AUTO-RECOVERY OF MICRO-SERVICES IN A LARGE DISTRIBUTED APPLIANCE MODEL

Kshitij Tanay
Raghu Rajendra Arur
Taj Uddin

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation
Tanay, Kshitij; Arur, Raghu Rajendra; and Uddin, Taj, 'AUTO-RECOVERY OF MICRO-SERVICES IN A LARGE DISTRIBUTED APPLIANCE MODEL', Technical Disclosure Commons, (February 05, 2019)
https://www.tdcommons.org/dpubs_series/1929
AUTO-RECOVERY OF MICRO-SERVICES IN A LARGE DISTRIBUTED APPLIANCE MODEL

AUTHORS:

Kshitij Tanay
Raghu Rajendra Arur
Taj Uddin

ABSTRACT

In large distributed application-based systems, an ensemble of co-existing micro-services perform individual functions and communicate with other micro-services. As these interactions and individual behaviors can be complex, it may be necessary to monitor micro-services for any abnormalities. Manual monitoring of all such micro-services, or establishing individual rule-based alerts for micro-services can be time-consuming. In addition to maintaining functionality of micro-services, it is important to prevent micro-services from becoming a performance bottleneck for other micro-services. In such situations, an alert may not suffice. For example, if a micro-service serves as a data store for all other services, a bottleneck at that micro-service may affect an entire system. By performing a predictive analysis of the health of micro-services, large distributed application-based systems can incorporate either preventive or reactive healing, thereby becoming auto-corrective in nature.

DETAILED DESCRIPTION

With the increase in complexity of applications, distributed applications and micro-service-based architecture has become common. Typically, a majority of the deployment of these applications is cloud-based, with administrators provided with access to dashboards in order to monitor performance. When performance issues occur, administrators can take actions to address the identified problems. However, these systems are typically rule-based, requiring users to program tools for alert generation.
Alert-based systems can be useful in hosted applications when administrators are monitoring the system and available to take corrective action. However, in appliance-based vendor-provided solutions, no active monitoring may occur. Even if performance statistics are provided, network administrators may be unable to address some problems. Therefore, it is desirable to provide appliance-based systems with auto-corrective functionality. Since many modern micro-service applications are combinations of open-source and custom-built services, generic rule-based approaches that cater to all the services are unlikely to be tenable. Present embodiments provide a health monitoring system that is auto-corrective in nature by learning correlations between dependent entities, as well as learning how to take corrective actions.

Present embodiments may utilize a time based probe, known as a service heartbeat, and may monitor service health statistics in order to deduce service degradation or abnormalities. Unlike conventional approaches, corrective actions can often be performed without necessitating a service restart. Furthermore, present embodiments may monitor health at a process or service level, rather than at a container level. By running clustering and correlation algorithms, present embodiments can come up with minimalistic, yet sufficient, groups of service metrics that uniquely identify symptoms. The corresponding corrective actions for each symptom is learned using reinforcement learning. Next, the corrective actions and the subsequent service state matrix is learned for each service. Corrective actions are moderated using the matrix to restore a service to a normal state. Using clustering and correlation algorithms, present embodiments may designate tightly-coupled micro-service clusters, and monitor the health of each such micro-service clusters. Graphical representations can be used to convey information regarding the inter-service dependencies to an end user.

Figure 1 is an example of a system architecture in accordance with a present embodiment.
The various components include one or more applications and/or micro-services, an auto-recovery engine (also known as a health monitoring system or HMS), and a policy enforcer. An application includes an ensemble of micro-services working in tandem to provide a unified functionality to the end user. From an application’s lifecycle point of view, landmark events may include registering with the HMS, updating service subscription requests, and deregistering with the HMS. Each application whose health is monitored announces itself to HMS along with a service list containing metadata about the services, which is also known as the service subscription request, or SSR. The SSR may be updated periodically. Deregistering with the HMS may occur when an application is uninstalled or deleted.

Figure 2 depicts an example of a micro-service with autonomic service healing.
From a service's lifecycle point of view, each service starts in a “normal” state. When a service starts, it registers for health monitoring and throughout its lifecycle sends a “heartbeat” containing metrics to the HMS. The HMS monitors the heartbeat and metrics of the service and, in response to detection of an abnormality, transitions a service to an “anomalous” state. When a service is in an anomalous state, the HMS incorporates preventive healing in order to restore normal functionality to the service. A service already in an anomalous state can further transition to a “diseased” state if the preventive actions and recommendations were unsuccessful. In response to a service entering a diseased state, the HMS may perform reactive healing in order to restore functionality to the service.

