A Configurable Software Aging Detection and Rejuvenation Agent for Android

Domenico Cotroneo, Luigi De Simone, Roberto Natella, Roberto Pietrantuono, Stefano Russo
DIETI - Università degli Studi di Napoli Federico II, Via Claudio 21, 80125 Napoli, Italy
{cotroneo, luigi.desimone, roberto.natella, roberto.pietrantuono, stefano.russo}@unina.it

***Abstract*—Empirical studies have shown that the Android operating system for mobile devices suffers from aging phenomena, which can be detected by monitoring specific system processes and correlating a number of workload and performance indicators. These vary depending on the operating system version and on the many hardware adaptations and customizations performed by the device manufacturer.**

This paper presents the design of ADARTA, an aging detection and rejuvenation tool for Android. The tool is a software agent which: *i)* performs selective monitoring of system processes and of trends in system and load indicators; *ii)* detects the aging state and estimates the time remaining to its failure, by means of heuristic rules; *iii)* schedules and applies rejuvenation, based on the estimated time-to-failure. The agent rules and parameters have been defined for ease of configuration and tuning by device designers. A stress testing experiment is described, showing ADARTA's configurability for the device under test, and its effectiveness in extending the time to failure.

***Index Terms*—Software Aging; Software Rejuvenation; Android operating system.**

I. Introduction

Software aging has been shown to affect many complex systems, such as such as web servers [1], operating systems [2], middleware [3] and cloud systems [4], due to issues such as memory and resource leaking known as *aging-related bugs* [5], [6]. The Android mobile operating system (OS) makes no exception, as it has become one of the largest open-source projects with millions of lines of Java and C++ code[1], and is heavily modified by manufacturers of mobile devices for adaptation to specific hardware and peripherals, as well as to deliver customized features. Empirical research showed that Android can experience a gradual decay in terms of responsiveness and resource utilization, and that it is prone to failures (e.g., the crash of key system processes) under stressful conditions [8]–[11].

In order to counteract software aging in Android devices, recent studies have investigated how to apply rejuvenation techniques in a mobile context. Qiao et al. [12] and Xiang et al. [13] proposed models (using Markov chains and Stochastic Petri Nets) for time-based detection and rejuvenation at the application or OS level, by taking into account the lifecycle of mobile applications. Other studies by Huo et al. [14], Qiao et al. [15], and Weng et al. [16] have pursued aging detection from a measurement-based perspective, by applying time series analysis and machine learning (e.g., ARIMA, SVM, neural networks) to forecast aging in resource utilization and performance metrics.

This paper presents the design of a configurable aging detection and rejuvenation agent for the Android OS, named ADARTA, developed in the context of an R&D project with Huawei Technologies Co., Ltd. ADARTA is designed based on the findings of own previous empirical study on Android aging [8]. The agent performs selective monitoring of Android processes (e.g., System Server) and performance indicators (e.g., free memory, garbage collection times) correlated to aging phenomena, as well as of workload indicators (e.g., applications' activation time). Then, it applies trend estimation techniques to such aging indicators. Finally, it computes a runtime indicator of the aging state of the Android device through heuristic rules, which can be easily interpreted and tuned by device designers. Based on such "global" indicator, rejuvenation is triggered. We also present a stress testing experiment on a real Android Huawei device, showing ADARTA's configurability and its effectiveness in detecting aging and computing the time-to-aging-failure. While there are several techniques and tools that can be adopted for individual tasks in OS aging detection and rejuvenation (e.g., the Monkey tool [17], used to perform stress tests [18]), we are not aware of any other such comprehensive software agent providing automated system and load monitoring, aging detection and rejuvenation triggering for Android.

The rest of the paper is organized as follows. Section II provides background definitions. Section III describes the architecture of ADARTA, while Section IV presents the monitoring and detection process. Section V describes the parameters which can be tuned by practitioners to tailor ADARTA to a specific platform. Section VI describes the experiment. Section VII provides final remarks.

II. Definitions

ADARTA monitors the following *aging indicators*:

- **Launch time of activities**: An *activity* in Android is related to all the interactions that a user can perform [19]. Indeed, this is the fundamental user-perceived performance metric, which we use to assess that the performance is degrading and the system is aging. Specifically,

[1]Lines of code (LOC) computed with the *SLOCCount* tool [7] on the Android Open Source Project (AOSP) – the baseline version of Android v. 5.0 - not including further LOC from external open-source projects (such as the Linux kernel and SQLite) and from vendor customizations.

the launch time is the time to complete the initialization of a launched activity;

- **Aging KPIs**: User-perceived performance in terms of launch time is not always directly measurable because the user might not perform multiple launches and the number of launch time samples is insufficient to reveal aging. In that scenario, aging is detected through several OS key performance indicators (*aging KPIs*), which have been shown empirically to be tightly related to the launch time. We selected aging KPIs, described in Section III, based on our previous empirical analysis of Android aging [8].

