Automated Prevention of Failure in Complex and Large Systems: Fighting Fire with Fire

Behzad Bordbar and Philip Weber

School of Computer Science, University of Birmingham, UK
{b.bordbar, p.weber}@cs.bham.ac.uk

Abstract - People, businesses and economies are increasingly dependent on Cloud and internet services. At the same time, the systems on which these services are built are becoming more complex and interdependent. The cost of failure is high, but systems are too complex for human detection of problems. We review methods for online fault diagnosis, process mining and Virtual Machine Introspection. We suggest bringing these techniques together for automated identification, diagnosis and prediction of risk of failure in large systems. We present examples from telecoms and Cloud industries in support of these ideas.

Keywords: Models, process mining, diagnosis, failure prevention, complex systems

1 INTRODUCTION

The Internet and the services it supports now play a key role in most people's daily lives, and in the day-to-day operation of business enterprises and nations. The Oxford Internet Surveys 2011 [1] surveyed over 2000 respondents in Britain and reported that over 80% of employees used the internet to obtain news and information. Usage for other purposes and by people in other 'life categories' was only slightly lower. Students were the largest consumers of media via the internet (over 90%), closely followed by other groups.

More importantly, the survey reported increasing levels of use of the internet for accessing critical services such as banking, grocery shopping and paying bills. Increasingly, citizens accessed government services online, with only 21% of households lacking internet access. Use of online services was highest among the young, wealthy and well-educated.

New technologies such as Cloud and virtualisation are enabling the move to internet-based systems architecture. Cloud enables computing resources to be provided on demand according to a utility model, using many instances of commodity hardware and software components shared between services. Costs of set up and ongoing provision of new services are thus reduced, since payment is only for the resources or time used. Businesses need no longer invest in costly infrastructure [2], [3]. Virtualisation abstracts services from the hardware, operating systems, storage and network. This makes the resources more flexible, increasing business value by increasing agility and resilience. Services are democratised by use of open Service-oriented Architectures (SoA) and standards such as SOAP and HTTP. The success of public clouds has seen private versions of the same concepts implemented within businesses.

However, new technologies and pace of change present new risks from system problems, and new opportunities for nefarious activities such as malware and cyber-attack. Heterogeneous systems (cloud and open standards) duplicated manyfold may all be affected by the same bug, security breach or performance problem [3]. Shared resources mean one problem may affect many services, and introduces the risk that the activities of one business may impact those of another. A business abstracting its infrastructure to a Cloud platform faces new questions of availability and performance unpredictability. Well documented outages to cloud services (see for example references from [2]) have taken many hours to resolve, each outage affecting many services.

Security can also be a concern. As far back as 2004, Byres *et al.* reported a steady rise in reported incidents of industrial problems caused by 'cyber attacks' [4]. They attributed this to increased use of heterogeneous interconnected systems, and the increasing attractiveness of targets due to the wide consequences possible. These factors are multiplied in today's online environments. Water [4] and power transmission industries [5] are given as examples of interconnected, critical, vulnerable industries which have been the subjects of attacks.

The security viewpoint provides extreme examples of the seriousness of potential damage caused by problems or attacks on highly interconnected systems, the difficulty of containing such problems and their potential to spread beyond the 'cyber' world to physical effects. Examples are the the Stuxnet attacks on Iranian nuclear facilities (e.g. [6]), and industrial 'cyber-crime' using 'botnets' (networks of many hijacked connected computers), reported to be responsible for disruption to Estonia's national networks in 2007, and in the 2008 Russia-Georgia war among others (also [6]).

We conclude that reliability of online services is of crucial importance. Problems may affect very many people simultaneously, prevent access to critical services, and have the greatest impact on the most economically and politically active groups. Service outages thus have the potential to impact economies, enterprises and national governments, both financially and through damaged reputations. 'The major problem for cloud computing is how to minimise such kinds of outage/failure to provide reliable services'[2]. A major challenge is how can we deliver such crucial services reliably while reducing cost?

In this paper we consider the internet- and cloud-based technologies underlying these services and describe three of the techniques used to tackle the above challenges. First we introduce model-based methods for automated detection of faults or undesirable scenarios using automated 'Diagnosers' (software modules or services) to diagnose occurrence of failure or undesirable scenarios in real time or near real time. Model-

based techniques are powerful, but sometimes there are no models of the system available or it is very costly or even impossible to produce a model of the system. For example, systems produced from merging of legacy systems are often too complex and large to be modelled. Often there is no access to the designers of such systems and it is costly to re-engineer a model. However, most modern systems produce logs capturing run-time information for various purposes. The second group of techniques discussed in this paper applies Process Mining to extend this diagnosis framework to situations where we do not have models of the system. Finally, we address the specific challenges of diagnosis of faults related to occurrences of malicious behaviour in Cloud. There are similarities between malicious behaviour as malware writers tend to use components available on the web which are used in existing malware. We present a framework which uses symptoms caused by using such components to discover newly emerging malware. We present examples from telecoms services and security in the Cloud to describe the three sets of methods.

The paper is organised as follows. In section 2 we introduce terminology and concepts necessary for the remainder of the paper, particularly to specify what are system failures or undesirable behaviours. Section 3 describes the problem in more detail. The core of the paper is sections 4, 5 and 6 in which we discuss in depth our methods for diagnosis. Section 7 concludes the paper.

2 PRELIMINARIES

We describe terminology and concepts necessary to understanding the rest of the paper.

2.1 Service-Oriented Architectures (SoA)

A SoA is a distributed business application architecture where heterogeneous components communicate and provide services to each other using open standards. Examples of open standards are WSDL [7] and XSD [8], which are used to define the interfaces between services, and communication protocols such as SOAP and HTTP. The use of such open standards means that the services and protocols can be changed with minimal impact on the service. A simplified SoA, for broadband failure resolution in a telecoms business, is illustrated in Fig. 1 (described in full in section 4).

Most essential is the business process describing how these services interact to complete the task such as resolving a fault.

