Toward Automatic Policy Refinement in Repair Services for Large Distributed Systems

Moises Goldszmidt[1], Mihai Budiu[1], Yue Zhang[2], Michael Pechuk[2]
[1]Microsoft Research and [2]Microsoft Windows Azure
Silicon Valley Campus

ABSTRACT

In order to be economically feasible and to offer high levels of availability and performance, large scale distributed systems depend on the automation of repair services. While there has been considerable work on *mechanisms* for such automated services, a framework for evaluating and optimizing the *policies* governing such mechanisms has been lacking. In this paper we propose one such framework and report on our initial experience in applying the framework to analyze and optimize the operation a geo-distributed cloud storage system at Microsoft.

1. INTRODUCTION

Researchers and practitioners have made a convincing argument for automated repair services in large distributed systems [9, 1]. This work been accompanied by significant amounts of research into the implementation of the necessary mechanisms for making these automated repair services a reality [6, 3]. However, we have seen little research on (a) evaluating the policies governing these services, and (b) refining and optimizing these policies. In this paper we present some first steps toward these goals. Our approach is grounded in survival analysis and statistical machine learning, in order (a) to estimate from log data the effects of a specific repair action, and (b) to automatically extract the factors that predict the effect of each repair action.

We analyze a repair service similar to Autopilot [6]. The unit of failure is a "device" rather than a process. The available remedies range from doing nothing, automated reboots and re-imaging, up to several levels of expert human intervention. Faults are detected using monitoring agents that send event signals to the repair service. The life-cycle of each device is managed by a state machine; the state changes as a function of the event signals from the agents, the current state of the device, the repair actions, and the (recent) repair history of the device. The repair policy is thus defined to be a mapping from the current state, the signals, and the repair history to a repair action. A diagram of the closed loop repair system is depicted in Figure 1. A detailed description of the repair service is provided in Section 2.

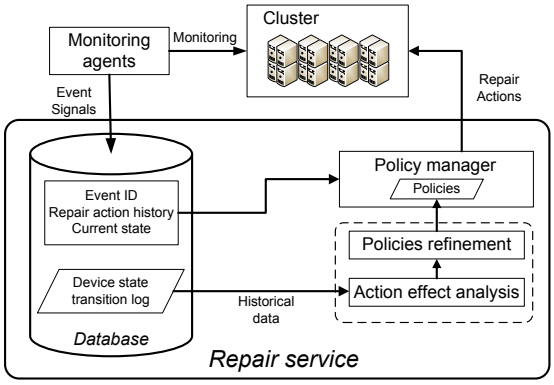

Figure 1: The closed loop system. This paper deals with the analysis of the repair action and the refinement of the policy (the dotted box).

The problem we address in this paper is that of refining the policy used by the repair service to improve the quality of service (QoS) in a cloud computing environment, especially the availability and cost metrics. The repair actions impact the QoS in two aspects: (1) (reduced) service availability, such as machine temporarily unavailable while the repair action is being executed (or while it takes effect), and (2) the economic cost of the action (e.g., billing by service operators). Given these costs, some examples of the policy refinements may be: (1) if we infer that some repair action cannot bring the service back to normal operating parameters, then we should choose a different repair action, decreasing cost due to downtime; (2) if we infer that a certain event triggered by the monitoring agent implies a problem that a Tier 1 operator cannot solve, it is better to directly escalate to a more experienced Tier 2 operator, to reduce the unnecessary cost spent on Tier 1.[1]

We refine the policy by concentrating on each specific repair action at a time. Our approach is in two parts. The first part consists of evaluating the effectiveness of each repair action. We make the assumption that this effectiveness is directly correlated to the amount of time that the device stays "healthy" after the repair action.[2] To

[1]This is a temporary solution; once the correct repair sequence is devised out by Tier 2, it will be "pushed down" to Tier 1.
[2]The time that the repair action needs to take effect is also part

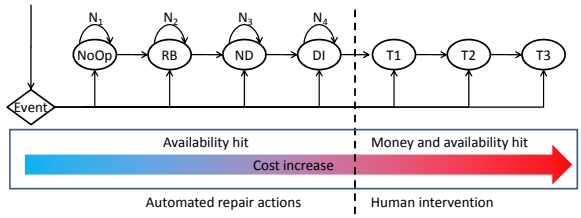

Figure 2: Repair action ladder. Where: NoOp = No operation, RB = reboot, NDI = non destructive re-imaging, DI = destructive re-imaging, US = update software, T1, T2 and T3 = tier 1, 2, 3 human intervention.

estimate availability we use techniques from statistical survival analysis [5].

