Discovering and Visualizing Operations Processes with POD-Discovery and POD-Viz

Ingo Weber*†, Chao Li*, Len Bass*†, Xiwei Xu*†, Liming Zhu*†
*SSRG, NICTA, Sydney, Australia
Email: {firstname.lastname}@nicta.com.au
†School of Computer Science and Engineering, University of New South Wales, Sydney, Australia

***Abstract*—Understanding the behavior of an operations process and capturing it as an abstract process model has been shown to improve dependability significantly [1]. In particular, process context can be used for error detection, diagnosis, and even automated recovery. Creating the process model is an essential step in determining process context and, consequently, improving dependability. This paper describes two systems. The first, POD-Discovery, simplifies the creation of such an abstract process model from operations logs. An activity that previously required many manual steps can now be done largely automatically and in minutes. Using the discovered model, the second system, POD-Viz, provides operators with the ability to visualize the current state of an operations process in near-real-time and to replay a set of events to understand how the process context changed over time. This allows operators to trace the progress of an operations process easily, and helps in analyzing encountered errors.**

I. Introduction

It is well known in the dependability community [2] that the dependability of a system or a collection of systems depends not only on the actions of the system(s) but also on the environment in which these systems are executing. Multiple industry surveys, e.g. [3], and research studies, e.g., [4], state that as many as half of the outages to mission critical systems are caused by problems in operations processes, such as software upgrades. Thus, the impact of operations processes on applications is an important factor in system dependability.

This is the basis for our earlier work [1], where we reported that, when viewing operations activities as abstract processes and using the process context as guidance, error detection can be substantially improved. In this case, the process context provides the connection between the operations processes and the environment in which the system(s) are operating. Exploiting this connection between process context and dependability depends critically on the ability to quickly and easily create a model that enables determination of the process context. That is the goal of POD-Discovery, the first tool we report on in this paper. At process runtime, detected errors and the current state of the operations process need to be made accessible in an effective way, which is the goal of the second tool we present in this paper, POD-Viz.

Consider the example of upgrading an application running on 8 virtual machines (VMs) on Amazon Web Services (AWS)[1], using rolling upgrade as implemented by Netflix Asgard[2]: after changing the launch configuration for the cluster, one by one each VM is terminated and replaced with a new VM, running the new version of the application. This process will serve as a running example throughout the paper. Rolling upgrade changes the performance metrics of the cluster, and hence, while running, renders traditional anomaly detection tools largely useless. A myriad of things can go wrong during the upgrade, for reasons spanning from cloud uncertainty over API unreliability to concurrent operations competing for the state of the same VMs. By utilizing a process model and runtime process context, we have shown – among others – that errors can be detected and diagnosed in a timely fashion [1], and that analysis of performance variations can consider if a variation is due to operations or due to an error [5].

Generation of the process model from logs using the methodology described in our earlier work [1], [6] was labor-intensive, highly manual, and error-prone. 5 different third-party tools were used, there were 13 distinct steps in the generation which included code in 3 different programming languages, taking an expert 3-6 hours to get from a set of log files to an acceptable process model. In this paper, we present the tool POD-Discovery, to simplify the creation of the process model from logs. POD-Discovery implements and optimizes all the above steps, up to the point where standard process mining tools can be used for the final process model discovery step. Using POD-Discovery, the same expert now goes through three steps and it takes on the order of 10 minutes.

In addition, we present POD-Viz, a tool that visualizes the current execution state of a process in near-real-time on models discovered using POD-Discovery. POD-Viz also displays error and diagnosis messages that it receives from respective services. Replaying a set of past process events in POD-Viz can help to understand how the process context changed over time, particularly when leading up to an error.

The tools described in this paper make novel contributions:

- Clustering low-level event traces into higher-level events with POD-Discovery, through a novel distance metric for log lines and a novel interactive control method for the clustering procedure through direct manipulation on a dendrogram.
- Visualizing progress on a process instance with POD-Viz, along with any error and diagnostic information, and the feature to rewind and replay earlier events.
- Both tools rely primarily on log messages from operations tools, and hence require no changes to the operations tools – they follow a *non-intrusive philosophy*

[1]https://aws.amazon.com, accessed 30/4/2015.
[2]https://github.com/Netflix/asgard, accessed 24/11/2014.

for process discovery, error detection, and monitoring.

Screencast videos demonstrating the use of POD-Discovery and POD-Viz are available through our website.[3] The paper proceeds by presenting the mechanics and features POD-Discovery and POD-Viz in the next two sections. Subsequently, we discuss limitations and related work in Section IV. Section V describes how the tools are going to be demonstrated at the conference, before Section VI concludes the paper.