2.2 Business Processes and Process Mining

Business processes describe activities carried out to fulfil a business function, and the relations between them. Among other aspects we may describe the process 'control-flow', i.e. how the activities are related; interactions between people and organisations; or business rules or constraints. In this paper we are concerned with the control-flow. Business process are commonly represented formally by languages such as BPEL, BPMN and Petri nets. Figure 2 shows a BPMN model of part of the business process for broadband fault resolution.

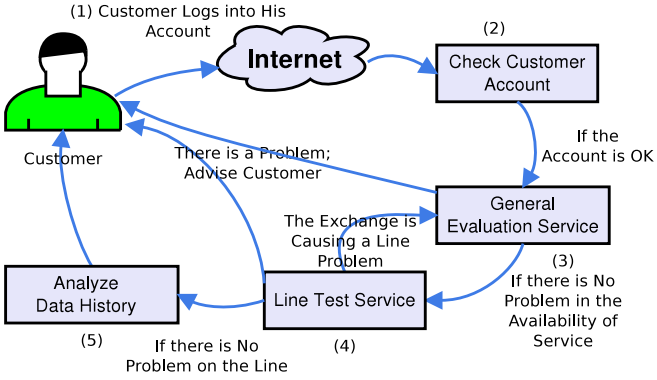

Figure 1: An interaction between the Customer and System

Let A be a set of business activities. A single pass through the business process from start to end task is a *case*, for example processing one order, The *events* of their occurrence are recorded in an *Event log* E. We assume that as a minimum, each event e is recorded with a case ID c, activity name $a \in A$ and timestamp t. An event log can then be represented by a non-empty set of triples

$$E = \{e : e = (c, a, t)\}^+, \tag{1}$$

assuming that timestamps are unique. Process mining algorithms [9], [10] use workflow logs to learn models of the business processes. We discuss process mining in section 5.

Process mining algorithms often assume that events are atomic (taking no time), are uniquely labelled (the same label always refers to the same event and vice versa), and make no use of additional information such as timing of events, merely the order in which they are recorded. We assume that the underlying process to be discovered is unchanging.

If the activities in our example service were encoded with symbols $a, b, \ldots$ from some alphabet Σ then (abstracting from detail) the 'trace' of one possible enactment of the process might be recorded in the event log as a string, e.g.'$abcdef$'. These strings are also called *traces*. A workflow log W is a multiset over traces,

$$W = \{x : x \in \Sigma^+\}^+, \tag{2}$$

e.g. $W = \{abcdef, abcdef, abcdeg, \ldots\}$.

2.3 Discrete Event System

A Discrete Event System (DES) is a 'discrete-state, event-driven system whose state depends on the occurrence of asynchronous discrete events over time' [11]. DES uses models to curb complexity. We can define a general model of a DES as a tuple $G = (X, \Sigma, \delta, x_0, A, L)$, where

- X is a set of states,
- Σ is a set of events,
- $\delta \subseteq X \times \Sigma \times X$ is a set of transitions between states,
- $x_0 \subseteq X$ is a set of initial states.
- A is an alphabet of event labels, with labelling function $L : \Sigma \to A$.

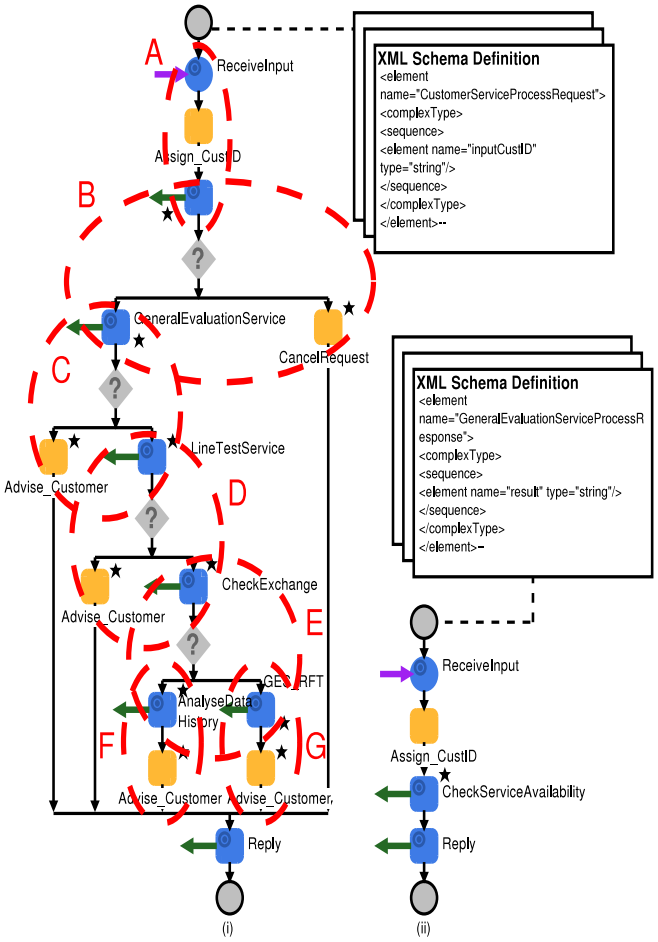

Figure 2: Customer Service BPEL, with some process structures highlighted (sequences A, F, G, XOR splits B, C, D, E).

Such a model can be realised using various modelling languages including automata, Petri nets, Workflow graphs models [12], [13] or ad hoc graphical representations. We do not give details here.

2.3.1 Observable and Un-Observable Events

Events $\tau \in \Sigma$ are partially observable (an event is either observable or unobservable), i.e. $\Sigma = \Sigma_O \cup \Sigma_{UO}$ where $\Sigma_O \cap \Sigma_{UO} = \emptyset$.

Some unobservable events indicate failure, i.e.

$$\Sigma_f \subseteq \Sigma_{UO}$$

We do not concern ourselves with observable failure events, which can be handled trivially, without need for diagnosers. There may be different types of failure, i.e. Σ_f is partitioned

$$\Sigma_f = \Sigma_{f_1} \cup \Sigma_{f_2}, \ldots, \Sigma_{f_n},$$

such that $\Sigma_{f_i} \cap \Sigma_{f_j} = \emptyset, 1 \leq i < j \leq n$.

