Statistics and Runtime Verification

Andrew Gauci
Dept. Of Computer Science
University Of Malta
agau0006@um.edu.mt

Gordon J. Pace
Dept. of Computer Science
University of Malta
gordon.pace@um.edu.mt

Christian Colombo
Dept. of Computer Science
University of Malta
christian.colombo@um.edu.mt

Abstract

The importance of correctness of systems is becoming more crucial as computers control more of our everyday activities. Various approaches have been advocated and used for the verification of such correctness, with one of the more promising ones being runtime verification. One important issue in runtime verification is the logic used to specify properties, since this influences both the overheads induced by the monitors, and the applicability of the approach to a particular domain. In this paper we propose techniques for the expression and runtime monitoring of statistical properties, enabling easier manipulation and expression of non-functional requirements. The logic is developed as an extension of the existing runtime verification tool LARVA*, and has been applied to an ftp server implementation, adding a new layer of probabilistic intrusion detection and system profiling.*

Index Terms

runtime verification, computational statistics, security, intrusion detection, anomaly-based detection

1. Introduction

Ever since the inception of software development, there has been the need for guarantees of a system's correctness. Moreover, as computer systems influence our lives in more frequent and crucial ways, the need for reliable software rapidly grows. Such guarantees can be provided through software verification, by ensuring that the system implementation corresponds to its specification, with the ultimate goal being a guarantee of the absence of bugs. Traditional techniques used include testing and model checking. However, given the intractability of verifying each possible system trace, and the lack of coverage implied by testing, an increasingly used approach is that of runtime verification, whose concern is the dynamic verification of the currently executing system trace. Runtime verification can be considered as an extension to testing with augmented guarantees through the use of more powerful specification languages.

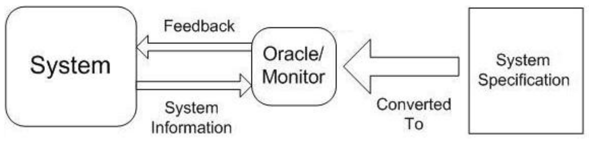
One important issue in runtime verification is that of the logic used to specify properties. This influences the computational complexity of the monitors, and hence the overheads induced. It also influences how effective the approach can be used in particular domains. For instance, using a logic enabling reasoning about discrete time can be effective in a synchronous system setting, whereas it could be challenging to express properties of systems with real-time constraints. In this paper, we explore the use of statistical properties within a runtime verification setting. The advantages and applications of such an approach are various — e.g. it is often better to watch for indicators of impending failure than to wait, and statistics allow for the straightforward specification of non-functional requirements, including specifications related to performance and reliability. Areas of application of a statistical runtime monitoring framework include performance profiling, intrusion detection, user modeling and quality of service, all of which benefit from the availability of such functionality.

We also investigate the use of the approach by implementing the logic through the runtime verification framework LARVA, an event-based runtime verification tool and logic for monitoring temporal and contextual properties. In this paper we give an overview of the resulting system LarvaStat, an extension to LARVA with statistical capabilities. These capabilities are introduced (to LARVA) as an additional statistical framework running on top of the current logic, and whose goals concisely are (i) the collection of a wide variety of statistics over runtime executions, in a manner which is intuitive and straightforward, and (ii) the fruitful integration of the runtime verification and statistical frameworks into one augmented framework, whose overall expressivity, applicability and suitability is greater than the sum of its constituent parts.

Section 2 introduces runtime verification, the tool LARVA, and also the issue of incrementally computable statistics. Section 3 presents LarvaStat, an event-based runtime verification tool extending LARVA with statistical capabilities. Section 4 presents a comprehensive case study implemented using LarvaStat, which defines a probabilistic intrusion detection system and integrated system profiler. Section 5 discusses alternate solutions for the collection of statistics over runtime executions, and also compares LarvaStat with current solutions. Section 6 concludes the paper and offers pointers to possible future work.

2. Background

Runtime verification [2], [5] is a dynamic software verification technique which verifies that the system trace generated at runtime by the currently executing system adheres to the system specification, and is summarised below.



