UDC 004.75

FAULT TOLERANCE REDUNDANCY METHODS FOR IOT DEVICES

DOI 10 36994 / 2788- 5518- 2022-02-04-06

Дмитренко О.А*. Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна. Olexandra.Dmytrenko@gmail.com*

Скулиш М.А*., д.т.н., проф, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна. mskulysh@gmail.com*

Abstract: *IoT technologies are emerging with high rate. Fault tolerance influences not only the general quality of work in a conglomeration of IoT devices, but also the security of the system. It might also affect the speed of communication. This paper describes how error detection can happen. Several methods of increasing an IoT system's fault tolerance are described and compared, showing their pluses and minuses, peculiarities of application. The methods are cold, warm and hot standby, fault tolerance event manager, forward and backward error correction. The ways of how common address and virtual routing redundancy can help with performing failover and increase the guarantees of the system stability are also described*

Key words: *IoT devices, fault tolerance, faults classification, redundancy algorithms, failover*

МЕТОДИ ВІДМОВОСТІЙКОСТІ РЕЗЕРВУВАННЯ ПРИСТРОЇВ IOT

Olexandra Dmitrenko *.National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute",Kyiv, Ukraine. Olexandra.Dmytrenko@gmail.co*

Marija Skulysh, *Dr.habil., Prof. National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute",Kyiv, Ukraine. . mskulysh@gmail.com*

Анотація: *IoT (Internet of Things) технології - це невеличкі прилади, які можуть підєднуватись до інтернету та часто містять сенсори, приводи (виконавчі механізми), або все разом. Розумний годинник з купою сенсорів, розумна лампа, яку можна увімкнути чи вимкнути віддалено, система зрошення рослин у тропічній теплиці, яка вмикається при занизькій вологості та передає інформацію в кінцевому рахунку в мобільний телефон щоб допомогти контролювати стан теплиці, це все приклади IoT приладів.*

Таких приладів стає все більше з кожним днем, та їх починають монтувати майже у всю техніку як в побуті, так і на підприємствах та фабриках. Відповідальність цих пристроїв не завжди критична, але є сфери, де помилки чи недостача даних з цих пристроїв є критичною або для правильності роботи алгоритмів прийняття рішень, або ж для машинного навчання та налаштування штучного інтелекту. Одна з великих ідей, яка стоїть поза застосуванням цих технологій всюди, це ідея створення розумного міста, де не треба буде персонально намагатись людям всюди встигнути та все доглянути, але догляд за по суті якістю життя містян відбуватиметься автоматично.

В даній статті наводяться особливості приладів інтернету речей, які відрізняють їх від загально відомого клієнту в клієнт-сервісному застосунку. Це маленька енергоємність, віддалене розташування від серверів чи хмари, куди завантажується інформація, а також головне, що стосується статті, це необхідність в безперервному надаванні даних а також часта неможливість повторити надсилання тих самих даних повторно.

Засновуючись на цих відмінностях, а також маючи на увазі поступовий, а не різкий спад активності застосунку заради забезпечення максимальної якості його роботи, проводиться аналіз методів надлишковості, а саме холодного, теплого та гарячого очікування, протоколу спільної надлишкової адреси та протоколу надлишковості віртуального роутингу. Також, аналізуються типи помилок та можливість зрозуміти справжню причину помилки. В статті також зазначається спосіб активації додаткового обладнання та метод слідкування за його роботою.

Метою статті є показати, які методи надлишковості придатні для роботи саме з IoT приладами.

Ключові слова: *IoT прилади, відмовостійкість, класифікація помилок, надлишкові алгоритми, відновлення системи після відмови.*

Introduction

IoT devices are the computing devices that by connecting to the Internet can exchange information with the other similar devices or connected managing applications. We can find them in smart TVs, lamps, washing machines and many other devices that compose a popular now smart home. They can also be found in enterprises. They may help in organizing work activities such as meetings, comfortable temperature in a building depending on amount of people or booking an available room. In order to facilitate device control and maintenances in the process of production in factories on assembly lines, IoT devices are also used.

For example, they can report on the need to replace some details that are getting warn out. If to adopt a robot on such an enterprise with the API to collect this information, this can eliminate need in people's involvement in production process, because the robot will be able to take the present details and repair the apparatus. Adding to this, if the process of details' ordering is standardized, no human involvement into the basic assembly line needs will be needed. This can be very helpful on the harmful production.

Nowadays, these edge devices gain much more popularity on the daily basics. A lot has to do with the idea of smart city that gains its popularity (fig. 1). The IoT devices get widespread in the agricultural sphere. There they allow easily adjust to the changes in weather and control the state of the soil and plants, turn on watering and report on the microelements and fertile that has to be added to soil for better plant growth.

