

Fault Injection for Online Failure Prediction Assessment and Improvement

A focus on data

v.1.4
(2018, March)

Ivano Irrera

Ph.D.

Departamento de Engenharia Informática
Universidade de Coimbra
Coimbra, Portugal

Abstract. This document is intended to provide a guide for understanding the data collected by Ivano Irrera during his Ph.D. at the Department of Informatics Engineering of the Coimbra University, Portugal, relative to the period that goes from 2009 to 2015. The data were intended to study the efficacy of using Fault Injection technique for assessing and improving Online Failure Prediction technique, a technique that allows forecasting failures occurring in a system monitoring the current system state (i.e., collecting data) and using models to predict future failure events.

The data were collected from several Windows XP-based systems, installed on hardware and in virtualized environments (the latter involved both to address limitations in using Fault Injection, and to follow the trend of software systems to be virtualized), running few different workloads.

Such Failure Data are divided in Failure Data, i.e., data collected during runs in which a failure event was detected, and Golden Data, i.e., data collected during runs in which no failure event was detected. Fault Injection technique was used to induce such systems to fail, thus helping to collecting Failure Data to train Failure Prediction models, and assessing Failure Prediction models' prediction performance, among others.

1 Online Failure Prediction and Fault Injection

Failure Prediction is a technique proposed in the past to predict failures by analyzing the system architecture and the development processes, or by learning from past failure data (e.g., the time between successive failures). Such technique evolved into **Online Failure Prediction**, which correlates past failure data with the current system state, increasing the quality of the prediction. In practice, the prediction of an incoming failure allows performing mitigation actions, such as saving data or restarting parts of a system, to lessen possible hazards.

Fault injection, on the other side, is an experiment-based approach that deliberately introduces faults into a computer system in a way that emulates real faults, with the goal of observing its behavior.

The objective of Ivano Irrera's Ph.D. thesis, titled "*Fault Injection for Online Failure Prediction Assessment and improvement*", was to address the difficulty in collecting Failure-related data from a particular system, when a Failure Prediction model needs to be used. The central idea is to use Fault Injection to generate Failure-related Data in a controlled time frame, without having to wait for the system to fail: in this way, ideally, Failure-related data can be collected in short time. In fact, a failure is caused by a fault, thus emulating the presence of faults increases the probability of driving the system to a failure.

In particular, the thesis had the following specific objectives:

1. **Using Fault Injection to improve the deployment of Online Failure Prediction models on a particular system installation, and evaluating their figures of merit**, using fault injection to generate the Failure-related data needed to achieve such goals.
2. **Using Fault Injection to support the continuous adaptation of failure prediction in dynamic systems**, as such systems does change over time, and failure prediction models are needed to be updated with up-to-date failure-related data.
3. **Using Fault Injection to identify the best variables to be used to predict failures**. The selection of the system parameters to monitor is not trivial, as the number of variables can be very high and the ones to be used are not known *a priori*. Focusing on the best ones is essential to correctly use a predictor and to improve its performance.

Failure-related data, in this context, are data collected from a specific software system using runtime monitoring of parameters or *variables* that portray the system state. Data are made of several variables, each variable made up of numerical values from the *Real Numbers* domain, each value associated to a timestamp, thus giving values a specific order.

In the context of this thesis, we injected **Software faults**, i.e., flaws that are present in software (caused by programming errors, design flaws, etc.), which may be the cause of software system failures. For the practical evaluation of the proposed theses, we used a tool implemented at the University of Coimbra, making use of the G-SWFIT recommendations ("*Emulation of Software Faults: A Field Data Study and a Practical Approach*", IEEE Transactions on Software

Engineering, 2006, [link](#)) to define and inject software faults directly in running Windows OSs' processes, as well as to remove an injected fault.

Furthermore, we adopted the **Online Failure Prediction characterization model** from Salfner and Malek (“*A survey of Online Failure Prediction models*”, ACM Computing Surveys 2010, [link](#)). According to the adopted characterization, the failure prediction task consists of assessing if, at a time t , a failure is going to occur within a precise time, called *lead-time* Δt_l . The prediction can be valid in a given time window, called *prediction window* Δt_p . The variation of the parameters Δt_l and Δt_p influences the performance of the prediction. In practice, at time t , a model (or predictor) should predict if a failure is going to occur in the interval $[t+\Delta t_l, t+\Delta t_l+\Delta t_p]$. As shown in Figure 1, a prediction performed at time t targets the *Prediction Window* starting at time $t+\Delta t_l$, and lasting Δt_p .

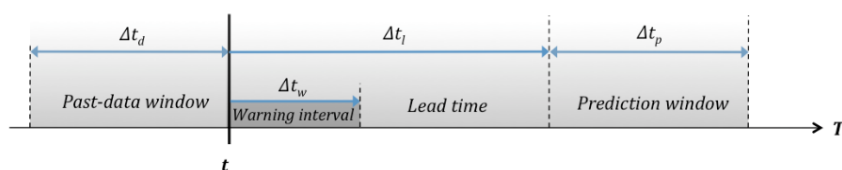


Figure 1. The Online Failure Prediction problem characterization

The prediction can be valid until $t+\Delta t_l+\Delta t_p$. As mentioned before, the predictor is built from a set of past data. As an example, considering a classifier as prediction system, one can assume that these data are a set of observations $x = \langle f_1, f_2, \dots, f_n \rangle$ of a target system. The *prediction task* is then to predict, from the observed features $x_{new} = \langle f_1, f_2, \dots, f_{n-1}, ? \rangle$, the target variable f_n , which can be either “failure” or “no failure” or, in general, a continuous measure indicating how much failure prone the current system state is. Thus, given previously unseen observation matrix x_{new} with an unknown class label at time t , the *prediction about the occurrence of a failure in the interval* $[t+\Delta t_l, t+\Delta t_l+\Delta t_p]$ is given by $f_n = C_l(x_{new})$, where C_l is the predictor. In particular, a prediction at time t is correct if the target event occurs at least once within the prediction period Δt_p .

2 The experimental evaluation environment and approach

To demonstrate the effectiveness of Fault Injection in assisting Failure Prediction, during this Ph.D. work we made use of an **experimental evaluation environment** made of a system that is injected with faults, and a machine controlling the injection and collecting data. In particular, such environment (represented in Figure 2) is made of a **Target system**, that is targeted by fault injection and from which the data are collected, and a **Controller system** (independent from the target system), whose responsibility were the collection of data, the management of fault injection and the restoring of the target system to a previous fault-free state, and so on. The collected Failure-related data are stored in a database on the Controller System.

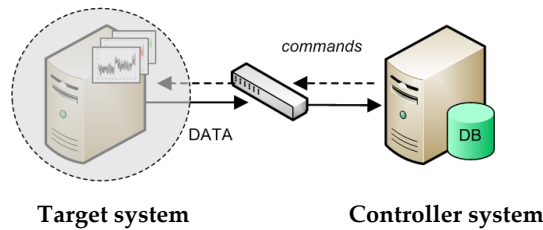


Figure 2. The experimental evaluation environment

A typical case study of the actual Ph.D. thesis work is based on an environment that includes a Windows XP SP3 machine (the *target system*), installed in a virtual machine running on top of a hypervisor (VMWare vSphere server or XEN server, in this work), or directly on the hardware. The *controller* machine is in charge of controlling the experiments and analyzing the failure data coming from the system. A typical configuration of the machines used is as follows:

- 1) **Machine #1 (target):** Intel i5-650@3.60GHz machine, 8GB RAM, running a Windows XP OS (SP3) in a VMWare vSphere server based on ESXi v5.0. Running the target system as a virtual machine on a VMWare vSphere server gave us the possibility of saving the state of the system at the beginning of the fault injection campaign, and restoring that saved state at the end of each run. This check-pointing functionality copies the configuration of the virtual machine, as well as the data contained in the virtualized storage disk and its running state (e.g., state and data of the processes in execution, values contained in the CPU registries, etc.).
- 2) **Machine #2 (controller):** Intel i5-650@3.60GHz machine, 8GB RAM, running a Windows XP OS (SP3), used to: (a) control the experiments (start/stop the experiments), (b) remotely command and control the fault injection tool, (c) force the reboot of the machines in case of failures (including in hanging situations), (d) collect the data and store them in a Microsoft SQL Server 2008, and (e) analyze data.

A detailed description of the specific setups used in the context of this Ph.D. can be found in Section 0.

The **approach** we defined for generating failure data through injecting faults includes a **procedure** and a set of **components** for controlling the fault injection process, collecting the data and building the dataset. Software faults are injected while the target system executes one or more operations (a group of these is called a *workload*), in a way that allows capturing the dynamics that lead to failures by monitoring several variables (numerical data, events, etc.). In practice, the approach includes the following components:

1. **Fault injector and faultload:** faults are defined and organized in a faultload. A fault injector emulates specific faults by modifying one or more components of the target system. The choice of the faultload is of utmost importance as it influences the data

generated, ultimately impacting on the overall results (different faults may lead to different types of failures).

2. **Workload:** for collecting information about the system behavior, faults must be injected while the target system runs a workload, and this procedure should be repeated several times. The workload is the set of operations that the target system performs in the field (realistic workload) or, alternatively, it may be a set of synthetic operations (a synthetic workload) that represents the usual tasks of the system, built specifically for failure data generation and collection. A synthetic workload is useful when the system has not been deployed yet, or when it is not possible to inject faults in the target system and/or the workload cannot be replicated.
3. **Monitoring and data collection infrastructure:** an infrastructure is used to gather the data that characterizes the behavior of the target system in the context of the observed failure events, while running a workload and injecting faults. Depending on the failure prediction mechanisms under study, besides failure-related data, one may need to collect also failure-free data. What is important is the data to include only the most relevant information for predicting failures.

The components above are the fundamental parts of the **experimental procedure**, which is divided in *four phases* (see also Figure 3):

1. **Definitions and set-up:** in this phase are defined the failures to predict, the system information to be monitored (e.g., a set of numerical variables or a set of events in the logs, including failure events), the workload and the faultload, and a set of parameters characterizing the scope of the failure prediction. This comprises building the concrete faultload to inject, installing and configuring the workload emulation tool, and installing and configuring the data monitoring and collection infrastructure and the fault injection tool. Other tasks include defining and setting up the *target* system and the *controller* system.
2. **Data generation and collection:** this is the core phase of the approach, where the data are collected while the target system executes the workload and faults are injected by a fault injection tool. This data may correspond to fault-free situations (Golden Data) and/or situations in which a failure is observed (Failure Data). Data collection is done during several time intervals and in each interval the monitoring infrastructure collects the values of the variables portraying the state of the target system.
3. **Dataset building:** the data collected are organized in datasets for being consumed later by the failure prediction models. This process depends on the failure prediction system to be trained (e.g., training anomaly detection systems only requires Golden Data), as well as on the types of failures being predicted. In particular, the monitored data are associated with the failures observed in Phase 2 considering the failure prediction parameters specified in Phase 1.

4. **Failure data accuracy estimation analysis:** accuracy is the property of the generated failure data to be similar to data that would be obtained in a real scenario. Due to the scarcity of real data, we estimate the correlation between synthetic and real failure data by applying metrics (specific of each condition) to two or more, independently generated, synthetic failure datasets. We use the concepts of *weak accuracy* and/or *strong accuracy*, as sufficient conditions for the generated failure data to be considered accurate. Strong accuracy metrics are applied directly on the datasets, while the weak accuracy metrics are applied to the prediction performance of the models trained with independent synthetic datasets.

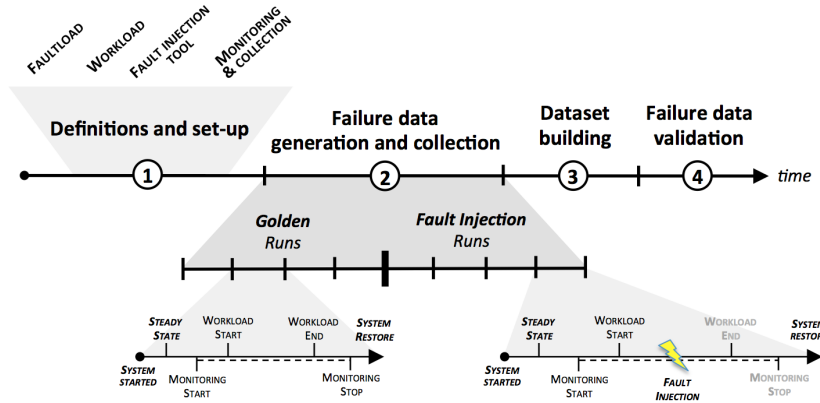


Figure 3. The four phases of the failure data generation

3 Failure Data Generation, Collection and Organization

The **Failure Data Generation and Collection** phase (*phase 2*) takes place throughout several time intervals (as shown in Figure 3), referred to as **runs**, during which the monitoring infrastructure collects the set of variables selected while the target system executes the operations defined by the workload. The **number of runs**, as well as their **duration**, depends on several parameters, such as the time needed to execute the workload, the specific set-up environment and the prediction parameters (e.g., for predicting a failure one hour in advance, each run must last for at least one hour).

