 401 (i.e., population 401a and population 401b), each made up of a plurality of device instances 402 (i.e., device instances 402a to 402a-n in population 401a and device instances 402b to 402b-n in population 401b). In embodiments, populations 401 are made up of device instances 402 of the same computing platform as populations 101.

In embodiments, populations 401 and device instances 402 could have overlap with populations 101 and device instances 102, though they could be entirely independent from one another.

**[036]** Like populations 101, populations 401 generate telemetry data 403 (i.e., telemetry data 403a corresponding to population 401a, and telemetry data 403b corresponding to population 401b). However, in computing environment 400 the populations 401 provide this telemetry data 403 to a prediction system 404. Although not expressly shown, populations 401 and prediction system 404 are connected via one or more communications channels, such as one or more networks, and the individual device instances 402 within populations 401 provide the telemetry data 403 over those communications channel(s). In embodiments, telemetry data 403 is of substantially the same type of data as telemetry data 103—i.e., one or more data points indicating energy consumption by at least one component at each device instance 402, and/or indicating energy consumption by each device instance 402 generally. Also, like telemetry data 103, telemetry data 403 includes telemetry data 403a from population 401a that was sensed or generated by device instances 402a at one time, and telemetry data 403b from population 401b that was sensed or generated by device instances 402b at a later time.

**[037]** The prediction system 404—which could be embodied on the same computing hardware as learning system 104, or on entirely separate computing hardware—includes a data obtaining component 405 and a characterization component 406 that, in embodiments, operate in substantially the same manner as data obtaining component 106 and characterization component 107 within learning system 104. If the prediction system 404 and the learning system 104 are embodied on the same computing hardware, in embodiments the data obtaining component 405 and the characterization component 406 are the same components as the obtaining component 106 and the characterization component 107. Thus, in general, the data obtaining component 405 obtains telemetry data 403 from populations 401. In addition, the data obtaining component 405 could also obtain metric data. Based on this obtained data, the characterization component 107 creates a data distribution for each population 401, and characterizes the differences between these data distributions by calculating one or more statistical results, such as by using the K-S test, the Welch test, or z-score. Like the learning system 104, the prediction system 404 also includes a model storage 409, which includes one or more ML models 410. In embodiments, one or more of ML models 410 corresponds to one or more of the ML models 111 that were generated by the learning system 104. Thus, as an example, ML model 410a could correspond to ML model 111a, which was trained by learning system 104 using supervised and/or unsupervised ML techniques, based on telemetry data 103 obtained from populations 101.

**[038]** Unlike the learning system 104, the prediction system 404 includes a prediction component 407, and could also include a root cause analysis component 408. In embodiments, the prediction component 407 receives, as input, the statistical results calculated by the characterization component 406 as well as an ML model, such as ML model 410a. As will be understood in view of the prior description of the characterization component 107, if there was a configuration change between population 401a and population 401b, these statistical results can indicate a likelihood of a performance defect having been introduced by that configuration change. For example, statistical results including a *p*-value (e.g., computed using the KS-test or the Welch test ) can indicate if a null hypothesis that the computing platform of population 401b is the same as the computing platform of population 401a (i.e., that the configuration change did not introduce a performance defect in population 401b) should be rejected or not. Additionally, or alternatively, the magnitude of the delta(s) between z-scores can indicate whether population 401b behaves statistically similar to population 401a, or not (which can be an indication of whether or not the configuration change introduced a performance defect in population 401b). In embodiments prediction component 407 inputs these statistical results to the ML model 410a. Since ML model 410a was trained using supervised ML with labeled training data set(s) that characterized the accuracy of similar statistical results for the same, or similar, populations, and/or using unsupervised ML with unlabeled training data set(s), ML model 410a provides an ML-validated prediction of whether or not there is a performance defect in the computing platform of which device instances 402b are comprised.

**[039]** By using ML in addition to statistical analysis more accurate performance defect detection is possible as compared with using statistical analysis alone. Because the ML has good generalization ability, accuracy of performance defect detection is high even where the configuration changes were not previously observed in the training data. By using ML in addition to statistical analysis, the need for expert human to interpret the statistical analysis is reduced.

**[040]** If included, the root cause analysis component 408 uses an automated analysis to determine a cause of an identified performance defect. For example, in many instances, performance defects introduced by software or firmware updates are due to a software (i.e., code) regression. Thus, in embodiments, the root cause analysis component 408 performs a software regression detection in one or more instances of the subject computing platform.

**[041]** As will be appreciated by one of skill in the art, operating systems, complex software applications, and other types of software products are typically developed by a large number of development teams organized into a hierarchy. Each team may submit or "check in" one or more changes and/or configuration modifications (collectively referred to herein as "payloads") to source code of a software application at a

corresponding level in the hierarchy daily, weekly, bi-weekly, monthly, or in other time intervals. Subsequently, the payloads checked in can be propagated to higher levels in the hierarchy which may also include payloads from other development teams. Periodically, a version of the source code of the software product with various payloads is compiled into executable instructions as a distinct "build" of the software product.

**[042]** One or more computing devices (e.g., servers in a testing lab or client devices of users signed up for testing) then execute the build of the software product and collect various performance metrics for analysis. Examples of performance metrics include power consumption, processor load, memory consumption, execution latency, and/or other suitable types of metrics. In practice, each build of a software product typically includes a large number of payloads. For instance, individual builds of an operating system include hundreds or even thousands of payloads. As such, determining impact of one of these payloads to the measured performance metrics of the software product can be rather difficult. In addition, as time goes on, payloads from lower levels can be propagated into higher levels of the hierarchy with additional payloads from other development teams. Thus, the additional payloads can mask impact of the payloads from the lower levels and render software regression detection difficult.

**[043]** In embodiments, the root cause analysis component 408 operates as, or utilizes, a regression detector that is configured to receive data representing measured performance metrics of multiple builds of a software product and a list of payloads included in each of the builds. In certain implementations, each payload is assigned a unique identification number or other suitable forms of identification and tracked for the multiple builds of the software product. In embodiments, the regression detector is configured to perform statistical analysis of the received dataset using the identification and presence/absence of each of the payloads as a denominator. For instance, the regression detector can be configured to apply multiple linear regression to the received dataset to generate a set of regression coefficients for the respective payloads. For instance, in a dataset with one dependent variable $Y_i$ (e.g., the "Metric") and multiple independent variables $X_{ip}$ where $p$ corresponds to, e.g., different payloads (e.g., "Payload A," "Payload B," and "Payload C"), a linear relationship between the dependent variable and the independent variables can be modeled using a disturbance term or error variable $\varepsilon$, as follows:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip} + \varepsilon_i, \text{ where } i = 1, 2, \ldots, n$$

The coefficients (i.e., $\beta_0$, $\beta_1$, ..., $\beta_p$) can be estimated using various techniques, such as least squares estimation, maximum likelihood estimation, ridge regression, least absolute deviation, etc. As such, by estimating the coefficients using a suitable technique, a multiple linear model of the received dataset can be obtained.