The above process admits multiple issues, the first of which is the choice of language for the expression of the system properties. Such languages are either logic based (such as duration calculus [3]) or automaton based (such as timed automata [1]). In general, no language is best, and the choice of language often depends on multiple factors such as the application domain and the required expressivity demanded by the system properties.

With the system properties expressed through a suitable logic, the next step is the actual verification that the system adheres to the specified properties. This verification is executed at runtime through an oracle/-monitor, encapsulating the specified system properties, which is running in parallel with the underlying system and observing its behaviour. A resulting crucial issue is the fact that this monitor consumes resources (both computational as well as memory) otherwise available to the system. This system overhead induced by the monitor may result in the alteration of the system behaviour itself, which implies that any monitor implementation should consume as little resources as possible. However, note that in general a tradeoff between logic expressivity and system overhead exists.

Another issue is that of instrumentation, which refers to the integration of the monitor with the underlying system, thus giving the monitor access to the required system information for verification to occur. Finally, an interesting issue worth mentioning is reparation. Whereas certain languages are content with simply identifying correct (or incorrect) system behaviour, others go one step further by specifying reparatory action in order to steer the system back to an acceptable state.

LARVA [6], standing for Logical Automata for Runtime Verification and Analysis, is an automaton-based logic and tool for runtime verification. Based on Dynamic Automata with Timers and Events (DATEs) as the underlying mathematical framework, LARVA listens to events within the underlying system and allows for the expression of (a) temporal properties dealing with (i) consequentiality (Event A must occur before B), (ii) real time properties (Event A must occur at most twice every five minutes), and (b) contextual properties with the possibility of monitoring objects either globally or grouped according to their context (by monitoring each account instance, ensure that every account must belong to a registered user). The defined core LARVA constructs include the property (specified through automaton-based DATEs), the timer (for real-time requirements), channel (used for automaton communication), the event (specifying any event of interest within the underlying system) and finally the *foreach* construct used for context specification.

The evaluation of statistics over runtime executions poses a dilemma, since whereas the computation of certain statistics are computationally expensive and potentially operate on a large data set, we require any monitor which executes such valuations to be as lightweight as possible. However, it turns out that a certain class of useful statistics can be characterised using an efficiently executable evaluation strategy, also known as incrementally computable statistics [7] defined below.

Definition 2.1: Given a sequence $\langle s_1, s_2, ..., s_n \rangle$, initial statistical valuation v_0, incrementally computable statistic α, characterised by function f(a,b), where a is a new valuation added to the statistic's input value set, and b is the current statistic valuation, the final valuation v resulting by the application of α to $\langle s_1, s_2, ..., s_n \rangle$ is

$$v = f(s_n, f(s_{n-1}, ..., f(s_1, v_0)))$$

In other words, an incrementally computable statistic evaluates its new valuation using only the current valuation and the new value added to the data set. Example statistics which can be evaluated in an online incrementally computable fashion are the average, sum as well as the count. Taking the example of the sum, its initial valuation and characterising function are defined as $v_0 = 0$, and $f(a,b) = a + b$. It is hence clear that incrementally computable statistics are efficiently computable, since all they require is the storage of the current valuation and the execution of the characterising function. However, not all statistics admit an incrementally computable characterisation, such as the median, histograms and modes [9]. In such cases, their evaluation needs to be handled with care using more intelligent ways, such as through *randomised* and *approximation* algorithms [9].

3. Statistics for Runtime Verification

The following section presents an event-based statistical framework augmented on top of the LARVA runtime verification framework. This statistical framework allows for the expression of the required statistics through three core constructs; the *point statistic*, *interval statistic*, and the *statistical event*. Moreover, there exists a tight binding between frameworks, whereby properties defined within the runtime verification framework have access to statistical valuations and can listen to events generated within the statistical framework, and statistical constructs are allowed access to the constructs (such as timers, channels and events) defined within the runtime verification framework.