Having major things depending on the quality of work of small microcontrollers reporting their information for the future analysis and decision making, it is getting hard to lose some data. For instance, if there are two channels for the information: one for the standard status and another for warning and error messages as it is described in *fault tolerance event manager* practice [1], and one channel stops sending messages. This might cause major issues in perception of the real state of things and cause wrong behavior and decision making. IoT devices won't get correct directives and will function either in the way they were programmed or in some default way. Speaking about agriculture, watering can happen during the maximum sun activity which might kill the plants.

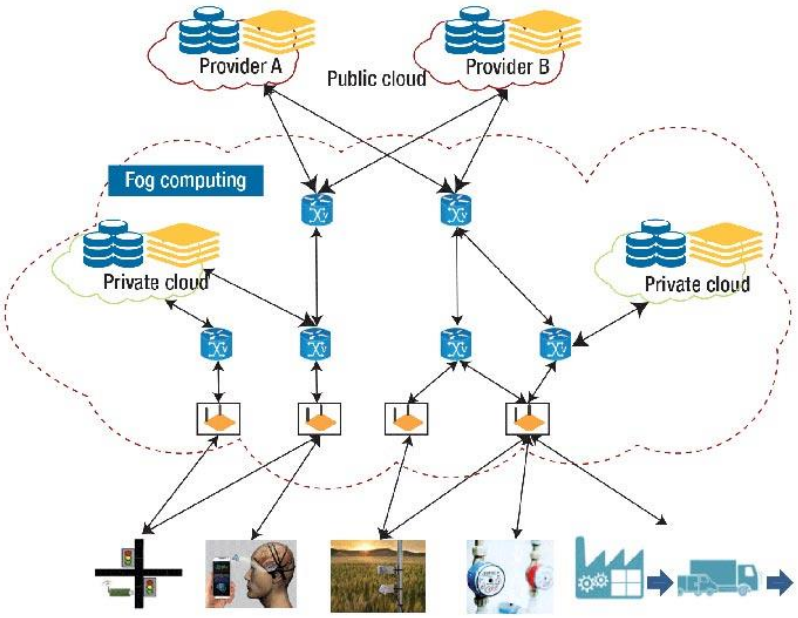

Fig.1. Distributed Systems and IOT. Source: https://www.icar.cnr.it/en/sistemi-distribuiti-e-internet-delle-cose/

This brings us to the need for *fault tolerance* [2]. Its idea is to let the system function correctly even if a fault or several took place or in the worst case *gracefully degrade*. The last means that the fall of part of the system will be obviously felt, but this won't prevent all the other parts of the system function correctly until the lacking part or its mirror is up. The main idea of this paper is to compare IoT devices with the standard clients in server-client applications, and based on the differences to propose solutions that will maximize the chances of receiving correct data on time.

Research

Peculiarities of IoT Devices Compared to Standard Clients

The question of fault tolerance is being on for many years and for sure there are many solutions on how things can be adjusted. Their present solutions were tested on systems with standard client-server applications. The question is how IoT devices are different from any other network device connected to the network.

One of the practical things is the small amount of energy being consumed during their work. It is very handy taking into consideration the sphere of application of edge devices. At the same time, having no electricity and/or no Wi-Fi, no data will be able to be sent. In order to prevent a failure of this basic level, an additional source of energy might be supplied, solar batteries and sufficient accumulators may be also placed to feed the electrical needs of a device.

Another peculiar quality is that the edge devices are often located at the distance from the reporting location. This means that there might be a need for a wired connection to the internet or for a connection to a local network through Wi-Fi. In case the connection breaks but the devices work properly, the third solution is to organize a moving connection that will come close to the devices several times a day and either read the data from the devices transmitting it further or give chance to the devices to connect to the Internet by themselves. In the case of using this technique, an IoT device has to be equipped with the proper amount of storage that will be cleaned only after the data gets transmitted.

An example of such a device is a smart ringer that records and stores all the suspicious behaviour it notices around the place. It has memory and a connection to the cloud where it transmits all the recordings. But when the connection is lost, the backup plan is to record the information on an inserted memory card and transmit it whenever the connection appears.

A different solution for having the Internet in distant places is SIM cards. As long as the carrier supports the connection in the location and the device supports SIM cards, it's possible to furnish every device with its card and be sure that if no data is received, the reason is not the network, but the malfunctioning of the device. Modern smartwatches support virtual SIM cards, so there is no need to sync the sleep information with the phone they are connected to, but sync it directly to the cloud.