Collected data can be divided into Failure Data, Golden Data, and Non-Failure Data. **Failure data** are data obtained by injecting faults during several runs (eventually evolving into failures), while **Golden data** are gathered when no faults are injected and no failures are observed¹. Finally, **Non-Failure Data** are relative to **non-failing runs**, i.e. runs in which a fault was injected but no failure was observed in the defined time frame. The use of each kind of data depends on the

¹ In fact, no fault is injected and no failure is observed does not mean that no fault was activated, as there is not guarantee that no residual faults are present in the system.

prediction models that will consume the generated data (e.g., anomaly detection based models just need golden data, while classifiers need both types of data).

A run with no faults injected and no failures observed is called **Golden Run (GR)**, and the corresponding data are *Golden Data (GD)*. An execution in which faults are injected is called **Fault Injection Run (FIR)**. If a failure is observed during a fault injection run then it is a **Failing Run**, and the data monitored are *Failure Data (FD)*. *Non-Failure Data (NFD)* are associated **Non-Failing Runs**. Although this kind of data may also provide information about the system failing behavior, their use was out of the scope of the thesis.

In each failing run, the **failure event** must be **detected** and later associated to the collected Failure Data. For this, different **failure detectors** (models that recognize failure patterns when they occur) may be needed.

When more than one failure mode or more than one workload is considered, the runs (and thus the failure data) can be grouped into **Scenarios**. In this work, the scenarios are identified by a failure mode \mathcal{F} and a workload W , or the tuple $\langle Workload, Failure\ mode \rangle$.

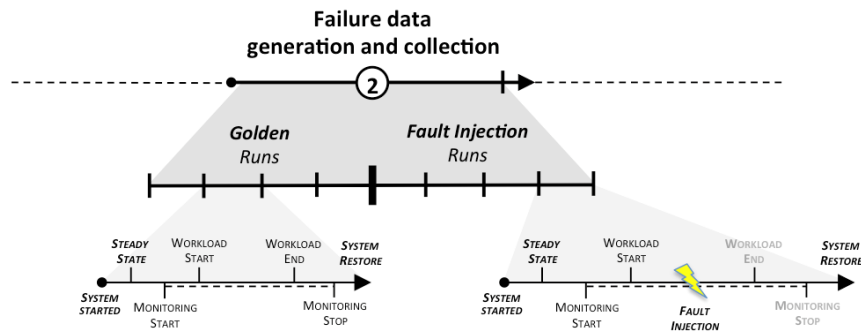


Figure 3.1. Failure data generation, collection and data organization phases

As detailed in Figure 3.1 the data are generated as follows:

1. Each run starts by booting the target system and waiting for it to **reach a steady state**, before the workload is executed. Having the system in a steady state means that it is ready for executing the workload in the best way possible, which is recommended, albeit not mandatory. The instant in which the system achieves its steady state is referred to as T_0 .
2. **The workload and the monitoring tools are then started.** The instant in which the workload execution starts is referred to as T_W , while T_M identifies the time at which the monitoring system is executed. The data collection may start at time T_M or T_W , depending on the specific needs (e.g., if data from the beginning of the workload execution are needed, the monitoring must be started before the workload). In practice, data is

composed of data samples collected from the different variables at a given instant of time, according to a specific sampling rate s .

3. In a **Fault Injection Run (FIR)**, a fault is injected at time T_{FI} while the target system is executing the workload and the monitoring tool is collecting data. In a **Golden Run (GR)** the system executes the workload, but no fault is injected.
4. The **run finishes** when a *failure* (T_F , FIR only) is detected (the failure detector associates the failure to the time T_F), or after the *workload* has completed its execution (T_{W_END}) or a *maximum* run execution time T_{MAX} is achieved. In such cases, two situations are possible:
 - a) In the case of **Golden Runs (GRs)**, if no failure is detected in the interval $[T_0; T_0+T_{MAX}]$, the data relative to the run are considered Golden Data (GDR_i , Golden Data relative to the i -th run). It is worth noting that a failure occurring in a Golden run is caused by an actual residual fault of the target system (i.e., not an injected one) and the data should also be considered as Failure Data.
 - b) For **Fault Injection Runs (FIRs)**, if no failure is detected in the interval $[T_0+T_{FI}; T_0+T_{MAX}]$, the run is considered to be failure-free, and the relative data to be Non-Failure Data ($NFDR_i$, relative to the i -th run). On the other hand, if a failure is detected in such interval, the collected data are considered Failure Data (FDR_i , relative to the i -th run).
5. After completing a run (and collecting the corresponding data), the target **system must be restored** to a state in which no faults injected are present. This ranges from *rebooting*, in the cases where the fault does not permanently affected parts of the system (e.g., data or files), to the correction of fault effects (e.g., substituting files previously backed-up) or the re-installation of the entire target system².

3.1 Failure Prediction Dataset building

For being used by failure prediction models, the collected Golden Data and Failure Data are to be organized in **datasets** and associated to information about the **failures observed** during each run.

Datasets are made of a concatenation of variables' values relative to different GRs and FIRs. As an example, a dataset can be made of V variables, which values were collected at a rate of 1 value/s in 10 GRs for V_N seconds, thus making a $|V| \times 10 \cdot |V_N|$ matrix. Successively, data are associated to the observed failures by **labeling** each data sample composing the collected data. **Data labeling** is a technique that associates a numerical label (e.g., 0, 1, etc.) to each data sample (i.e., a set of values of each monitored variable), depending on the meaning that each label has in the particular modeling or prediction scenario (e.g., a sample is labeled 0 if the target system was working correctly at the moment of the sample's collection, or conversely is labeled 1 if the system was presenting an erratic behavior). In this particular context, data is labeled according to

² Virtualization is a solution that allows restoring the target system (both software and – emulated – hardware), by using check-pointing and restoring operations.

the **failure time** T_F and the **failure prediction lead-time and prediction window** $(\Delta t_l, \Delta t_p)$, defined in the failure prediction problem characterization adopted (as presented in Section 1). A detail about labeling according to such model is presented in Figure 4: Starting from the failure time $T_{Failure}$, the label 1 is put backwards until covering the interval $[T_{Failure} - \Delta T_L + \Delta T_P, T_{Failure}]$. It is worth noting that a different label could be used for identifying the different intervals $\Delta T_L, \Delta T_P$, etc.

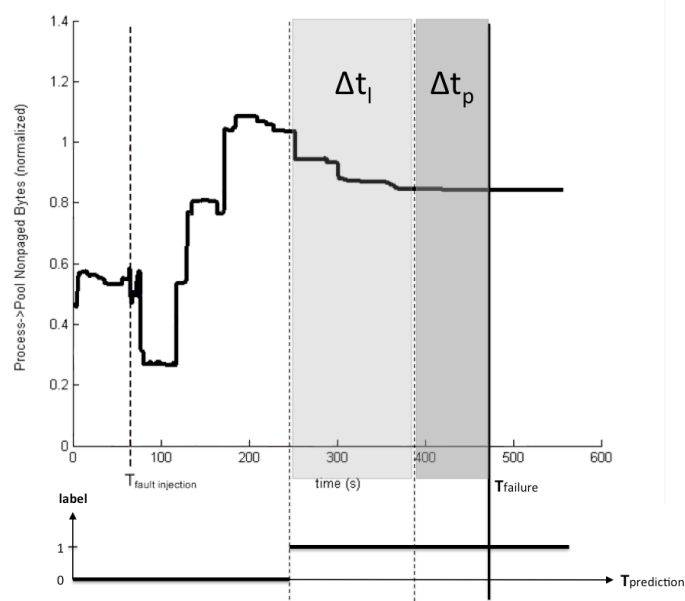


Figure 4. Example of labeling a single run (only one variable showed), according to the failure time $T_{Failure}$

Data from a given run r is composed of n different variables $\underline{v}^r = \langle v^r_1, v^r_2, \dots, v^r_n \rangle$, where v^r_i is the i -th variable collected from the target system. For each time instant k , each variable v^r_i has a given value $v^r_i(k)$, representing a variable value collected at the time instant k . Hence, a *data sample* relative to time k is defined as:

$$(1) \quad \underline{v}^r(k) = \langle v^r_1(k), v^r_2(k), \dots, v^r_n(k) \rangle$$

A *data sample* $\underline{v}^r(k)$ collected during a *Golden Run* (when no failure occurred), is associated a **label** $l^r(k)=0$, for each time k . On the other hand, given T^r_F the time at which a failure was detected during the *Failure Run* r , and the prediction indexes $(\Delta t_l, \Delta t_p)$ (valid for all the runs), a label $l^r(k)=1$

is associated to a data sample $\underline{v}^r(k)$ if a failure occurred in the interval $[T_F^r - (\Delta t_l + \Delta t_p), T_F^r - \Delta t_l]$, otherwise it is 0³. Hence, for each time instant k and each run r , a labeled sample is:

$$(2) \quad \underline{v}^{r*}(k) = \langle v_1^r(k), v_2^r(k), \dots, v_n^r(k), \mathbf{f}(k) \rangle$$

Collected data labeled according to the failure prediction indexes $(\Delta t_l, \Delta t_p)$ and the failure time T_F can be considered a **dataset**. More generally, several couples $(\Delta t_l, \Delta t_p)$ can be specified, and varying the values of Δt_l and Δt_p let the labels associated to each data sample to change accordingly. In this case, being $\underline{\Delta t}_l = \langle \Delta t_{l1}, \Delta t_{l2}, \dots, \Delta t_{ln} \rangle$ and $\underline{\Delta t}_p = \langle \Delta t_{p1}, \Delta t_{p2}, \dots, \Delta t_{pn} \rangle$, one can define a dataset with which N **sets of labels** are associated, where $N = |\Delta t_l| \times |\Delta t_p|$, or alternatively, define N different datasets, each one associated to a specific tuple $(\Delta t_l, \Delta t_p)$. Such dataset can be built once and used for training and testing a failure prediction model using a couple $(\Delta t_l, \Delta t_p)$ at a time, or together in a meta-model fashion. It is worth noting that a dataset made of Golden Data will present 0s for all the values of the couple $(\Delta t_l, \Delta t_p)$.

In addition to this, different types of failures can affect the target system, and several different workloads can be used as well. **Failure types** and **workloads** define a single scenario, and each scenario $\langle \text{Workload}, \text{Failure mode} \rangle$ is associated to a different set of data, as data reflect different failure modes and workloads (see Figure 5). In the context of this specific Ph.D. work, failure prediction models are trained and tested using data from a single scenario.

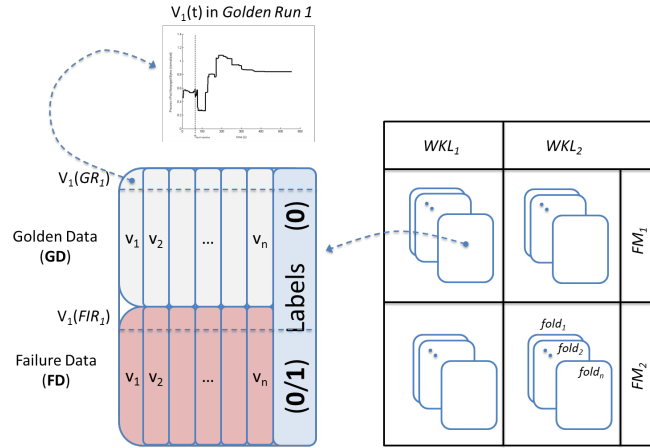


Figure 5. Datasets and scenarios (two workloads and two failure modes)

³ It must be noted that the label values chosen can be any two different numerical values (other widely used values for labeling data are (-1, +1) – especially when using Support Vector Machine classifiers – (5, 10), and so on).

For training and validating failure prediction models, data must be organized into **training datasets (TDSs)** and **testing datasets (TTDSs)**, whose goal is to support the assessment of prediction performance. Such division is usually based on grouping single *data samples*. However, in our work we decided to group *Golden* and *Failure Data* in training and testing datasets by considering the **runs** to which they belong to, thus implementing a **runs-wise dataset**. The reason that stays behind this decision is that the collected data represents time series and the division in *samples* may alter the *continuity* and *ordering* among samples, which may finally impact the prediction performance (e.g., when training regression models).

An **example of dataset** is presented in Figure 6 (a) and (b): Figure 6 (a) represents data collected from the i -th Fault Injection Run and labeled with N different couples of $(\Delta t_i, \Delta t_p)$ values, while Figure 6 (b) presents a dataset made of GR_G Golden Runs and FIR_F Fault Injection Runs, highlighting the difference between labeling Golden and Failure Data, being the first labeled with only 0s and the latter with 0s and 1s.

4 Experiments characterization and used testbeds

In this section, we present the characterization of the experimental evaluations performed in this Ph.D. work.

We adopted a Windows-based **software fault injection tool** implemented at University of Coimbra following the G-SWFIT recommendations for the fault injection task. Such tool is able to inject software faults at machine-code level both in binary files and in running processes (user-mode only). However, due to the fact that the Windows OS includes a protection for avoiding certain system files from being changed, the fault injector was limited to inject software faults in running processes of the operating system, but faults were injected before starting the collection of data, thus simulating residual faults from the perspective of the data collection process.