**[044]** Using the obtained multiple linear model, the regression detector can be configured to determine which one or more of the payloads have statistically significant impact (i.e., above a noise level) to the performance metrics in the received dataset. For instance, example coefficients for the foregoing payloads may be as follows:

| Payload No. | Payload A | Payload B | Payload C |
|---|---|---|---|
| Coefficient | -0.21 | .25 | 10.4 |

In the above example, the estimated coefficients for Payload A and Payload B are -0.21 and 0.25, respectively. The estimated coefficient for Payload C is 10.4 while an intercept ($\beta_0$) for the model is 100.5. As such, the multiple linear model for the received dataset can be expressed as follows:

$$Y_i = 100.5 - 0.21X_{i1} + 0.25X_{i2} + 10.4X_{i3} + \varepsilon_i, \text{ where } i = 1, 2, \ldots, n$$

**[045]** In embodiments, the regression detector is configured to detect, identify, or select one or more payloads as impacting the performance metrics of the software product (referred to herein as a "significant payloads") from the multiple linear model based on a preset threshold. For instance, in the example above, the regression detector can be configured to select "Payload C" as a significant payload based on a threshold of ±0.5. In certain implementations, the threshold is set corresponding to a noise level in the received dataset. In other implementations, the threshold is set in other suitable manners with other suitable values. In embodiments, when the regression detector detects, identifies, or selects one or more payloads as impacting the performance metrics of the software product, the root cause analysis component 408 identifies those payload(s) as a potential root cause of the performance defect. In embodiments, the root cause analysis component 408 further identifies the code check-in(s) that introduced those payload(s).

**[046]** To further demonstrate operation of prediction system 404, Figure 5 illustrates a flow chart of an example method 500 for detecting a performance defect at an electronic computing platform resulting from a configuration change within the computing platform. Method 500 will be described with primarily respect to the components and data of computing environment 400. In embodiments, when referring to a configuration change within a computing platform, this configuration change comprises at least one of a software configuration change within the computing platform or a firmware configuration change within the computing platform.

**[047]** As shown, method 500 comprises an act 501a of obtaining a pre-change telemetry distribution from a device population, and an act 501b of obtaining a post-change telemetry distribution from a device population. In some embodiments act 501a includes obtaining a first distribution of first telemetry data

obtained from a first plurality of instances of the computing platform, the first telemetry data corresponding to a pre-change configuration, the first telemetry data including data corresponding to each of the first plurality of instances of the computing platform and indicating energy consumption by at least one component at the corresponding instance of the computing platform, while act 501b includes obtaining a second distribution of second telemetry data obtained from a second plurality of instances of the computing platform, the second telemetry data corresponding to a post-change configuration, the second telemetry data including data corresponding to each of the second plurality of instances of the computing platform and indicating energy consumption by the at least one component at the corresponding instance of the computing platform. In an example, the data obtaining component 405 obtains pre-change telemetry data 403a from population 401a (i.e., a plurality of device instances 402a of a particular computing platform), and obtains post-change telemetry data 403b from population 401b (i.e., a plurality of device instances 402b of the particular computing platform). In embodiments, populations 401a and 401b have partially, or fully, overlapping device instances 402. Thus, in embodiments, a first set of devices comprising the first plurality of instances of the computing platform overlaps with a second set of devices comprising the second plurality of instances of the computing platform. However, in other embodiments, populations 401a and 401b could be entirely separate sets of devices.

**[048]** Acts 501a and 501b are illustrated as having no particular ordering among one another. As such, even though telemetry data 403a is from prior to a configuration change, and telemetry data 403a is from after the configuration change, the data obtaining component 405 could obtain telemetry data 403a prior to obtaining telemetry data 403b, could obtain telemetry data 403a subsequent to obtaining telemetry data 403b, and/or could obtain telemetry data 403a and telemetry data 403b at least partially in parallel.

**[049]** As discussed, examples of telemetry data 403 includes data points relating to energy consumption at each of those device instances 402, such as energy consumption by the device as a whole, energy consumption by one or more hardware components of the device, and/or energy consumption caused by one or more software components running at the device. Thus, in embodiments, the data indicating energy consumption by at least one component at a corresponding instance of the computing platform comprises at least one of data indicating energy consumption by a hardware device at the corresponding instance of the computing platform, or data indicating energy consumption caused by executing a software process at the corresponding instance of the computing platform.

**[050]** In embodiments, the first plurality of instances of the computing platform (e.g., device instances 402a) and the second plurality of instances of the computing platform (e.g., device instances 402b) share

a substantially identical hardware configurations. For example, each instance of the computing platform could have the same hardware configuration (e.g., the same model identifier, such as SKU). Thus, in embodiments, the first plurality of instances of the computing platform comprise a first plurality of computing devices having a common model identifier, and the second plurality of instances of the computing platform comprise a second plurality of computing devices having the common model identifier. However, instances of the computing platform could have substantially similar, but not necessarily identical, hardware configurations (e.g., being part of the same model family).

**[051]** Method 500 also comprises an act 502 of characterizing shift(s) between the pre-change and post-change telemetry distributions. In some embodiments act 502 includes computing one or more results using at least a portion of the first telemetry data and at least a portion of the second telemetry data as input, the one or more results characterizing one or more differences between the first distribution and the second distribution. In an example, the characterization component 406 identifies a first data distribution of one or more data points from the first telemetry data, and identifies a second data distribution of one or more data points from the second telemetry data. Then, the characterization component 406 uses these data distributions as inputs for computing one or more statistical results. As discussed, in embodiments the statistical results are produced by one or more of the two-sample K-S test, the Welch test, or z-score. Thus, in embodiments, the one or more results comprise at least one of a z-score at a particular percentile of the first distribution and the second distribution, a first *p*-value generated by a Welch test, or a second *p*-value generated by a K-S test.

**[052]** Notably, when using z-scores, the inventors have observed that accurate results are obtained in an efficient manner where z-scores are calculated at one or more of the $10^{th}$ percentile (i.e., P-10), the $50^{th}$ percentile (i.e., P-50), or the $90^{th}$ percentile (i.e., P-90), though other percentiles could be used. Thus, in embodiments, the one or more results comprise a z-score at one or more of a $10^{th}$ percentile, a $50^{th}$ percentile, or a $90^{th}$ percentile of the first distribution and the second distribution. In addition, when using z-scores, the inventors have observed it is be less accurate to calculate z-scores at one or more of the $0^{th}$ percentile (i.e., P-0), the $25^{th}$ percentile (i.e., P-20), the $75^{th}$ percentile (i.e., P-75), or the $100^{th}$ percentile (i.e., P-100). Thus, in embodiments, the one or more results exclude a z-score at one or more of a $0^{th}$ percentile, a $25^{th}$ percentile, a $75^{th}$ percentile, or a $100^{th}$ percentile of the first distribution and the second distribution.

**[053]** Notably, rather than relying on raw z-scores, and the deltas therebetween, the inventors have observed it is beneficial in some cases to use an averaged z-score delta based on mixing and iterating over the subject distributions. This process is described later in connection with Figure 6.