The third distinguishing point of IoT devices is the regularity of the messages they send. There are many devices for measuring characteristics all the time, e.g., humidity, or recording 4k video on the streets and transmitting it to the police station. When it comes to bank robbery, which is a rare case yet possible, it's of high importance that it is discovered at its start, not when the money is already stolen and people are injured. In such institutions, it's better to have replicated devices that will preserve from a great disaster from happening.

If a device is simple and has no memory inside, which is a common case, failure to send the data results in data loss. This can break not only a single decision made based on not complete data, but ruin the system of machine learning if such is present. The process of teaching artificial intelligence is important, time and resource-consuming, so better not to repeat several times. This means that it's better to have guarantees of maximum correct data that will result in correct algorithms for future decision-making.

Below there are several methods describing how to better organize the structure of IoT devices, data from which is of high importance.

Basic Structure of IoT System and its Needs

IoT devices can communicate with each other or with the gateway, which secures the connection in between the device and the cloud or server that collects the data from all the sources and permits a connected system to make decisions based on the collected information and respond with the directives (fig. 2). Taking into consideration that some devices produce loads of information, for example, the moving of particles in an electron collider is registered every millisecond, it is getting too complicated to transmit the data as soon as it appears.

That is why there some devices perform bandwidth optimization - have integrated data processing capabilities that minimize the amount of data that has to be sent to the data centres by picking only the most important data. This technique is called *edge computing*. For example, this can be done by sending the new data only when an update happens or by sending delta in changes. This technique improves not only security footprint, taking into consideration that fewer data has to be encrypted and transferred over the network. It also reduces response time because data doesn't have to travel to and from a remote location for processing [4].

For now, there are still many unsolved issues with the security set-up. In this article, the accent is made on describing the faults that may happen in the system and possible ways how to handle them are shown.

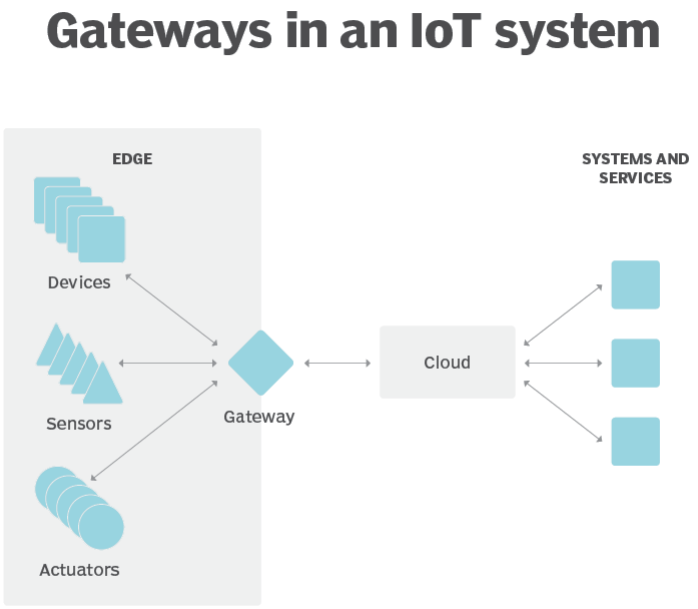

Fig.2. IoT device and Cloud management organization.

Source: https://www.techtarget.com/searchnetworking/definition/edge-device

The Faults Classification

In order to understand a cause of an error, it is important to understand the origin of an issue. If an issue is seen on a server, it could have originated not necessarily there, but possibly on any server dependency. Looking on Image 2 it can be said that if a system element fails to do its work correctly, the reason for it could be erroneous data on the cloud that could be caused by problems with the gateway or by issues with any edge or IoT devices. So basically, anything could cause an error. Below, the onion fault model is described. It contains the basic classes of faults.

The onion fault model presents the basic faults division into five main classes (fig.3). The structure describes that any upper faults reasons include any reason from the bottom layer. For example, a cause of an arbitrary fault can be a crash of a server that belongs to the fail-stop level [3].

- *Arbitrary* faults are the faults that happen once in a while, but other than that, a server works correctly.
- *Timing* - a server fails to respond in a given amount of time, but the late response would be correct.
- Omission fault is when a server doesn't respond or accept requests.
- Crash or fail-silent model of fault is when a server suddenly stops working, and it is not clear to all the working components. In synchronous systems, such faults can be detected by long timeouts.
- Fail-Stop faults happen when a server halts in the process of task execution and this gets known by relative working services.