We distinguish five different states, relevant from the aging point of view, in which the device can operate. These states are typically used in aging models. They are:

- **Failure state**: The state in which Android actually exhibits a user-perceived failure, typically in the form of a *crash* or *hang* (namely, the device is not responding to any input within a defined timeout);
- **Aging Failure state**: A state where Android is *certainly* affected by aging and from which it will eventually enter a *Failure state*, unless rejuvenation is performed;
- **Aging state**: A state in which Android is *suspected* to be affected by software aging. This is also known as a *failure-probable* state in the literature [20]. In such a state, an *aging alert* is raised when an individual aging indicator is recognized (with a configurable confidence value $C\%$) to exhibit a trend. From such a state, the aging phenomenon will continue (if it is not transient), leading to an *Aging Failure state*;
- **Healthy state**: A state where Android is *unsuspected* to be affected by aging. In a healthy state, monitored indicators do not show significant aging trends. We consider *significant* a trend if it can be assessed with a (configurable) confidence value $C\%$ (by default set to 95%). Not significant trends are likely to be due to transient phenomena;
- **Rejuvenation state**: A state in which a rejuvenation action has been triggered, aimed at leading Android back to a *Healthy state* (from an *Aging state*).

Clearly, it is desirable to trigger rejuvenation only when the OS is *actually* aging, yet while it is still in an *Aging state*, so as to prevent it to enter an *Aging Failure state*. To this aim, an *aging alarm* is raised in an *Aging state* when multiple alerts occur. This process is described in detail in Section IV-C

We define the **Time To Aging Failure (TTAF)** as the predicted time lapse until the device enters an Aging Failure state, an event that ADARTA aims to prevent.

III. Architecture of ADARTA

Figure 1 shows the architecture of the proposed agent. It entails the following modules:

- **Aging KPIs and Load Monitors**: The responsibilities of this module are to collect periodically (with configurable sample period), to pre-process, and to store the *i)* aging KPIs, and *ii)* the information about the current load (measured by workload indicators, WI), such as the number and type of running applications/processes/activities, the percentage of CPU usage, detailed in the following). The output of this module is the time series to be used by the *Aging Detector* module;

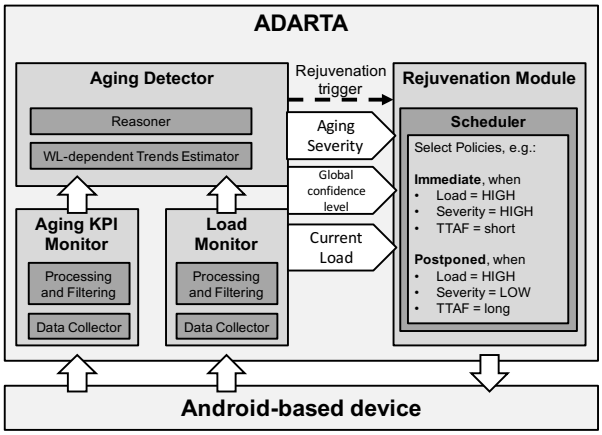

Fig. 1. Architecture of the ADARTA agent

- **Aging Detector**: The responsibilities of this module are: *i)* assessment of trend significance and slope in each time series; *ii)* based on the previous step, detection of the occurrence of an *aging state*, i.e., a state in which there is a progressive degradation of performance with an established (configurable) degree of *confidence* that it will persist. Besides the confidence levels configured for each KPI, the aging detector will provide in output *i)* an estimation of the *severity* of the aging in terms of TTAF; *ii)* an assigned *global confidence level* to the detected aging, based on how many and which KPIs concur to determine it; *iii)* the *load level*, discretized into 5 levels: very high, high, medium, low, very low. These three outputs will be the base for determining the best rejuvenation policies, according to the notification process described in Section IV-C;
- **Rejuvenation Module**: This module uses the information provided by the aging detector (confidence, TTAF, and load level), in order to decide *when* to rejuvenate the system. For example, a policy could be to trigger an immediate rejuvenation if the TTAF is *short* regardless the current load, or if the TTAF is *longer*, the rejuvenation could be postponed as soon as the current load will be low. Rejuvenation is performed by restarting key services found to be the main responsible for aging in our previous study; a finer grain action is under investigation, which cleans the single Java containers within the critical processes; its details are left to future work.