The **faultload** is based on the fault types defined by G-SWFIT recommendations. Based on previous experience, we mostly focused the fault injection on the code of the **svchost.exe** process and of the linked **dynamic library kernel32.dll** (containing functions for handling the OS memory usage), which are key resources of the Windows XP OS. The fault injection tool was able to automatically generate thousands of *code mutants* by analyzing the fault locations matching a specific pattern depending on the type of software fault, being each fault identified by the tuple $\langle \text{fault type}, \text{fault location}, \text{code mutant} \rangle$. In order to design a feasible experiment, a subset of the faults was selected based on the relevance of their *locations* (details on the number of faults injected and their impact are presented in the next subsection). For this, we used a profiling tool (*Luke Stackwalker*), which helped identifying the functions and modules executed along several runs of the workloads considered. As previously discussed, the selection of the most executed modules of the target system does not invalidate the representativeness of the injected software faults.

Regarding the **failures**, we empirically focused on **Crashes** and **Hangs**, which are the two failure modes observed injecting faults in a part of the Windows XP OS, by using the G-SWFIT tool. A **failure detector** able to detect the occurrence of the two failure modes mentioned above was implemented. In practice, the detector continuously monitors the target system to detect failures in the following way:

1. a **crash** is detected when the system does not respond to a *ping* (implementing an *heartbeat* mechanism) for a certain time T_{\max_ping} . The failure time T_F is obtained by considering the first time instant in which the system became unresponsive;
2. a **hang** is detected if the target system responds to a *ping*, but it hangs on executing a given set of operations. Again, the failure time T_F is obtained by considering the first time instant in which the system became unresponsive, identified by the time instant when the first not executed operations were sent to the system⁴.

The Target system runs several different **workloads**, namely:

1. **WinRAR** application (**WKL₁**), compressing a file using the RAR algorithm with the low compression option;
2. **COSBI OpenSourceMark** computer benchmarking suite (**WKL₂**), a more complex workload that includes computation and input/output intensive tests, compression algorithms, disk and memory accesses, etc. (we consider that these workloads include generic operations that computer systems perform frequently, being thus adequate for the present case study). Tomcat application server, which executes the workload of the TPC-W benchmark;
3. **Tomcat application server** (**WKL₃**), which executes the workload of the TPC-W benchmark (details in the dissertation associated to this Ph.D. thesis work). Three versions of the Tomcat application server were used, namely 6.0.36 (**WKL₃₁**), 7.0.19 (**WKL₃₂**), 7.0.40 (**WKL₃₃**).

The combination <Workload, Failure mode> allows defining four different **scenarios** for the analysis: <WKL₁, Crash (FM₁)>, <WKL₁, Hang (FM₂)>, <WKL₂, Crash (FM₁)>, and <WKL₂, Hang (FM₂)>, etc.

Regarding the **variables to monitor**, we typically considered a set of variables reflecting the state of the operating system and the usage of the hardware resources, as the symptoms of the failures considered may manifest at the OS and at lower levels (e.g., an increase in the number of context switches/s). In most of the cases, we monitored 233 numerical variables, at the sample rate of one value per second, using the Logman tool that is included in Windows OSs family, and afterwards conducted a three-step feature selection to reduce the number of variables. Some of the

⁴ In the case of a Hang failure, in absence of the information about the time of failure detected, one can estimate the failure time by using the value of the timeout given to the request to the Target to reboot (usually 60 seconds in average, value used in all the campaigns), and the duration of the experiment. Hence, the occurrence time of a Hang failure can be estimated to be $T_{exp}-60$.

variables were not considered in our analyses, as the ones having a constant or *null* value in all the runs, and successively variables having a linear correlation coefficient (Pearson's coefficient) greater than 0.9 between each other.

The **failure prediction model** used in the context of this Ph.D. work was **SVM** (Support Vector Machine), the state of the art among classification models. In particular, the **libSVM** libraries implementing the SVM predictor were used.

The specific **setups** used in the context of this Ph.D. thesis were the following:

1. “*Towards Identifying the Best Variables for Failure Prediction using Injection of Realistic Software Faults*” (PRDC 2010, [link](#)): a **single Controller** machine running Windows OS and Microsoft SQL Server, and a **single Target** machine running Windows XP SP3 OS
2. “*Assessing the Impact of Virtualization on the Generation of Failure Prediction Data*”, (LADC 2013, [link](#)): a **single Controller** machine running Windows OS and Microsoft SQL Server, and **five Target** machines, configured as follows:
 - **Machine #1**: Intel i5-650@3.60GHz; 8GB RAM; Windows XP OS (SP3); **no virtualization** (hosts the original system).
 - **Machines #2 and #3** (*virtualized, Type II Hypervisors*): Intel i5-650@3.60GHz; 8GB RAM; virtualized Windows XP OS (SP3). Machine#2 uses a **Citrix XEN server v5.6.10**, and Machine#3 runs a **VMWare vSphere server** based on ESXi v5.0. These provide two virtual versions hosted on top of Type II Hypervisors.
 - **Machines #4 and #5** (*virtualized, Type I Hypervisors*): Intel P4 HT@3.00GHz; 2GB RAM; virtualized Windows XP OS (SP3). Machine#4 runs **Oracle's VirtualBox**, and Machine#5 runs **VMWare Player**, both on top of Windows XP OSs. These provide two virtual versions hosted on top of Type I Hypervisors.
3. “*The time dimension in predicting failures: a Case Study*” (LADC 2013, [link](#)): a **single Controller** machine running Windows OS and Microsoft SQL Server, and a **single, virtualized Target** machine running on top of a Citrix XEN Server.
4. “*On the need for training Failure Prediction algorithms in evolving software systems*” (HASE 2014, [link](#)): **three virtualized Controller** machines and **three virtualized Target** machines running on top of a Citrix XEN Server. The controller machines ran a Windows 7 OS and a Microsoft SQL Server, while target machines ran a Windows XP SP3 OS. The machines are organized as in Figure 7.

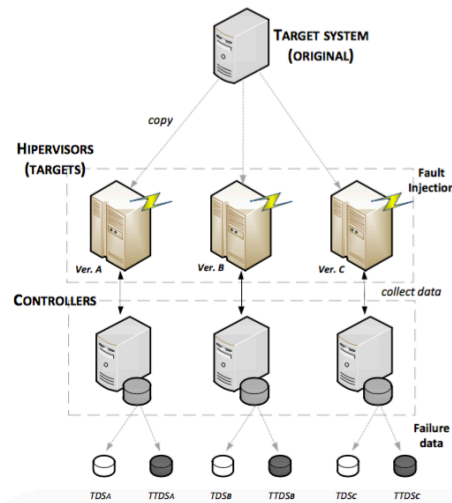


Figure 7. Replicated, virtualized Targets and Controllers

5. “A Practical Approach for Generating Failure Data for Assessing and Comparing Failure Prediction Algorithms” (PRDC 2014, [link](#)): a **single Controller** machine running Windows OS and Microsoft SQL Server, and a **single Target** machine running Windows XP SP3 OS.
6. “Adaptive Failure Prediction for Computer Systems: a Framework and a Case Study” (HASE 2015, [link](#)): the framework was implemented using a **single Controller**, and a **virtualized Target** machine running into a XEN server, and a **Sandbox Hypervisor** made of a Citrix Xen Server, hosting a (**virtualized**) **Replica of the Target** machine, used to inject faults and generate failure data. In Figure 8 a representation of the framework.

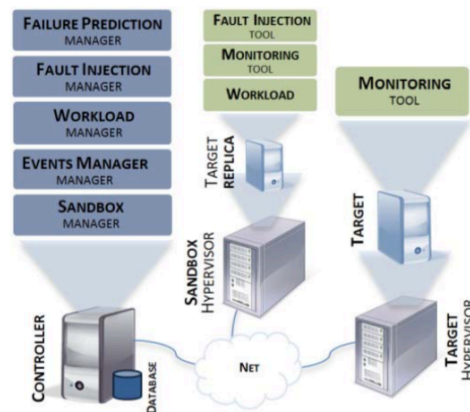


Figure 8. The Adaptive Failure Prediction framework

5 Experimental evaluation: the collected data

Data were collected from Target System using Microsoft Logman monitoring tool, which is natively included in Windows OSs family. All the data are stored in a Microsoft SQL Server (2008-2010 versions) on the Controller machine, along with information about Fault Injection, Failure Detection and Failure Prediction. Each database contains monitored data (in tables created and operated by the Microsoft Logman monitoring tool), and data relative to the management of Fault Injection and Failure Prediction, as *faultloads*, *datasets*, *failure prediction parameters*, *prediction assessment results*, etc., stored in tables created on purpose.

Several different databases created were relative to a specific setup (e.g., VMWare environment, XEN environment, real machine, etc.), and a specific campaign (e.g., 1000 GRs and 5000 FIRs, with faults injected in svchost.exe process). Each database, furthermore, has been divided into **sets**, which are part of a whole campaign, for scalability and fault tolerance purposes: in fact, if a database is corrupted, only a single set has to be re-executed. Few examples are in Figure 9.

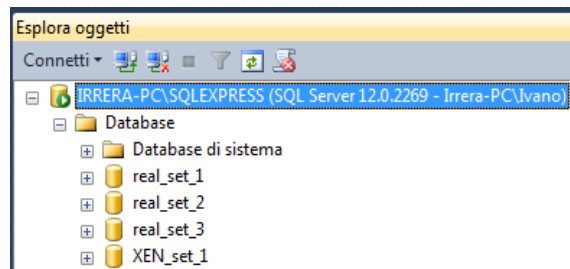


Figure 9. Example of different databases and several sets:
real_set_1, *real_set_2*, *real_set_3*, *XEN_set_1*

The *Logman* tool automatically organizes the data collected in three tables, **CounterData**, **CounterDetails** and **DisplayToID** (see Table 1). Each monitored component is defined by the tuple $\langle \textit{Machine}, \textit{Object}, \textit{Instance}, \textit{Counter} \rangle$, where *Counter* is a single *variable*. Specifically:

- **Machine** identifies the physical machine where the collected information belongs (e.g., "Server_XYZ");
- **Object** identifies a component (intended as macro-object) of a Machine, made of various parts (micro-objects), for instance Memory, Operating System, Physical Disk, etc.;

- **Instance** identifies an instance of a particular Object. In fact, an object can have one or more instances, as for example CPU, in a multicore system, can have the instantiation “CPU0”, “CPU1”, etc. In the case an Object has one instance only, the Instance value is “NULL”;
- **Counter** identifies a variable describing the properties of a particular Object, for instance “page faults/s” belonging to the Memory object, or “written bytes/s” belonging to the Physical Disk object.

The Logman tool automatically creates the tables above when the monitoring starts. For scalability purposes, after each run has ended, we copy the current **CounterData** table to a **CounterData_N_TYPE** table, where *N* is the number of the last experiment, and *TYPE* is the type of experiment executed, which can be one between *GR* (Golden Run) and *FIR* (Fault Injection Run).

An example of an actual Microsoft SQL database obtained from our case studies is in Figure 10.

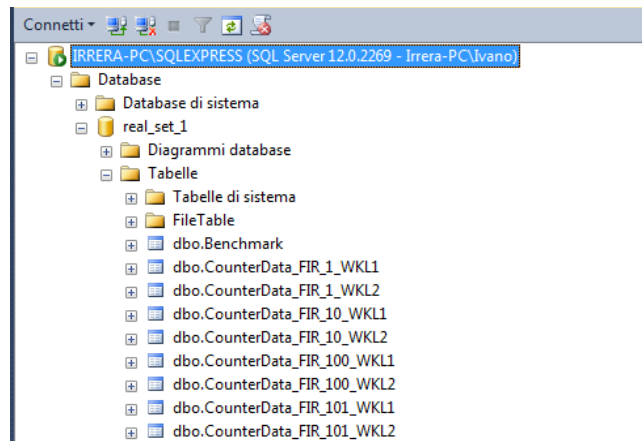


Figure 10 – Datasets and scenarios (two workloads and two failure modes)

The collected data are organized according to the *Relational model*. All data were stored in a Microsoft SQL Server.

5.1 Data collected from 2009 to 2015

The data collected from the fault injection campaign have been stored in several databases, divided in sets to be easily manageable and reduce the total size of data:

1. **real_set_X**, data collected from a real machine running a Windows XP OS, faults injected in the svchost.exe process, running WKL₁ and WKL₂;
2. **vSphere_set_X**, data collected from a virtualized machine running on top of a VMware vSphere server (ESXi), running a Windows XP OS, faults injected in the svchost.exe process, running WKL₁ and WKL₂;
3. **XEN_set_X**, data collected from a virtualized machine running on top of a Citrix XEN server, running a Windows XP OS, faults injected in the svchost.exe process, running WKL₁ and WKL₂;
4. **XEN_Tomcat_set_X**, data collected from a virtualized machine running on top of a Citrix XEN server, running a Windows XP OS, faults injected in the svchost.exe process, running WKL_{3x};
5. **Training_set_X**, data collected from a virtualized machine running on top of a Citrix XEN server, running a Windows XP OS, faults injected in the svchost.exe process, running WKL_{3x} (data used for actual Failure Prediction Model assessment/Benchmark – *extended tables DB*);
6. **Simulation_set_X**, data collected from a virtualized machine running on top of a Citrix XEN server, running a Windows XP OS, faults injected in the svchost.exe process, running WKL_{3x} (data used for actual Failure Prediction Model assessment/Benchmark – *extended tables DB*).