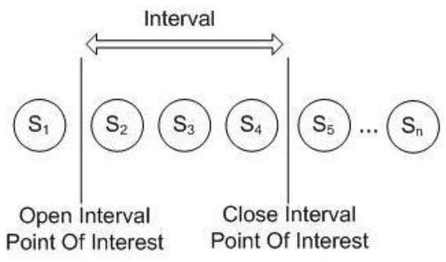
3.1. The Point Statistic

The primary basic statistical construct in our framework is that of the *point statistic*, which keeps track of its current valuation, listens to events and updates its valuation accordingly. Take the example of a statistic keeping count of bad login attempts. Certainly we require a running valuation storing the count of attempted bad logins up to that instant, which is incremented each time another unsuccessful login occurs. This logic can be extended to a variety of statistics (such as the maximum, minimum and the average), and hence a point statistic is defined through (i) the current statistic valuation (and its corresponding type), (ii) a point of interest, entailing the triggering event expression as well as a boolean condition (allowing for the differentiation between events), and (iii) a statistic update executed each time the associated point of interest is triggered, updating the statistic valuation as required.

3.2. The Interval Statistic

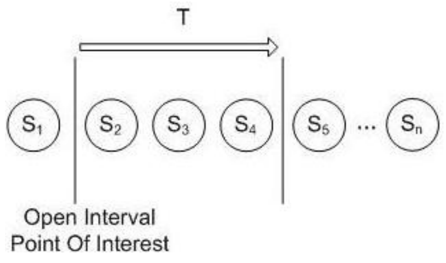
Sometimes point statistics is only to be executed on a subsequence of the whole event trace, which gives rise to the notion of an *interval statistic*. If for example we require the byte count sent during a given stream connection, the only event sequence of interest lies between the opening and the closing of the connection. Consequently, an interval statistic augments the point statistic with the notion of an interval, implying that an interval statistic is characterised through (i) the current statistical valuation, (ii) a point of interest, (iii) a statistic update, and (iv) an interval. Intervals form an expressive backbone of the statistical framework, and admit either a static or dynamic nature.

The opening and closing of static intervals are characterised by fixed points within the event trace, whereby we present two variations.


Interval
S_1
S_2
S_3
S_4
S_5
...
S_n
Open Interval Point Of Interest
Close Interval Point Of Interest

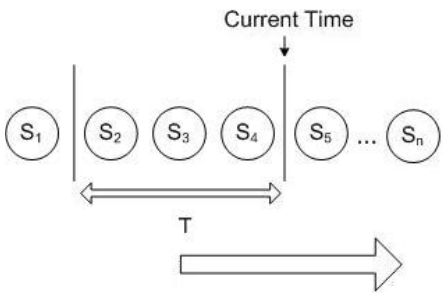
The event interval (above) is the first static interval variation, whereby the opening and closing of the interval are characterised by an interval point of interest, which is essentially a point of interest and an additional action execution (used for initialisation or finalisation purposes). Given such an interval characterisation, an event interval statistic executes the internally defined point statistic between the interval opening and closing. Continuing from the previous example evaluating the byte count, the opening and closing interval points of interest characterise the start and closing of the stream connection, and the internal point statistic counts the bytes sent within this duration.

The second static interval variation is the duration interval (below), and is in fact a specialisation of the previous interval. In essence, whereas the opening interval characterisation remains identical (to before), the interval closes a duration T time units after the interval opening, as described below.



An example duration interval statistic is the download count for the first thirty minutes since user login.

Dynamic intervals are somewhat different than their static equivalent, whereby dynamic intervals essentially "move" with time without any fixed interval opening or closing. Practical applications for dynamic intervals include the definition of statistics such as the largest file downloaded within the last hour, or the network throughput for the last minute. The time interval is the single dynamic interval variant defined within the statistical framework, and is defined below.



Hence, a time interval is defined through (i) the time window duration T, and (ii) an interval updating action, executed periodically in order to keep the statistic valuation in line with the dynamic interval semantics. In other words, whereas the execution of static intervals solely require the listening of events, dynamic intervals require an updating mechanism distinct from any system execution. This is provided through polling, whereby a lightweight updating action is executed every δ (much smaller than T) time units so as to update the interval in accordance to its dynamic semantics, thus simulating a "moving" interval. However, this implies that simulating dynamic intervals can be computationally expensive, as well as the fact that statistics using dynamic intervals admit an outdated valuation by at most δ time units. Alternate methods for simulating dynamic intervals more efficiently are currently being explored.