Let string $\sigma = \tau_1\tau_2\tau_3'\tau_4\tau_5\tau_6', \ldots$ represent a sequence of events generated by G. Of these events only a subset are observable, e.g. τ_3', τ_6'.

Definition 1 (Projection to Observable Events)**.** We define a mapping P to project sequences to just the events which are observed,

$$P : \Sigma \rightarrow \Sigma_O \cup \{\epsilon\} \text{ such that} \tag{3}$$

$$P(\alpha) = \begin{cases} \epsilon & \text{if } \alpha \notin \Sigma_0, \text{ i.e. } \alpha \text{ is not observable,} \\ \alpha & \text{otherwise, where} \end{cases}$$

ϵ is the identity of the alphabet, i.e. $\alpha\epsilon = \epsilon\alpha = \alpha, \alpha \in \Sigma_O$.

Definition 2 (Extend P to Sequences of Events)**.** Let

$$P : \Sigma^* \rightarrow (\Sigma_0 \cup \{\epsilon\})^*, \text{ where}$$
$$P(\alpha_0\alpha_1 \ldots \alpha_n) = P(\alpha_0)P(\alpha_1) \ldots P(\alpha_n). \tag{4}$$

For example, $P(\tau_1\tau_2\tau_3'\tau_4\tau_5\tau_6') = \tau_3'\tau_6'$.

2.4 Cloud and Introspection of Virtual Machines

Benefits of moving to Cloud are well publicized; adopting could result in lower cost of IT due to the economics of scale, reduce the up-front cost for infrastructure, decrease the time to market by using off-the-shelf components, and boost the 'Green' credentials of the company [14]. However, in order for the Cloud environment to be profitable, there is temptation to homogenize the applications and operating systems used. But as the Cloud becomes more homogeneous, it will provide bigger and richer targets for attackers; places where the attacker may be confident of finding lucrative information or where disruption will have the greatest impact. As a result, ensuring security of the cloud is seen as a major engineering challenge [15]. In the next two subsections we shall give a brief description of two of the technologies used in Cloud.

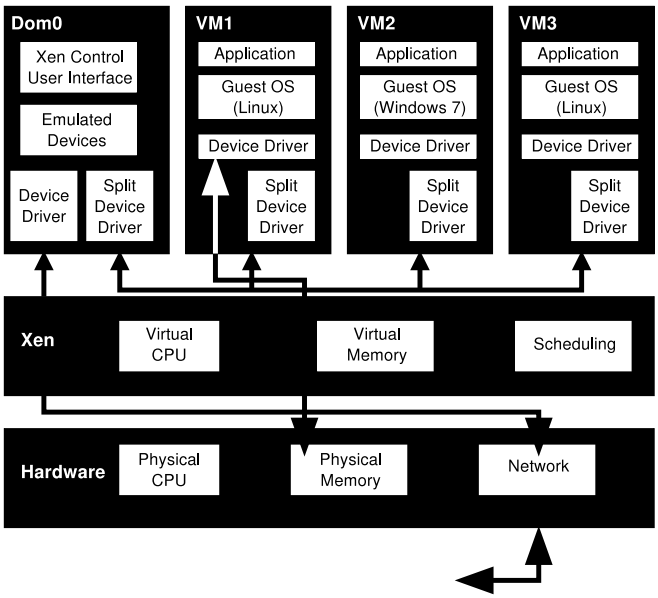

Figure 3: Virtual Machine Architecture

2.5 Virtualisation

Virtualisation is 'A framework or methodology of dividing the resources of a computer hardware into multiple execution environments. . .' [15]. Virtualisation relies on Virtual Machines (VMs), software that emulates or simulates the capabilities of the hardware. It is capable of running a complete operating system along with any applications that run on top of that OS [16]. Fig. 3 depicts a high level view of Xen [15], which is an open source virtualisation software based on 'paravirtualization' technology. In this architecture, the Virtual Machine Monitor (VMM) is an abstraction of the underlying physical hardware and provides hardware access for the different virtual machines. Xen includes a special VM called Domain 0 (Dom0). Only Domain 0 can access the control interface of the VMM, through which other VMs can be created, destroyed, and managed. This powerful VM is used to create other Virtual Machines that can access the hardware through secure interfaces provided by Xen. In addition it is possible to create other virtual machines that can access the physical resources provided by Domain 0'a s control and management interface in Xen. Virtual Machines are heavily used within the Cloud. In addition to the advantage of running multiple operating systems simultaneously, Virtualisation reduces the cost of infrastructure implementation and the associated cost of maintenance by optimising the utilisation of resources. A user can ask for new VMs when extra resources are required and decommission some of the VMs, when they are no longer required. Virtualisation also makes it possible to secure the VMs by a powerful technique, which is commonly known as Virtual Machine Introspection.

2.6 Virtual Machine Introspection

Virtual Machine Introspection (VMI) can be defined as a virtualisation based technique that enables one guest VM to monitor, analyse and modify the state of another guest VM by observing its virtual memory pages. Such introspection can be carried out by a VMM that hosts the VM or another VM which has been granted special privileges by the VMM. VMI will allow product developers and researchers to move the security related software out of a probable target host or VM and take advantage of the host's lack of awareness to detect any malicious events or code that is being executed in runtime. One of the early methods of introspecting a Virtual Machine from an external VM is by Garfinkel and Rosenblum [14]. They used VMI to develop an Intrusion Detection System (IDS), called Livewire, for a customized version of VMWare Workstation for Linux. VMI techniques have also been used in Digital Forensics [17] and [18]. Hyperspector [19] implemented another Intrusion Detection System for distributed computer systems using VMI to isolate the IDS from the servers that they monitor. These isolated IDSs are located inside distinct VMs which are termed as IDS VM. There are also commercial products built using VMI technology [20].

3 PROBLEM STATEMENT

As discussed in the introduction, we see a proliferation of online public services provided over the internet. These services are provided by multiple suppliers, whose information systems interact through standards-based 'services' (e.g. SOAP, HTTP). At the same time, the information systems providing these public services are evolving towards Cloud infrastructures comprised for example of many homogeneous commodity servers with standardised operating systems, applications and hardware. This allows computing services to be provided as a utility, with virtualised hardware and applications, and the consumer of services unaware of how or where they are hosted.