IV. Monitoring and Detection

A. Aging Metrics

User-level aging is measured in terms of responsiveness, i.e., in terms of launch time of activities during experiments (and the same will be done during validation). As anticipated, we may not detect aging directly through launch time since the

user might not request multiple launches of activities. Indeed, the system could be aging, but the number of samples related to the launch time may be insufficient to detect the aging state. Thus, the best aging indicators to be used during monitoring and detection are the aging KPIs, which we have observed to be strongly related to the launch time [8]. These indicators can be measured at any time, thus allowing a prompt reaction in case of aging. We also collect launch times every time an activity is launched from scratch, so as we can use them, when they are available, together with the other indicators. Besides launch times, the following aging KPIs will be recorded and stored periodically within a sampling period denoted as T:

- Proportional Set Size (PSS) of the processes `system_server`, `systemui`, `surfaceflinger`, `mediaserver`. The PSS indicates the portion of main memory occupied by a process;
- Free and Cached memory;
- ZRAMinSWAP, ZRAMPhysicalUsed, LostRAM, KSM-Saved, KSM-Shared, KSM-Unshared, KSM-Volatile, Used PSS, Used Slab, Used Buffers;
- Garbage collector pause time (GC-paused) and total time (GC-total) of the processes `system_server`, `systemui` and `huawei.systemManager`.

We remark that this is a conservative set of metrics, i.e., there are more indicators than strictly needed. The indicators related to `system_server` have a much heavier weight to determine if aging is occurring or not, as they are strongly related to the activities launch time [8].

The aging detection strategy based on these metrics is described in Section IV-C.

B. Characterizing the Healthy State

In order to properly compute the TTAF when aging is detected, and select the best rejuvenation time, the aging detector needs to characterize the *healthy state* first. In this phase, the aging detector "learns" the average values of the indicators (including both *launch times* and KPIs) when the system is in a state where no aging has been detected yet. These averages are used to characterize the healthy state.

Let us refer to a generic indicator KPI_i of the ones listed before (let us postpone the discussion on *launch time* later in this section). Let us indicate with t_0 the *system start-up time* or the *time soon after a rejuvenation* action has been applied. We assume that at this time the system is in a healthy state. The aging detector starts sampling and recording, for each KPI_i, the current value at each sample period T. After a minimum value of S samples gathered, the aging detector computes the average (we opt for the *median* as the average indicator) and the confidence interval of the average $C_{avg} = 95\%$ over the set of S samples. The aging detector checks if the average is accurate within $R = 5\%$ of error (i.e., it verifies that the confidence interval is tighter than the interval $[avg - 5\%avg; avg + 5\%avg]$). If it is not the case, it adds further samples to the set S and keeps on computing the average on S until the condition is satisfied (i.e., until the average is accurate within 5% of error). If the average is accurate, the aging detector stores the value (let us denote it as $\bar{S}_1$), discard the S samples, starts recording new S samples, and repeating the process described above.

This process goes on until: *i)* either a maximum number N of storable average value or a maximum time Z is reached (i.e., the history is too long to be stored entirely, in which case the oldest $\bar{S}_i$ values are discarded), or *ii)* an aging detection alarm is raised (i.e., aging has been detected, see subsection IV-C), i.e., the system is no longer in a healthy state. If aging is detected before the condition on the accuracy of the average is met, the aging detector stores the average of the current set of samples and use that one as a healthy state indicator. The parameters S, R, C_{avg}, N, and Z are configurable, in order to tune the aging detector for the specific platform.

The described process provides, for each KPI, a series of values $\bar{S}_i$, which are the medians of values observed over time in the healthy state. The agent considers the 95*th* percentile of these values as upper bound, denoted as U_{KPI_i}. For each indicator, a threshold of acceptable performance is computed as a percentage of its upper bound value. Specifically, the threshold for the indicator KPI_i is set as $Thr_{KPI_i} = U_{KPI_i} + x\%U_{KPI_i}$, or, equivalently, as $Thr_{KPI_i} = U_{KPI_i} \cdot x$, with x being the slowing down factor. This means that, if the decision were based only on the i-th KPI, the rejuvenation module should rejuvenate as soon as the value of KPI_i is expected to exceed the threshold Thr_{KPI_i}. In that case, the TTAF is the expected time to achieve Thr_{KPI_i}.

We remark that, for the launch time metric, we can not establish if and when a new sample will be available for a given activity. Thus, it is likely that the collected samples (which are averaged and stored, similarly to the $\bar{S}_i$ samples for KPIs), will be insufficient for trend analysis. In this scenario, if an accurate average can be computed, we use it to determine the average launch time in the healthy state. If the average is not enough accurate, or insufficient samples are available to raise an aging alarm, then this indicator is not used until the next rejuvenation (or device restart) occurs. This choice is adopted since there would be no baseline to decide what is the average launch time in the healthy state. Consequently, aging detection could be based only on aging KPIs. In that case, ADARTA still tries to detect aging by sampling every T seconds all the configured KPIs, which are strongly correlated to the aging on the launch time metric as demonstrated in [8].