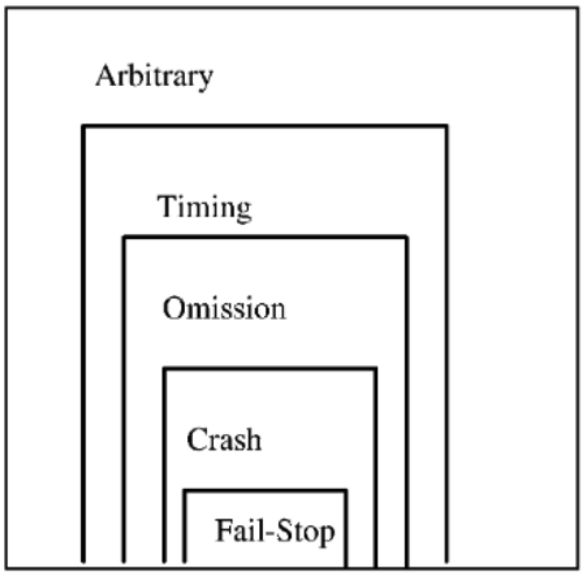

Fig.3. Onion fault model

To sum this all up, it's important to mention that at every stage there should be some guarantee that the faults won't happen because of 'stupid' reasons. If it is possible to perfect an important system with additional guarantees that a simple occasion will not destroy all the algorithms and analysis, it should be done. Below, I study the methods that may be used to prevent a basic failure and adapt them to varieties of IoT devices giving examples.

Adding Redundancy to the System

Failures can happen in any system at any moment. It's about being ready for it. Implementing a standby system might be very useful. This means that additional elements like database, servers, power or any other component will be in a ready state. In case something fails, there are other parts ready to pick up the work.

When picking a location for the standby elements, many factors should be considered. If that is a server or a database, a probability of an earthquake, war or food may be considered. This means that on one hand, it is politically more secure not to reveal user data through basing databases in other countries, but on the other hand, the probability of physical corruption should be small enough to pose a server in a nearby location.

There are several kinds of redundancy methods that may be implemented over the system depending on its needs. The methods are cold, warm and hot standbys.

Cold Standby

This is a redundancy method when another system that duplicated the running one, is installed and started once in order to be activated. After all the setup activities were done, the system is turned off and is started only in case of failure of the current working system [4].

This technique can be used in systems that either is not designed to give momentary responses to users or in the ones that don't incorporate stored data. For such systems, other standbys can fit better.

As an example of a cold standby, can serve an autonomous electricity generator that doesn't work without any need. If a sudden issue happens with the power station, so that it can no longer provide electricity, an electricity generator will turn on automatically and continue the work of the system.

Warm Standby

This technique is different from the cold standby by having the running duplicating system in parallel that time-to-time syncs data with the primary system so that in case of its fallback it will be able to pick up work and continue performing it, but only from the last back up time. If mirroring of the system is

done often, it's good for backup reliability, but the cost of mirroring should also be taken in consideration. It might be performed only at times when the system itself is not overloaded [5].

Distributed or NoSQL databases can serve as an example. Usually, applications are written in a way that servers are stateless and all the information is read from the config files or from the database. As long as replicas of the database contain the latest information, there is no issue to continuing full work from the moment and when the issue with the primary system is solved, sync all the data on the main storage.

Hot Standby

This technique is peculiar by having a duplicating system running all the time, simultaneously with the primary system. All the data is mirrored in real-time. Because of this, it is also called 'hot spare'. This is useful for systems with a high chance of failure or with a need to restart from time to time. Such a system provides not only graceful degradation but a switch on the supplementary system might not be felt by many users. Yet, often such systems are implemented in a way that when a secondary system works, it gives responses only on read-only activities. Writes are configured only on primary system which is never off for a long time [6]. A message about temporary write unavailability may be posted or the right requests might be collected in a que and performed as soon as the initial system is on and running.

Examples of such a system can be easily found in enterprise networks, as long as a second turned-on printer can serve as a replication of a primary one and perform all its activities if the user redirects the request. In a similar way, any IoT device can be replicating a base one and be ready to perform whenever it is requested by a managing system.

Hot standby reminds load balancing a lot. But is it really it? Load balancing is a technique of having two or more servers that can process the same requests. Ideally is when all the amount of the requests is divided evenly in between all the servers, so that the same number of requests will go for every server and none is overloaded [7]. Yet, this technique is not needed for IoT devices primarily because they are the source of the information and accept data only in the form of their configurations with no need to spend sufficient time on its processing.

Common Address Redundancy Protocol

The idea of a common address redundancy protocol is to share common IP address and a common virtual ID in between multiple hosts in same network segment. This conglomeration of hosts is called redundancy group. One host is a master host and it owns the IP address. Other hosts are ready to pick up if the master is not responding [8].