At the same time, people, corporations and governments are more dependent on these services. So the cost of service failure is high, while at the same time risks are multiplied. Heterogeneity of systems mean a system problem or successful attack can have rapid, widespread effect, while pooled resources increase the attractiveness of targets. However the complexity and interconnectedness of systems means they are impossible for humans to diagnose.

The problem we face is how to detect, diagnose and predict problems in such system architectures. We discuss this under three headings. Firstly we look at model-based online fault diagnosis. Next we discuss using Process mining techniques where models are not available, and finally we discuss some results in prediction of problems in cloud-based systems.

4 MODEL-BASED DIAGNOSIS OF FAILURE

Models representing Internet-based systems are partially observable. The growing trends of using Service oriented Architecture means that services are developed and their interfaces are made available for the users. As a result, business processes models are produced that capture external behaviour of the system by accessing the interfaces of the services while the internal behaviour remains hidden from the

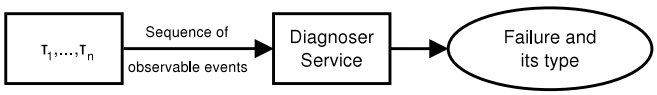

Figure 4: Diagnosis of Fault

users. As a result such models are inherently partially observable. In this context, as depicted in Fig. 4, diagnoser services (sometimes called Monitors or simply Diagnosers) are themselves services which receive sequences of observable events produced by the system and identify if a failure has happened or may have happened, in addition to the type of failure. Producing Diagnosers deals with two challenging issues:

1. is the system Diagnosable? i.e. whether it is possible to create a Diagnoser, and
2. creation of algorithms to construct Diagnosers from any given model.

The theory of Diagnosability of partial observable systems for Discrete Event Systems is well developed. Sampath *et al.* [21] in their seminal paper formulate Diagnosability and present a necessary and sufficient condition for Diagnosability. They also provide an algorithm for creating Diagnosers for Regular Languages. In their approach failure is modelled as transitions. Sampath *et al.* [21] has been extended to larger categories of models such as Petri nets [22]–[24] and even temporal logic [25], among others. The following definition from [26] extends the classic definition of diagnosability [21].

Definition 3. Consider a Petri net $\mathcal{N}$ with an initial marking M_0, which has no deadlock after firing of a transition which represents failure. We say $\mathcal{N}$ is Diagnosable if there are no two firing sequences s_1 and s_2 satisfying the following conditions:

1. $P(s_1) = P(s_2)$,
2. no failure transition appears in s_1,
3. there exists at least one failure transition in s_2
4. It is possible to make s_2 arbitrarily long after the occurrence of a fault.

The above definition states that in a diagnosable system it is not possible to come across any two execution sequences with the same observable behaviour ($P(s_1) = P(s_2)$), so that only one of them has a failure transition. The part about '. . . arbitrarily long after the occurrence . . . ' is to ensure that the systems continues long enough after occurrence of failure and is also present in [21]. The classic theory of Diagnosability which was originally designed for DES has now been adopted to develop Diagnosers for Service oriented Architectures and Telecom services [27]–[33], [13]. In these approaches, a number of services are considered in a SoA, as depicted in Fig. 5. We assume that models of such systems exist and failure which is going to be diagnosed is also modelled. Then, if the system (consisting of all involved services) is Diagnosable a new service is created and *integrated* in the infrastructure to use observable events and establish occurrence and type of

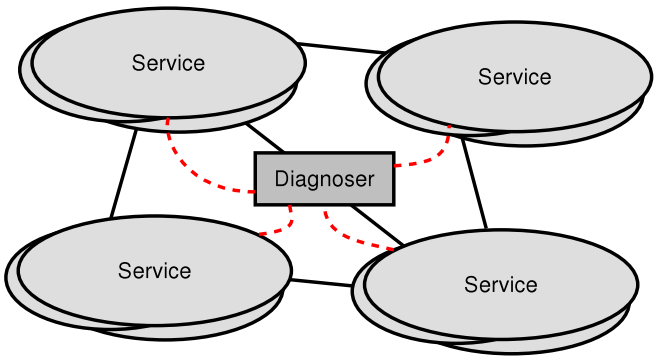

Figure 5: Diagnoser in a SoA

failure. Our recent work also uses code generation techniques to produce the Diagnosers and interfaces for integrating them into the system automatically [30]–[32].We shall explain this process with the help of an example [32].

Example 1. Right-First-Time failure. Consider a simplified interaction between a customer and a number of services in a typical Telecommunication Company for technical support related to the Broadband connection.

As depicted in Figure 1, the customer logs[1] onto the company website and enters details such as the account number. Choosing the 'Broadband problem' option, he submits his form online. Next, the company's Check Customer Account (CCA) service determines whether the customer account is in a satisfactory condition in order to progress the fault report. If the current status of the account is not satisfactory the customer is advised to phone the call center and the process ends. If the account status is satisfactory, the CCA invokes a request to another service called General Evaluation Services (GES). The GES examines the availability of service at the exchange side and ensures that everything is up and running, in which case the process moves to the next step. If GES identifies any problem with the availability of the services at the exchange side, the customer is informed of the status and a separate process is invoked to deal with this problem (not shown as part of this example). If everything is fine on the exchange side, the Customer Services sends a request to Line Test Service (LTS), which is an automated service to check line status up to the customer premises. However, LTS can also indicate problems on the exchange side which were not detected by the GES. There are three possible outcomes: 1) the line has no problem, move to next step, 2) the line has some problems, advise the customer or 3) There is no problem with the line, although there is likely a problem with the exchange. Option 3 is shown by the bold arrow in Figure 1. If case 3 happens, a failure emerges which means that GES should repeat its course of action violating Right-First-Time. Finally, LTS sends a request to analyse data history in the customer router. If it is possible to carry out analysis then get a decision from the analysis algorithm (either all OK so the customer has to call technical support, or the analysis finds

[1] We assume that the Customer can log into the company's website, for example supposing the customer is not happy with the speed of his Broadband connection.

the problem and customer is advised what to do).