C. Detecting the Aging State

Given the time series of each metric and their thresholds, the task of the aging detector is to notify when the device enters an aging state. The process is illustrated in Figure 2 and explained hereafter.

1) Monitoring: Starting from t_0, the aging detector monitors, for each aging KPI, the value at each sample period T. After a window $W_1 = \{s_1, s_2, \dots s_w\}$ of W samples (we set $W = 50$), the detector applies the Mann-Kendall (MK) test with a given confidence C% (e.g., $C\% = 95\%$) and the *Sen* procedure to the samples in the current window W_1. The test will give an outcome MK_1 = YES/NO for the MK test, and

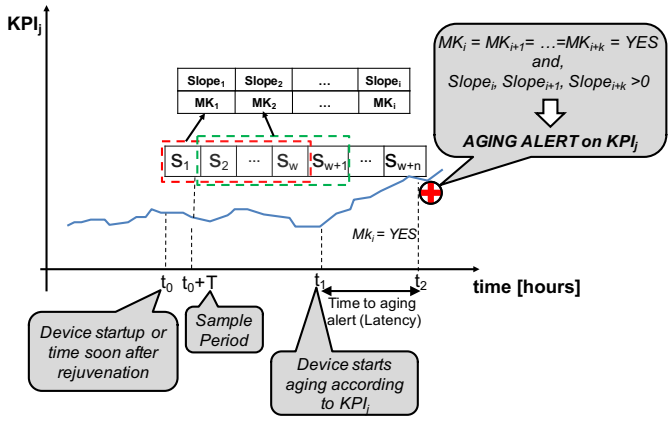

Fig. 2. Aging detection process

a value of the trend slope, $Slope_1$, through the Sen procedure (if MK_1 = NO, then $Slope_1$ is set to 0). At the subsequent sample, after T seconds, the window slides of one position, so as to obtain $W_2 = \{s_2, s_3, \dots s_{w+1}\}$. The MK and Sen procedures are applied again (in other words, the test is applied always to the last W samples). Alternatively, the MK and Sen tests can be applied every $n \cdot T$ samples with $n > 1$ to reduce the overhead, at the price of increasing the detection time.

For the i-th time series for the j-th KPI, there are:

- a series $MK_{i,j}$ of YES/NO outcomes indicating if there was a trend or not at each T;
- a series $Slope_{i,j}$ of values indicating the slope (if any, otherwise it is 0) of the trends computed at each T.

Besides the KPIs, the monitor collects and stores the launch time of any activity. The MK test and Sen procedure are applied each time a new activity is launched. This will make available two additional series (MK_i and $Slope_i$) for launch times, having a different size for KPIs series.

2) Detection: In order to be robust to noise, we adopt a mechanism based on counting. For the *i*-th series of the j-th KPI, $MK_{i,j}$, there could be, at some time (e.g., at time t_1 in Figure 2), an outcome YES (i.e., there is a trend on the last W samples). The user can select the confidence level to claim that there is aging (for instance, $C = 95\%$), and this means that in C% of cases the detected trend has not occurred randomly, but it is a systematic trend. Despite the already high confidence level, we adopt an additional criterion to further increase it. Specifically, we check for the *persistence* of the detected trend over time, which is revealed if the following conditions are jointly met:

- the outcome $MK_{i,j}$ is YES (with C% of confidence) for at least k (e.g., we choose default value equal to k=5) consecutive times[2], and;
- the $Slope_{i,j}$ is always positive in this k times (of course, for indicators where "positive" means more severe aging, like PSS, while "always negative" for the opposite cases, e.g., Free Memory).

[2]The value k is a trade-off between the prompt and correct detection of aging and the likelihood of raising false alarms.

In both conditions hold, we say that there is an **aging alert** that refers to the KPI j (time t_2 in Figure 2).

Given the information about the MK test and the slope of the computed trend by means of Sen's procedure, the TTAF for the i-th KPI, is estimated according to the equation:

$$TTAF_{KPI_i} = \frac{Thr_{KPI_i} - intercept_{KPI_i}}{slope_{KPI_i}} - t_{aging_{KPI_i}} \quad (1)$$

where Thr_{KPI_i} is the threshold of acceptable performance, the $slope_{KPI_i}$ and $intercept_{KPI_i}$ are the output of the Sen's procedure [21], and $t_{aging_{KPI_i}}$ is the aging detection time.