II. POD-Discovery

In order to achieve the increased dependability unveiled in our earlier work [1], one must obtain a process model from the log events a given tool emits. This is done through the use of process mining techniques [7]. However, process mining techniques are designed for business-level events, e.g., as generated by ERP or CRM systems. These techniques are not directly applicable to low-level system logs, such as from operating systems, middleware, platforms, and infrastructure-as-code approaches. In order to deal with such low-level, voluminous log messages, the user needs to cluster them into meaningful process events, such that they are comparable to business-level events. When traces of such higher-level process events have been obtained through clustering, standard process mining techniques can be used.

The first tool we present here has been developed to make clustering efficient and to reduce complexity, so that the interface is more scalable. By a more scalable interface we mean allowing the user to abstract from fine-grained details, so that the UI is still usable on thousands of events.

A. Overview

Fig. 1 shows an overview of the approach implemented by POD-Discovery. We describe the high-level logic in this section, before delving into details. First, logs are collected from the systems that are to be monitored. These logs form the input to POD-Discovery; the scope of POD-Discovery is indicated by the dashed box on the left of Fig. 1. POD-Discovery computes a specialized distance metric between pairs of log lines, and then performs Hierarchical Agglomerative Clustering (HAC, see e.g., [8]) based on that distance. The user adapts the clusters according to her understanding of the process being captured, and finally assigns names to the clusters. The clusters and respective cluster names then form the basis for deriving transformation rules for the original log. Using the transformation rules, the whole log is converted into an event trace, which is exported in a standardized format. This format is accepted by various process mining tools, which can discover a model from the event traces – see the "Current Model" in Fig. 1 for an example. Process models allow expressing manifold behavior in a compact fashion, including – among many other constructs – parallelism, (exclusive) choice, and repetition.

The user evaluates if the model fits her understanding – if not, she adapts the clusters and cluster names and re-runs the remaining steps. Once the model is satisfactory, the procedure completes with the export of the process model and the transformation rules, for import into runtime error detection

[3] http://reliableops.com

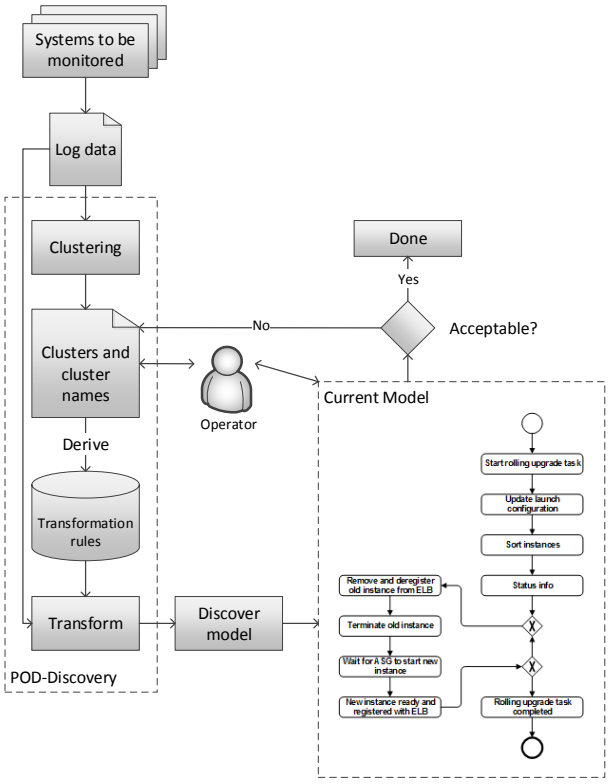

Fig. 1: Overview of the POD-Discovery approach

services and POD-Viz, as discussed in Section III. We now discuss each of the steps in more detail.

B. Log Line Distance and Clustering

The goal of the clustering is to group log lines that belong to the same *activity*, i.e., a step of the process at a desired level of abstraction. What that desired level of abstraction is depends on the system to be monitored, and is up to the user to decide. Typically, activities of interest have an effect of interest, such as an observable change of state. To give a concrete example, take the launch of a new virtual machine. Periodic status outputs of "still waiting for the VM to start" can usually be grouped together. The interesting change in status is reached when the VM actually completed the launch and boot-up sequence and is ready to serve requests.

In order to provide a meaningful clustering result, the key is in calculating a meaningful log line distance. To achieve this, POD-Discovery splits each log line into a hierarchy of *tokens*. An example of this tokenization is shown in Fig. 2. Several characters are interpreted as splitting points between tokens, such as various kinds of braces, tab stops, etc. As can be seen from Fig. 2, POD-Discovery recognizes specific tokes such as time, date, IP addresses, URLs, etc., as well as specific Amazon Web Services (AWS) identifiers, including instance IDs, e.g., `i-1a2b3c4d`, and AMI IDs, e.g., `ami-2b3c4d5e`.