For the details of the method of automated production of the Diagnoser we refer the reader to [32], where four methods of integration of the Diagnoser are also described. We make use of the models of the system that represent the interaction between the involved services. Figure 2 shows the example of the model used in BPEL. We converted this model based on the formalism suggested by Vanhatalo *et al.* [34] which draws on Petri net theory so that to apply Petri net Diagnosability theory techniques. Without such a model and formulation of failure, it is not possible to design Diagnosers on the basis of this technology. In the next section, we focus on techniques which are applicable to scenarios where models of the system are not available.

5 MONITORING OF LARGE SYSTEMS VIA LOGS USING PROCESS MINING

In the previous section we discussed automatically producing Diagnosers for near identifying failure in near real time. These require a model of the system to be diagnosed. In this section, we ask what we can do if there is no model, for instance if the services are built on legacy systems or is too complex or poorly understood to model. In this case we first need to find a model of the system to which we can apply Diagnosers. For this we use Process Mining.

Figure 1 showed a simplified problem resolution process from telecoms, implemented using a SoA. In a more complex example, this might be spread across several service providers (business entities). Each part of the process will involve information systems, so events pertaining to the business processes (e.g. Fig. 2) may be recorded in multiple event logs E_i. Assuming event logs defined as in (1), the E_i can be easily merged and traces extracted into a single workflow log W. This log contains full process traces from start to end activity, e.g. from the customer loggin in to the website, to resolution of the problem.

Process mining [9], [10] is the discovery and analysis of models of business processes from workflow logs. A process discovery algorithm Φ uses a minimal log such as W to attempt to recover a model M of the 'control flow' of the underlying process, such as that in Fig. 2, i.e.

$$\Phi(W) \rightarrow M \tag{5}$$

The recovered model M represents the 'true' business process and can be compared with an 'assumed process' M' (Fig. 6), used to troubleshoot differences, check adherence to business rules, SLA and audit requirements. Mined model M can be extended (e.g. with performance information) and used for performance analysis and identifying bottlenecks. M may also be used for planning, e.g. of business change, load balancing or energy efficiency by using as a basis for modifications, simulating the changed model. Models showing the interactions between people or organisations can be used to analyse the efficiency of work practices.

Many algorithms have been proposed for the control-flow discovery aspect of process mining. These start from different theoretical bases, or focus on different priorities. We refer

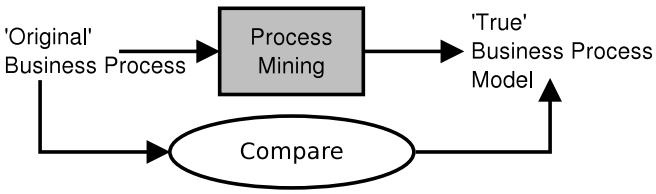

Figure 6: Process Mining

the interested reader to references in [9], [10], [35] for further details of algorithms. Business processes are often characterised by structuredness and concurrency: process models are (ideally) composed of substructures such as sequences and matching splits and joins, and activities or parts of the process may take place in parallel.

As a simplified example of a process discovery algorithm we outline the Alpha algorithm [36]. Consider two events a and b from the set of activities A belonging to a process M recorded in a workflow log W. These two events must be related in one of four relations, defined as follows.

- $a \rightarrow b$ (a may appear immediately before b in traces in W, never b before a), or conversely
- $b \rightarrow a$,
- $a \parallel b$ (sometimes a appears immediately before b, sometimes immediately after),
- $a \# b$ (a and b are always separated by at least one other activity).

The algorithm processes workflow log W to determine the relation between each pair of activities $(a, b) \in A \times A$. From this set of relations compiled for each pair of activities, a Petri net is created that satisfies all these relations. Note that this assumes that events are always recorded correctly in W.

One key question that arises is, if a is seen before b thousands of times and b before a only once, should this be interpreted as a mistake in the log or a rare scenario? In general this question can only be answered with knowledge of the business environment or service, and different algorithms make different assumptions.

So using a process mining algorithm such as Alpha we can discover process models as a basis for the diagnosis techniques described in the previous section. However,workflow logs can be large, and processing them can be computationally expensive (or data can be expensive or time-consuming to collect). Can we minimise the amount of data we need to use? How many process traces do we need to be confident that the model we have mined is the correct one? We next look at these questions.

5.1 Real Time Business Process Mining (RTBPM)

In this section we outline a probabilistic framework for considering process mining questions (for a fuller presentation see [35]). This provides a rigorous basis for answering questions such as 'how many traces do we need to be confident in the results of mining?', 'how different are two mod-

els?', and 'what is the probability that a detected fault is real and not an artefact of the data?'.

We here describe using this framework to determine the probability of identifying an undesirable scenario. Given a workflow log W, what is the probability P_f of identifying a failure or undesirable scenario, if we only use $X\%$ of the log? Conversely, given a desired P_f, can we calculate X?

To answer such questions involving uncertainty, we first need a probabilistic framework within which to consider business processes and process mining. Whereas business processes have traditionally been viewed as languages over activities, with no probabilistic structure, we consider business processes as probability distributions over strings of activities ('traces'). The primary task of a process discovery algorithm is to learn these distributions.

As introduced in section 2.2 we represent activities as symbols from a finite alphabet Σ, and traces as strings $x \in \Sigma^+$. We assume a probabilistic model for the generation of event traces, i.e. that traces are drawn into the event log *i.i.d.* (independently and identically drawn). The *true* business process M is modelled by a probability distribution P_M over traces, where the probability of a trace x is $P_M(x)$, such that $\sum_{x\in\Sigma^+} P_M(x) = 1$. As before, the workflow log W is a finite multiset over Σ^+, now understood to be drawn *i.i.d.* from P_M. The task of a process mining algorithm is to learn from W a distribution $P_{M'}$, to approximate P_M.

We are now assuming that we have a correct process model M of the system, e.g. previously mined from a 'large' log. We want to use process mining to monitor the system for failure, so the question becomes how many traces n do we need to use from the log W to be confident in mining M correctly? If we answer this question, we can be confident that if we use n traces and mine a significantly different model M', then the underlying process changed and we may have a fault scenario.