The second part uses the survival analysis results to refine the policy. For this purpose we use statistical machine learning techniques. In particular we automatically induce a classifier-based model of the relationship between (a) the history of the device before the repair action, (b) the specific events triggered by the agents, and (c) the probability of survival after the repair action. We use the history to separate the devices that survive more than a threshold T from those that survive less. The threshold T is computed based on the economics of the service, such as the worth of the machine, time to repair etc. The dotted line in Figure 1 displays the two components that we are engineering, analyzing, and optimizing.

There are two additional questions that we can address with the techniques explored in this paper. The first one is that of debugging the monitoring system itself: e.g., are there signals that are abnormally frequent? The second one is finding statistically significant differences in the behavior of different datacenters.

The data that drives the refinement process is extracted from the logs of the repair service under consideration. These logs contain the state transitions of each device, the repair action being executed, and the timestamps of the events. The logs come from a large geo-distributed cloud storage system at Microsoft that spans four data centers, with thousands of machines with commodity hard disks. The storage capacity is several petabytes, serving the needs of hundreds of millions of end-users, performing billions of daily requests, amounting to hundreds of terabytes.

2. REPAIR ACTIONS AND POLICIES

The automated repair system (RS) associates a "state" with each device.[3] The actual instance of the state is decided by the RS using event signals sent from the set of monitoring agents. For the purposes of this paper the set of possible states is:

1. Healthy (H) - device is considered available for the service and does not have any abnormal reports from monitoring agents.
2. Failed (F) - there is some abnormal report which has an associated event ID.
3. Probation (P) - device is recovering from failure after the application of a repair action (thus, signals from the monitoring agents are basically ignored).
4. Down (D) - device needs human intervention.

To this set of basic states we also add states induced by the repair actions (see below for the description of repair actions), such as Updated (U) and Rebooted (R).[4]

Several repair actions can be used to bring a device from the failed state back to the healthy state (see Figure 2). The actions are either executed by the RS in automatic fashion, or the may require human intervention. A subset of the available actions is: [5]

1. Automated actions.
   (a) Do nothing (NoOp).
   (b) Reboot device (RB).
   (c) Perform non-destructive reimaging, saving all stored data (NDI).
   (d) Perform destructive reimaging, loosing all stored data (DI).
   (e) Update the device with new software (US).
2. Actions requiring human intervention
   (a) Call for Tier 1 repair person, who usually perform a scripted set of actions) (T1).
   (b) Call for Tier 2 repair person, who has extended knowledge about the system (T2).
   (c) Call for Tier 3 repair person, an expert able to fix virtually any problem (T3).

The actions above are listed in a specific order: first it is assumed that each repair action can resolve all problems that can be solved by prior actions. Second, the actions increase monotonically in cost. It is also assumed that a Tier 3 repair action (requiring human intervention) can resolve any problem.

Some additional remarks about the repair service are in order:

1. There is no one-to-one mapping between the event signals generated by monitoring agents and the actual underlying problems.
2. A single failure could trigger more than one event signal from the agents, but only one of these events will trigger a repair action.

of the equation. We ignore this factor in this paper with no consequence for the analysis presented.

[3]We describe the essential features of the automated repair system that are relevant to this paper. Readers interested in further details about this system are encouraged to consult [6].

[4]The process modifies the basic state-machine by adding states in the "middle" of transitions, encoding the repair events. There are other states in our actual state-machines, but describing them in this paper does not change the nature of the results.

[5]We note that the human intervention actions may generate, in turn, other actions. We ignore this complication in the paper.

3. Some of the automated repair actions may be executed more than once to have the appropriate effect (shown by N_i in Figure 2).

In this context, a policy is a function that takes as input an event from the monitoring agents, the state of the device, and a fragment of the device history (e.g., how many times a given repair action has been executed), and returns the entry point in the repair action ladder (Figure 2), and the number of times that successive repair actions will be executed.[6] The "policy manager" from Figure 1 triggers the repair action by starting the automated action on the device or filing a ticket request for human intervention.