**[054]** Method 500 also comprises an act 503 of inputting characterization(s) to a trained ML model. In some embodiments act 503 includes inputting the one or more results to a trained ML model. In an example, the characterization component 406 provides the results calculated in act 502 to the prediction component 407. Since making a prediction solely on these results can at times be unreliable, the prediction component 407 provides these results an ML model (e.g., ML model 410a). In embodiments, this ML model has previously been trained by the learning system 104 using telemetry data 103 obtained from populations 101 that are the same as, or similar to, populations 401. Thus, in embodiments, the trained ML model has been trained using training data comprising statistical results computed from one or more portions of historical first and second telemetry data. In some embodiments, the statistical results are labeled with known indications of performance defects introduced by historical configuration changes associated with the one or more portions of historical first and second telemetry data. In other embodiments, the statistical results are unlabeled. In embodiments, the trained ML model comprises at least one of a logistic regression algorithm, a support vector machine algorithm, a random forest algorithm, a k-nearest neighbors algorithm, a Naïve Bayes algorithm, or a DBSCAN algorithm, and is trained based on training data comprising one or more of a first set of z-score values, a second set of *p*-values generated by a Welch test, or a third set of *p*-values generated by a K-S test.

**[055]** Method 500 also comprises an act 504 of obtaining a prediction of whether the shift(s) indicate that the configuration change introduced a performance defect. In some embodiments act 504 includes obtaining a prediction of whether the one or more differences between the first distribution and the second distribution indicate that a performance defect was introduced by the configuration change. In an example, based on the prediction component 407 having provided the statistical results an ML model (e.g., ML model 410a) in act 503, the ML model provides an ML-validated prediction of whether or not there is a performance defect in the computing platform of which device instances 402b are comprised. As mentioned, a performance defect can be a defect caused by the configuration change. In the case of the configuration change being a software or firmware update, the performance defect could comprise at least one of a regression in hardware performance or a regression in software performance.

**[056]** In some embodiments, method 500 also comprises an act 505 of, based on the prediction indicating that the configuration change introduced a performance defect, identifying a root cause of the performance defect. In some embodiments act 505 includes, based on an indication that a performance defect was introduced by the configuration change, identifying a root cause of the performance defect by analyzing the configuration change. In an example, the root cause analysis component 408 performs one or more types of automated analysis to identify a potential root cause for the performance defect. As an

example, the root cause analysis component 408 might perform a regression detection, as discussed above, to identify a regression in a software or firmware code update, whether that be due to a change in code or a change in configuration. As such, in method 500, identifying the root cause of the performance defect could comprise at least one of determining that a firmware code difference occurred from the pre-change configuration to the post-change configuration, determining that a firmware configuration difference occurred from the pre-change configuration to the post-change configuration, determining that a software code difference occurred from the pre-change configuration to the post-change configuration, or determining that a software configuration difference occurred from the pre-change configuration to the post-change configuration. In addition, when the root cause is a code change that cause a regression, the root cause analysis component 408 might identify a code check-in that introduced that code change. Thus, determining that a software code or a firmware code difference caused the performance defect could comprise identifying a software or firmware check-in that caused a regression.

**[057]** As mentioned, the inventors have observed it can be more accurate to use an averaged z-score delta based on mixing and iterating over the subject distributions, because performance defects are detected more accurately as compared to using a regular, non-averaged z-score delta. To demonstrate this process, Figure 6 illustrates a flow chart of an example method 600 for calculating an averaged z-score delta. As discussed, a classic z-score is calculated by subtracting the population mean from an individual raw score and then dividing the difference by the population standard deviation. In method 600, an averaged z-score delta at a particular percentile of two distributions is computed by dividing the z-score delta at a particular percentile by a standard deviation computed based on iteratively mixing the two distributions and computing z-score deltas.

**[058]** As shown, method 600 comprises an act 601 of identifying a particular z-score percentile and an iteration count. In an example, the characterization component 406 identifies a subject percentile, such as P-10, P-50, P-90, and the like, as well as a number of times to iterate within method 600 (e.g., 1,000 iterations, 10,000 iterations, 100,000 iterations, etc.).

**[059]** Method 600 also comprises an act 602 of, at the particular z-score percentile, calculating an original delta between the pre-change and post-change telemetry distributions. In some embodiments act 602 includes calculating an original delta between the first distribution and the second distribution at the particular percentile. In an example, the characterization component 406 calculates a first z-score for a first distribution at the subject percentile, and calculate a second z-score for a second distribution at the subject percentile. Then, the characterization component 406 calculates a delta between this first and second z-scores (e.g., as an absolute value of the difference between those z-scores).

**[060]** Method 600 also comprises an act 603 of mixing data points from the pre-change and the post-change telemetry distributions. In some embodiments act 603 includes mixing data points from the first telemetry data and the second telemetry data. In an example, the characterization component 406 mixes the data points from the first and second distributions used in act 602 to create a new mixed distribution.

**[061]** Method 600 also comprises an act 604 of creating two distinct mixed telemetry distributions from the mixed data points. In some embodiments act 604 includes dividing the mixed data points to create a first mixed distribution and a second mixed distribution. In an example, the characterization component 406 divides the mixed distribution into a first new distribution and a second new distribution. In embodiments, this division is performed randomly. Thus, in embodiments of act 604, dividing the mixed data points to create the first mixed distribution and the second mixed distribution comprises randomly dividing the mixed data points. However, in other embodiments the division is done with some predictable pattern that would nonetheless create a different first mixed distribution and second mixed distribution with each iteration of act 604.

**[062]** Method 600 also comprises an act 605 of, at the particular z-score percentile, calculating a corresponding delta between the mixed telemetry distributions. In some embodiments act 605 includes calculating corresponding delta between the first mixed distribution and the second mixed distribution at the particular percentile. In an example, the characterization component 406 calculates a delta between the first mixed distribution and the second mixed distribution, at the particular percentile, in the same manner that it calculated the original delta in act 602.

**[063]** Method 600 also comprises an act 606 of saving the corresponding delta. In an example, the characterization component 406 saves the delta computed in act 605 in some list or database.

**[064]** Method 600 also comprises an act 607 of determining if the iteration count met. If the iteration count was met, method 600 proceeds to act 608. Otherwise, if the iteration count was not met, act 607 includes incrementing the iteration count and returning to act 604 (i.e., for creating two distinct mixed telemetry distributions from the mixed data points). Thus, method 600 repeats acts 604 through 606 until the iteration count is met. Since, in act 604, the two mixed telemetry distributions are created randomly or with some predictable pattern, each iteration of act 604 creates a distinct set of mixed distributions. Then, in acts 605 and 606 a different corresponding delta is created, and saved, based on these mixed distributions.

**[065]** After iterating, act 608 comprises calculating a standard deviation among the corresponding deltas. In an example, the characterization component 406 computes a standard deviation among all of the deltas saved in act 606 across the iterations of acts 604 to 606.

**[066]** Method 600 also comprises an act 609 of dividing the original delta by the standard deviation. In an example, the characterization component 406 divides the original delta calculated in act 602 by the standard deviation calculated in act 608, resulting in a weighted delta between the original two distributions at the selected percentile.

**[067]** Accordingly, the embodiments described herein detect performance defects in a computing platform based on analyzing power consumption data obtained from a population of a plurality of instances of the computing platform. After a configuration change in the computing platform, these embodiments obtain a post-change distribution of power consumption telemetry data from the population of instances of the computing platform, and compare with this post-change distribution of power consumption telemetry data with a pre-change distribution data of power consumption telemetry data generated by that population (or from a similar population) prior to the configuration change. The comparison computes one or more characterizations of differences in the distribution of power consumption telemetry data found in the pre-change distribution as compared to the distribution of power consumption telemetry data found in the post-change distribution. These computed characterization(s) are provided to a trained ML model, which produces a prediction as to whether or not the differences between the post-change distribution and the pre-change distribution are indicative that the configuration change introduced a performance defect in the computing platform.