For each of the token types, specific distance token metrics can be defined. These are applied whenever two tokens of the same type are compared. Fig. 3 shows an example of the comparison based on the tokenization and token type-specific distance calculation. The default token type is *String*,

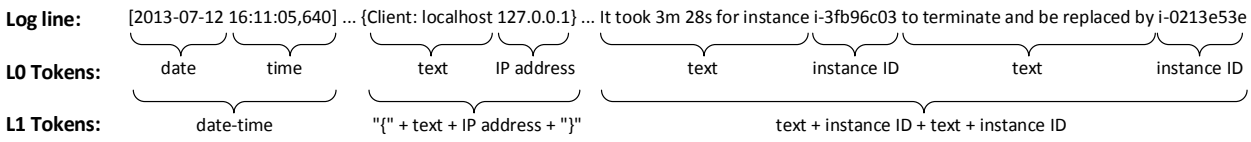

Fig. 2: Tokenizing log lines

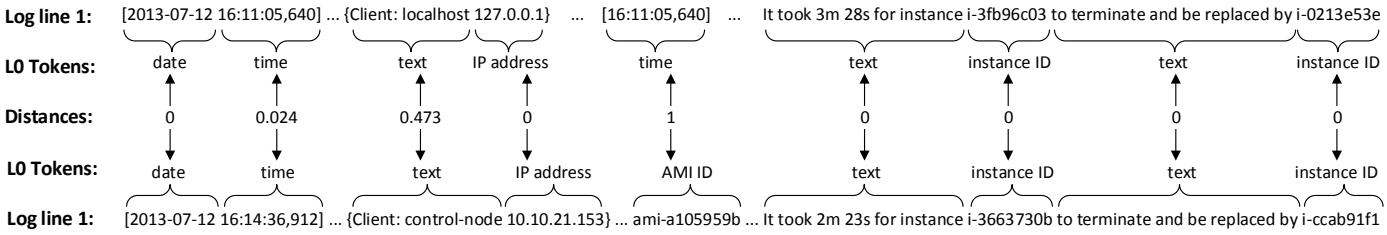

Fig. 3: Calculating log line distance based on token distances

and the distance metric applied for String comparison is the Levenshtein distance [9]. If one token is compared with a token of another type, the maximal distance, "1", is assigned.

In order to identify activities, it is often helpful to ignore varying parameters, such as IP addresses and IDs. Therefore, the default metric is to simply return "0" as a distance for any pair of, say, IP addresses. For other token types, such as *date-time* (consisting of a date and a time token), a more specific calculation can be made, such as the total time elapsed between the two date-time tokens.

The set of token types can easily be adapted if need be, through a plugin mechanism: the token types are defined in a configuration file, each with a regular expression. The order in the file specifies the order in which the regular expressions are matched. A given string is interpreted as a token of the first type whose regular expression matches. Comparison implementations, i.e., token type-specific distance metrics, are referred to by Java class name in the configuration file, where the class is required to implement a given interface. When POD-Discovery compares two tokens of a type, the respective Java class is instantiated. New token types can thus be added by adding a regular expression and a Java class implementing the comparison of two tokens of that type.

As a final step, all token distances are multiplied by the token length, added up, and divided by the length of the longer log line for normalization. The set of distances is then passed into an implementation of the HAC "average-link" algorithm [8, Chapter 4] for clustering.

C. Dendrogram-based Cluster Selection

The HAC results are shown to the user as a Dendrogram, which forms the blue tree depicted in the POD-Discovery screenshot of the respective screen, Fig. 4. A dendrogram, in general, is a tree of nested clusters. The root cluster (top node in Fig. 4) represents a cluster containing all items, i.e., log lines in our case. The leaf nodes on the bottom of Fig. 4 represent clusters of size 1, i.e., individual log lines. The nodes in between each represent a cluster that forms the union of the two child node clusters. The maximum distance between any pair of log lines in a cluster increases proportionally with the height of the node. Observe the orange scale[4] on the left of Fig. 4: this shows the distance at which two child clusters are merged into a parent cluster. For instance, the left-most orange circle is around a dendrogram node that merges its two sub-trees into one cluster at a distance of 28%.

The user can interact with the dendrogram screen of POD-Discovery in several ways. The dendrogram represents many possible options how to cluster the log lines. For instance, for the left-most log line, should it form a cluster by itself, or be part of the cluster at 2%, or the one at 4%, at 18%, or even the one at 28%? The user can make that decision by selecting the nodes representing these clusters. In POD-Discovery, the selected nodes are shown in orange, as shown in Fig. 4.