Since processes are distributions over traces, we use distances between distributions, such as the Euclidean distance, to test for significant difference between models, e.g.

$$d_2(P_M, P_{M'}) = \sqrt{\sum_x \left(P_M(x) - P_{M'}(x)\right)^2} > \epsilon, \qquad (6)$$

for small $0 < \epsilon \ll 1$

We use the Alpha algorithm [36] as an example. First we consider the basic substructures from which business processes are constructed, highlighted for example in Fig. 2. For acyclic processes, Alpha can discover sequences of activities, exclusive (XOR) splits (to alternative sequences of activities) and parallel (AND) splits (to parts of the process that may execute concurrently) and the corresponding join structures. Next we analyse the probabilistic behaviour of the algorithm to produce formulae for the probability of successful mining of these substructures, in terms of the probabilities in the model and n, the number of traces used for mining. These probabilities for discovery of structures can be combined to give the probability of successful mining by Alpha of the whole model M.

The discovery of structures in the model can be treated as conditional on the discovery of 'earlier' structures in the model, so if M is the example model in Fig. 2, then

$$P_\alpha(M) = P_\alpha(A) \times P_\alpha(B|A) \times P_\alpha(C|B) \times \ldots, \qquad (7)$$

where $P_\alpha(S)$ is the probability of Alpha correctly mining structure S, $P_\alpha(M)$ the probability of mining the full model. These probabilities are given in terms of n (the number of traces in the workflow log used for mining) and probabilities of substrings in the log.

To obtain the number of traces n needed to ensure that with confidence P_c the algorithm will produce the correct model, we invert the equation and fix a desired confidence in the mining results, $P_\alpha(M) = P_c$. Thus when a model is mined from a log of n traces, if the distance between the true and mined models $d(M, M') > \epsilon$ (equation 6), then with probability $P_f = P_c$ we have identified a fault.

The Alpha algorithm is relatively simple and makes many assumptions, e.g. no noise in the recording of the traces, and that the underlying process can be modelled by a restricted Petri net (Structured Workflow Net). However the same method can in principle be applied to any process mining algorithm.

6 MONITORING EMERGING MALICIOUS BEHAVIOUR

Having discussed using model-based Diagnosers to identify known faults, and process mining to learn unknown business process models from logs, in this section we ask whether we can diagnose a new fault which we have not seen before. This seems impossible in general. However it is possible in some cases to discover failure which is associated to emerging behaviour which has not seen before. In this section we give an example of such failure detection technology. The proposed method can be compared to the use of symptoms in human pathology, in which study of symptoms directs physicians to diagnosis of a disease or possible causes of illness. Observing unusual symptoms, even a physician cannot identify the illness, he will be alerted to conduct further experiments or to ask for expert advice. In that sense, from the observation of unusual symptoms the possibility of illness is discovered. In this section we argue that modern malware is becoming component wise. We also argue that in an environment such as Cloud in which introspection is possible, components used in malware produce symptoms. As a result, similar to pathology, observing of the symptoms can lead to discovery of possible malicious behaviour which can be new malware, or malware created from components used in old malware.

6.1 Reuse of Components and Techniques in Modern Malware

A malware writer must overcome a large number of obstacles to reach his objective. Among them, there are problems related to how to gain entry to a machine, how to install malicious code, how to evade detection, how to prevent the infected machine informing the owner, how to propagate, how to make analysis difficult, how to stop other malware writers

to gain access to an infected machine and so on. Considering the sophisticated nature of modern defence, solving all these problems demands huge resources. In addition, a low quality malware might 'give the game away' resulting in alerting security experts of the vulnerabilities of the target system. As a result, malware developers reuse the existing components, algorithms and techniques to improve the quality of the code. Some of the reuse is of legitimate components, for example using existing encryption libraries, and some are illegal software available online [37]. Consequently, it is common to come across variants of the same script within various malware products [38].

6.2 Symptoms That Point to Malicious Activities

Reusing code or techniques can leave symptoms behind. For example, a wide range of malware disables the defences of the system by stopping the antivirus software. Conficker [39] for example is a well-known computer worm that targets the Microsoft Windows operating system and forms a botnet. in the *Conficker C*, 23 processes are immediately aborted whenever they are discovered running on the victim host, including *sysclean*, *tcpview*, *wireshark*, *confik* and *autorun*. See page 12 of [39] for a list. Absence of Antivirus software from the Process Table of a system can be seen as a symptom that points to the possibility of malicious behaviour. Of course, it is possible that the Antivirus has been stopped for various legitimate reasons. Other examples of symptoms are unusual values for registry keys, or existence of high entropy code associated with encryption, which is essential for the malware when communicating with the malware writer. For a list of symptoms see [40], where we have included a list of symptoms which we have come across when studying well known malware.

The key point is that appearance of the symptoms can be a reason for further investigation. In particular, observing more than one symptom can convince us of the greater possibility of an undesirable behaviour. This is similar to the patient who is suffering from a disease which has caused multiple symptoms. Shifting the attention to looking for the symptoms, as opposed to looking for the malware that creates the symptoms, can alert us of existence of malicious behaviour.

6.3 FVMs and Monitoring of Malware While Remaining Hidden

To cope with sophisticated defence mechanisms deployed in modern systems, malware writers have developed techniques to remain hidden. For example, a common practice is to stop an infected system from contacting security vendors such as antivirus providers. In some extreme cases, malware writers can completely incapacitate the system by conducting aggressive actions such as killing the operating system to cover their tracks [37]. However, it is very difficult to remain invisible to someone viewing from 'outside' when VMI is used. Relying on VMI, the external viewer can observe the changing state of a VMs memory, processes that take inordinately long times to initialize, snippets of program code