The *aging alerts* on KPIs do not cause any rejuvenation action yet. To trigger rejuvenation, we consider all the KPIs together and define the following criteria. Based on the experimental evidence in [8], we claim that there is an **aging alarm**, which will be notified to the rejuvenation module that will activate the rejuvenation, if:

- **There is an aging alert on the *Launch Time* indicator, and no aging alerts on KPIs**. The TTAF is computed with respect to the threshold of the Launch Time. The shorter the time to reach the threshold, the more severe is the aging. This scenario is very unlikely since whenever there was aging on launch time, we observed aging in some KPIs before (especially in the `system_server` PSS, where it was observed in every case);
- **There is an aging alert on *Launch Time* indicator and on other KPIs**. The OS TTAF is computed as the shortest among the KPIs' TTAF and the launch time TTAF;
- **There is no aging alert on Launch Time, but there is an aging alert only on `system_server` PSS**. "No aging alert in Launch Time" means either that there is no aging at all at the user level, or that there are not enough launch time samples to detect it. In this scenario, the alarm is raised and the TTAF is the one computed on the `system_server` PSS. Note that from results of experiments in [8], we know that whenever there is aging in the `system_server` PSS, there is a very high likelihood that there is aging on launch times: thus, `system_server` PSS is treated as a very important indicator for aging detection.
- **There is no aging alert on launch time, but there is an aging alert on `system_server` PSS and some other KPIs too**. In this case, the alarm is raised and the TTAF is the minimum among the KPIs TTAF;
- **There is no aging alert on Launch Time and on `system_server` PSS, but an aging alert on at least one of specific KPIs combinations**. The considered combinations of KPIs are the following:
  - Free memory, cached memory, Lost Ram;
  - Used PSS, ZRAMinSWAP, ZRAMPhysicalUsed;
  - GC paused and total time of *systemui* and *huawei.systemManager* processes.

  The estimated TTAF is computed as the minimum among the TTAF of all indicators.

TABLE I
SETS OF AGING INDICATORS, AND CORRESPONDING CONFIDENCE LEVEL

Aging indicators	**Global confidence level**
Launch Time	HIGH
`system_server` PSS	HIGH
`system_server` PSS and *Launch Time*	VERY HIGH
`system_server` PSS and GC Paused or Total Time	VERY HIGH
Free Memory, *Cached Memory*, *Lost RAM*	MEDIUM
Used PSS, ZRAMinSWAP, ZRAMPhysicalUsed	MEDIUM
`systemui`, `huawei.systemManager` GC Paused/Total Time	LOW
One of *KSM-*, Used slab, Used buffers*	LOW
One of `systemui`, `surfaceflinger`, `mediaserver` PSS	VERY LOW

Besides the KPI pre-defined confidence levels, we assign a *global confidence level* to the detected aging, based on how many and which KPIs concur to determine the alert. The higher the global confidence level, the more likely an ongoing aging phenomenon is heavily affecting the entire system. ADARTA provides this further mechanism to allow finding the trade-off between the ability to timely detect aging and the number of false alarms. To this aim, ADARTA builds on the findings described in [8]: Table I reports the sets of aging indicators identified experimentally in [8], and the corresponding global confidence level. This list is used by the rejuvenation module, along with the TTAF estimation, to decide if rejuvenation is needed. Note that the combination of indicators in the left column are the *sufficient condition* to mark the confidence with the level in the right column: for instance, an aging alert on *Launch Time* is sufficient by itself to mark the aging with a `HIGH` confidence; if there is also a corresponding aging on `system_server` PSS, then this is a `VERY HIGH` confidence (regardless of the other indicators, which might have aging or not).

D. Workload Monitoring

Understanding the current load allows the rejuvenation module to decide when to perform rejuvenation based on the current aging state. Specifically, ADARTA includes a *load monitor* that collects the following workload indicators (WI) periodically within time T:

- Percentage of foreground processes over all processes;
- Percentage of visible processes over all processes;
- Percentage of service processes over all processes;
- Percentage of background processes over all processes;
- Percentage of CPU usage;
- Average PSS of running applications over the RAM size;
- Percentage of the sum of PSS of running applications over the RAM size.

These indicators are treated separately. Each indicator is discretized into five classes (`VERY LOW`: 0%-20%; `LOW`: 20%-40%; `MEDIUM`: 40%-60%; `HIGH`: 60%-80%; `VERY HIGH`: 80%-100%). The overall workload level is computed as follows: *i)* counting the number of workload indicators (WI) that fall into each level; *ii)* choosing the level with the maximum number of WIs. For instance, if two WIs are in the `LOW` level, four WIs in the `MEDIUM`, and one WI in the `HIGH` level, the overall workload level is said to be `MEDIUM`. Such an indication on the current load is further used by the rejuvenation module as triggering factor for rejuvenation action.