In order to make the decision about the clusters, it is important to know which log lines are clustered in which node. This can be observed after clicking on any node, which leads to the contained log lines being shown in the text box below the dendrogram.

The orange circles can be modified in two ways. The first option is to click on the orange scale on the left, say at 20%. Then, the nodes closest to 20% will be selected for each sub-tree that intersects with the 20% threshold. This is fairly coarse-grained, and in our experience hardly ever led to satisfactory results. However, for a first selection it is very useful. The second option allows fine-grained tuning, by dragging-and-dropping the orange circle from a selected node to another node, which has to be either on the sub-tree below the selected node or on a path from the selected node to the root. The selected nodes will then be updated to match this change. For instance, if the left-most orange circle is moved to one of its children, then the other child will be selected as well. All manipulations are implemented such that each leaf node is always contained in exactly one selected cluster.

Finally, if the log contained thousands of lines, the dendrogram would become very wide, and it would be very hard to get an overview. Therefore, the user can *abstract* from nodes

[4]Note that the distance is rounded to 2% for clarity of presentation. This is a customizable setting.

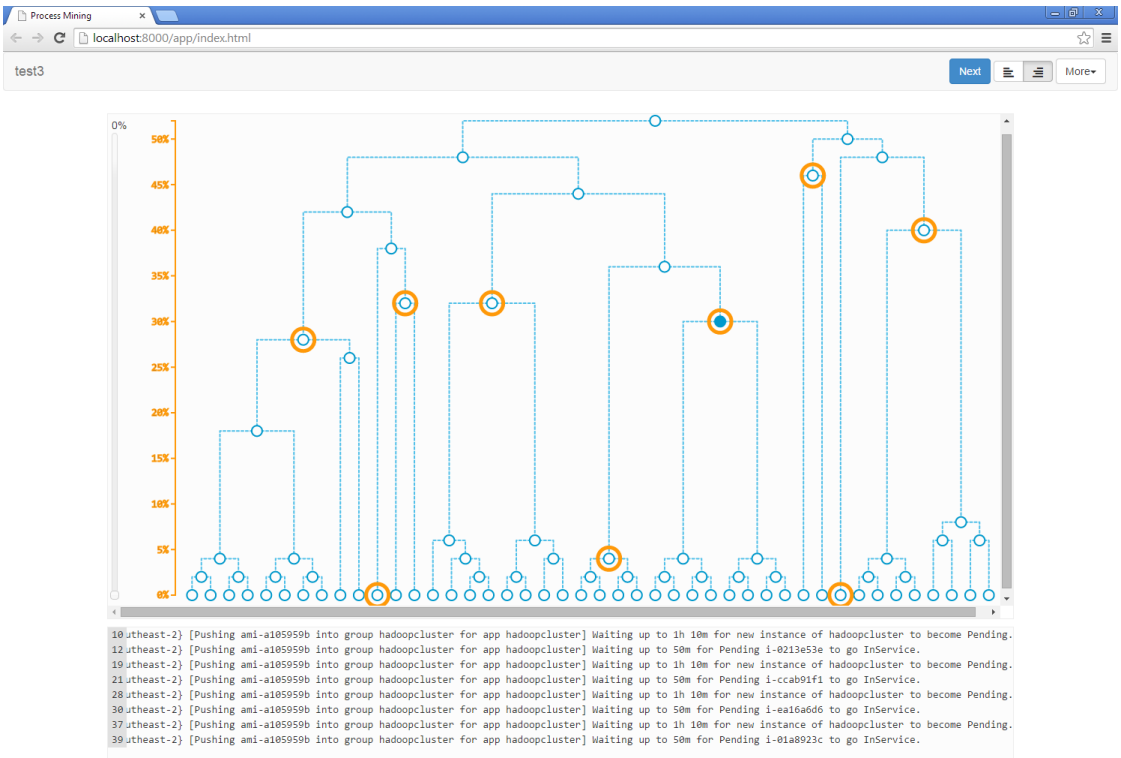

Fig. 4: Selecting clusters in the dendrogram (POD-Discovery screenshot)

with a high similarity. This is done using the white slider next to the orange scale – set to 0% (no abstraction) in Fig. 4. By pulling the lever up, all nodes below the selected abstraction level are hidden – so long as they are part of a selected higher-level cluster. Selected lower-level clusters are shown as before, but all their children are hidden.

The demonstration video of POD-Discovery mentioned in the introduction shows all these interactions.