3.3. The Statistical Event

Since we build our approach on a model working with events, we package all statistical results in a special form of event — the *statistical event* — an event carrying the latest statistical valuation, which is generated each time a statistic valuation is updated. This enables statistical events to be listened to by properties and other statistics alike, and allows for considerable flexibility within the framework.

The first scenario allows for statistic valuations — possibly representing non-functional requirements — to affect system behaviour (for example, if the bad login count valuation goes above the set threshold of three, an attempted intrusion is detected and the user is blacklisted), whereas the second scenario allows for the expression of multilayered statistics. Multilayered statistics essentially allow for the combination of multiple statistics (a sequential chaining of statistics), and allows for the expression of statistics such as the maximum average, or the least variance.

A crucial issue worth mentioning is that of overlapping intervals, whereby certain statistics are interested in select intervals of the event trace, which however overlap in time. Such situations are especially common in multithreaded systems, and an example statistic admitting such an issue is the counting of bytes sent over multiple concurrent stream connections. This implies the requirement of a mechanism which distinguishes between intervals, in such a manner whereby each event of interest (to the statistic) occurring within the underlying system is associated with a unique (from the possible overlapping) interval. The solution to the issue of overlapping intervals is presented through the association of context with each possible overlapping interval. In other words, analogously to the use of context within LARVA, each interval statistic admitting an overlapping interval is associated with an object acting as its context, such that this object is used as a determiner regarding the interval instance affected by an event occurrence.

Although the evaluation of non-incrementally computable statistics is potentially inefficient, we require our statistical framework to be general, capable of evaluating a wide variety of statistics. Hence, both the point and interval statistic are augmented with the possibility of declaring additional variables as required for the evaluation of non-incrementally computable statistics. However, it is the user's responsibility to ensure that any additional constructs do not imply an exponential requirement of computational and memory resources by the statistic.

The statistical framework also defines additional functionality permitting control over the act of statistics collection itself. Hence, statistics can be paused, resumed, updated, manipulated and also reset.

4. Case Study

We have investigated the use of LarvaStat by implementing a probabilistic intrusion detection system and integrated system profiler applied over a third party Java ftpd server implementation [1]. This resulting monitor is to implement two components, these being (i) the observation and quantification of system performance, and (ii) the observation of user behaviour, attempting to characterise incorrect as well as abnormal user behaviour. Moreover, given that the monitoring of users is expensive, we develop a mechanism which probabilistically chooses which user sessions to monitor. This choice depends on two factors, these being the system load as well as the user risk factor (discussed below). Hence, given a high risk user, and/or the ftpd server being under a small load, the user in question will probably be monitored.

- **System Profiler:** The system performance profiling component follows the assumption that the ftpd server's performance is tightly bound to the bandwidth usage, as well as the number of logged in users.
- **Intrusion Detection System:** The second component implements an intrusion detection system, and is based on the statistical anomaly detection techniques presented in [8]. Hence, the intrusion detection system comprises two further components. The first component involves a markov chain analysing the event sequence executed by the user. Each action is associated with a risk factor, and hence if the chain of events is deemed too risky or plain illegal, the user is deemed to be malicious. The markov chain is simulated using a point statistic keeping count of the event sequence risk by multiplying the current risk factor with that associated with the previously executed action. The second component involves the use of statistical moments, more specifically the user average behaviour as well as the variance, for the characterisation of abnormal user behaviour. In essence, if the user adheres to a statistically predictable acceptable behaviour, only to suddenly start placing considerable stress on the system is an indication of intrusion.

Full details of the case study can be found in [10]. The full implementation of the specified logic entails the specification of twenty statistics, all of which are incrementally computable in nature. Through the case study prototype, it is encouraging that a comprehensive non-trivial implementation has not only been expressed through LarvaStat, but also implemented without altering a single line of the original ftpd server code. This offers potential with respect to pluggability, whereby whole security policies can effectively be plugged in or out without altering the original system, leading to better separation of concerns and an increase in modularity.

1. Available under the GNU General Public Licence at http://www.anomic.de/AnomicFTPServer/index.html

5. Related Work

Although runtime verification has received considerable attention in recent years, it is perhaps surprising that there is a lack of research into the augmentation of runtime verification with statistical capabilities, especially given the theoretical and practical applications. We identify three existing approaches to the field, with the first approach presented in [9] solely focused on the collection of statistics over runtime executions. Through the extension of linear temporal logic [4], this approach presents a framework which is said to evaluate queries on individual trace positions (experiments), and combine multiple evaluations using aggregate statistics.