V. CONFIGURABILITY

The proposed ADARTA agent allows users to tune the following parameters:

- **Enable/disable rejuvenation** (default value is disabled);
- **Sample period** T, default is 30 seconds. The longer is this period, the longer the latency between aging occurrence and aging detection, but the less will be the overhead in terms of memory usage, collection time, and energy consumption;
- **Enable/disable the collection of some KPIs**. The following default KPIs cannot be disabled: `system_server`, *Launch time*, *Free memory*. The more KPIs are enabled the more accurate the detection can be, but the more overhead is caused in terms of memory usage, collection time, and energy consumption;
- **The minimum confidence threshold of the Mann-Kendall test** to state that there is a trend in data (it indicates how much likely the trend will persist). It is configurable to four levels: 80%, 90%, 95%, 99% (the last three values are commonly used in statistics). If the result of the test is greater than the set confidence, a trend is said to be present in that time series;
- **Threshold of acceptable performance for each KPIs**. As described in section IV-B, the thresholds will be computed as the percentage of the upper bound value U_{KPI_i} experienced in the healthy state, to be independent of the device where they are applied ($Thr_{KPI_i} = U_{KPI_i} + \text{x}\%U_{KPI_i}$). The x value is configurable.

Further configurable parameters are:

- **Minimum number of samples** S, after which the aging detector computes the average to determine the healthy state after time t_0. For the computation of the average in the healthy state, the **confidence of the average** C_{avg} and **accuracy of the average** R are configurable (default values respectively 95% and 5%). Also the maximum "history", i.e. the number of storeable averages, N, and/or maximum time Z, are configurable (see section IV-B);
- **The window size** W, namely the number of collected samples over which the MK test and the Sen procedure are performed every T seconds. We prefer short windows (e.g., $W \leq 50$) for reacting as fast as possible whenever aging occurs. As stated in Section IV-C, to reduce false alarms, the aging detector raises an alert only when there is an aging trend for at least k consecutive times, with $k = 5$ by default. Note that samples within W are deleted once the MK test and Sen's procedure are over. The outcome of these tests, namely the series $MK_{i,j}$, $Slope_{i,j}$, are discarded unless an aging alarm occurs (in which case the last k results are kept).

The agent parameters need to be tuned to the specific Android device under test. These will determine the trade-offs

between accuracy/performance of the solution and the incurred overhead, as shown in the next section.

VI. Experiment

This Section presents an experiment showing how to configure ADARTA to effectively detect aging in an Android device and to rejuvenate its state.

The mobile device selected for the experiment is a P8 smartphone by Huawei. It is equipped with an *Android Open Source Project* (AOSP) OS, version 6.0.1, and with third party applications chosen by Huawei for the aging analysis conducted in [8], i.e., `com.tencent.mm`, `com.sina.weibo`, `com.qiyi.video`, `com.youku.phone`, `com.taobao.taobao`, `com.tencent.mobileqq`, `com.baidu.searchbox`, `com.baidu.BaiduMap`, `com.UCMobile`, and `com.moji.mjweather`. The Huawei device is stressed via the *Monkey* tool [17], started by using the Android Debug Bridge (`adb`) command-line tool [22].

For this device, assume the manufacturer's software engineer wishes to configure ADARTA to monitor the `total PSS` KPI of the `system_server` process, which is demonstrated in our previous work to be one of the best indicators of aging in Android [8]. The aging detector module is configured with the following parameters:

- Sampling period (T): 30 s;
- Upper KPI ratio value (i.e., the slowing down factor for acceptable performance): 1.3;
- Confidence for computing threshold: 95%;
- Minimum number of samples (the window size W): 50;
- Minimum number of samples on which calculate health state (i.e., S): 10;
- Maximum storable medians (i.e., N): 15.

Figure 3 shows an overview of the experiment result, plotting the total PSS of the `system_server` over time (hours). In the initial phase (marked with ①), the aging detector computes the healthy state of the device, and in particular the "healthy" total PSS (see section IV-B), that is, the memory consumption of the device when there is still no aging effect. We assume that at the starting of the experiment the device has just been rebooted or freshly booted, thus there is no aging. Moreover, the aging detector computes the thresholds for the total PSS among the other KPIs; these thresholds will be adopted for the computation of the TTAF. According to the set slowing down factor, the device is considered aging-failed when the total PSS of the `system_server` process will exceed +30% of the "healthy" total PSS (see ②). After about 2 hours, the device already exhibits a degradation of performance, and the memory consumption of the `system_server` grows anomalously (see ③).