Virtual Router Redundancy Protocol (VRRP)

This technique is useful for cases when the physical connection with a device can break and this device will be (temporary) replaced with another having its own address. In order to eliminate issues with those, a virtual address can be used that will connect all the addressees with a working element through a virtual address, while a real one will be known only to a master router that manages all the connections. It has to be mentioned that the master router is also backed up by a group of other routers ready to take his job if it fails.

Remote measuring of water level can serve as an example of a group of devices where only one takes an action depending on its state. Several IoT devices can be set up on different height in a channel. If one of the sensors changes its state, e.g., it was flooded but now it gets not, the level of water has fallen below required and a command to open dam's gates will be sent. When water reaches another sensor, using the same IP address it sends another command which is to close the gates. There is no need for those devices to have different identifiers because they do the same function.

How to Initiate a Failover

In order for any redundancy technique to take place, in servers and IoT world the technique of heartbeats usually takes place. Working equipment sends its heartbeats with a defined regularity to a replica or to the management server. In case of any change in a regularity of the heartbeats, the

preprepared redundant equipment has to take an effect. It can be activated manually. In this case it is called 'automated with manual approval configuration'

Conclusion

This paper explains the redundancy algorithms whose main idea is to supply the network with additional either physical or logical connections or devices. Primarily, it is about determining a real cause of an issue. It is important not to take all issues too generally, but when the issue happens, have its sufficient description to eliminate problems in middleware that could possibly cause the same issue. An onion fault model describes the order of issues' severity and inclusion in one another. If an application is multipart, a substantial problem like an outage could happen with it and it will not start working by itself, but to another part of an application, it will appear as a minor issue that some indicators cannot be provided.

When a system experiences a fault, a big thing is a graceful degradation. This means that if a part of a system stopped working this shouldn't kill the whole system. All the working parts can continue supplying their functionality. When some mechanisms include the broken others to perform fully, a warning with the fault description and preferably the fix time could be shown.

In order to provide this kind of behaviour, and to cover the need for infinite non-stopping data supply from the IoT devices, the system could be supplied with additional devices that will be in standby mode waiting to be needed and turned on in the whole flow. Depending on the speed an additional device has to infiltrate and on the presence of state (memory or set up in which a working partner is), there are three types of standbys. A cold standby is when a device is initially set up and ready to participate, but is off until it will be needed. A warm standby is when one device syncs its data/state with a supplementary device. A hot standby is when a second device is in a completely ready physical and logical state to pick up work as long as sinking happens all the time.

It is not only about backup for a device. The backup should also be on the protocol level, meaning that both devices should have a common IP or virtual address or be in one redundancy group. The protocols that support this kind of addressing to different hosts are called common address redundancy protocol and virtual router redundancy protocol.

References

1. M. Melo and G. Aquino, "FaTEMa: A Framework for Multi-Layer Fault Tolerance in IoT Systems," *Sensors*, Oct. 2021. vol. 21, no. 21, p. 7181.
2. A. Sari and M. Akkaya, "Fault Tolerance Mechanisms in Distributed Systems," *International Journal of Communications, Network and System Sciences*, 2015. vol. 08, no. 12, pp. 471–482.
3. C. J. Walter and N. Suri, "The customizable fault/error model for dependable distributed systems," *Theor Comput Sci*, Jan. 2003. vol. 290, no. 2, pp. 1223–1251.
4. P. Vedavalli and C. Deepak, "Enhancing Reliability and Fault Tolerance in IoT," in *2020 International Conference on Artificial Intelligence and Signal Processing (AISP)*, Jan. 2020, pp. 1–6.
5. R. Peng, Q. Zhai, and J. Yang, *Reliability Modelling and Optimization of Warm Standby Systems*. Singapore: Springer Singapore, 2021.
6. J. Wen, Y. Cao, L. Ma, and S. Du, "Design and Analysis of Double One Out of Two with a Hot Standby Safety Redundant Structure," *Chinese Journal of Electronics*, May 2020. vol. 29, no. 3, pp. 586–594.
7. [7] Jen-Hao Kuo *et al.*, "An Evaluation of the Virtual Router Redundancy Protocol Extension with Load Balancing," in *11th Pacific Rim International Symposium on Dependable Computing (PRDC'05)*, pp. 133–139.
8. [8] R. Nur, Z. Saharuna, I. Irmawati, I. Irawan, and R. Wahyuni, "Gateway Redundancy Using Common Address Redundancy Protocol (CARP)," *IJITEE (International Journal of Information Technology and Electrical Engineering)*, Feb. 2019. vol. 2, no. 3, p. 71.