In the following, we present the details of the databases containing the data collected during the period of the presented Ph.D. work.

Note:

- *Tables organization* = the particular set of tables used to organize the data, *described in paragraph 5.2*
- *Variables set* = the particular set of information collected from the Target systems (variables), *described in paragraph 5.3*

Earlier data

Name	Date	(GR, FIR)	Failures (Hang, Crash, Controller)	Workload	Tables org.	Var. set	Data
real_set_1	15-19/3/2012	(100, 500), (100, 500)	(25, 2, 2)	WKL ₁ , WKL ₂	T ₁	V ₁	yes
real_set_2	13-28/10/2011	(2, 501), (2, 501)	(18, 2, 0)	WKL ₁ , WKL ₂	T ₁	V ₁	yes
real_set_3	9-15/11/2011	(-, 500), (-, 500)	(17, 2, 0)	WKL ₁ , WKL ₂	T ₁	V ₁	yes
real_set_4	17-23/11/2011	(500, 500), (500, 500)	(19, 1, 3)	WKL ₁ , WKL ₂	T ₁	V ₁	yes
real_set_5	26/11- 4/12/2011	(-, 500), (-, 500)	(20, 1, 0)	WKL ₁ , WKL ₂	T ₁	V ₁	yes
real_set_6	17-23/11/2011	(400, 500), (400, 500)	(35, 3, 1)	WKL ₁ , WKL ₂	T ₁	V ₁	yes
vSphere_set_1	19/4-4/5/2012	(500, 500), (500, 500)	(-, -, -)	WKL ₁ , WKL ₂	T ₁	V ₁	yes
vSphere_set_2	21/11- 1/12/2011	(500, -), (500, 500)	(-, -, -)	WKL ₁ , WKL ₂	T ₁	V ₁	yes
vSphere_set_3	14-17/12/2011	(1, 500), (1, 500)	(15, 0, 0)	WKL ₁ , WKL ₂	T ₁	V ₁	yes
vSphere_set_4	14-18/1/2011	(1, 500), (1, 500)	(24, 3, 0)	WKL ₁ , WKL ₂	T ₁	V ₁	yes
vSphere_set_5	-	-	-	-	T ₁	V ₁	yes
vSphere_set_6	10-15/5/2011	(1, 500), (1, 500)	(41, 3, 0)	WKL ₁ , WKL ₂	T ₁	V ₁	yes
XEN_set_1	16-24/3/2012	(100, 500), (100, 500)	(60, 9, 162)	WKL ₁ , WKL ₂	T ₁	V ₁	yes

Note: $T_{Hang,Failure} \approx T_{exp} - 60$, $T_{Hang,Failure} = T_{exp}$

XEN server, Controller 1 (C1_)

Name	Date	(GR, FIR)	Failures (Hang, Crash, TPCW, Controller, OS)	Workload	Tables org.	Var. set	Data
XEN_Tomcat_set_1	26-28/5/2013	(6, 500)	(15, 0, 0, 0, -)	WKL ₃₁	T ₁	V ₂	yes
XEN_Tomcat_set_2	30/5-2/6/2013	(5, 500)	(51, 0, 3, 16, -)	WKL ₃₁	T ₁	V ₂	yes
XEN_Tomcat_set_3	2-5/6/2013	(5, 500)	(5, 0, 15, 3, -)	WKL ₃₁	T ₁	V ₂	yes
XEN_Tomcat_set_4	5-9/6/2013	(5, 500)	(1, 0, 1, 9, -)	WKL ₃₁	T ₁	V ₂	yes
XEN_Tomcat_set_5	12-14/6/2013	(5, 500)	(2, 0, 0, 1, -)	WKL ₃₁	T ₁	V ₂	yes
XEN_Tomcat_set_1 (extra)	3-11/9/2013	(25, 500)	(13, 0, 17, 0, -)	WKL ₃₃₁	T ₁ /T ₂	V ₂	yes
XEN_Tomcat_set_2 (extra)	13-24/9/2013	(25, 500)	(0, 0, 9, 0, -)	WKL ₃₃₁	T ₁ /T ₂	V ₂	yes
Op_Test_Training_3_set_1	22-25/8/2014	(20, 150)	(0, 0, 0, 0, -)	WKL ₃₃	T ₂	V ₂	yes
Op_Test_Training_3_set_2	25-26/8/2014	(-, -)	(0, 0, 0, 0, -)	WKL ₃₃	T ₂	V ₂	yes
Training_set_1	27-30/8/2014	(50, 250)	(10, 8, 0, 0, -)	WKL ₃₁	T ₂	V ₂	yes
Training_C1_set_1	8-12/9/2014	(50, 184)	(16, 0, 0, 0, -)	WKL ₃₃	T ₂	V ₂	yes
Simulation_C1	7-16/9/2014	(475, 450)	(9, 0, 1, 0, -)	WKL ₃₃	T ₂	V ₂	no
Simulation_C1_Oct_2014	15-29/10/2014	(50, 297)	(15, 0, 1, 0, -)	WKL ₃₁	T ₂	V ₂	no
Training_C1_Oct_2014_set_1	11-14/2/2015	(25, 250)	(14, 0, 0, 0, -)	WKL ₃₇	T ₂	V ₂	yes
Training_C1_Oct_2014	14-17/2/2015	(25, 250)	(14, 0, 0, 0, -)	WKL ₃₇	T ₂	V ₂	yes

Note: $T_{Hang,Failure} \approx T_{exp} - 60$, $T_{Hang,Failure} = T_{exp}$

XEN server, Controller 2 (C2_)

Name	Date	(GR, FIR)	Failures (Hang, Crash, TPCW, Controller, OS)	Workload	Tables org.	Var. set	Data
XEN_Tomcat_set_1	30/5-2/6/2013	(6, 500)	(22, 0, 7, 0, -)	WKL ₃₁	T ₁	T ₂	yes
XEN_Tomcat_set_2	2-6/6/2013	(5, 500)	(52, 0, 7, 14, -)	WKL ₃₁	T ₁	T ₂	yes
XEN_Tomcat_set_3	8-11/6/2013	(5, 500)	(4, 0, 0, 9, -)	WKL ₃₁	T ₁	T ₂	yes
XEN_Tomcat_set_4	12-14/6/2013	(5, 500)	(2, 0, 1, 3, -)	WKL ₃₁	T ₁	T ₂	yes
XEN_Tomcat_set_5	18-20/6/2013	(5, 500)	(2, 0, 2, 4, -)	WKL ₃₁	T ₁	T ₂	yes
XEN_Tomcat_set_1 (extra)	3-10/9/2013	(25, 500)	(23, 0, 1, 0, -)	WKL ₃₃₂	T ₁ /T ₂	T ₂	yes
XEN_Tomcat_set_2 (extra)	14-23/10/2013	(25, 500)	(44, 0, 245, 0, -)	WKL ₃₃₂	T ₁ /T ₂	T ₂	yes
Training_C2_set_1	8-10/9/2014	(50, 149)	(10, 8, 0, 0, -)	WKL ₃₅	T ₂	T ₂	yes
Simulation_C2_set_2	8-12/9/2014	(200, 200)	(0, 0, 0, 0, -)	WKL ₃₄	T ₂	T ₂	no
Simulation_C2	12-17/9/2014	(50, 187)	(0, 0, 0, 0, -)	WKL ₃₄	T ₂	T ₂	no

Note: $T_{Hang,Failure} \approx T_{exp} - 60$, $T_{Hang,Failure} = T_{exp}$

XEN server, Controller 3 (C3_)

Name	Date	(GR, FIR)	Failures (Hang, Crash, TPCW, Controller, OS)	Workload	Tables org.	Var. set	Data
XEN_Tomcat_set_1	26-28/5/2013	(5, 500)	(23, 0, 0, 0, -)	WKL ₃₁	T_1	V_2	yes
XEN_Tomcat_set_2	30/5-2/6/2013	(5, 500)	(50, 0, 4, 0, -)	WKL ₃₁	T_1	V_2	yes
XEN_Tomcat_set_3	2-5/6/2013	(5, 500)	(4, 0, 0, 6, -)	WKL ₃₁	T_1	V_2	yes
XEN_Tomcat_set_4	8-11/6/2013	(5, 500)	(1, 0, 1, 20, -)	WKL ₃₁	T_1	V_2	yes
XEN_Tomcat_set_5	14-17/6/2013	(5, 500)	(1, 2, 0, 33, -)	WKL ₃₁	T_1	V_2	yes
XEN_Tomcat_set_1 (extra)	3-11/9/2013	(25, 500)	(23, 0, 3, 0, -)	WKL ₃₃₃	T_1/T_2	V_2	yes
XEN_Tomcat_set_2 (extra)	23-25/9/2013	(25, 500)	(0, 0, 67, 0, -)	WKL ₃₃₃	T_1/T_2	V_2	yes
Simulation_C3_Oct_2014	15-22/10/2014	(50, 354)	(19, 0, 0, 0, -)	WKL ₃₁	T_2	V_2	no
Simulation_C3_set_1 20140820_01_34 bak	16-18/8/2014	(50, 150)	(4, 0, 80, 0, 7)	WKL ₃₁	T_2	V_2	yes
Simulation_C3_set_1 20140827_17_46 bak	24-26/8/2014	(75, 150)	(29, 0, 0, 0, 8)	WKL ₃₃	T_2	V_2	yes
Simulation_C3_set_2	8-11/9/2014	(100, 250)	(17, 0, 0, 1, -)	WKL ₃₃	T_2	V_2	no
Simulation_C3_set_3	13-16/9/2014	(50, 200)	(12, 0, 0, 0, -)	WKL ₃₃	T_2	V_2	no
Training_C3	8-11/9/2014	(50, 250)	(27, 0, 1, 0, -)	WKL ₃₃	T_2	V_2	yes
Training_C3_Oct_2014	8/3/2015	(25, 200)	(0, 0, 1, 0, -)	WKL ₃₇	T_2	V_2	no
Training_C1_set_1	3/8-2/9/2014	(25, 250)	(10, 0, 0, 0, -)	WKL ₃₄	T_2	V_2	yes

Note: $T_{Hang,Failure} \approx T_{exp} - 60$, $T_{Hang,Failure} = T_{exp}$

5.2 Organization of the collected data

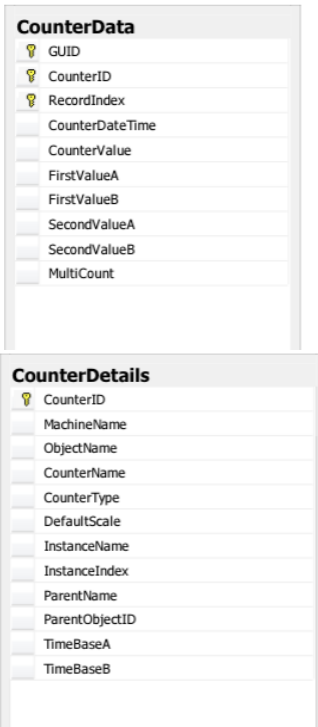
The organization of the data collected is presented in paragraphs 5.2.1 and 5.2.2, the former presenting and describing the tables contained in a first version of the databases, while the latter presenting a successive version of the databases, presenting an extended set of collected information, hence more tables.

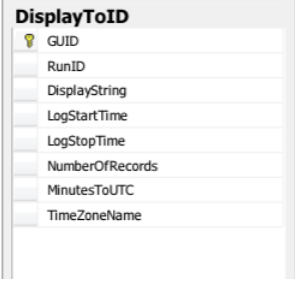
5.2.1 First Database version (2009-2013) (T_1)

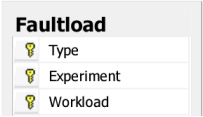
In this paragraph we present the organization of the data collected from the earlier experiences (2009-2013). The organization of data slightly changed with the latter experiences, and is presented in the next paragraph.

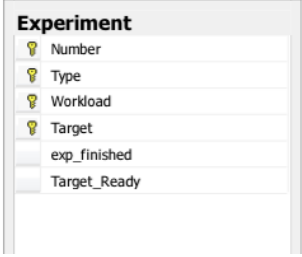
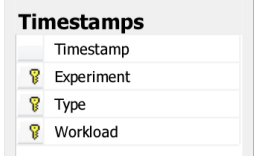
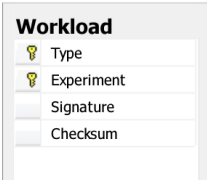
To keep data integrity, *primary keys*, *unique values* and *null rules* were used, while no *referential integrity rule* was used.