The HMS may include several components, such as a data store, representational state transfer (REST) application program interface (API) service, diagnostics engine, and rule engine. The data store may store relevant information received from services. In addition to service metrics and heartbeat information, the data store may also store internal information required to run predictive algorithms on the collected data. The HMS exposes certain REST APIs for the end entities (e.g., services, a policy enforcer, etc.) to communicate with the HMS. The diagnostics engine determines the health of a service. As part of the application and service registration, the following information is conveyed to HMS: a list of service metrics to be monitored using a rule-based approach, a list of service metrics to be monitored using a trend-based approach, a list of service metrics with no indication of a monitoring approach.
For each metric in question, a threshold is also provided to the HMS. The rule engine determines abnormal behavior for services using a rule-based approach. Each time the HMS receives metrics from a service, the rule engine may compare the metrics to thresholds to determine whether the metrics are acceptable or not. Once metrics are collected for each service over a period of time, the rule engine may determine if a given service has exhibited anomalous behavior based on the threshold comparisons. If a service is determined to have exhibited anomalous behavior a sufficient number of times within a time period, then the HMS performs one or more preventive healing procedures to attempt to restore service functionality and prevent further service degradation. It communicates with a corrective actions and recommendations engine (CARE) for initiating preventive healing. Using the trend engine, the HMS analyzes the time series data for each metric for each service and utilizes approaches such as an Exponential Weighted Average or a Hidden Markov Model (HMM) to determine if an abnormality has occurred. Once the trend engine detects an abnormality, the response of the trend engine is similar to the rule engine.

When no monitoring approach is indicated for a service, the HMS may run clustering algorithms to generate minimalistic, yet sufficient, groups of service metrics that uniquely identify a symptom. The corresponding corrective actions for each symptom are learned using reinforcement learning. Figure 3 depicts an example of statistics collected from a micro-service.
As indicated, a number of statistics are collected for a service, a covariance matrix is constructed, and Pearson correlation coefficient (PCC) values are calculated for each $S(i,j)$. The clustering algorithm then sets 0.75 as the threshold for $|\text{Abs} \{\text{PCC} (i,j)\}$, and then creates service statistic cluster $S_2$, $S_4$, $S_6$. Thus, it can be inferred that the services $S_2$, $S_4$, and $S_6$ are tightly coupled. Present embodiments may use this approach for forming microservice clusters. For example, when an anomaly is observed in service $S_2$, actions may be automatically taken to prevent degradation of services $S_4$ and/or $S_6$.

The scheduler component of the HMS is used to execute tasks in a periodic manner. Tasks that are performed periodically may include heartbeat monitoring, which check to determine that a service is alive, and statistics monitoring, which monitors for degradation. The scheduler may run clustering algorithms to identify clusters for statistics monitoring. Population of databases with information such as feedback information and system information may be scheduled. Furthermore, the CARE may be periodically sent notifications in order to generate corrective actions.
Figure 4 depicts an example of the CARE architecture in accordance with present embodiments.

Each time a service is deemed to be either in an anomalous or diseased state, a snapshot of the service metrics and heartbeat trends is captured and sent to the CARE by the scheduler. The CARE then determines the symptoms that best classify the service state. Depending upon the state of the service, the CARE triggers preventative or reactive healing procedures.

Figure 5 depicts an example of states of a service during reinforcement learning in accordance with present embodiments.

When a node is decommissioned, the HMS may observe changes in metric trends. The HMS may then flag an anomaly (e.g., as indicated by state 2 in Figure 5), and the CARE may be provided with information to deduce corrective actions that can be performed with regard to the node, as well as any other services that may be tightly coupled with the node’s performance. The CARE may then perform one or more corrective actions,
and monitor services for a change in state. The CARE may determine that the service has moved to the diseased state (as indicated by state 3 of Figure 5), which is an even more unfavorable move. Thus, the corrective action that was performed may be associated with a penalty corresponding to a negative value. The CARE may execute another corrective action, and monitor the service for a change in state. The CARE may determine that the service has moved to a normal state (as indicated by state 4 of Figure 5), which is a favorable move. Thus, the corrective action that was performed may be associated with an award corresponding to a positive value. However, if the move from state 1 to state 4 has an overall negative score (e.g., award value minus penalty value), then it may not be the best route.

Figure 6 depicts an example of states of a service during reinforcement learning in accordance with present embodiments.

![Diagram of service states]

**Figure 6**

Another node may be decommissioned, and the HMS may begin to observe the change in trends of the service’s metrics. The HMS may flag an anomaly, and the CARE may be supplied with information to deduce one or more corrective actions pertaining to the service, and if applicable, for other services that are coupled to the service’s performance. The CARE, having been provided with the cost of the corrective action (e.g., from the previous example discussed with reference to Figure 5), may execute a different corrective action and monitor the change in state (as indicated in Figure 6 by the change from state 2 to state 5). The CARE may detect that the service has moved to another anomalous state, and may execute yet another corrective action, thus restoring the service to a normal state (as indicated in Figure 6 by the change from state 5 to state 4). Since state
4 is a normal state, the CARE may update the reward/penalty value for the corrective actions that were performed. Thus, in the future, one or more corrective actions may be selected that minimizes penalties and maximizes rewards.