Establishing the entry points for each event at the appropriate level on the repair ladder translates into minimized costs, maximized QoS, and increased availability. A proper policy can also eliminate false alarms from uncorrelated events caused by monitoring agents.

3. DATA AND ANALYSIS TOOL

The data in this study includes information from about 1600 devices over a period of two months. The information contains the transitions between states for each device. Each transition is time-stamped, and it is annotated with a numeric *event identifier*, which corresponds to the signal from the monitoring agent that triggered the state change.

We used Artemis [4] to analyze and visualize this data. Artemis is a toolkit for evaluating the performance of distributed systems; it offers an interactive workflow comprising four different tools: data collection, data storage, data visualization and data analysis. For this application we did not use the data collection service; instead, we used the logs from the monitoring system as input data. Artemis itself is oblivious to the semantics of the data, but it provides a plug-in mechanism, which allows the implementation of domain-specific data analyses.

The state transition data can be naturally visualized as a time-series: for each machine we can plot the current state as a function of time. For this domain, the main object manipulated by the Artemis plugins is the *trace fragment*: a sequence of consecutive transitions (represented as a string). The complete trace of states of a machine is a particular instance of a trace fragment. We have implemented two trace-specific Artemis plugins to analyze the data. The *trace statistics* plugin computes basic statistics for each trace fragment: the count of occurrences of each state and event, the time spent in each state, the normalized time spent in each state (as a percentage of the length of the trace), the average time spent in each state. The *regular expression* plugin is used to decompose a trace into fragments; it receives as argument a regular expression representing a string of states, written using the C# syntax. The plugin matches this regular expression with each trace in the data set, and produces a trace fragment for each matching instance of the regular expression. This plugin is also a very convenient interactive query engine that the analyst can use to slice and dice the data.

We use the *regular expression* plugin to extract information about the effect of repair actions. For example, to investigate the effectiveness of manual repair of a machine we use the following regular expression: `(H[^HDUR]*D)[^DHU]*(?<rep>H)`.[7] The sub-traces matching this expression all start in the healthy state, then eventually reach the down state, without intervening reboots or updates, and then move back to healthy. The results of the plug-in execution include the monitoring signals for each trace matching fragment, and the time that each device spent on each sub-trace. In this example, the first part of the regular expression in parenthesis will cause Artemis to generate statistics about the behavior of the devices prior to entering the down state.

4. EVALUATING REPAIR ACTIONS

For the purposes of this paper, we equate the effectiveness of an automated repair action on a given device with the time this device will remain available in the datacenter. There are simplifications in this assumption, yet, it is obvious that this factor is a prime component in the ultimate equation determining value.

We estimate $P(t_a|A)$: the probability that the device will remain available for a time at least t_a, given that the repair action A was performed on the device. To estimate this probability we borrow techniques from the field of *survival analysis*. We used the Kaplan-Meier estimator [7], defined as follows: given data with N samples with availability times $t_1 \leq t_2 \leq \ldots \leq t_n$

$$\hat{P}(t_a|A) = \Pi_{t_i<t} \frac{n_i - f_i}{n_i} \quad (1)$$

where n_i is the number of devices available just prior to time t_i, and f_i is the number of devices failing prior to t_i. External factors may truncate the device availability period by early termination (*censoring* in statistical terms). In this case, n_i is the number of machines available less the number of machines unavailable due to early termination. For example, in our case censoring is due to software updates or to problems in logging the data.

Figure 3 presents the graph of this probability function for one of the repair actions requiring human intervention (let's call this repair action HRA). The data t_i was extracted from the logs using the Artemis regular expression plugin (see Section 3). t_i are the lengths of the traces

[6]The number of times a successive action is executed may be zero, which effectively results in a subset of the actions being part of the final policy.

[7]The ?<rep> annotation in the last part indicates that the last healthy state can be reused as the start of a new sub-trace; in other words, trace fragments can overlap. The parentheses designate trace fragments that are of particular interest.