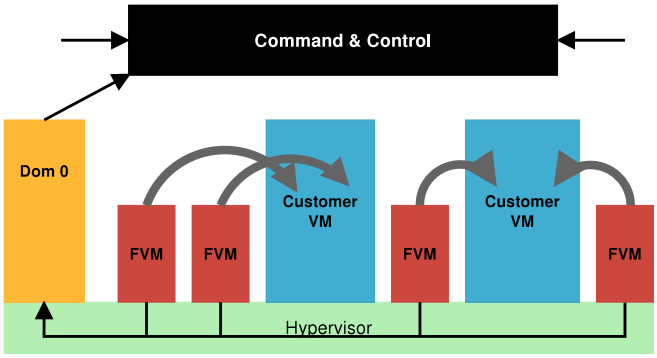

Figure 7: Forensic Virtual Machine Architecture

that has been obfuscated, snippets of code containing known crypto algorithms, or any modifications to the system code. Fig. 7 depicts the outline of the approach suggested in this paper. It shows a number of small independent VMs, called Forensic Virtual Machines (FVMs), which have been given the capability to inspect the memory pages of specific Customer Virtual Machines. Once a symptom has been detected, then the FVM reports its findings to other FVMs via secure multicast. In such cases, other FVMs will be prompted to inspect the VM for additional symptoms. In addition, when a symptom is discovered, this fact is reported, via Dom0, to a Command & Control centre. The Command and Control Centre correlates this information with information from other sources to identify an appropriate mitigation. For instance, the Command & Control, through the Dom0 and hypervisor, can 'freeze' the customer's VM by denying it any CPU cycles as a result to stop the malicious activity. The memory will remain frozen until it can be forensically examined or copied for further analysis.

FVMs make use of the computational resources that could otherwise be allotted to the customers VMs. As a result, management of the efficient allocation of the resources to the FVMs is crucial. In particular, creating and deploying an FVM is computationally intensive. In addition, permanent monitoring of an FVM is costly and wasteful, as the symptoms are expected to appear sparsely. We have designed the FVMs so that they regularly change their target Customer VM. To achieve this, a distributed algorithm is created to allow the FVM to schedule moving its searching process from one Custome's VM to another. We refer to such algorithms as mobility algorithms. For an example of a mobility algorithm see [40].

6.4 Limitations

The proposed approach has a number of limitations. Firstly, the suggested approach cannot cope with malware products which do not make use of component or existing algorithms. This although it seems unlikely is not impossible. Secondly, compromising Dom0 will allow taking over the virtualisation layer. To the best of our knowledge this has not happened yet. Securing the virtualisation layer is the subject of extensive research and technical innovations and will possibly define the battleground between malware writers who focus on Cloud.

Thirdly, it is possible to detect if a system is running on a virtualised environment. This would alert malware writers who wish to remain undetected to stay away from Cloud and focus on systems which are not virtualised.

7 CONCLUSION

In this paper we argue that the problem of ensuring correct functioning of modern systems is essential, due to their ubiquity and involvement in every area of modern life. We presented three examples of how we can approach these problems. Firstly, when we have a model of the system to be diagnosed, and secondly using logs to produce such a model when one does not already exist. Finally we discussed the situation when we are interested in emerging behaviour, such as detecting new malware threats in the Cloud, from the symptoms they present.

These examples all deal with very large and complex problems, where the size, complexity and amount of computation involved means it is not possible to manually avoid or even detect any the failures in the above categories. Therefore we have no no choice but to use computational resources to deal with the problems. As a result we are 'fighting fire with fire', using modern, distributed computing techniques to deal with faults caused within modern, highly distributed computer based systems.

REFERENCES

[1] W. H. Dutton and G. Blank. Next Generation Users: The Internet in Britain. *Oxford Internet Survey*, 2011. Oxford Internet Institute, University of Oxford.

[2] R. B. Prasad, E. Choi, and I. Lumb. A Taxonomy and Survey of Cloud Computing Systems. In Jinhwa Kim, D. Delen, Jinsoo Park, F. Ko, Chen Rui, Jong Hyung Lee, Wang Jian, and Gang Kou, editors, *NCM*, pages 44–51. IEEE Computer Society, 2009.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, Gunho Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, 2010.

[4] E. Byres and J. Lowe. Myths and facts behind cyber security risks for industrial control systems. *Engineering Technology*, 7(10):48 – 50, 2004-2005.

[5] P. Mohajerin Esfahani, M. Vrakopoulou, K. Margellos, J. Lygeros, and G. Andersson. Cyber attack in a two-area power system: Impact identification using reachability. In *2010 American Control Conference (ACC 2010)*, pages 962 – 7, Piscataway, NJ, USA, 2010.

[6] J. P. Farwell and R. Rohozinski. Stuxnet and the Future of Cyber War. *Survival*, 53(1):23–40, 2011.

[7] R. Chinnici, J. Moreau, A. Ryman, and S. Weerawarana. Web Services Description Language (WSDL) Version 2.0. `http://www.w3.org/TR/wsdl20/`, 2006. W3C.

[8] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures. `http://www.w3.org/TR/xmlschema-1/`, 2004. W3C.

[9] W. M. P. van der Aalst and A. J. M. M. Weijters. Process Mining: a Research Agenda. *Comput. Ind.*, 53(3):231–244, 2004.

[10] A. Tiwari, C. J. Turner, and B. Majeed. A Review of Business Process Mining: State-of-the-Art and Future Trends. *Bus. Process Manage. J.*, 14(1):5 – 22, 2008.

[11] P. J. Ramadge and W. M. Wonham. Modular Supervisory Control of Discrete Event Systems. In *Analysis and Optimization of Systems. Proceedings of the Seventh International Conference*, pages 202 – 14, Berlin, 1986.

[12] J. Vanhatalo, H. Völzer, and F. Leymann. Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In B. J. Krämer, Kwei-Jay Lin, and P. Narasimhan, editors, *ICSOC*, volume 4749 of *Lecture Notes in Computer Science*, pages 43–55. Springer, 2007.

[13] M. Alodib and B. Bordbar. A Modelling Approach to Service Oriented Architecture for On-line Diagnosis. *Service Oriented Computing and Applications*, pages 1–17, 2012.

[14] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *NDSS*. The Internet Society, 2003.

[15] D. E. Williams and J. R. García. *Virtualization With Xen: Including XenEnterprise, XenServer, and XenExpress*. Syngress Media. Syngress, 2007.

[16] R. P. Goldberg. Survey of Virtual Machine Research. *Computer*, 7:34–45, 1974.