Figures 7A and 7B depict examples of states of a service during reinforcement learning in accordance with present embodiments.

![Figure 7A](image1)
![Figure 7B](image2)

In Figure 7A, corrective actions are causing a loop in which a service moves from one anomalous state to another. Likewise, in Figure 7B, corrective actions are causing a loop in which a service moves from one diseased state to another. If corrective actions result in a loop, these corrective actions may be pruned from a rule set, and in some embodiments, no further corrective actions may be applied.

A rule set, or root cause set, serves as a map between symptoms and corrective actions. This mapping is determined by reinforcement learning. The input to the rule book is either provided by an administrator, or the CARE may pre-populate corrective actions in the rule book for well-known open-source infraservices. For custom built services, such corrective actions may be provided by an end user. If no such input is available, restarting a service may be taken as the default corrective action. Another action that may be performed as part of the preventive healing process may be to automatically provision more of a particular resource to a service that is constrained for the resource.

Another critical component of the CARE is to record and learn from its proposed actions and recommendations. As described above, the diagnostics engine labels a service as either anomalous or diseased. A service may be labeled as anomalous if there is a
degradation in the performance, and a service may be labeled diseased if there is a severe performance degradation that affects other services, or the service itself is inactive. The feedback engine may use reinforcement learning to monitor the services that were labeled either anomalous or diseased. The information about anomalous or diseased services is stored, and based on the symptoms, appropriate corrective actions are executed. Once a corrective action is taken, the CARE monitors the state of the service. In some embodiments, the longer that a service remains stable, the greater the reward that is associated with that corrective actions. However, if a service were to re-enter an anomalous or diseased state, the corrective action is correspondingly penalized.

The CARE may thus establish intelligent modifications to actions and recommendations by constructing a weighted service action and state transition directed acyclic graph (DAG). A DAG consists of nodes and edges. Figure 8 depicts a DAG in accordance with present embodiments.

![Figure 8](https://www.tdcommons.org/dpubs_series/1929)

The initial and final nodes of the DAG represent a service in a normal state. All other nodes represent a service in an anomalous or diseased state. A directed edge represents either the trigger or action that causes a service transition from previous state to current state. For the DAG depicted in Figure 8, nodes 1 and 7 represent initial and final
normal states of a service. Nodes 2, 3, and 5 represent anomalous state, and nodes 4 and 6 represent diseased states. A corrective action CA#XY may refer to a corrective action taken from node X to node Y, and a trigger action Tr#XY may likewise indicate a trigger that causes a service to move from node X to node Y. A trigger may only originate from the initial node. For example, in the above DAG, Tr#12, Tr#13 and Tr#15 are the triggers.

Figure 9 depicts an example of reactive healing in accordance with present embodiments.

![Diagram of nodes 1, 2, and 3]

**Figure 9**

As depicted, a Tr#12 causes a service to move to diseased state. CA#23 was able to restore the normalcy. Tr#12 may be an “OutofMemory Crash” trigger, and CA#23 may be a “Restart service” corrective action.

Consider an application that has micro-services A, B, C, D, and E. Service B may be a REST API gateway, and service C may communicate with services D and E for processing flows. After generating a covariance matrix and applying clustering algorithm, the CARE may establish that services A and B belong to one cluster, and services C, D, and E belong to another cluster. In an example scenario, service A may experience an anomaly due to high search and response time. Given this information and the determined service clusters, we now know that service B may also experience service degradation, such as increased query response times, inability to render information on a GUI, etc. Therefore, the CARE may perform a corrective action in which the connection timeout value is adjusted when establishing a connection with A. This corrective action may help in maintaining the availability of B, and the GUI may be able to fetch a response from B instead of receiving errors as output. Additionally, a corrective action may be performed on service A, and the state of service A may be monitored. If the corrective action is not fully effective and the service remains in an anomalous state or moves to a diseased state, reactive healing is sought, and corrective actions are further applied to return the service to an operational state.
The HMS relays the actions and recommendations received from the CARE to a policy enforcer. The policy enforcer may execute corrective actions or share the recommendations with an end user. The policy enforcer may consult a policy database to determine whether a given Corrective action is in violation of a service policy agreement. In such cases where a violation is found, the action may be omitted in any recommendations to an end user. For example, a policy database may have a hard limit on a number of restarts for a service within a selected time interval. In such a case, the action of restarting a service may not be executed.