Figure 4 shows the output of the aging detector (as stored in the device at a predefined location) of the last five-time windows of PSS samples from the time when the first aging alert is detected and the alarm is raised (see ④). In particular, the aging detector checks the presence of a trend periodically, every 50 samples of the total PSS, according to the minimum number of samples set. Figure 4 shows that 5 consecutive

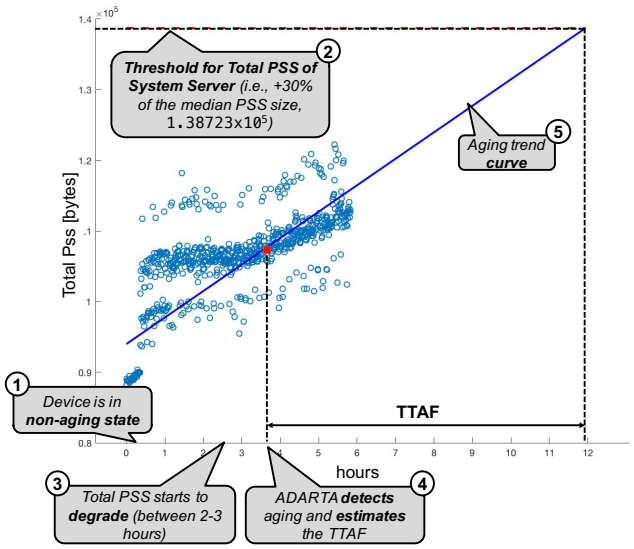

Fig. 3. Overview of experiment result.

aging trends are detected (i.e., the field `Trend: yes` in the `MannKendall` section), and the estimated slope (i.e., the value `TheilSen slope`) of the aging trend is updated over time, in order to follow the behavior of the device. The output in Figure 4 reports also the list (abbreviated) of 50 samples in the time windows indicating both the timestamp and the sampled total PSS value. Furthermore, the output includes also the computation of the Mann-Kendall test and the Sen's slope procedure, the statistical confidence of the trend detection (for example, 98% confidence), and the slope and the intercept of the aging trend (see aging trend curve ⑤). Finally, in the $i+5th$ window the device experiences five consecutive trends and positive slopes, and according to criteria in section IV-C, the aging detector raises an *aging alarm* due to an *aging alert* on the total PSS KPI. At this point, the aging detector computes the estimation of TTAF in milliseconds. We obtain the TTAF equal to approximately 8 hours. This means that starting from the time the aging alarm is raised, in about 8 hours the device will reach a state in which the performance will be degraded to such an extent that leads is to be unusable.

In the conducted experiment, rejuvenation is performed by restarting key services found to be the main responsible for aging in our previous study, and specifically the `WiFi service` and the `Activity manager`. We are currently implementing a finer grain rejuvenation action, which aims at cleaning the single Java containers (e.g., `Heap`, `ArrayLists`, etc.) within the selected critical services of the system server process. The main idea behind our approach is to profile processes and services for understanding how the memory usage of objects evolves. Subsequently, we apply some heuristics for identifying and cleaning-up the Java containers without causing side effects on the application. The used heuristics could indicate only containers whose size grows over time or containers that have at least one object with a long lifetime. Details are left to future work.

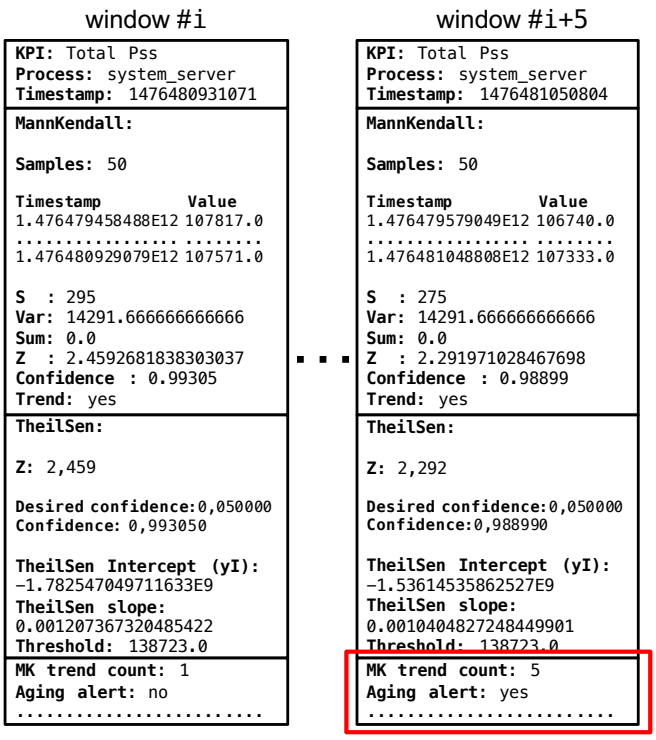

Fig. 4. Aging detector output.