D. Cluster Fine-tuning and Naming

While the dendrogram allows already for fairly fine-granular selection of clusters, it only permits changes that follow its tree structure. Further fine-tuning of the clusters may be needed, e.g., when relatively similar log lines belong to different activities. In POD-Discovery, this can be done using the screen shown in Fig. 5. This step comes after the dendrogram manipulation, by clicking the button labeled "Next" shown in the right upper corner of Fig. 4. It shows all clusters and their log lines in text boxes. The user can drag-and-drop log lines from one cluster to another, remove them, create new clusters or remove empty ones.

In addition, the user has the task of naming the clusters. Each cluster becomes an activity on the process level, so the assigned names become the activity names in the process model. Hence it is important to assign speaking names.

E. Transformation Rules

Once the cluster contents and names have been finalized for the current iteration, the tool can derive and apply transformation rules. A transformation rule specifies which name to annotate to a given log line. The rule conclusion is thus the annotation of the cluster name. The rule condition is a regular expressions that captures all log lines in a cluster, and generalizes selectively over this set.

In particular, each cluster is a group of log lines, split into tokens. For token types, the behavior during export can be controlled in a specific way, similar to token type-specific distance metrics. In the absence of specific instructions, POD-Discovery interprets tokens as Strings, except if the distance metric is statically "0": then any occurrence of the token is replaced by the regular expression for the token type. For example, instance ID `i-1a2b3c4d` is replaced by `i-[0-9a-f]{8}`, where the part after the "i-" refers to exactly 8 hexadecimal values. For date, time, and date-time tokens the same is applied. Furthermore, any numerical values in a log line that differ between otherwise identical log lines are replaced by `\d+`, i.e., a regular expression that accepts any number. For the remaining text portions of the log lines, a regular expression is derived by constructing minimal acyclic DFA (deterministic finite automaton) as described by Watson [10], and by turning this DFA into a regular expression.

By then applying the transformation rules constructed as per above to the original log, we obtain an activity annotation for each log line.

F. Export

Process mining is an active field of research, see e.g., [7], and various companies offer tools in that space. In order to allow these tools to operate on our annotated log lines, we export them in a standardized format. The standard for

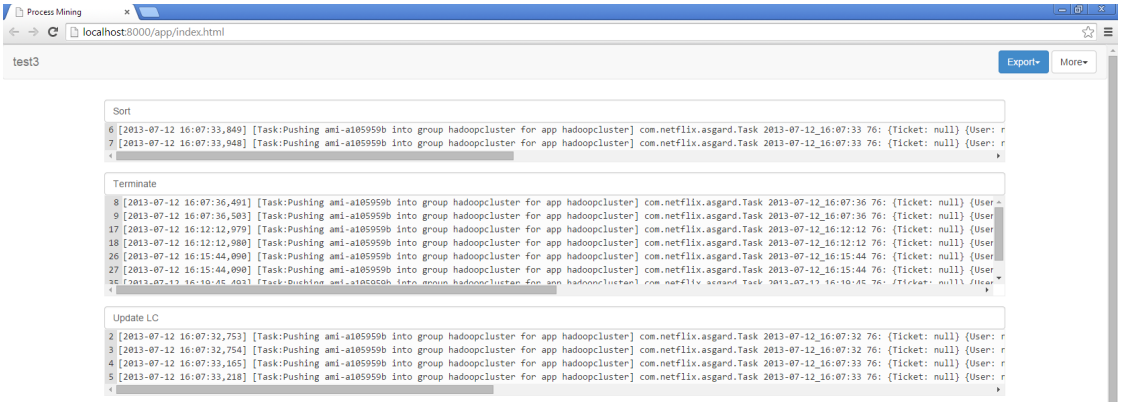

Fig. 5: Fine-tuning and naming clusters (partial POD-Discovery screenshot)

process mining event traces is called extensible event stream (XES)[5]. We use the reference implementation, OpenXES 2.0, which is available from the standard's website. An alternative output format that we support for historic reasons are comma-separated value (CSV) files. Also, POD-Discovery can export the set of transformation rules, which can be imported by various other services in our overall architecture, such as POD-Detection services and POD-Viz.

G. Process Discovery

Finally, the exported event traces can be imported to a process discovery tool, such as ProM[6] or Disco[7]. The result of discovering a process model in Disco from the event trace exported from Fig. 5 is shown in Fig. 6.

A myriad of process discovery algorithms exist in the literature, therefore it would not be practical to re-implement these in POD-Discovery. Instead, we make use of the XES standard for exporting event traces from POD-Discovery, so that the tool and algorithm of choice can be selected. POD-Discovery is used at design time. Once a suitable process model has been derived, it can be used at runtime.