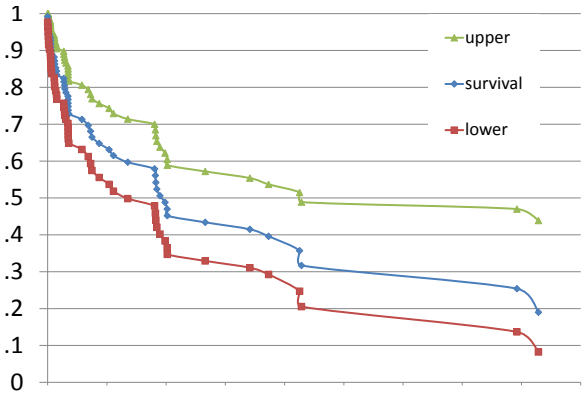

Figure 3: Probability that a machine will stay healthy at least to time t following a HRA repair action. The x axis is intentionally obscured. The *survival* line represents the actual estimated value, and the *upper* and *lower* lines depict the 95% confidence interval.

matching the regular expressions from the point the device reaches the healthy state until it reaches a fail state (after a HRA action). The curves above and below represent the 95% confidence interval for the estimate; we use the Greenwood formula to estimate the variance $\hat{V}$ and rely on the usual approximations for normality and standard deviations:

$$\hat{V}(\hat{P}(t_a|A)) = \hat{P}(t_a|A)^2 \sum_{t_a<t} \frac{f_i}{n_i(n_i - f_i)} \quad (2)$$

Let us use T to denote the particular "availability time" of interest after HRA, as displayed in Figure 3. Namely, performing HRA is effective only if the device remains available for a time $t \geq T$. Otherwise it makes sense to escalate to the next human intervention tier directly.

Let's assume hypothetically that only 40% of the devices survive longer than T. If we can use a predictor to distinguish these from the 60% that fail to meet the T time, we can improve the policy by bypassing the repair action to the next tier. The methodology for doing this is described in the next section.

5. REFINING THE POLICY

As explained at the end of the last section, given T as a reference, we intend to separate the class of devices that after the repair action HRA will fail to meet this "deadline" from those that will do. In order to influence the repair policy, we need to make this distinction when the policy intends to invoke HRA. Let us denote by D_0 the devices that survive *less* than T after HRA, and by D_1 those that survive *more*. For devices belonging to D_0 we will bypass HRA, since for all practical purposes this repair action is ineffective.

One way to accomplish this is to automatically build a *model* from the information about the device history; the model is used to determine whether the device is in D_0 or in D_1. A pattern classifier is one such model, as employed by statistical machine learning. In our case the information about the history of the device is comprised from the events emitted by the monitoring agents, and from the duration that the device spends in the healthy or probation state prior to the current failure. In our case this information is extracted using Artemis, and it corresponds to subtrace matching the first part of the regular expression `(H[^HDUR]*D)` (see Section 3), namely, the path from H to D.[8]

The classifier we experimented with in this work is based on logistic regression with $L1$ regularization. There are two main reasons for this choice. First, it is known that this approach is very effective at picking the most relevant signals when automatically building its model, even in cases when the number of possible signals is bigger than the number of samples [8]. Second, we have had very good experience with this classifier model in domains related to automated diagnosis of datacenter machines [2].

In our case, the first aspect is very relevant, since the number of events from the monitoring agents grows fast with the number of devices and trace length. Also, the number of devices requiring costly actions may be small, especially as the system becomes more and more reliable. In our particular case, we had on the order of 30 to 40 cases with interesting repair actions, and over 60 signals to select. As will be seen below, in the particular model we inferred, only 2 of these signals are good predictors, in addition to the time spent by the machine in the healthy or probation states.

The output of this classifier is a linear model of the form:

$$\mathcal{D} = \sum_i \beta_i \times S_i + \beta_0 \quad (3)$$

where the device belongs to D_1 if and only if $\mathcal{D} > 0$. The parameters of the model, namely β_i, are fitted from data, while S_i are the features used by the model. To estimate the quality of the model, we follow standard practices from statistics and estimate the *accuracy* of the model when identifying whether a device is in D_1 or in D_0 using ten-fold cross-validation.

Once the threshold T is determined, the model is automatically built and evaluated using the information in the logs. To close the loop, updating the policy, equation 3 is evaluated on line. The updates and roll-out to the policies can be done offline, following a suitable schedule.