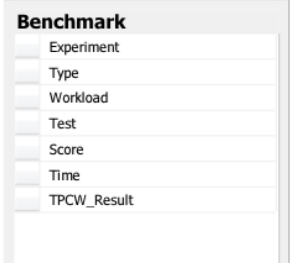
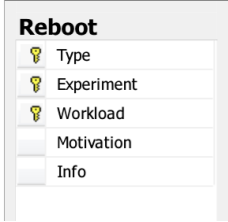
Table 1 – Collected data and other information: first version (reduced version), 9 tables.

	Benchmark Database tables	Description	Columns description																				
Data Tables		<p>The Microsoft Logman tool creates the tables here presented. The CounterData table contains, among other data, the values of each variable (<i>CounterValue</i>), and the time at which the value was registered (<i>CounterDateTime</i>). Each data is related to a single Counter by a <i>CounterID</i>, and each value is related to a (<i>CounterID</i>, <i>RecordIndex</i>) tuple.</p> <p>Details about each counter (identified by <i>CounterID</i> key) are contained in CounterDetails table.</p> <p>Finally, the DisplayToID table (often removed from</p>	<p style="text-align: center;">CounterData</p> <table border="1"> <tr> <td>GUID</td> <td>(Global Unique Identifier) single logging session ID (hexadecimal)</td> </tr> <tr> <td>CounterID</td> <td>monitored variable ID ([1...N_{var}])</td> </tr> <tr> <td>RecordIndex</td> <td>sequence number of the specific variable's value (sequential, [1...T_{max}])</td> </tr> <tr> <td>CounterDate Time</td> <td>timestamp of the specific variable's value (YYYY-MM-DD HH:MM:SS.mss)</td> </tr> <tr> <td>CounterValue</td> <td>specific variable's value</td> </tr> <tr> <td>FirstValueA</td> <td><not used></td> </tr> <tr> <td>FirstValueB</td> <td><not used></td> </tr> <tr> <td>SecondValueA</td> <td><not used></td> </tr> <tr> <td>SecondValueB</td> <td><not used></td> </tr> <tr> <td>MultiCount</td> <td><not used></td> </tr> </table>	GUID	(Global Unique Identifier) single logging session ID (hexadecimal)	CounterID	monitored variable ID ([1...N _{var}])	RecordIndex	sequence number of the specific variable's value (sequential, [1...T _{max}])	CounterDate Time	timestamp of the specific variable's value (YYYY-MM-DD HH:MM:SS.mss)	CounterValue	specific variable's value	FirstValueA	<not used>	FirstValueB	<not used>	SecondValueA	<not used>	SecondValueB	<not used>	MultiCount	<not used>
			GUID	(Global Unique Identifier) single logging session ID (hexadecimal)																			
			CounterID	monitored variable ID ([1...N _{var}])																			
			RecordIndex	sequence number of the specific variable's value (sequential, [1...T _{max}])																			
			CounterDate Time	timestamp of the specific variable's value (YYYY-MM-DD HH:MM:SS.mss)																			
			CounterValue	specific variable's value																			
			FirstValueA	<not used>																			
			FirstValueB	<not used>																			
			SecondValueA	<not used>																			
			SecondValueB	<not used>																			
			MultiCount	<not used>																			
			CounterDetails																				
			CounterID	single monitored variable's ID ([1...N _{var}])																			
MachineName	name of the system from																						

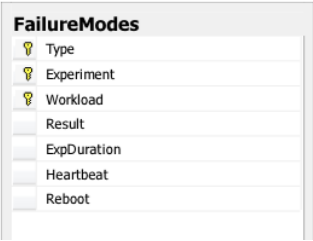
		<p>the database) has additional information about the logging session, identified by the GUID (Global Unique Identifier), as Log start and stop time, number of records, etc.</p> <p>A guide describing each column and relative values is available at Microsoft Technet website.</p>	<table border="1"> <tr> <td></td> <td>which the variable was collected (e.g., Lab-PC-1)</td> </tr> <tr> <td>ObjectName</td> <td>component of a system, made of various parts (e.g., Memory, OS, Disk ...)</td> </tr> <tr> <td>CounterName</td> <td>specific variable's name, belonging to an Object and a Machine</td> </tr> <tr> <td>CounterType</td> <td><not used></td> </tr> <tr> <td>DefaultScale</td> <td><not used></td> </tr> <tr> <td>InstanceName</td> <td><not used></td> </tr> <tr> <td>InstanceIndex</td> <td><not used></td> </tr> <tr> <td>ParentName</td> <td><not used></td> </tr> <tr> <td>ParentObjectID</td> <td><not used></td> </tr> <tr> <td>TimeBaseA</td> <td><not used></td> </tr> <tr> <td>TimeBaseB</td> <td><not used></td> </tr> <tr> <td colspan="2" style="text-align: center;">DisplayToID</td> </tr> <tr> <td>GUID</td> <td>(Global Unique Identifier) single logging session ID (hexadecimal)</td> </tr> <tr> <td>RunID</td> <td><not used></td> </tr> <tr> <td>DisplayString</td> <td><not used></td> </tr> <tr> <td>LogStartTime</td> <td><not used></td> </tr> <tr> <td>LogStopTime</td> <td><not used></td> </tr> <tr> <td>NumberOfRecords</td> <td><not used></td> </tr> <tr> <td>MinutesToUTC</td> <td><not used></td> </tr> <tr> <td>TimezoneName</td> <td><not used></td> </tr> </table>		which the variable was collected (e.g., Lab-PC-1)	ObjectName	component of a system, made of various parts (e.g., Memory, OS, Disk ...)	CounterName	specific variable's name, belonging to an Object and a Machine	CounterType	<not used>	DefaultScale	<not used>	InstanceName	<not used>	InstanceIndex	<not used>	ParentName	<not used>	ParentObjectID	<not used>	TimeBaseA	<not used>	TimeBaseB	<not used>	DisplayToID		GUID	(Global Unique Identifier) single logging session ID (hexadecimal)	RunID	<not used>	DisplayString	<not used>	LogStartTime	<not used>	LogStopTime	<not used>	NumberOfRecords	<not used>	MinutesToUTC	<not used>	TimezoneName	<not used>	
	which the variable was collected (e.g., Lab-PC-1)																																											
ObjectName	component of a system, made of various parts (e.g., Memory, OS, Disk ...)																																											
CounterName	specific variable's name, belonging to an Object and a Machine																																											
CounterType	<not used>																																											
DefaultScale	<not used>																																											
InstanceName	<not used>																																											
InstanceIndex	<not used>																																											
ParentName	<not used>																																											
ParentObjectID	<not used>																																											
TimeBaseA	<not used>																																											
TimeBaseB	<not used>																																											
DisplayToID																																												
GUID	(Global Unique Identifier) single logging session ID (hexadecimal)																																											
RunID	<not used>																																											
DisplayString	<not used>																																											
LogStartTime	<not used>																																											
LogStopTime	<not used>																																											
NumberOfRecords	<not used>																																											
MinutesToUTC	<not used>																																											
TimezoneName	<not used>																																											

Faultload table		<p>The Faultload table contains, for each experiment (identified by tuple <Type, Experiment, Workload>), the target of the fault injection, external library modules (<i>dlls</i>, if used), and the fault injection result, i.e., success or error.</p>	Type	type of run (<i>GR</i> or <i>FIR</i>)
			Experiment	run's sequential number ($[1 \dots N_{runs}]$)
			Workload	workload used during the specific run (e.g., <i>WKL1</i>)
			Target	fault injection target in the specific run (e.g., <i>svchost.exe</i> process)
			Module	specific fault injection target (<i>optional</i>)
			FaultType	type of fault injected, according to G-SWFIT recommendations (e.g., <i>MFC</i>)
			IntervalStart	< <i>fault injection tool-specific information</i> > index of the first location at which a fault is injected
			IntervalEnd	< <i>fault injection tool-specific information</i> > index of the last location at which a fault is injected
			Injection	1 = successful injection 0 = injection failure
			Description	injection tool output
Fault	specific fault (assembly-level mutation code)			

Experiments table		<p>The Experiment table collects the data about the current run, including the run sequence number, the workload used, the type (GR or FIR), the target of the fault injection (e.g., svchost.exe), and the “experiment finished” and “target system ready” flags.</p> <p>This table is used to synchronize Controller system and the Target system.</p>	Type	type of run (<i>GR</i> or <i>FIR</i>)
			Number	run's sequential number ($[1 \dots N_{runs}]$, “ <i>Experiment</i> ” column in other tables)
			Workload	workload used during the specific run (e.g., WKL ₁)
			Target	fault injection target in the specific run (e.g., svchost.exe process)
			exp_finished	flag indicating the current run is complete 1 = completed 0 = not completed
			Target_Ready	flag indicating the target is ready for executing a new run (e.g., if the target is still booting, the flag is 0) 1 = ready (after ramp-up) 0 = not ready
Timestamps table		<p>The Timestamps table contains information about the time at which a fault was injected, in each experiment.</p>	Type	type of run (<i>GR</i> or <i>FIR</i>)
			Experiment	run's sequential number ($[1 \dots N_{runs}]$, “ <i>Experiment</i> ” column in other tables)
			Workload	workload used during the specific run (e.g., WKL ₁)
			Timestamp	time at which a fault is injected, in a specific run (YYYY-MM-DD HH:MM:SS.mss)
Workload table		<p>The Workload table is a custom table. It contains additional information about the workload used (used for WKL₁).</p>	Type	type of run (<i>GR</i> or <i>FIR</i>)
			Experiment	run's sequential number ($[1 \dots N_{runs}]$, “ <i>Experiment</i> ” column in other tables)
			Signature	WKL ₁ results
			Checksum	WKL ₁ error-control checksum

Benchmark table		<p>The Benchmark table is a custom table, containing information about other workloads (WKL₂ and WKL_{3x}) used.</p>	Type	type of run (<i>GR</i> or <i>FIR</i>)
			Experiment	run's sequential number ($[1 \dots N_{runs}]$, " <i>Experiment</i> " column in other tables)
			Workload	workload used during the specific run (e.g., WKL ₁)
			Test	WKL ₂ single test result or WKL ₃ name
			Score	WKL ₂ single test score or WKL ₃ score
			Time	WKL ₂ single test time or WKL ₃ total time
			TPCW_Result	WKL ₃ result
			Reboot table	
Experiment	run's sequential number ($[1 \dots N_{runs}]$, " <i>Experiment</i> " column in other tables)			
Workload	workload used during the specific run (e.g., WKL ₁)			
Motivation	<p>motivation of the target system's reboot</p> <p style="text-align: center;"><i>Target-side</i></p> <p><i>Target</i> = no failure occurred <i>Heartbeat</i>⁵ = Crash failure (Failure mode FM₁, the Target's heartbeat stopped) <i>Reboot</i>⁴ = Hang failure (Failure mode FM₂, Target rebooted by the Controller)</p> <p style="text-align: center;"><i>Controller-side</i></p> <p><i>Controller</i> = failure or error occurred to controller machine <i>TPCW-RBE</i> = TPCW client error (client used by WKL₃)</p>			
Info				

⁵ Corrected from a previous version, where the definitions of Crash and Hang were inverted

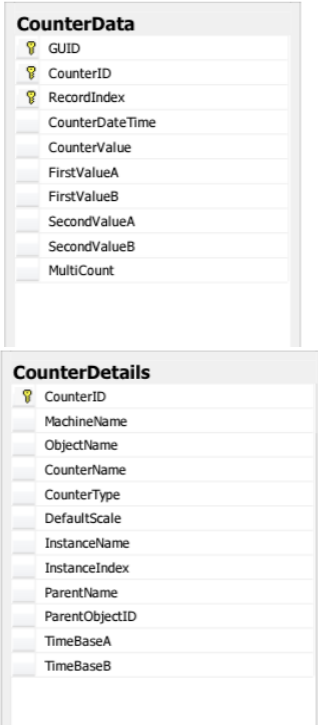
Failure Modes table		<p>The FailureModes contains information about the failures occurred during a certain experiment.</p> <p><i>This table is not actually used.</i></p>	Type	type of run (<i>GR</i> or <i>FIR</i>)
			Experiment	run's sequential number ($[1 \dots N_{runs}]$, " <i>Experiment</i> " column in other tables)
			Workload	workload used during the specific run (e.g., <i>WKL₁</i>)
			Result	workload result
			ExpDuration	run duration
			Heartbeat	Hang failure (Failure Mode <i>FM₂</i>)
			Reboot	Crash failure (Failure Mode <i>FM₁</i>)

5.2.2 Latest Database version (2013-2015) (T_2)

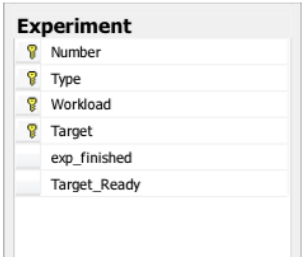
Databases were modified for addressing data organization drawbacks and limitations, and provide support to Failure Prediction models and their training and assessment. In particular, *Fault Injection* table, *Faultload* table and *Reboot* table were extended with additional information. On the other hand, new tables added were *StartingFaultload*, *FailurePredictor*, *FailurePredictionVariables* and *OnlineFailurePredictionResults*. In particular, the tables *StartingFaultload* and *Faultload* are used to contain the entire faultload, and to keep trace of the single specific fault injected in a single run.