**[068]** The embodiments described herein can consume fewer computing resources than prior techniques, since they operate on relatively easy to generate telemetry data (i.e., power consumption data), rather than monitoring specific thresholds at individual computer systems and running test scenarios at individual computer systems. In addition, they can detect many performance defects that would be difficult to anticipate and develop appropriate thresholds and/or test scenarios for. Further, since the performance defect detection techniques described herein operate on telemetry data from a potentially large number of computer systems, they can detect performance defects that may be difficult to reproduce.

**[069]** Notably, while the embodiments described herein can operate on any granularity of telemetry data, a distinct advantage of the embodiments described herein is that they can reliably detect performance defects in a computing environment even when the telemetry data obtained from instances of that computing environment is relatively course (e.g., power consumption data relating to a device or device component generally, rather than detailed information about how applications and processes are executing) or even noisy. Use of course data enables technical efficiencies across the board. At the instances of the subject computing environment, gathering and transmitting courser data versus more

granular data conserves processing resources, memory resources, storage resources, and communications bandwidth. At the prediction system receiving and processing courser data also conserves processing resources, memory resources, storage resources, and communications bandwidth.

**[070]** In addition, by operating on relatively course data the embodiments described herein can also provide distinct privacy advantages as compared to conventional performance defect detection techniques. For instance, even if the embodiments herein operate on telemetry data that includes the power consumption caused by executing a particular process at a computing device, this power consumption information contains no sensitive information about how the process was actually being used or what it was doing at the computing device. For instance, if the process in question corresponds to a web browser, information about the power usage that the web browser caused provides no sensitive information about how the web browser was actually being used (e.g., there is no telemetry on users, content consumed, websites visited, a number of tabs open, etc.). Nonetheless, the embodiments herein can use this course power consumption data to detect performance defects introduced by updates to that web browser.

**[071]** Although the subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the described features or acts described above, or the order of the acts described above. Rather, the described features and acts are disclosed as example forms of implementing the claims. In addition, although the methodological acts may have been discussed in a certain order, and may have been illustrated in a flow chart as occurring in a particular order, no particular ordering is required unless specifically stated, or required because an act is dependent on another act being completed prior to the act being performed.

**[072]** Embodiments of the present invention may comprise or utilize a special-purpose or general-purpose computer system that includes computer hardware, such as, for example, one or more processors and system memory, as discussed in greater detail below. Embodiments within the scope of the present invention also include physical and other computer-readable media for carrying or storing computer-executable instructions and/or data structures. Such computer-readable media can be any available media that can be accessed by a general-purpose or special-purpose computer system. Computer-readable media that store computer-executable instructions and/or data structures are computer storage media. Computer-readable media that carry computer-executable instructions and/or data structures are transmission media. Thus, by way of example, and not limitation, embodiments of the

invention can comprise at least two distinctly different kinds of computer-readable media: computer storage media and transmission media.

**[073]** Computer storage media are physical storage media that store computer-executable instructions and/or data structures. Physical storage media include computer hardware, such as RAM, ROM, EEPROM, solid state drives ("SSDs"), flash memory, phase-change memory ("PCM"), optical disk storage, magnetic disk storage or other magnetic storage devices, or any other hardware storage device(s) which can be used to store program code in the form of computer-executable instructions or data structures, which can be accessed and executed by a general-purpose or special-purpose computer system to implement the disclosed functionality of the invention.

**[074]** Transmission media can include a network and/or data links which can be used to carry program code in the form of computer-executable instructions or data structures, and which can be accessed by a general-purpose or special-purpose computer system. A "network" is defined as one or more data links that enable the transport of electronic data between computer systems and/or modules and/or other electronic devices. When information is transferred or provided over a network or another communications connection (either hardwired, wireless, or a combination of hardwired or wireless) to a computer system, the computer system may view the connection as transmission media. Combinations of the above should also be included within the scope of computer-readable media.

**[075]** Further, upon reaching various computer system components, program code in the form of computer-executable instructions or data structures can be transferred automatically from transmission media to computer storage media (or vice versa). For example, computer-executable instructions or data structures received over a network or data link can be buffered in RAM within a network interface module (e.g., a "NIC"), and then eventually transferred to computer system RAM and/or to less volatile computer storage media at a computer system. Thus, it should be understood that computer storage media can be included in computer system components that also (or even primarily) utilize transmission media.

**[076]** Computer-executable instructions comprise, for example, instructions and data which, when executed at one or more processors, cause a general-purpose computer system, special-purpose computer system, or special-purpose processing device to perform a certain function or group of functions. Computer-executable instructions may be, for example, binaries, intermediate format instructions such as assembly language, or even source code.

**[077]** Those skilled in the art will appreciate that the invention may be practiced in network computing environments with many types of computer system configurations, including, personal computers, desktop computers, laptop computers, message processors, hand-held devices, multi-processor systems,

microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, mobile telephones, PDAs, tablets, pagers, routers, switches, and the like. The invention may also be practiced in distributed system environments where local and remote computer systems, which are linked (either by hardwired data links, wireless data links, or by a combination of hardwired and wireless data links) through a network, both perform tasks. As such, in a distributed system environment, a computer system may include a plurality of constituent computer systems. In a distributed system environment, program modules may be located in both local and remote memory storage devices.

**[078]** Those skilled in the art will also appreciate that the invention may be practiced in a cloud computing environment. Cloud computing environments may be distributed, although this is not required. When distributed, cloud computing environments may be distributed internationally within an organization and/or have components possessed across multiple organizations. In this description and the following claims, "cloud computing" is defined as a model for enabling on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services). The definition of "cloud computing" is not limited to any of the other numerous advantages that can be obtained from such a model when properly deployed.

**[079]** A cloud computing model can be composed of various characteristics, such as on-demand self-service, broad network access, resource pooling, rapid elasticity, measured service, and so forth. A cloud computing model may also come in the form of various service models such as, for example, Software as a Service ("SaaS"), Platform as a Service ("PaaS"), and Infrastructure as a Service ("IaaS"). The cloud computing model may also be deployed using different deployment models such as private cloud, community cloud, public cloud, hybrid cloud, and so forth.

**[080]** Some embodiments, such as a cloud computing environment, may comprise a system that includes one or more hosts that are each capable of running one or more virtual machines. During operation, virtual machines emulate an operational computing system, supporting an operating system and perhaps one or more other applications as well. In some embodiments, each host includes a hypervisor that emulates virtual resources for the virtual machines using physical resources that are abstracted from view of the virtual machines. The hypervisor also provides proper isolation between the virtual machines. Thus, from the perspective of any given virtual machine, the hypervisor provides the illusion that the virtual machine is interfacing with a physical resource, even though the virtual machine only interfaces with the appearance (e.g., a virtual resource) of a physical resource. Examples of physical resources including processing capacity, memory, disk space, network bandwidth, media drives, and so forth.