6. RESULTS

We applied our classifier to the HRA for a period of 3 months, inducing an initial model. After an analysis of the model we diagnosed several problems in the service operation, which we handed to the operators. We

[8]What constitutes relevant prior history, namely how much of the prior trace should be considered, is a parameter that can be altered. As we show below, in our case this choice of history contains enough information to almost perfectly separate D_0 from D_1.

then collected data for 3 more months, and induced a new model. Here is a description of some of our significant findings.

The initial application of the classifier to the HRA, using threshold T, yielded a very accurate model: the accuracy of the prediction of D_1 is perfect, and the accuracy of D_0 is 96%. What drew our attention was that two signals (amongst the 60 possible) were enough to achieve that accuracy. Looking closely at the cases where those signals were involved, we saw that in fact these signals were robust predictors of failures. Very shortly after the HRA, these signals would reappear and cause the state of the machine to change to F or D. The service operators analyzed this data and determined that the specific sensor triggering one of these signals was not properly calibrated. Three months after calibrating the signal, the new induced model shows that the faulty sensor problem has been repaired. The new model accuracy is 88%, and uses more than 6 distinct signals.

The initial model also indicated that a repairable fault was consistently correlated with machines remaining unhealthy after intervention. With this information the operators determined that the actual repair procedure was incorrect. The repair action was changed, and the new models don't show a particular strong correlation between this signal and machines that do not reach the healthy state after HRA.

Finally, we have used the estimator to determine whether there is a quality difference between the 4 datacenters hosting the devices. We noticed significant differences, and we are currently trying to find the root cause, differences in hardware being the prime suspect. The main technique used for this study was the comparison of survival functions using contingency tables [5].

7. DISCUSSION AND FUTURE WORK

There are many immediate paths that we intend to follow. First, we intend to test the limits of the models as we collect more data. Second, we consider many ways to improve the models in Sections 4 and 5. These include more sophisticated models for survival (and further partitioning the cases by particular events), looking at the order in which the sequence of signals is received, including event signals as part of the information, increasing the past history window, and looking beyond classifiers as models. Third, we continue to evaluate the implementation of our model's recommendation and then verify that indeed the service availability is increased (and cost is decreased) in a continuously running production setting. Fourth,we plan to implement a more general approach that includes preventive maintenance and repair schedules in the optimization loop.

We would like to note that this approach does not depend in any way on the particulars of the Artemis tool, or on details of the cloud service under investigation. Artemis was used as a convenient tool for visualization and manipulation of the data, and the statistic techniques of pattern classification and survival analysis are clearly applicable to other cloud computing platforms running different services.

The repair system under investigation has been manually optimized since its inception for a period of over six months, prior to our optimization efforts. It is a testimony to the power of the framework and techniques we are proposing in this paper that we managed to help in finding false alarms, and mistakes in the repair procedures.

8. REFERENCES

[1] L. A. Barroso and U. Holzle. *The Datacenter as a Computer — an introduction to the design of warehouse-scale machines*. Synthesis Series on Computer Architecture. Morgan and Claypool, 2009.

[2] P. Bodík, M. Goldszmidt, and A. Fox. HiLighter: Automatically building robust signatures of performance behavior for small- and large-scale systems. In *SysML*, 2008.

[3] G. Candea, J. Cutler, and A. Fox. Improving availability with recursive microreboots: a soft-state system case study. *Performance Evaluation*, 56(1-4):213–248, 2004.

[4] G. F. Creţu-Ciocârlie, M. Budiu, and M. Goldszmidt. Hunting for problems with Artemis. In *USENIX Workshop on the Analysis of System Logs (WASL)*, San Diego, CA, December 7 2008.

[5] D. Diez. Survival analysis in R. http://www.stat.ucla.edu/david/teac/surv/R_survival.pdf.

[6] M. Isard. Autopilot: Automatic data center management. *Operating Systems Review*, 41:60–67, 2007.

[7] E. L. Kaplan and P. Meier. Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53:457–481, 1958.

[8] K. Koh, S.-J. Kim, and S. Boyd. An interior-point method for large-scale L1-regularized logistic regression. *Journal of Machine Learning Research*, 8:1519–1555, 2007.

[9] D. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupamn, and N. Treuhaft. Recovery oriented computing (ROC): Motivation, definition, techniques, and case studies. Technical report, UC Berkeley, March 2002.