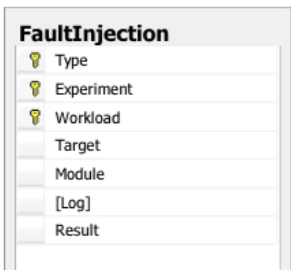
The tables are presented in **Table 2**.

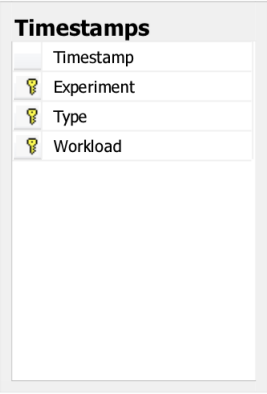
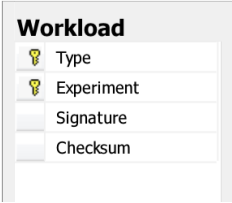
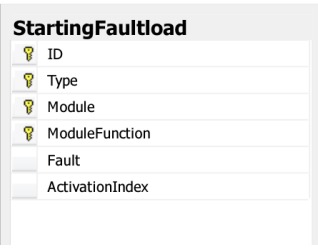
Table 2 – Collected data and other information: a newer, extended version (15 tables).

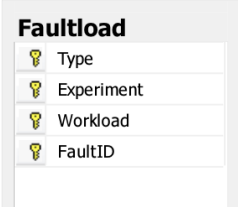
	Benchmark Database tables	Description	Columns description	
Data Tables		<p>The Microsoft Logman tool creates tables presented aside. The CounterData table contains, among other data, the values of each variable (<i>CounterValue</i>), and the time at which the value was registered (<i>CounterDateTime</i>). Each data is related to a single Counter by a <i>CounterID</i>, and each value is related to a (<i>CounterID</i>, <i>RecordIndex</i>) tuple.</p> <p>Details about each counter (identified by <i>CounterID</i> key) are contained in CounterDetails table.</p> <p>Finally, the DisplayToID tables has additional information about each value of a single Counter, identified by the <i>GUID</i> (<i>Global Unique Identifier</i>)</p>	CounterData	
			GUID	(Global Unique Identifier) single logging session ID (hexadecimal)
			CounterID	monitored variable ID ([1...N _{var}])
			RecordIndex	sequence number of the specific variable's value (sequential, [1...T _{max}])
			CounterDate Time	timestamp of the specific variable's value (YYYY-MM-DD HH:MM:SS.mss)
			CounterValue	specific variable's value
			FirstValueA	<not used>
			FirstValueB	<not used>
			SecondValueA	<not used>
			SecondValueB	<not used>
			MultiCount	<not used>
			CounterDetails	
			CounterID	single monitored variable's ID ([1...N _{var}])
			MachineName	name of the system from which the variable was collected (e.g., Lab-PC-1)
			ObjectName	component of a system, made of various parts (e.g.,

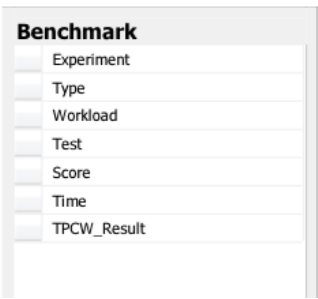
	<div data-bbox="284 309 580 595"> <p>DisplayToID</p> <ul style="list-style-type: none"> GUID RunID DisplayString LogStartTime LogStopTime NumberOfRecords MinutesToUTC TimeZoneName </div>	<p>key, as Log start and stop time, number of records, etc.</p> <p>A guide describing each column and relative values is available at Microsoft Technet website.</p>	<table border="1"> <tr> <td></td> <td>Memory, OS, Disk ...)</td> </tr> <tr> <td>CounterName</td> <td>specific variable's name, belonging to an Object and a Machine</td> </tr> <tr> <td>CounterType</td> <td><not used></td> </tr> <tr> <td>DefaultScale</td> <td><not used></td> </tr> <tr> <td>InstanceName</td> <td><not used></td> </tr> <tr> <td>InstanceIndex</td> <td><not used></td> </tr> <tr> <td>ParentName</td> <td><not used></td> </tr> <tr> <td>ParentObjectID</td> <td><not used></td> </tr> <tr> <td>TimeBaseA</td> <td><not used></td> </tr> <tr> <td>TimeBaseB</td> <td><not used></td> </tr> <tr> <td colspan="2" style="text-align: center;">DisplayToID</td> </tr> <tr> <td>GUID</td> <td>(Global Unique Identifier) single logging session ID (hexadecimal)</td> </tr> <tr> <td>CounterID</td> <td>monitored variable ID ([1...N_{var}])</td> </tr> <tr> <td>RecordIndex</td> <td>sequence number of the specific variable's collected value (sequential, [1...T_{max}])</td> </tr> <tr> <td>CounterDate Time</td> <td>timestamp of the specific variable's collected value (YYYY-MM-DD HH:MM:SS.mss)</td> </tr> <tr> <td>CounterValue</td> <td>specific variable's collected value</td> </tr> <tr> <td>FirstValueA</td> <td><not used></td> </tr> <tr> <td>FirstValueB</td> <td><not used></td> </tr> <tr> <td>SecondValueA</td> <td><not used></td> </tr> </table>		Memory, OS, Disk ...)	CounterName	specific variable's name, belonging to an Object and a Machine	CounterType	<not used>	DefaultScale	<not used>	InstanceName	<not used>	InstanceIndex	<not used>	ParentName	<not used>	ParentObjectID	<not used>	TimeBaseA	<not used>	TimeBaseB	<not used>	DisplayToID		GUID	(Global Unique Identifier) single logging session ID (hexadecimal)	CounterID	monitored variable ID ([1...N _{var}])	RecordIndex	sequence number of the specific variable's collected value (sequential, [1...T _{max}])	CounterDate Time	timestamp of the specific variable's collected value (YYYY-MM-DD HH:MM:SS.mss)	CounterValue	specific variable's collected value	FirstValueA	<not used>	FirstValueB	<not used>	SecondValueA	<not used>	
	Memory, OS, Disk ...)																																									
CounterName	specific variable's name, belonging to an Object and a Machine																																									
CounterType	<not used>																																									
DefaultScale	<not used>																																									
InstanceName	<not used>																																									
InstanceIndex	<not used>																																									
ParentName	<not used>																																									
ParentObjectID	<not used>																																									
TimeBaseA	<not used>																																									
TimeBaseB	<not used>																																									
DisplayToID																																										
GUID	(Global Unique Identifier) single logging session ID (hexadecimal)																																									
CounterID	monitored variable ID ([1...N _{var}])																																									
RecordIndex	sequence number of the specific variable's collected value (sequential, [1...T _{max}])																																									
CounterDate Time	timestamp of the specific variable's collected value (YYYY-MM-DD HH:MM:SS.mss)																																									
CounterValue	specific variable's collected value																																									
FirstValueA	<not used>																																									
FirstValueB	<not used>																																									
SecondValueA	<not used>																																									

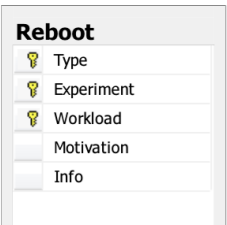
Experiments table		<p>The Experiment table collects the data about the <i>current</i> run, including the run sequence number, the workload used, the type (GR or FIR), the target of the fault injection (e.g., svchost.exe), and the “experiment finished” and “target system ready” flags.</p>	Type	type of run (<i>GR</i> or <i>FIR</i>)
			Number	run's sequential number ([1... <i>N_{runs}</i>], “ <i>Experiment</i> ” column in other tables)
			Workload	workload used during the specific run (e.g., <i>WKL_i</i>)
			Target	fault injection target in the specific run (e.g., svchost.exe process)
			exp_finished	flag indicating the current run is complete 1 = completed 0 = not completed
			Target_Ready	flag indicating the target is ready for executing a new run (e.g., if the target is still booting, the flag is 0) 1 = ready (after ramp-up time) 0 = not ready

Fault Injection table		<p>(Updated: added “Log” and “Result”)</p> <p>The Fault Injection table (former <i>Faultload</i>) contains, for each experiment (identified by tuple <Type, Experiment, Workload>), the target of the fault injection, external library modules (<i>dlls</i>, if used), and the fault injection result, i.e., success or error.</p> <p>This table is used to synchronize Controller system and the Target system.</p>	Type	type of run (<i>GR</i> or <i>FIR</i>)
			Experiment	run's sequential number ([1... <i>N_{runs}</i>], “ <i>Experiment</i> ” column in other tables)
			Workload	workload used during the specific run (e.g., <i>WKL_i</i>)
			Target	fault injection target in the specific run (e.g., svchost.exe process)
			Module	specific fault injection target (<i>optional</i>)
			Log	(former “Description”) injection tool output
			Result	Injection result 0 = Injection failed 1 = Injection succeeded

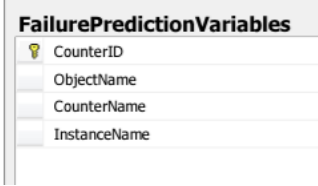
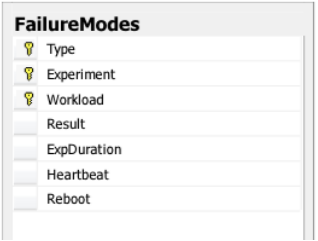
Timestamps table		<p>The Timestamps table contains information about the time at which a fault was injected, for each experiment.</p>	Type	type of run (<i>GR</i> or <i>FIR</i>)
			Experiment	run's sequential number ($[1 \dots N_{runs}]$, " <i>Experiment</i> " column in other tables)
			Workload	workload used during the specific run (e.g., WKL_1)
			Timestamp	time at which a fault is injected, in a specific run (YYYY-MM-DD HH:MM:SS.mss)
Workload table		<p>The Workload table is a custom table. It contains additional information about the workload used (used for WKL_1).</p>	Type	type of run (<i>GR</i> or <i>FIR</i>)
			Experiment	run's sequential number ($[1 \dots N_{runs}]$, " <i>Experiment</i> " column in other tables)
			Signature	WKL_1 results
			Checksum	WKL_1 error-control checksum
StartingFaultload table		<p><i>*New*</i></p> <p>The StartingFaultload table contains a set of faults that can be injected in the target system. <i>The information contained into this table is specific to the injection tool used in this Ph.D. thesis work.</i></p> <p>It is worth noting that the set of all injectable faults can be very large, thus the user may have to select a part of such faults. This table is used to build a Faultload table, which contains the actual faults to inject, as such set can change when using, for instance, a different dataset.</p>	ID	fault numerical ID
			Type	type of fault (e.g., MFC – Missing function call, MIFS – Missing IF statement, ...)
			Module	target module (e.g., ntdll.dll)
			ModuleFunction	target function (e.g., Memory Allocation), number
			Fault	mutation operator
			ActivationIndex	fault activation rate, which can be found experimentally (e.g., 8% for MFC faults)

Faultload table		<p>(Updated: added "FaultID", details about fault moved)</p> <p>The Faultload table contains the set of faults that have to be effectively injected. In this case, each fault is identified with a <i>FaultID</i>, and the details about the faults, as well as the effective code mutant, is available in the <i>StartingFaultload</i> table.</p>	Type	type of run (<i>FIR</i> only)
			Experiment	run's sequential number ($[1 \dots N_{runs}]$, " <i>Experiment</i> " column in other tables)
			Workload	workload used during the specific run (e.g., <i>WKL₁</i>)
			FaultID	ID of the fault injected in the run (details about the fault are in table <i>StartingFaultload</i>)

Benchmark table		<p>The Benchmark table is a custom table, containing information about other workloads (<i>WKL₂</i> and <i>WKL_{3x}</i>) used.</p>	Experiment	type of run (<i>GR</i> or <i>FIR</i>)
			Type	run's sequential number ($[1 \dots N_{runs}]$, " <i>Experiment</i> " column in other tables)
			Workload	workload used during the specific run (e.g., <i>WKL₁</i>)
			Test	<i>WKL₂</i> single test result or <i>WKL₃</i> name
			Score	<i>WKL₂</i> single test score or <i>WKL₃</i> score
			Time	<i>WKL₂</i> single test time or <i>WKL₃</i> total time
			TPCW_Result	<i>WKL₃</i> result

Reboot table		<p>(Updated: added "Info")</p> <p>The Reboot table contains information about the <i>failure</i> occurred in the target system, as the failure type and the status of the controls used to detect the failure.</p>	Type	type of run (<i>GR</i> or <i>FIR</i>)
			Experiment	run's sequential number ($[1..N_{runs}]$, " <i>Experiment</i> " column in other tables)
			Workload	workload used during the specific run (e.g., WKL_i)
			Motivation	motivation of the target system's reboot <i>Target-side</i> <i>Target</i> = no failure occurred <i>Heartbeat^t</i> = Crash failure (Failure mode FM_1 , the Target's heartbeat stopped) <i>Reboot^t</i> = Hang failure (Failure mode FM_2 , Target rebooted by the Controller) <i>Controller-side</i> <i>Controller</i> = failure or error occurred to controller machine <i>TPCW-RBE</i> = TPCW client error (client used by WKL_s)
			Info	status of the controls used to detect the failure (containing the actual failure detection rule for the specific experiment)