[17] K. L. Nance, B. Hay, and M. Bishop. Investigating the Implications of Virtual Machine Introspection for Digital Forensics. In *ARES*, pages 1024–1029. IEEE Computer Society, 2009.

[18] B. Dolan-Gavitt, B. D. Payne, and W. Lee. Leveraging Forensic Tools for Virtual Machine Introspection. Technical Report GT-CS-11-05, Georgia Institute of Technology, 2011. `http://www.bryanpayne.org/storage/GT-CS-11-05.pdf` (accessed 19/10/2012).

[19] K. Kourai and S. Chiba. HyperSpector: virtual distributed monitoring environments for secure intrusion detection. In Michael Hind and Jan Vitek, editors, *VEE*, pages 197–207. ACM, 2005.

[20] Hypertection. Hypervisor-Based Antivirus. in Hypertection Team. Web, `www.hypertection.com` (accessed 13/09/2011).

[21] M. Sampath, R. Sengupta, and S Lafortune. Diagnosability of Discrete-Event Systems. *IEEE Transactions on Automatic Control*, 40:1555–75, Sept. 1995.

[22] S. Genc and S. Lafortune. Distributed Diagnosis of Discrete-Event Systems Using Petri Nets. In *Proceedings of the 24th International Conference on Applications and Theory of Petri Nets (ICATPN 2003),*, pages 316–336. Springer-Verlag, 2003.

[23] G. Jiroveanu, R. Boel and, and B. Bordbar. On-line Monitoring of Large Petri Net Models Under Partial Observation. *Discrete Event Dynamic Systems*, 2008.

[24] A. Giua and C. Seatzu. Observability of Place/Transition Nets. *IEEE Transactions on Au-*

tomatic Control, 47(9):1424–1437, 2002.

[25] S. Jiang and R. Kumar. Failure Diagnosis of Discrete-Event Systems With Linear-Time Temporal Logic Specifications. *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, 49(6):934–945, 2004.

[26] M. P. Cabasino, A. Giua, and C. Seatzu. Diagnosability of Bounded Petri Nets. In *CDC*, pages 1254–1260. IEEE, 2009.

[27] Y. Wang, T. Kelly, and S. Lafortune. Discrete Control for Safe Execution of IT Automation Workflows. In *EuroSys*, pages 305–314, 2007.

[28] Yuhong Yan and P. Dague. Modeling and Diagnosing Orchestrated Web Service Processes. In *IEEE International Conference on Web Services*, volume 9, pages 51 – 59, 2007.

[29] W. Hamscher, L. Console, and J. de Kleer, editors. *Readings in Model-Based Diagnosis*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[30] M. Alodib, B. Bordbar, and B. Majeed. A Model Driven Approach to the Design and Implementing of Fault Tolerant Service Oriented Architectures. In *3rd International Conference on Digital Information Management (ICDIM)*, 2008.

[31] M. Alodib and B. Bordbar. A Model Driven Architecture Approach to Fault Tolerance in Service Oriented Architectures, a Performance Study. In *3rd International Workshop on Modeling, Design, and Analysis for Service-oriented Architectures (MDA4SOA)*, 2008.

[32] M. Alodib and B. Bordbar. A Model-Based Approach to Fault Diagnosis in Service Oriented Architectures. In R. Eshuis, P. W. P. J. Grefen, and G. A. Papadopoulos, editors, *ECOWS*, pages 129–138. IEEE Computer Society, 2009.

[33] Yuhong Yan, Y. Pencole, M.-O. Cordier, and A. Grastien. Monitoring Web Service Networks in a Model-based Approach. In *ECOWS05*, Sweden, 2005.

[34] J. Vanhatalo, H. Völzer, and F. Leymann. Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In *ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing*, pages 43–55. Springer-Verlag, 2007.

[35] P. Weber, B. Bordbar, and P. Tiňo. A Framework for the Analysis of ProcessMining Algorithms. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, PP(99):1 –15, 2012. accepted.

[36] W. M. P. van der Aalst, T. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–1142, 2004.

[37] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. M. Youssef, M. Debbabi, and L. Wang. On the Analysis of the Zeus Botnet Crimeware Toolkit. In *PST*, pages 31–38. IEEE, 2010.

[38] Sheng Yu, Shijie Zhou, Leyuan Liu, Rui Yang, and Jiaqing Luo. Malware Variants Identification Based on Byte Frequency. In *Networks Security Wireless Communications and Trusted Computing (NSWCTC),2010 Second International Conference on*, volume 2, pages 32 –35, 2010.

[39] P. Porras, H. Saidi, and V. Yegneswaran. Conficker C analysis. *SRI International*, 2009.

[40] K. Harrison, B. Bordbar, S. T. T. Ali, C. Dalton, , and A. Norman. A Framework for Detecting Malware in Cloud by Identifying Symptoms. In *16th IEEE International EDOC Conference*, 2012. accepted.

Behzad Bordbar Behzad Bordbar has his BSc, MSc and Ph.D in Mathematics (PhD from Sheffield, UK). Following his PhD, he worked as a researcher on a number of projects at University of Ghent, Belgium and University of Kent, UK. He is currently affiliated to the School of Computer Science, University of Birmingham, UK, where he teaches courses in Software Engineering and Distributed Systems. In recent years, he has had close collaborative research with various academic and industrial organizations, among them Ghent University, Osaka University, Colorado State University, BT, IBM and HP research laboratories. His research activities are mostly aimed at using modelling to produce more dependable software and systems in shorter development cycles and at a lower cost. His current research projects are dealing with Formal methods, Model Analysis, Software Tools, Model Driven Development and Fault-tolerance in Service Oriented Architectures and Cloud.

Philip Weber Philip Weber received the BSc degree in Computer Science from Loughborough University, UK, in 1994, and the MSc in Advanced Computer Science from Birmingham University, UK, in 2009. Between these he worked in industry designing, analysing and implementing IT systems, and in systems administration. He is currently working towards the Ph.D. degree in Computer Science at the University of Birmingham, UK. His research interests include Process Mining, Machine Learning, Data Mining and information management.