Lola [7] is another approach, and presents a functional stream computation language allowing for the expression of past and future specifications, numerical queries as well as guaranteeing bounded memory requirements.

EAGLE [2] is a third approach offering a language independent tool and rule based logic. Although parsimonious in its construct definitions, the defined logic is sufficiently expressive to encode multiple formalisms such as interval logics, finite state automata, extended regular expressions and even logics for the expression of statistical properties.

It is immediately apparent that our approach is somewhat different than current approaches. Firstly, (to our knowledge) our approach is the first attempt at collecting statistics over runtime executions using an automata-based formalism (all three alternate approaches are logic-based). Also, LarvaStat integrates real-time with statistics collection, thus allowing for the specification of significant class of statistical queries otherwise not expressible (such as the *count* of user downloads *within the last hour*). Moreover, although [9] introduces the application of statistics collection over system trace intervals, we extend this notion through the introduction of dynamic intervals. Finally, and perhaps most crucially, whereas alternate approaches are perhaps rather non-trivial in nature, we believe our approach to be the most intuitive approach yet, whereby collecting potentially complex statistics boils down to a few lines of code. On the other hand, Lola [7] offers functionality not present in LarvaStat, namely the guarantee of bounded memory requirements (given that Lola cannot express non-incrementally computable statistics), as well as its encoding of future and past time logics.

6. Conclusions and Future Work

We have presented LarvaStat, an extension to the LARVA runtime verification tool and logic with statistical capabilities. The presented logic is concise and intuitive, yet sufficiently expressive to express a wide variety of statistics. In fact, the presented framework is also to our knowledge one of the first approaches augmenting statistics collection with real time functionality, while also presenting innovative concepts such as dynamic intervals. Moreover, the presented case study is particularly encouraging, since not only did it exemplify the need for the augmentation of runtime verification with statistical capabilities, but also justified LarvaStat by showing that the presented framework is applicable to a wide variety of non-trivial scenarios. For more information see [10], where the presented framework is formally analysed and an operational semantics specified, as well as specifying a language and associated compiler for the expression and implementation of the motivated framework.

Three core issues are identified for further study. The first issue regards the development of a probabilistic framework on top of the statistical framework. This framework should allow for the expression of properties such as “property X should be adhered to 80% of the time” or “check property X for 80% of the time”. The second issue regards the further in depth characterisation of intervals, placing emphasis on dynamic intervals (which currently can be computationally costly and are executed in inexact fashion due to polling), as well as the application of intervals beyond statistical requirements within runtime verification. The final issue worth further analysis is the system overhead induced by the presented statistical framework, with plans for further framework optimisation.

References

[1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[2] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Rule-based runtime verification. pages 44–57. Springer, 2004.

[3] Zhou Chaochen, C.A.R Hoare, and Anders P.Ravn. A calculus of durations. *Oxford University Computing Laboratory, Programming Research Group*, 1991.

[4] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. The MIT Press, The MIT Press Massachusetts Insititute Of Technology Cambridge, Massachusetts 02142, 1999.

[5] Séverine Colin and Leonardo Mariani. Run-time verification. In *Model-Based Testing of Reactive Systems*, pages 525–555. Springer, 2004.

[6] Christian Colombo. Practical runtime monitoring with impact guarantees of java programs with real-time constraints. Master’s thesis, University of Malta, 2008.

[7] Ben D’Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B. Sipma, Sandeep Mehrotra, and Zohar Manna. Lola: Runtime monitoring of synchronous systems. In *12th International Symposium on Temporal Representation and Reasoning (TIME’05)*, pages 166–174. IEEE Computer Society Press, June 2005.

[8] Dorothy E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13:222–232, 1987.

[9] Bernd Finkbeiner, Sriram Sankaranarayanan, and Henny B. Sipma. Collecting statistics over runtime executions. In *In Proceedings of Runtime Verification (RV’02) [1]*, pages 36–55. Elsevier, 2002.

[10] Andrew Gauci. Statistics and runtime verification. University Of Malta, 2009.