III. POD-Viz

At runtime, it is important for operators to gain visibility into processes that affect the systems under their control. To that end, we developed POD-Viz, a visualization application that is fed with live information from running operations processes. The two main challenges for operators which POD-Viz addresses are:

- Obtaining a view of the current state of the operations process, including alerts – if any.
- Tracing the order of events that led to an alert, for diagnosis.

A. POD Runtime Architecture

The POD runtime design follows the microservice architecture approach [11]: several services provide solely a dedicated, small function, and communicate through defined interfaces with each other. Each component is self-contained and can be registered to receive updates from other services.

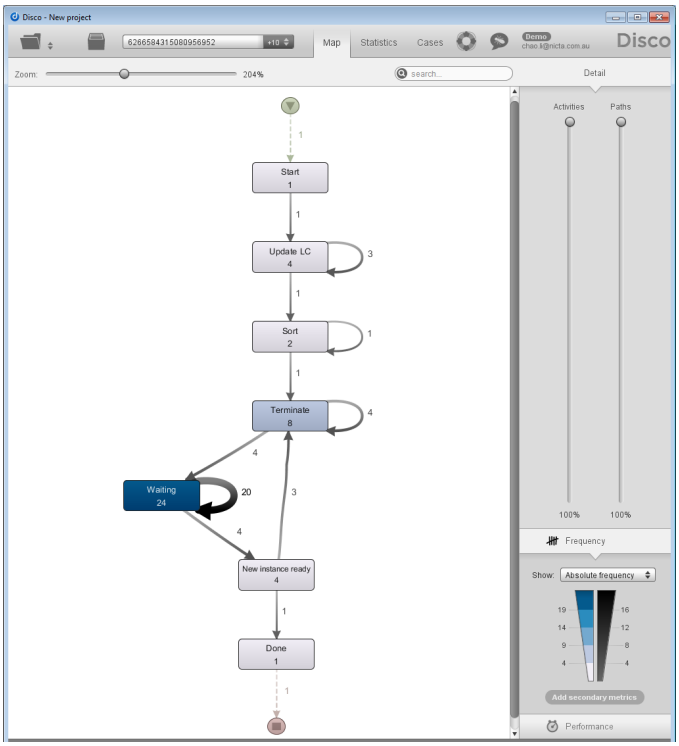

Fig. 6: Discovered process model (Disco screenshot)

POD runtime includes error detection services, error diagnosis services, and recovery services. If any service is not present, the remaining set of services can still deliver their respective functionality. Any service can be registered to receive messages from another service, in a publish-subscribe fashion. Error detection services can include timeout values: if an expected event is not received at most x seconds after the previous event, this triggers an alarm as well.

POD-Viz is another service in this architecture. In the following sections we describe the kinds of messages it subscribes to and how it visualizes the information contained in them in

[5]http://www.xes-standard.org/, accessed on 21/11/2014.
[6]http://www.promtools.org/, accessed on 21/11/2014.
[7]http://www.fluxicon.com/disco/, accessed on 21/11/2014.

Fig. 7: Observing process state without errors in POD-Viz

relation to process models discovered through POD-Discovery.

B. Messaging and Message Processing

Out of the different types of messages that can be sent between the services, two are of particular relevance to POD-Viz: log messages and alerts. Log messages originate from an operations tool, such as Netflix Asgard. We use Logstash[8] to detect new log lines, filter them, and ship relevant log lines to registered POD services. POD-Viz offers a service interface where it accepts log messages. Equipped with the process model and the transformation rules from POD-Discovery, POD-Viz can process the original log lines and understand to which process activity they belong. With this information, it can update the view on the process model.

A set of error detection and diagnosis services of POD have been described in [1]. If present, the error detection services receive any new log line from Logstash as well. When an error is detected, an alert message is sent to the diagnosis service and POD-Viz, which displays the alert as a possible error. The diagnosis service then tries to find the (root) cause of the error. Since there is a small rate of false positives, and since some errors are transient, the diagnosis service might determine that no error is actually present, and send a message to that effect to POD-Viz. If, in contrast, an error is confirmed and a cause for the error determined, a message containing more details is sent to POD-Viz – and potentially further alarms are sent directly to pagers/mobile phones of operations staff on duty.

[8]http://logstash.net/, accessed 24/11/2014.

C. POD-Viz Visualization

POD-Viz is a Web application that has two types of pages: (i) an overview list of all running and past process instances (not shown), and (ii) a page for displaying details of a process instance. Fig. 7 depicts the details page during a normal run of a process, and Fig. 8 shows the same page after an error has been detected. A collapsible dashboard is available on both pages. It shows important information about all running process instances, such as the current status – see the box labeled “Process Instances” in Fig. 7.