**[081]** The present invention may be embodied in other specific forms without departing from its essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope. When introducing elements in the appended claims, the articles "a," "an," "the," and "said" are intended to mean there are one or more of the elements. The terms "comprising," "including," and "having" are intended to be inclusive and mean that there may be additional elements other than the listed elements.

## CLAIMS

What is claimed:

1. A method, implemented at a computer system (404) that includes at least one processor, for detecting a performance defect at an electronic computing platform resulting from a configuration change within the computing platform, the method comprising:

obtaining (405) a first distribution of first telemetry data (403a) obtained from a first plurality of instances (402a) of the computing platform, the first telemetry data corresponding to a pre-change configuration, the first telemetry data including data corresponding to each of the first plurality of instances of the computing platform and indicating energy consumption by at least one component at the corresponding instance of the computing platform;

obtaining (405) a second distribution of second telemetry data (403b) obtained from a second plurality of instances (402b) of the computing platform, the second telemetry data corresponding to a post-change configuration, the second telemetry data including data corresponding to each of the second plurality of instances of the computing platform and indicating energy consumption by the at least one component at the corresponding instance of the computing platform;

computing (406) one or more results using at least a portion of the first telemetry data and at least a portion of the second telemetry data as input, the one or more results characterizing one or more differences between the first distribution and the second distribution; and

inputting (407) the one or more results to a trained machine learning model (410) to obtain a prediction of whether the one or more differences between the first distribution and the second distribution indicate that a performance defect was introduced by the configuration change.

2. The method of claim 1 wherein the trained machine learning model has been trained using training data comprising statistical results computed from one or more portions of historical first and second telemetry data, the statistical results labeled with known indications of performance defects introduced by historical configuration changes associated with the one or more portions of historical first and second telemetry data.

3. The method of any preceding claim, wherein the one or more results comprise at least one of a z-score at a particular percentile of the first distribution and the second distribution, a first *p*-value generated by a Welch test, or a second *p*-value generated by a Kolmogorov-Smirnov test.

4. The method of claim 3, further comprising calculating a z-score delta based on at least:

a) calculating an original delta between the first distribution and the second distribution at the particular percentile;

b) mixing data points from the first telemetry data and the second telemetry data;

c) dividing the mixed data points to create a first mixed distribution and a second mixed distribution;

d) calculating corresponding delta between the first mixed distribution and the second mixed distribution at the particular percentile;

e) repeating b) through d) a plurality of times while creating a distinct first mixed distribution for each iteration and creating a distinct second mixed distribution for each iteration;

f) calculating a standard deviation among the calculated corresponding deltas; and

g) dividing the original delta by the standard deviation.

5. The method of claim 4, wherein dividing the mixed data points to create the first mixed distribution and the second mixed distribution comprises randomly dividing the mixed data points.

6. The method of any preceding claim, wherein the performance defect comprises at least one of a regression in hardware performance or a regression in software performance.

7. The method of any preceding claim, wherein the first plurality of instances of the computing platform comprises a first plurality of computing devices having a common model identifier, and wherein the second plurality of instances of the computing platform comprises a second plurality of computing devices having the common model identifier.

8. The method of any preceding claim, wherein the data indicating energy consumption by at least one component at a corresponding instance of the computing platform comprises at least one of:

data indicating energy consumption by a hardware device at the corresponding instance of the computing platform; or

data indicating energy consumption caused by executing a software process at the corresponding instance of the computing platform.

9. The method of any preceding claim, wherein the trained machine learning model comprises at least one of a logistic regression algorithm, a support vector machine algorithm, a random forest algorithm, a *k*-nearest neighbors algorithm, or a Naïve Bayes algorithm.

10. The method of any preceding claim, wherein the trained machine learning model is trained based on training data comprising one or more of a first set of z-score values, a second set of *p*-values generated by a Welch test, or a third set of *p*-values generated by a Kolmogorov-Smirnov test.

11. The method of any preceding claim, wherein a first set of devices comprising the first plurality of instances of the computing platform overlaps with a second set of devices comprising the second plurality of instances of the computing platform.

12. The method of any preceding claim, wherein the configuration change comprises at least one of a software configuration change or a firmware configuration change.

13. The method of any preceding claim, further comprising, based on an indication that a performance defect was introduced by the configuration change, identifying a root cause of the performance defect by analyzing the configuration change.

14. The method of claim 13, wherein identifying the root cause of the performance defect comprises at least one of:

determining that a firmware code difference occurred from the pre-change configuration to the post-change configuration;

determining that a firmware configuration difference occurred from the pre-change configuration to the post-change configuration;

determining that a software code difference occurred from the pre-change configuration to the post-change configuration; or

determining that a software configuration difference occurred from the pre-change configuration to the post-change configuration.

15. The method of claim 14, wherein identifying the root cause of the performance defect comprises determining that a software code difference occurred from the pre-change configuration to the post-change configuration, wherein determining that the software code difference caused the performance defect comprises identifying a software check-in that caused a regression.

## REVENDICATIONS



Ce qui est revendiqué :

1. Un procédé, mis en œuvre au niveau d'un système informatique (404) incluant au moins un processeur, destiné à détecter un défaut de performance au niveau d'une plateforme informatique électronique résultant d'un changement de configuration dans la plateforme informatique, le procédé comprenant :

l'obtention (405) d'une première distribution de premières données de télémétrie (403a) obtenues à partir d'une première pluralité d'instances (402a) de la plateforme informatique, les premières données de télémétrie correspondant à une configuration avant changement, les premières données de télémétrie comprenant des données correspondant à chacune des instances de la première pluralité de la plateforme informatique et indiquant la consommation d'énergie par au moins un composant au niveau de l'instance correspondante de la plateforme informatique ;

l'obtention (405) d'une deuxième distribution de deuxièmes données de télémétrie (403b) obtenues à partir d'une deuxième pluralité d'instances (402b) de la plateforme informatique, les deuxièmes données de télémétrie correspondant à une configuration après changement, les deuxièmes données de télémétrie comprenant des données correspondant à chacune des instances de la deuxième pluralité de la plateforme informatique et indiquant la consommation d'énergie par au moins un composant au niveau de l'instance correspondante de la plateforme informatique ;

le calcul (406) d'un ou plusieurs résultats à l'aide d'au moins une partie des premières données de télémétrie et au moins une partie des deuxièmes données de télémétrie comme entrées, le ou les résultats caractérisant une ou plusieurs différences entre la première distribution et la deuxième distribution ; et

l'introduction (407) du ou des résultats dans un modèle entraîné d'apprentissage machine (410) afin de prédire si la ou les différences entre la première distribution et la deuxième distribution indiquent qu'un défaut de performance a été causé par le changement de configuration.

2. Le procédé selon la revendication 1, dans lequel le modèle entraîné d'apprentissage machine a été entraîné à l'aide de données d'entraînement comprenant des résultats statistiques calculés à partir d'une ou plusieurs parties de premières et deuxièmes données de télémétrie historiques, les résultats statistiques étiquetés avec des indications connues de défauts de performance causés par des changements historiques de configuration associés à la ou aux parties de premières et deuxièmes données de télémétrie historiques.