Failure Predictor table	<div style="border: 1px solid gray; padding: 5px; margin-bottom: 10px;"> <p>FailurePredictor</p> <ul style="list-style-type: none"> PredictorMetaID ModelID DataFeatures DataMean DataDeviation PredictionMethod PredictionModel PredictionOptThreshold delta_L delta_P FM PredictionOtherParameters OtherInformation updated </div>	<p style="color: red; text-align: center;"><i>*New*</i></p> <p>The FailurePredictor table contains information about the failure prediction models used. In particular, we use an instantiation of a predictor for each parameter of the failure prediction problem, i.e., for each defined value of the tuple $\langle \mathit{delta_L}, \mathit{delta_P}, \mathit{data\ features} \rangle$. The present table contains the prediction model (<i>PredictionModel</i>), details about the training process (e.g., <i>DataFeatures</i>, <i>DataMean/Deviation</i>, <i>PredictionOptThreshold</i> and <i>FM</i> - the failure mode to predict), and a flag indicating if the predictor was updated.</p>	<p>PredictorMetaID</p>	<p>ID associated to the predictor type (e.g., SVM_time_window_classifier_gaussian)</p>
			<p>ModelID</p>	<p>ID associated to the specific prediction scenario (e.g., SVM_tw_3_FM2_10_10)</p>
			<p>DataFeatures</p>	<p>variables used in the specific prediction execution</p>
			<p>DataMean</p>	<p>mean of the data used in the specific prediction execution</p>
			<p>DataDeviation</p>	<p>deviation of the data used in the specific prediction execution</p>
			<p>PredictionMethod</p>	<p>description of the prediction method (e.g., SVM)</p>
			<p>PredictionModel</p>	<p>model obtained after the training phase (predictor status)</p>
			<p>PredictionOptThreshold</p>	<p>predictor's optimal threshold (<i>outputs of classifiers are thresholded in order to obtain one of the output classes</i>)</p>
			<p>delta_L</p>	<p>value of Δt_L used in the specific prediction execution</p>
			<p>delta_P</p>	<p>value of Δt_P used in the specific prediction execution</p>
			<p>FM</p>	<p>type of failure predicted (e.g., Hang - FM₂)</p>
			<p>PredictionOtherParameters</p>	<p>other predictor's parameters</p>
			<p>OtherInformation</p>	<p>other predictor's info</p>
			<p>updated</p>	<p>0 = predictor not updated in the specific prediction execution 1 = predictor updated</p>

Failure Prediction Variables table		<p><i>*New*</i></p> <p>The FailurePredictionVariables table contains information about the variables that, among all the ones monitored, were used to train the predictor. Each predictor can have a different set of variables, and that such variables are associated to each predictor instantiation in the FailurePrediction table through the <i>DataFeatures</i> column, containing a vector of CounterID values.</p> <p><i>This table was actually not used.</i></p>	CounterID	single monitored variable's ID
			ObjectName	monitored system component (e.g., CPU, Memory, Disk ...)
			CounterName	specific variable's name (e.g., exceptions/s)
			InstanceName	specific object instance name (e.g., CPU-1)
Failure Modes table		<p>The FailureModes is a help table containing information about the failures occurred during a certain experiment. The columns identify some Failure Modes we take into account for the benchmark.</p> <p><i>This table is not used.</i></p>	Type	type of run (GR or FIR)
			Experiment	run's sequential number ($[1 \dots N_{runs}]$, "Experiment" column in other tables)
			Workload	workload used during the specific run (e.g., WKL ₁)
			Result	workload result
			ExpDuration	run duration
			Heartbeat	Hang failure (Failure Mode FM ₂)
			Reboot	Crash failure (Failure Mode FM ₁)

Online Failure Prediction Results table

OnlineFailurePrediction_Results		Type	type of run (GR or FIR)
Type		Experiment	run's sequential number ([1...Nruns], "Experiment" column in other tables)
Experiment		Workload	workload used during the specific run (e.g., WKL _i)
Workload		PredictionMetaID	ID associated to the predictor type (e.g., SVM_time_window_classifier_gaussian)
PredictorMetaID		ModelID	ID associated to the specific prediction scenario (e.g., SVM_tw_3_FM2_10_10)
ModelID		PredictionValues	raw prediction values
PredictionValues		PredictionValues Labelled	labelled prediction values
PredictionValuesLabelled		datasetLength	length of the dataset
datasetLength		LabelledRun	labels associated to the selected run
LabelledRun		TP	predictor's performance associated to the selected run: True positives
TP		TN	pp: True Negatives
TN		FP	pp: False Positives
FP		FN	pp: False Negatives
FN		Precision	pp: Precision
Precision		Recall	pp: Recall
Recall		FMeasure	pp: F-Measure
FMeasure		NegativePrecision	pp: Precision computed according to non-failure situations
NegativePrecision		NegativeRecall	pp: NegativeRecall
NegativeRecall		NegativeFMeasure	pp: NegativeFMeasure
NegativeFMeasure		CutOffThreshold	optimal threshold obtained by ROC analysis
CutOffThreshold		SE	standard error computed between labeled data and predictor's output
SE		ROCxr	ROC x-axis values
ROCxr		ROCYr	ROC y-axis values
ROCYr		ROCAUC	ROC-AUC
ROCAUC			

New

The **OnlineFailurePrediction Results** table contains the results of a single predictor's execution, using data coming from a single run.

5.3 Monitored data

In the following, we present the information collected from the Target systems, grouped in singular variables. In particular, in the first experiments we collected 233 variables, and reduced this number to 170, pruning the variables that resulted in being not useful (e.g., constant-valued variables, variables with no value at all, etc.), and differentiating some other (e.g., page faults/s from a specific process as “svchost.exe”, instead of from the *total* number of page faults from all the running processes). The two sets are presented in Table 3, being V_1 the earlier set with 233 variables, and V_2 being the latter set with 170 variables. Table 3 presents the two sets in a comparative fashion. Note that only the variables names are presented: the full description of each variable can be found in the Performance Monitor tool, present in each Microsoft Windows’s family OSs.

Note:

- x = variable not present in the second set of variables V_2
- $diff$ = variable present in the second set, but associated to a different instance of the monitored object (*differentiated*)

Table 3

V_1 (233 vars)				V_2 (170 vars)		
ObjectName	CounterName	Instance Name	Notes	ObjectName	CounterName	Instance Name
.NET CLR Exceptions	# of Exceps Thrown	Global	x			
.NET CLR Exceptions	# of Exceps Thrown / sec	Global	x			
.NET CLR Exceptions	# of Filters / sec	Global	x			
.NET CLR Exceptions	# of Finallys / sec	Global	x			
.NET CLR Exceptions	Throw To Catch Depth / sec	Global	x			
.NET CLR LocksAndThreads	# of current logical Threads	Global	x			
.NET CLR LocksAndThreads	# of current physical Threads	Global	x			
.NET CLR LocksAndThreads	# of current recognized threads	Global	x			

.NET CLR LocksAndThreads	# of total recognized threads	Global	x			
.NET CLR LocksAndThreads	Contention Rate / sec	Global	x			
.NET CLR LocksAndThreads	Current Queue Length	Global	x			
.NET CLR LocksAndThreads	Queue Length / sec	Global	x			
.NET CLR LocksAndThreads	Queue Length Peak	Global	x			
.NET CLR LocksAndThreads	rate of recognized threads / sec	Global	x			
.NET CLR LocksAndThreads	Total # of Contentions	Global	x			
Cache	Async Copy Reads/sec	NULL		Cache	Async Copy Reads/sec	NULL
Cache	Async Data Maps/sec	NULL	x			
Cache	Async Fast Reads/sec	NULL	x			
Cache	Async MDL Reads/sec	NULL	x			
Cache	Async Pin Reads/sec	NULL	x			
Cache	Copy Read Hits %	NULL		Cache	Copy Read Hits %	NULL
Cache	Copy Reads/sec	NULL	x			
Cache	Data Flush Pages/sec	NULL		Cache	Data Flush Pages/sec	NULL
Cache	Data Flushes/sec	NULL		Cache	Data Flushes/sec	NULL
Cache	Data Map Hits %	NULL	x			
Cache	Data Map Pins/sec	NULL		Cache	Data Map Pins/sec	NULL
Cache	Data Maps/sec	NULL		Cache	Data Maps/sec	NULL
Cache	Fast Read Not Possibles/sec	NULL	x			
Cache	Fast Read Resource Misses/sec	NULL	x			
Cache	Fast Reads/sec	NULL	x			
Cache	Lazy Write Flushes/sec	NULL		Cache	Lazy Write Flushes/sec	NULL
Cache	Lazy Write Pages/sec	NULL		Cache	Lazy Write Pages/sec	NULL
Cache	MDL Read Hits %	NULL	x			
Cache	MDL Reads/sec	NULL	x			
Cache	Pin Read Hits %	NULL	x			
Cache	Pin Reads/sec	NULL	x			
Cache	Read Aheads/sec	NULL	x			
Cache	Sync Copy Reads/sec	NULL	x			

Cache	Sync Data Maps/sec	NULL		Cache	Sync Data Maps/sec	NULL
Cache	Sync Fast Reads/sec	NULL	x			
Cache	Sync MDL Reads/sec	NULL	x			
Cache	Sync Pin Reads/sec	NULL	x			
Job Object Details	% Privileged Time	Total	x			
Job Object Details	% Processor Time	Total	x			
Job Object Details	% User Time	Total	x			
Job Object Details	Creating Process ID	Total	x			
Job Object Details	Elapsed Time	Total	x			
Job Object Details	Handle Count	Total	x			
Job Object Details	ID Process	Total	x			
Job Object Details	IO Data Bytes/sec	Total	x			
Job Object Details	IO Data Operations/sec	Total	x			
Job Object Details	IO Other Bytes/sec	Total	x			
Job Object Details	IO Other Operations/sec	Total	x			
Job Object Details	IO Read Bytes/sec	Total	x			
Job Object Details	IO Read Operations/sec	Total	x			
Job Object Details	IO Write Bytes/sec	Total	x			
Job Object Details	IO Write Operations/sec	Total	x			
Job Object Details	Page Faults/sec	Total	x			
Job Object Details	Page File Bytes	Total	x			
Job Object Details	Page File Bytes Peak	Total	x			
Job Object Details	Pool Nonpaged Bytes	Total	x			
Job Object Details	Pool Paged Bytes	Total	x			
Job Object Details	Priority Base	Total	x			
Job Object Details	Private Bytes	Total	x			
Job Object Details	Thread Count	Total	x			
Job Object Details	Virtual Bytes	Total	x			
Job Object Details	Virtual Bytes Peak	Total	x			
Job Object Details	Working Set	Total	x			

Job Object Details	Working Set Peak	Total	x			
Job Object	Current % Kernel Mode Time	Total	x			
Job Object	Current % Processor Time	Total	x			
Job Object	Current % User Mode Time	Total	x			
Job Object	Pages/Sec	Total	x			
Job Object	Process Count - Active	Total	x			
Job Object	Process Count - Terminated	Total	x			
Job Object	Process Count - Total	Total	x			
Job Object	This Period mSec-Kernel Mode	Total	x			
Job Object	This Period mSec - Processor	Total	x			
Job Object	This Period mSec - User Mode	Total	x			
Job Object	Total mSec - Kernel Mode	Total	x			
Job Object	Total mSec - Processor	Total	x			
Job Object	Total mSec - User Mode	Total	x			
LogicalDisk	% Disk Read Time	Total		LogicalDisk	% Disk Read Time	Total
LogicalDisk	% Disk Time	Total		LogicalDisk	% Disk Time	Total
LogicalDisk	% Disk Write Time	Total		LogicalDisk	% Disk Write Time	Total
LogicalDisk	% Free Space	Total	x			
LogicalDisk	% Idle Time	Total		LogicalDisk	% Idle Time	Total
LogicalDisk	Avg. Disk Bytes/Read	Total		LogicalDisk	Avg. Disk Bytes/Read	Total
LogicalDisk	Avg. Disk Bytes/Transfer	Total		LogicalDisk	Avg. Disk Bytes/Transfer	Total
LogicalDisk	Avg. Disk Bytes/Write	Total		LogicalDisk	Avg. Disk Bytes/Write	Total
LogicalDisk	Avg. Disk Queue Length	Total		LogicalDisk	Avg. Disk Queue Length	Total
LogicalDisk	Avg. Disk Read Queue Length	Total		LogicalDisk	Avg. Disk Read Queue Length	Total
LogicalDisk	Avg. Disk sec/Read	Total		LogicalDisk	Avg. Disk sec/Read	Total
LogicalDisk	Avg. Disk sec/Transfer	Total		LogicalDisk	Avg. Disk sec/Transfer	Total
LogicalDisk	Avg. Disk sec/Write	Total		LogicalDisk	Avg. Disk sec/Write	Total
LogicalDisk	Avg. Disk Write Queue Length	Total		LogicalDisk	Avg. Disk Write Queue Length	Total
LogicalDisk	Current Disk Queue Length	Total		LogicalDisk	Current Disk Queue Length	Total
LogicalDisk	Disk Bytes/sec	Total		LogicalDisk	Disk Bytes/sec	Total