VII. Conclusion

We have presented ADARTA, a software agent for aging detection and rejuvenation in Android platforms. It is designed to automate the whole process from monitoring system- and workload-related aging indicators, to detecting unhealty system states, up to scheduling and performing a proper rejuvenation action.

We believe the configurability feature of the proposed agent will support practitioners in tailoring it for manufacturer's specific needs. We hope this will contribute to fill the gap between the many results made available by the research on mobile software aging and rejuvenation in the very last years, and their incorporation into the engineering practices of future mobile devices based on the popular Android operating system.

Acknowledgments

This work has been partially supported by the Italian PRIN 2015 project "GAUSS" funded by MIUR (Grant n. 2015KWREMX_002).

References

[1] M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi, "Analysis of software aging in a web server," *IEEE Transactions on Reliability*, vol. 55, no. 3, pp. 411–420, Sep. 2006.

[2] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "Software Aging Analysis of the Linux Operating System," in *2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2010, pp. 71–80.

[3] G. Carrozza, D. Cotroneo, R. Natella, A. Pecchia, and S. Russo, "Memory leak analysis of mission-critical middleware," *Journal of Systems and Software*, vol. 83, no. 9, pp. 1556–1567, 2010.

[4] R. Pietrantuono and S. Russo, "A survey on software aging and rejuvenation in the cloud," *Software Quality Journal*, pp. 1–32, 2019.

[5] D. Cotroneo, R. Natella, and R. Pietrantuono, "Predicting aging-related bugs using software complexity metrics," *Performance Evaluation*, vol. 70, no. 3, pp. 163–178, 2013.

[6] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "A Survey of Software Aging and Rejuvenation Studies," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 10, no. 1, p. 8, 2014.

[7] David A. Wheeler, "SLOCCount HomePage," 2019. [Online]. Available: https://dwheeler.com/sloccount/

[8] D. Cotroneo, F. Fucci, A. K. Iannillo, R. Natella, and R. Pietrantuono, "Software Aging Analysis of the Android Mobile OS," in *Proc. 27th IEEE International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2016, pp. 478–489.

[9] C. Weng, J. Xiang, S. Xiong, D. Zhao, and C. Yang, "Analysis of Software Aging in Android," in *2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2016, pp. 78–83.

[10] F. Qin, Z. Zheng, X. Li, Y. Qiao, and K. S. Trivedi, "An empirical investigation of fault triggers in Android operating system," in *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 2017, pp. 135–144.

[11] Y. Qiao, Z. Zheng, and F. Qin, "An empirical study of software aging manifestations in Android," in *2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2016, pp. 84–90.

[12] Y. Qiao, Z. Zheng, Y. Fang, F. Qin, K. S. Trivedi, and K.-Y. Cai, "Two-level rejuvenation for Android smartphones and its optimization," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 633–652, 2018.

[13] J. Xiang, C. Weng, D. Zhao, J. Tian, S. Xiong, L. Li, and A. Andrzejakb, "A new software rejuvenation model for Android," in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2018, pp. 293–299.

[14] S. Huo, D. Zhao, X. Liu, J. Xiang, Y. Zhong, and H. Yu, "Using machine learning for software aging detection in Android system," in *2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)*. IEEE, 2018, pp. 741–746.

[15] Y. Qiao, Z. Zheng, and Y. Fang, "An empirical study on software aging indicators prediction in Android mobile," in *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2018, pp. 271–277.

[16] C. Weng, D. Zhao, L. Lu, J. Xiang, C. Yang, and D. Li, "A rejuvenation strategy in Android," in *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2017, pp. 273–279.

[17] Android Open-Source Project, "UI/Application Exerciser Monkey," 2019. [Online]. Available: https://developer.android.com/studio/test/monkey

[18] J. Araujo, V. Alves, D. Oliveira, P. Dias, B. Silva, and P. Maciel, "An Investigative Approach to Software Aging in Android Applications," in *2013 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2013, pp. 1229–1234.

[19] Android Open-Source Project, "Android Documentation," 2017. [Online]. Available: https://developer.android.com/reference/android/app/Activity

[20] M. Grottke, R. Matias, and K. S. Trivedi, "The fundamentals of software aging," in *2008 IEEE International Conference on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2008.

[21] R. O. Gilbert, *Statistical methods for environmental pollution monitoring*. John Wiley & Sons, 1987.

[22] Android Open-Source Project, "Android Debug Bridge," 2019. [Online]. Available: https://developer.android.com/studio/command-line/adb