3. Procédé selon l'une quelconque des revendications précédentes, dans lequel le ou les résultats comprennent au moins un score z à un centile particulier de la première et la deuxième distribution, une première valeur *p* générée par un test de Welch ou une deuxième valeur *p* générée par un test de Kolmogorov-Smirnov.

4. Procédé selon la revendication 3, comprenant en outre le calcul d'un delta de score z basé sur au moins :

a) le calcul d'un delta initial entre la première distribution et la deuxième distribution au centile particulier ;

b) le mélange de points de données provenant des premières données de télémétrie et des deuxièmes données de télémétrie ;

c) la division des points de données mélangés pour créer une première distribution mélangée et une deuxième distribution mélangée ;

d) le calcul d'un delta correspondant entre la première distribution mélangée et la deuxième distribution mélangée au centile particulier ;

e) la répétition de b) à d) plusieurs fois tout en créant une première distribution mélangée distincte pour chaque itération et en créant une deuxième distribution mélangée distincte pour chaque itération ;

f) le calcul d'un écart type parmi les deltas calculés correspondants ; et

g) la division du delta initial par l'écart type.

5. Le procédé selon la revendication 4, dans lequel la division des points de données mélangés pour créer la première distribution mélangée et la deuxième distribution mélangée comprend la division aléatoire des points de données mélangés.

6. Le procédé selon l'une quelconque des revendications précédentes, dans lequel le défaut de performance comprend au moins une régression de la performance matérielle ou une régression de la performance logicielle.

7. Le procédé selon l'une quelconque des revendications précédentes, dans lequel la première pluralité d'instances de la plateforme informatique comprend une première pluralité de dispositifs informatiques ayant un identifiant de modèle commun, et dans lequel la deuxième pluralité d'instances de la plateforme informatique comprend une deuxième pluralité de dispositifs informatiques ayant l'identifiant de modèle commun.

8. Le procédé selon l'une quelconque des revendications précédentes, dans lequel les données indiquant la consommation d'énergie d'au moins un composant d'une instance correspondante de la plateforme informatique comprennent au moins :

des données indiquant la consommation d'énergie d'un dispositif matériel de l'instance correspondante de la plateforme informatique ; ou

des données indiquant la consommation d'énergie causée par l'exécution d'un processus logiciel au niveau de l'instance correspondante de la plateforme informatique.

9. Le procédé selon l'une quelconque des revendications précédentes, dans lequel le modèle entraîné d'apprentissage machine comprend au moins un algorithme de régression logistique, un algorithme de machine à vecteurs de support, un algorithme de forêts aléatoires, un algorithme des plus proches voisins *k* ou un algorithme par classification bayésienne naïve.

10. Le procédé selon l'une quelconque des revendications précédentes, dans lequel le modèle entraîné d'apprentissage machine est entraîné sur la base de données d'entraînement comprenant un ou plusieurs éléments d'un premier ensemble de valeurs de score z, d'un deuxième ensemble de valeurs *p* générées par un test de Welch ou d'un troisième ensemble de valeurs *p* générées par un test de Kolmogorov-Smirnov.

11. Le procédé selon l'une quelconque des revendications précédentes, dans lequel un premier ensemble de dispositifs comprenant la première pluralité d'instances de la plateforme informatique chevauche un deuxième ensemble de dispositifs comprenant la deuxième pluralité d'instances de la plateforme informatique.

12. Le procédé selon l'une quelconque des revendications précédentes, dans lequel le changement de configuration comprend au moins un changement de configuration de logiciel ou un changement de configuration de micrologiciel. LU101632

13. Le procédé selon l'une quelconque des revendications précédentes, comprenant en outre, sur la base d'une indication qu'un défaut de performance a été causé par le changement de configuration, l'identification d'une cause profonde du défaut de performance en analysant le changement de configuration.

14. Le procédé selon la revendication 13, dans lequel l'identification de la cause profonde du défaut de performance comprend au moins l'un des éléments suivants :

le fait de déterminer qu'une différence de code de micrologiciel est survenue entre la configuration avant changement et la configuration après changement ;

le fait de déterminer qu'une différence de configuration de micrologiciel est survenue entre la configuration avant changement et la configuration après changement ;

le fait de déterminer qu'une différence de code de logiciel est survenue entre la configuration avant changement et la configuration après changement ; ou

le fait de déterminer qu'une différence de configuration de logiciel est survenue entre la configuration avant changement et la configuration après changement.

15. Le procédé selon la revendication 14, dans lequel l'identification de la cause profonde du défaut de performance comprend le fait de déterminer qu'une différence de code de logiciel est survenue entre la configuration avant changement et la configuration après changement, dans lequel le fait de déterminer que la différence de code de logiciel a provoqué le défaut de performance comprend l'identification d'un enregistrement de logiciel qui a provoqué une régression.

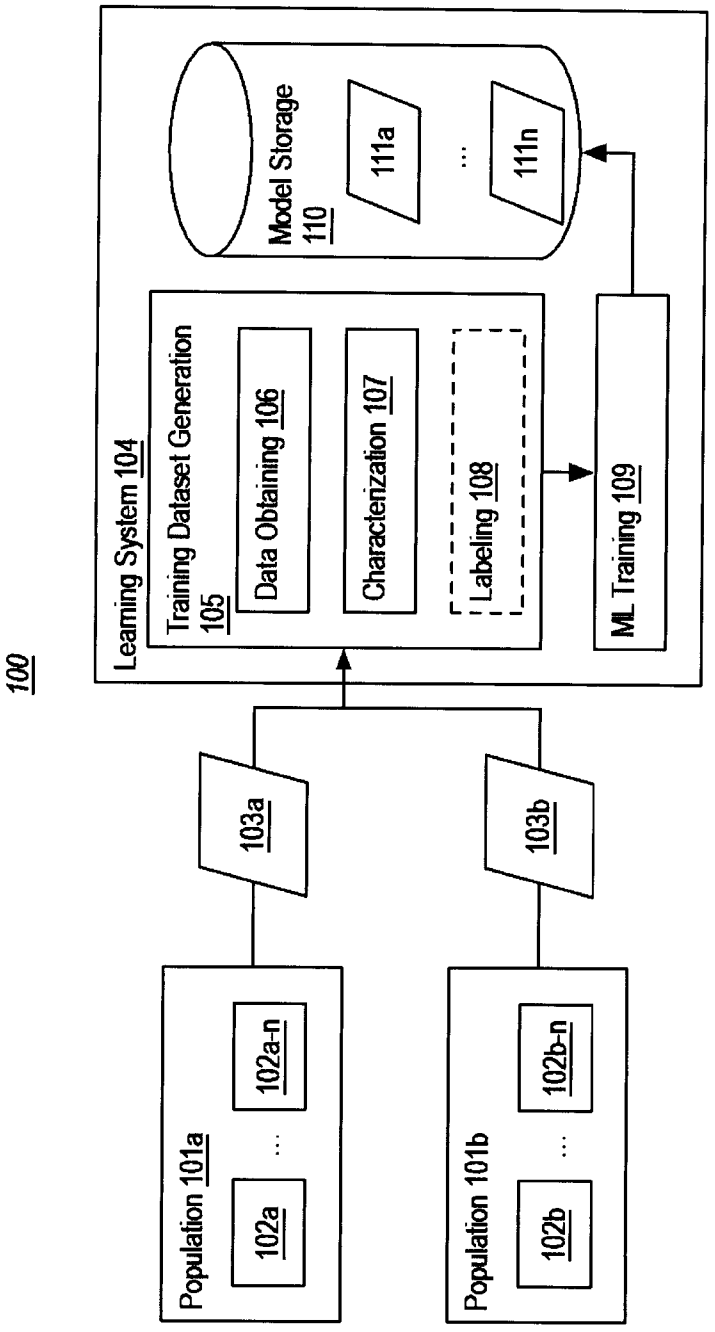
100
Learning System 104
Training Dataset Generation 105
Data Obtaining 106
Characterization 107
Labeling 108
ML Training 109
Model Storage 110
111a
...
111n
103a
103b
Population 101a
102a
...
102a-n
Population 101b
102b
...
102b-n