LogicalDisk	Disk Read Bytes/sec	Total	x			
LogicalDisk	Disk Reads/sec	Total	x			
LogicalDisk	Disk Transfers/sec	Total		LogicalDisk	Disk Transfers/sec	Total
LogicalDisk	Disk Write Bytes/sec	Total		LogicalDisk	Disk Write Bytes/sec	Total
LogicalDisk	Disk Writes/sec	Total		LogicalDisk	Disk Writes/sec	Total
LogicalDisk	Free Megabytes	Total	x			
LogicalDisk	Split IO/Sec	Total		LogicalDisk	Split IO/Sec	Total
Memory	% Committed Bytes In Use	NULL		Memory	% Committed Bytes In Use	NULL
Memory	Available Bytes	NULL	x			
Memory	Available KBytes	NULL	x			
Memory	Available MBytes	NULL	x			
Memory	Cache Bytes	NULL		Memory	Cache Bytes	NULL
Memory	Cache Bytes Peak	NULL	x			
Memory	Cache Faults/sec	NULL		Memory	Cache Faults/sec	NULL
Memory	Commit Limit	NULL	x			
Memory	Committed Bytes	NULL		Memory	Committed Bytes	NULL
Memory	Demand Zero Faults/sec	NULL		Memory	Demand Zero Faults/sec	NULL
Memory	Free System Page Table Entries	NULL		Memory	Free System Page Table Entries	NULL
Memory	Page Faults/sec	NULL		Memory	Page Faults/sec	NULL
Memory	Page Reads/sec	NULL		Memory	Page Reads/sec	NULL
Memory	Page Writes/sec	NULL		Memory	Page Writes/sec	NULL
Memory	Pages Input/sec	NULL		Memory	Pages Input/sec	NULL
Memory	Pages Output/sec	NULL		Memory	Pages Output/sec	NULL
Memory	Pages/sec	NULL		Memory	Pages/sec	NULL
Memory	Pool Nonpaged Allocs	NULL		Memory	Pool Nonpaged Allocs	NULL
Memory	Pool Nonpaged Bytes	NULL		Memory	Pool Nonpaged Bytes	NULL
Memory	Pool Paged Allocs	NULL		Memory	Pool Paged Allocs	NULL
Memory	Pool Paged Bytes	NULL		Memory	Pool Paged Bytes	NULL
Memory	Pool Paged Resident Bytes	NULL		Memory	Pool Paged Resident Bytes	NULL
Memory	System Cache Resident Bytes	NULL		Memory	System Cache Resident Bytes	NULL

Memory	System Code Resident Bytes	NULL	x			
Memory	System Code Total Bytes	NULL	x			
Memory	System Driver Resident Bytes	NULL	x			
Memory	System Driver Total Bytes	NULL	x			
Memory	Transition Faults/sec	NULL	x			
Memory	Write Copies/sec	NULL	x			
Objects	Events	NULL		Objects	Events	NULL
Objects	Mutexes	NULL		Objects	Mutexes	NULL
Objects	Processes	NULL		Objects	Processes	NULL
Objects	Sections	NULL		Objects	Sections	NULL
Objects	Semaphores	NULL		Objects	Semaphores	NULL
Objects	Threads	NULL		Objects	Threads	NULL
Paging File	% Usage	_Total	x			
Paging File	% Usage Peak	_Total		Paging File	% Usage Peak	_Total
PhysicalDisk	% Disk Read Time	_Total	x			
PhysicalDisk	% Disk Time	_Total	x			
PhysicalDisk	% Disk Write Time	_Total	x			
PhysicalDisk	% Idle Time	_Total	x			
PhysicalDisk	Avg. Disk Bytes/Read	_Total	x			
PhysicalDisk	Avg. Disk Bytes/Transfer	_Total	x			
PhysicalDisk	Avg. Disk Bytes/Write	_Total	x			
PhysicalDisk	Avg. Disk Queue Length	_Total	x			
PhysicalDisk	Avg. Disk Read Queue Length	_Total	x			
PhysicalDisk	Avg. Disk sec/Read	_Total	x			
PhysicalDisk	Avg. Disk sec/Transfer	_Total	x			
PhysicalDisk	Avg. Disk sec/Write	_Total	x			
PhysicalDisk	Avg. Disk Write Queue Length	_Total	x			
PhysicalDisk	Current Disk Queue Length	_Total	x			
PhysicalDisk	Disk Bytes/sec	_Total	x			
PhysicalDisk	Disk Read Bytes/sec	_Total	x			

PhysicalDisk	Disk Reads/sec	Total	x			
PhysicalDisk	Disk Transfers/sec	Total	x			
PhysicalDisk	Disk Write Bytes/sec	Total	x			
PhysicalDisk	Disk Writes/sec	Total	x			
PhysicalDisk	Split IO/Sec	Total	x			
Process	% Privileged Time	Total	<i>diff</i>	Process	% Privileged Time	Total
				Process	% Privileged Time	explorer
				Process	% Privileged Time	java
				Process	% Privileged Time	svchost
Process	% Processor Time	Total	<i>diff</i>	Process	% Processor Time	Total
				Process	% Processor Time	explorer
				Process	% Processor Time	java
				Process	% Processor Time	svchost
Process	% User Time	Total	<i>diff</i>	Process	% User Time	Total
				Process	% User Time	explorer
				Process	% User Time	java
				Process	% User Time	svchost
Process	Creating Process ID	Total	x			
Process	Elapsed Time	Total	<i>diff</i>	Process	Elapsed Time	explorer
				Process	Elapsed Time	java
				Process	Elapsed Time	svchost
Process	Handle Count	Total	<i>diff</i>	Process	Handle Count	Total
				Process	Handle Count	explorer
				Process	Handle Count	java
				Process	Handle Count	svchost
Process	ID Process	Total	x			
Process	IO Data Bytes/sec	Total	<i>diff</i>	Process	IO Data Bytes/sec	Total
				Process	IO Data Bytes/sec	explorer
				Process	IO Data Bytes/sec	java
				Process	IO Data Bytes/sec	svchost

Process	IO Data Operations/sec	Total	<i>diff</i>	Process	IO Data Operations/sec	Total
				Process	IO Data Operations/sec	explorer
				Process	IO Data Operations/sec	java
				Process	IO Data Operations/sec	svchost
Process	IO Other Bytes/sec	Total	<i>diff</i>	Process	IO Other Bytes/sec	Total
				Process	IO Other Bytes/sec	explorer
				Process	IO Other Bytes/sec	java
				Process	IO Other Bytes/sec	svchost
Process	IO Other Operations/sec	Total	<i>diff</i>	Process	IO Other Operations/sec	Total
				Process	IO Other Operations/sec	explorer
				Process	IO Other Operations/sec	java
				Process	IO Other Operations/sec	svchost
Process	IO Read Bytes/sec	Total	<i>diff</i>	Process	IO Read Bytes/sec	Total
				Process	IO Read Bytes/sec	explorer
				Process	IO Read Bytes/sec	java
				Process	IO Read Bytes/sec	svchost
Process	IO Read Operations/sec	Total	x			
Process	IO Write Bytes/sec	Total	<i>diff</i>	Process	IO Write Bytes/sec	Total
				Process	IO Write Bytes/sec	explorer
				Process	IO Write Bytes/sec	java
				Process	IO Write Bytes/sec	svchost
Process	IO Write Operations/sec	Total	x			
Process	Page Faults/sec	Total	<i>diff</i>	Process	Page Faults/sec	Total
				Process	Page Faults/sec	explorer
				Process	Page Faults/sec	java
				Process	Page Faults/sec	svchost
Process	Page File Bytes	Total	<i>diff</i>	Process	Page File Bytes	Total
				Process	Page File Bytes	explorer
				Process	Page File Bytes	java
				Process	Page File Bytes	svchost

Process	Page File Bytes Peak	Total	x			
Process	Pool Nonpaged Bytes	Total	diff	Process	Page Faults/sec	Total
				Process	Page Faults/sec	explorer
				Process	Page Faults/sec	java
				Process	Page Faults/sec	svchost
Process	Pool Paged Bytes	Total	diff	Process	Page File Bytes	Total
				Process	Page File Bytes	explorer
				Process	Page File Bytes	java
				Process	Page File Bytes	svchost
Process	Priority Base	Total	x			
Process	Private Bytes	Total	diff	Process	Private Bytes	Total
				Process	Private Bytes	explorer
				Process	Private Bytes	java
				Process	Private Bytes	svchost
Process	Thread Count	Total	diff	Process	Thread Count	Total
				Process	Thread Count	explorer
				Process	Thread Count	java
				Process	Thread Count	svchost
Process	Virtual Bytes	Total	diff	Process	Virtual Bytes	Total
				Process	Virtual Bytes	explorer
				Process	Virtual Bytes	java
				Process	Virtual Bytes	svchost
Process	Virtual Bytes Peak	Total	x			
Process	Working Set	Total	diff	Process	Working Set	Total
				Process	Working Set	explorer
				Process	Working Set	java
				Process	Working Set	svchost
Process	Working Set Peak	Total	x			
Processor	% C1 Time	Total	diff	Processor	% C1 Time	0
				Processor	% C1 Time	1

Processor	% C2 Time	Total	<i>diff</i>	Processor	% C2 Time	0
				Processor	% C2 Time	1
Processor	% C3 Time	Total	x			
Processor	% DPC Time	Total	<i>diff</i>	Processor	% DPC Time	0
				Processor	% DPC Time	1
Processor	% Idle Time	Total	<i>diff</i>	Processor	% Idle Time	0
				Processor	% Idle Time	1
Processor	% Interrupt Time	Total	<i>diff</i>	Processor	% Interrupt Time	0
				Processor	% Interrupt Time	1
Processor	% Privileged Time	Total	<i>diff</i>	Processor	% Interrupt Time	0
				Processor	% Interrupt Time	1
Processor	% Processor Time	Total	<i>diff</i>	Processor	% Processor Time	0
				Processor	% Processor Time	1
Processor	% User Time	Total	<i>diff</i>	Processor	% Processor Time	0
				Processor	% Processor Time	1
Processor	C1 Transitions/sec	Total	<i>diff</i>	Processor	% Processor Time	0
				Processor	% Processor Time	1
Processor	C2 Transitions/sec	Total	<i>diff</i>	Processor	C2 Transitions/sec	0
				Processor	C2 Transitions/sec	1
Processor	C3 Transitions/sec	Total	x			
Processor	DPC Rate	Total	<i>diff</i>	Processor	DPC Rate	0
				Processor	DPC Rate	1
Processor	DPCs Queued/sec	Total	<i>diff</i>	Processor	DPCs Queued/sec	0
				Processor	DPCs Queued/sec	1
Processor	Interrupts/sec	Total	<i>diff</i>	Processor	Interrupts/sec	0
				Processor	Interrupts/sec	1
System	% Registry Quota In Use	NULL	x			
System	Alignment Fixups/sec	NULL	x			
System	Context Switches/sec	NULL		System	Context Switches/sec	NULL
System	Exception Dispatches/sec	NULL		System	Exception Dispatches/sec	NULL

System	File Control Bytes/sec	NULL		System	File Control Bytes/sec	NULL
System	File Control Operations/sec	NULL		System	File Control Operations/sec	NULL
System	File Data Operations/sec	NULL		System	File Data Operations/sec	NULL
System	File Read Bytes/sec	NULL		System	File Read Bytes/sec	NULL
System	File Read Operations/sec	NULL	x			
System	File Write Bytes/sec	NULL		System	File Write Bytes/sec	NULL
System	File Write Operations/sec	NULL	x			
System	Floating Emulations/sec	NULL	x			
System	Processes	NULL		System	Processes	NULL
System	Processor Queue Length	NULL	x			
System	System Calls/sec	NULL		System	System Calls/sec	NULL
System	System Up Time	NULL	x			
Thread	% Privileged Time	_Total		Thread	% Privileged Time	_Total
Thread	% Processor Time	_Total		Thread	% Processor Time	_Total
Thread	% User Time	_Total		Thread	% User Time	_Total
Thread	Context Switches/sec	_Total		Thread	Context Switches/sec	_Total
Thread	Elapsed Time	_Total	x			
Thread	ID Process	_Total	x			
Thread	ID Thread	_Total	x			
Thread	Priority Base	_Total	x			
Thread	Priority Current	_Total	x			
Thread	Start Address	_Total	x			
Thread	Thread State	_Total	x			
Thread	Thread Wait Reason	_Total	x			
.NET CLR Exceptions	# of Exceps Thrown	_Global	x			