The process instance details page consists of three main components, besides the dashboard. The time line widget on the left shows current metrics from the cluster of machines, such as average CPU utilization and network traffic – these can be retrieved from monitoring services such as AWS CloudWatch. When an event occurs, it is shown as a vertical bar at the particular point in time when it was registered. The process model view on the right shows the current execution state of the process. Specifically, previously executed activities are shown in blue, not-yet-executed activities in grey, the last activity for which an event was registered is highlighted in green (as in Fig. 7), and an activity associated with an error in red (as in Fig. 8). The same color code is used for event bars on the time line. Finally, details of the latest events are shown at the bottom of the page.

Fig. 8 further demonstrates how an error can be analyzed. It depicts an error message, prominently displayed with a red background, as well as the details of corresponding log lines. POD-Viz has two modes in general: a *replay* and a *live mode*. These are controlled by the music player-like controls below

Fig. 8: Error display and analysis in POD-Viz

the time line widget. Clicking "pause" switches to the replay mode, where the user can jump backward and forward, event by event, through clicks on the "previous" / "next" buttons. Clicking "play" switches to the live view, where the page updates whenever new events trickle in: the content of the time line widget is continuously scrolling to the left, so that current values and events appear on the right; updates to activity states in the process model and new log lines appear as soon as they are received. Clicking on an event bar in the time line switches to replay mode, showing the state right after that event happened. Replay enables the operator to understand the sequence of events leading up to an error.

Processing of new log lines continues in the background, regardless of the mode the UI is currently in. While in "pause" mode, any new messages, such as new errors, are only visible through the dashboard.

IV. Discussion

A. Limitations

POD-Discovery assumes that log files are plain text files, where each line represents an event. This is the standard format for many tools. The tool applies only to log files where this is given, or feasible to achieve through pre-processing. Furthermore, POD-Discovery implements one standard way of performing clustering, which may or may not work well on a given kind of log files. In order to customize the clustering, POD-Discovery offers a plugin mechanism to adapt the configuration and implementation, as discussed in Section II-B, and there is the fine-tuning option discussed in Section II-D. With these options, clustering should often be possible, but might not be as convenient as when the default implementation works well. Process mining in general has a few assumptions, see e.g. [7].

POD-Viz assumes that it receives information promptly, but is dependent on external systems (e.g., for error detection). POD-Viz can be very helpful in situations where a somewhat structured process is followed, and where operators benefits from a process-based view.

Error detection services may be specific to certain processes. Purely log-based detection services, such as conformance checking [1], require only the customization offered by POD-Discovery – but in turn have strengths and limitations. In short, conformance checking can detect errors that are unknown *a priori*, but is limited to errors that actually change the log behavior. Assertion evaluation services, as those discussed in [1], can detect in principle any observable errors, but require *a priori* knowledge and coding.

B. Related Work

On the topic of clustering data for process mining, there have been a number of works. The approach with the closest relation to POD-Discovery, van Dongen and Adriansyah [12], provides a simple clustering mechanism *during* process discovery. The goal is to discover process models from lower-level and higher-level events, and to display performance information on top of the discovered model. The clustered lower-level events become part of the process model. This approach is infeasible if there are hundreds or thousands of log events, since it overburdens the process discovery algorithm with the additional task of clustering. Many other approaches, such as [13], [14], [15], focus on clustering log *traces*, not events. The goal of these approaches is to find events that belong to the same trace, whereas our focus is orthogonal: POD-Discovery clusters events that belong to the same higher-level activity.

In terms of clustering tools using dendrograms, we are only

aware of tools that use a single threshold on the dendrograms. For instance, a blog entry [16] describes a tool that interactively displays dendrograms. It still applies a single threshold over the whole dendrogram for the clustering. In contrast, POD-Discovery allows the user to select a separate threshold (and thus clustering granularity) for each subtree – so long as this results in every leaf node being part of exactly one cluster.

POD-Viz, on the other hand, offers functionality that is similar to parts of what several process execution engines offer, e.g., Intalio BPMS[9]. Management interfaces of these execution engines allow to explore the current status of the process model, visually, through data variable content, etc.; and they display error messages. However, the difference is that these interfaces provide information about what is happening *inside* the execution engine. In contrast, POD-Viz provides information about processes running in *other* tools or sets of tools. It is thus part of the POD tool set, which can provide non-intrusive error detection and diagnosis. Furthermore, the error/diagnostic messages displayed stem from other services in the POD tool family, not the tool producing the original log messages. We are not aware of any tool similar to POD-Viz in this respect.