FIG. 1

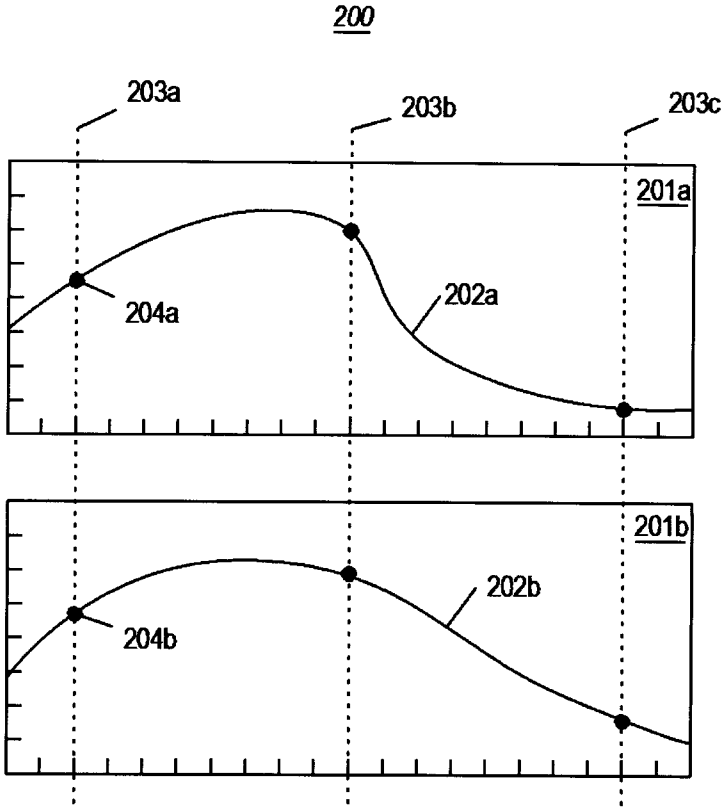


FIG. 2

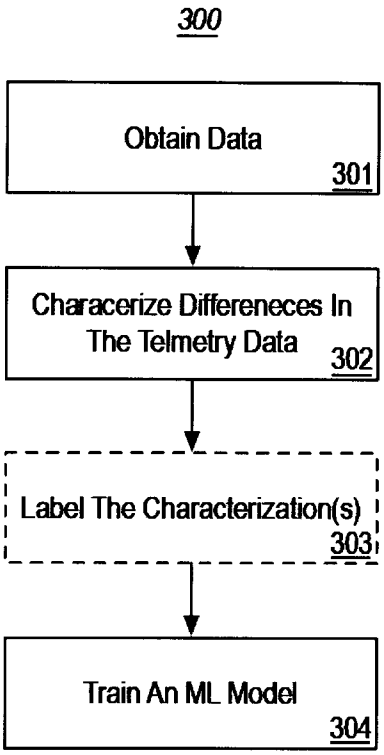
300
Obtain Data
301
Characerize Differeneces In The Telmetry Data
302
Label The Characterization(s)
303
Train An ML Model
304

FIG. 3

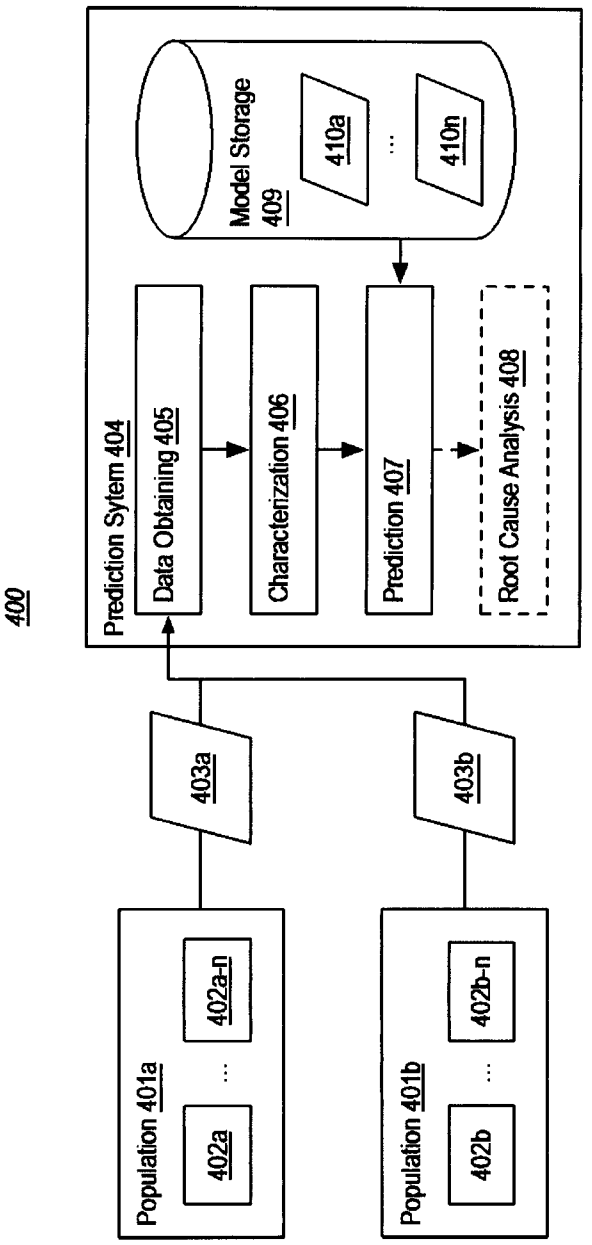
400
Prediction Sytem 404
Data Obtaining 405
Characterization 406
Prediction 407
Root Cause Analysis 408
Model Storage 409
410a
410n
403a
403b
Population 401a
402a
402a-n
Population 401b
402b
402b-n

