To what degree can tools and automation help us to recover from system faults?

There are a variety of reasons that can make it difficult for the human to control the system while using it. Developing an autonomic system that can self-control self-recovery has increased in recent years. However, the software may fail in different ways. It might be even malfunctioning to start the self-healing function.

Google has designed a large scale distributed system tracing infrastructure called Dapper. It is such an infrastructure that can generate a lot of useful information without application involvement. This is designed because the system is continuously developed by different teams meaning "no one can be an expert in the internals of all of them [1]".

The paper written by Amjad A. Hudaib has addressed that Software Fault Detection and Recovery (SFDR) method detects the cases of a fault occurs with software components such as component deletion, replacement or modification [2]. It can automatically diagnose monitor and recover the fault component in the target software. The application of this method has been implemented to demonstrate to be better than the other industry approaches such as Microsoft windows restore, ASURE in the case of recover error result from deleting software components and fault recovery.

Research has shown that there are many ways that we can recover the system fault even at a very low level, such as using machine learning in the assembly levels. In this paper written by L. Seabra Lopes, they addressed that through the use of machine learning techniques, the supervision architecture will be given capabilities for improving its performance in the unforeseen event [3]. Through training the machine learning model using the raw sensor data. An error classification algorithm is subsequently generated by induction algorithm SKIL. This approach has been tested on an industry system on a robot called SCARA to prove its fault recoverability.

Furthermore, a system can be a self-fault tolerant system itself by using big data [4]. By using the technology related to the Internet of Things such as wireless sensor and cloud computing. We can make the fault diagnosis being data driven instead of knowledge-driven which means the full automation testing can be achieved driven by big data. In this paper [4]. it has been stated that by using deep learning network such as hierarchical diagnosis network or deep belief learning (DBN) to extra the feature from the data can help us to gain the insight view into the reason of the fault. It also has a range of industrial applications such as electrocardiogram fault diagnosis system.

References

[1] Sigelman, **Dapper, A Large Scale Distributed Systems Tracing Infrastructure** https://blog.acolyer.org/2015/10/06/dapper-a-large-scale-distributed-systems-tracing-infrastructure/ *Date Accessed: 16 February 2019*

[2] A. Hudaib and H. Fakhouri, **An Automated Approach for Software Fault Detection and Recovery** https://www.scirp.org/journal/PaperInformation.aspx?PaperID=69412 *Date Accessed: 15 February 2019*

[3] L. Seabra Lopes and L. M. Camarinha-Matos, **A machine learning approach to error detection and recovery in assembly** https://ieeexplore.ieee.org/document/525884 *Date Accessed: 15 February 2019*

[4] Y. Xu, Y. Sun, J. Wan, X. Liu and Z. Song, **Industrial Big Data for Fault Diagnosis: Taxonomy, Review, and Applications** https://ieeexplore.ieee.org/document/7990488 *Date Accessed: 16 February 2019*

Problem: It takes too long time to figure out what is wrong when system goes down.

Suggestion 1: Using automatic management and provisioning

Automatic provisioning and installation.
- Provisioning and installation manually will be costly and error-prone.
- Small configuration differences make problem determination much more difficult.

Configuration changes made in production must produce an audit log record.
- Frequently scan all servers to ensure their current state matches the intended state.
- Helps catch install and configuration failures, detects server misconfigurations early, and finds non-audited server configuration changes.

Recover at the service level.
- Handle failures and correct errors at the service level, not in lower software levels.
- For example, build redundancy into the service rather than depending upon recovery at the lower software layer.

Fail services regularly.
- Take down data centers, shut down racks, and power off servers.
- Regular controlled brown-outs will go a long way to exposing service, system, and network weaknesses.
- Without production testing, recovery will not work when called upon.

Problem: The dev team only found system down after customers called the CEO directly.

Suggestion 2: Using auditing, monitoring and alerting

Configurable logging.
- Support configurable logging that can optionally be turned on or off as needed to debug issues.
- Having to deploy new builds with extra monitoring during a failure is very dangerous.

Track all fault tolerance mechanisms.
- Track every time a retry happens, or a piece of data is copied from one place to another, or a machine is rebooted or a service restarted.
- Avoid fault tolerance hiding little failures, minimize possibility for them to become big failures.

Audit all operations, keep historical data and record all significant actions.
- Every time somebody does something, especially something significant, log it.
- Historical performance and log data is necessary for trending and problem diagnosis.
- Having important action record helps immensely in debugging problems.