V. Demonstration Outline

At the conference, we will demonstrate the tools by explaining the assumptions, approach, and architecture, followed by showing how POD-Discovery and POD-Viz operate on practical examples. The demonstrations will proceed along the lines of the screencast videos. Both tools will be demonstrated using prepared input data sets, where POD-Discovery will use real life logs from Netflix Asgard. For POD-Viz, we will use similar logs, but with altered timestamps so as to keep the demo interesting and not keep the audience waiting for several minutes while there is nothing happening in Asgard. To this end, we will use a testing tool we developed, which sends log events from a given log to POD-Viz every second. We will further highlight the error notification and replay features.

In addition, if time and setup permit that, we will offer to use POD-Discovery live, on data provided by visitors – either on USB flash drives or through a public URL. The motto for this part of the demo is: *if you bring your log data, we will try to find a meaningful process model in 15 minutes or less.*

VI. Conclusion

In this paper, we presented two tools that contribute to increased dependability through use of process context. The first tool, POD-Discovery, is the *enabler* for all the benefits process context can offer to dependability: it simplifies and speeds up the generation of high-level process models from low-level system logs. This is achieved through a novel distance metric for log lines, standard clustering based on that metric, a novel interactive control method for the clustering procedure through direct manipulation on a dendrogram, and a manual fine-tuning feature for the clusters.

The second tool, POD-Viz, allows the operator to understand the current state of a process by using the process model discovered through POD-Discovery. POD-Viz further shows any error or diagnosis messages. It also allows to replay the observed events, which is useful to get insights into how a sequence of events led up to a fault. POD-Viz is part of a microservice architecture of services for detecting and diagnosing services at runtime, and recovering from them.

Screencast videos of both tools are available, see footnote 3 on page 2.

[9] http://bpms.intalio.com/, accessed 26/11/2014

Acknowledgments

We thank Rijia (Michael) Guo for his development efforts. NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program. We thank Fluxicon for providing academic licenses for Disco.

References

[1] X. Xu, L. Zhu, I. Weber, L. Bass, and W. Sun, "POD-Diagnosis: Error diagnosis of sporadic operations on cloud applications," in *DSN'14: IEEE/IFIP Intl. Conf. on Dependable Systems and Networks*, June 2014.

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Tr. on Dependable and Secure Computing*, vol. 1, no. 1, Jan 2004.

[3] R. Colville and G. Spafford, Gartner, Oct. 2010, RAS Core Research Note G00208328.

[4] D. Yuan, Y. Luo, X. Zhuang, G. R. Rodrigues, X. Zhao, Y. Zhang, P. U. Jain, and M. Stumm, "Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems," in *OSDI'14: Symposium on Operating Systems Design and Implementation*, Oct. 2014.

[5] X. Xu, L. Zhu, M. Fu, W. Sun, A. B. Tran, S. Dwarakanathan, and L. Bass, "Crying wolf and meaning it: Reducing false alarms in monitoring of sporadic operations through POD-Monitor," in *COUFLESS'15: Workshop on Complex Faults and Failures in Large Software Systems*, May 2015.

[6] X. Xu, I. Weber, H. Wada, L. Bass, L. Zhu, and S. Teng, "Detecting cloud provisioning errors using an annotated process model," in *MW4NextGen'13: Workshop on Middleware for Next Generation Internet Computing*, December 2013.

[7] W. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[8] H. Marmanis and D. Babenko, *Algorithms of the Intelligent Web*. Manning, 2009.

[9] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Doklady Akademii Nauk SSSR*, vol. 163, no. 4, 1965, English translation: Soviet Physics Doklady, 10(8):707-710, 1966.

[10] B. W. Watson, "A new algorithm for the construction of minimal acyclic DFAs," *Science of Computer Programming*, vol. 48, no. 23, 2003.

[11] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, 2015.

[12] B. van Dongen and A. Adriansyah, "Process mining: Fuzzy clustering and performance visualization," in *Business Process Management Workshops*, 2010.

[13] G. Greco, A. Guzzo, L. Pontieri, and D. Sacca, "Discovering expressive process models by clustering log traces," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 8, Aug 2006.

[14] M. Song, C. W. Günther, and W. M. van der Aalst, "Trace clustering in process mining," in *Business Process Management Workshops*, 2009.

[15] D. Ferreira, M. Zacarias, M. Malheiros, and P. Ferreira, "Approaching process mining with sequence clustering: Experiments and findings," in *BPM'07: Intl. Conf. on Business Process Management*, 2007.

[16] Creative Data Solutions Blog, "Interactive heatmaps (and dendrograms) - A shiny app," 7 July 2013, accessed 2 December 2014. [Online]. Available: https://imdevsoftware.wordpress.com/2013/07/07/interactive-heatmaps-and-dendrograms-a-shiny-app/