FIG. 4

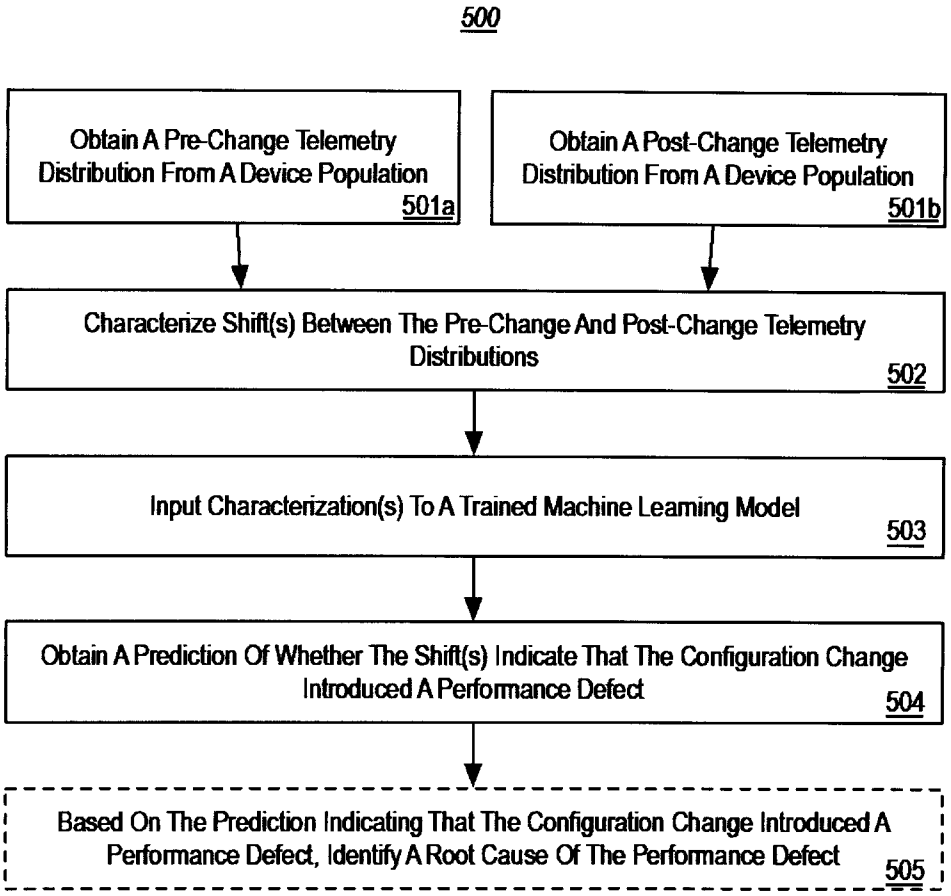
500
Obtain A Pre-Change Telemetry Distribution From A Device Population
501a
Obtain A Post-Change Telemetry Distribution From A Device Population
501b
Characterize Shift(s) Between The Pre-Change And Post-Change Telemetry Distributions
502
Input Characterization(s) To A Trained Machine Learning Model
503
Obtain A Prediction Of Whether The Shift(s) Indicate That The Configuration Change Introduced A Performance Defect
504
Based On The Prediction Indicating That The Configuration Change Introduced A Performance Defect, Identify A Root Cause Of The Performance Defect
505

*FIG. 5*

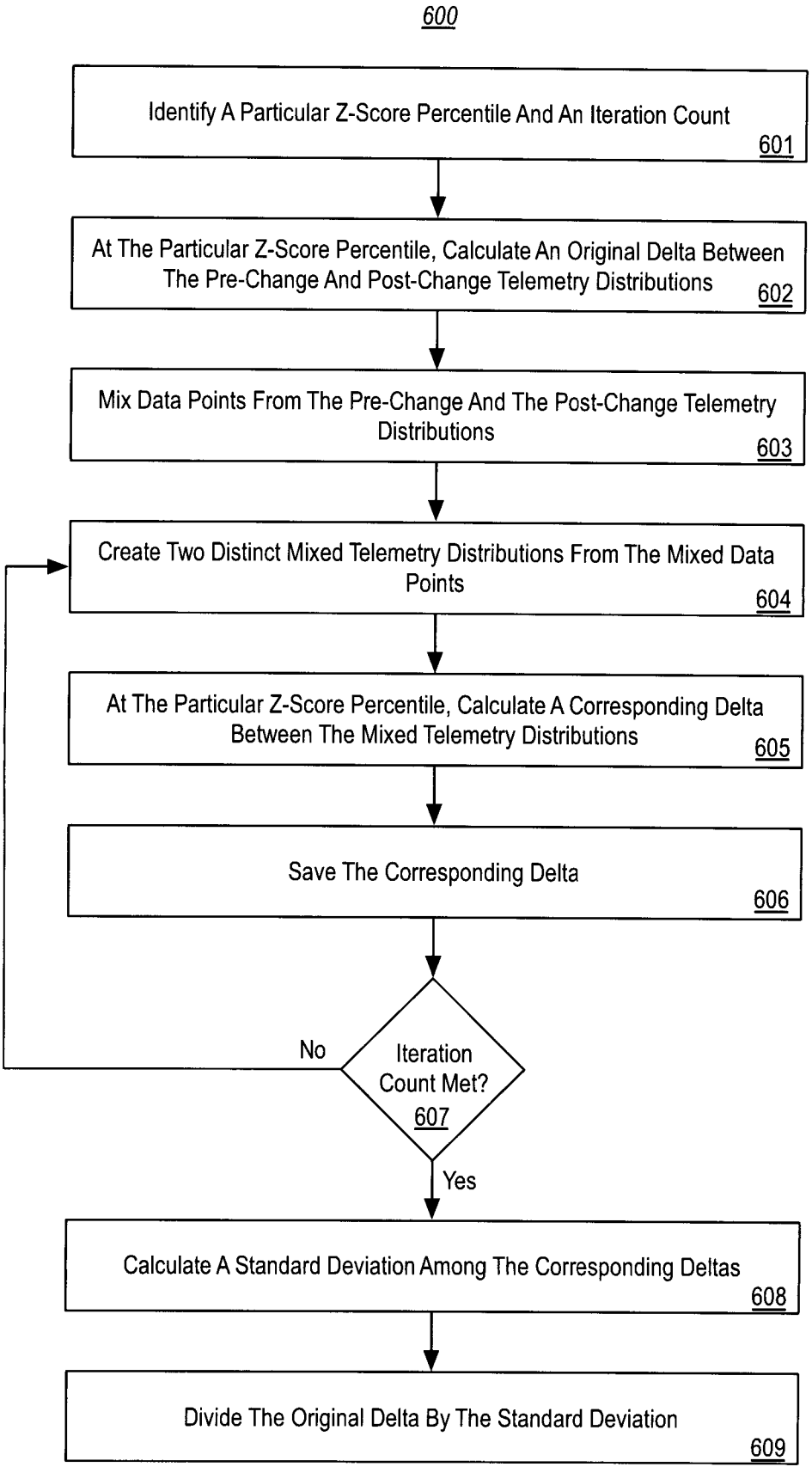
600
Identify A Particular Z-Score Percentile And An Iteration Count
601
At The Particular Z-Score Percentile, Calculate An Original Delta Between The Pre-Change And Post-Change Telemetry Distributions
602
Mix Data Points From The Pre-Change And The Post-Change Telemetry Distributions
603
Create Two Distinct Mixed Telemetry Distributions From The Mixed Data Points
604
At The Particular Z-Score Percentile, Calculate A Corresponding Delta Between The Mixed Telemetry Distributions
605
Save The Corresponding Delta
606
Iteration Count Met?
607
No
Yes
Calculate A Standard Deviation Among The Corresponding Deltas
608
Divide The Original Delta By The Standard Deviation
609

FIG. 6