The Circuit Breaker Pattern targeted to Future IoT Applications

Gibeon Aquino$^{1[0000-0001-6950-8169]}$, Rafael Queiroz$^{1[0000-0002-7088-4492]}$, Geoff Merrett$^{2[0000-0003-4980-3894]}$, and Bashir Al-Hashimi^{2}

[1] Department of Informatics and Applied Mathematics, UFRN, Brazil
`gibeon@dimap.ufrn.br,rafaelqueiroz@ufrn.edu.br`
[2] School of Electronics and Computer Science, University of Southampton, UK
`{gvm,bmah}@ecs.soton.ac.uk`

Abstract. In the context of the Internet of Things (IoT), there is a growing trend towards increasing the integration and collaboration between IoT systems to create relevant end-to-end solutions. Accordingly, addressing dependability in the future IoT applications will surely be more challenging. In this work, we examine a popular microservices pattern known as Circuit Breaker (CB). This pattern aims at preventing failure from cascading to dependent services. In the context of IoT, it can be used as an intermediary in the communication between critical IoT nodes to increase the dependability of the whole. Notwithstanding, some particularities present in IoT must be considered to allow this pattern to yield similar benefits. Therefore, we compile several aspects concerning the design and implementation of the CB tailored to IoT applications as a taxonomy. Also, we conduct an experimental validation to compare the benefits of the CB in a prototype of a traffic light system.

Keywords: Circuit Breaker · Internet of Things · Microservices Architecture · Dependability · Software Architecture

1 Introduction

There is a growing trend towards increasing the integration and collaboration between IoT systems to create relevant end-to-end solutions [2]. Under the collaborative perspective, they form a cluster of systems cooperating to solve harder problems. This phenomenon is known as System of Systems (SoS) [1], and it is well applicable to a significant part of existing IoT systems [4]. Indeed, this kind of IoT solutions has been advancing very fast and, shortly, they might represent most of the IoT deployments. The concern is that designing, implementing, and operating IoT applications acting as SoSs is even more complex and introduces new challenges. Therefore, advanced development strategies are required to address dependability in these applications adequately [6].

Meanwhile, Microservices Architecture (MSA) has been increasingly lauded as a successful approach to achieving dependability in information systems. There is also a growing position in favor of applying MSA to in IoT [3, 7]. Accordingly, IoT applications could adopt several of the MSA development strategies to reap similar benefits. Among the MSA strategies, the Circuit Breaker (CB) is a prevalent pattern to deal with the resilience of distributed services. It works by preventing the failure propagation to dependent services. For IoT systems, it can be used as an intermediary in the communication between critical IoT nodes in order to increase the dependability of the whole solution.

This work seeks to explore the options of designing and implementing the CB in IoT solutions. Although many IoT devices have high computing capabilities (e.g., smartphones, home voice assistants), a significant part has limited capabilities (e.g., memory, processing power, energy availability, connectivity), dedicated systems, and non-preemptive execution [8, 5]. Therefore, the CB in IoT applications must address these constraints properly. For this reason, we investigate several CB possibilities target to IoT and organize them as a taxonomy. Moreover, to examine the applicability and trade-offs, we conduct an experimental study using a prototyped IoT application. This prototype simulates a solution of smart traffic lights based on the collaboration of multiple nodes.

2 Circuit Breaker Narrowed to IoT Systems

The CB is a simple and effective pattern for fault tolerance. In the interaction between different microservices, it assumes the responsibility of detecting failures and preventing its propagation. The use of CB brings several benefits directly related to availability and reliability, such as: (a) It prevents to perform the action that is doomed to fail; (b) It allows handling the error quickly and gracefully; (c) The callers do not have to cope with the failure themselves; (d) Custom fallback plans can be used; (e) All callers are spared from calling the crashed service; (f) It can also spare the service from being overwhelmed by large numbers of requests (e.g., it can implement a local cache).

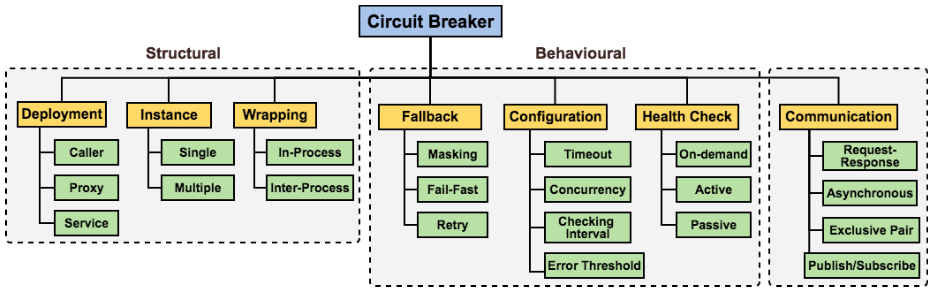

Fig. 1. A Taxonomy for the Design of the Circuit Breaker Targeted to IoT applications.

Although the CB is a simple pattern, there are several issues to take into account when implementing it. For the best of our knowledge, the literature about it is usually focused on information systems and fail to address specific needs required by IoT. For this reason, we sought to compile the main aspects, and possibilities regarding the CB tailored to IoT and organized them as a taxonomy (Fig. 1). It defines three main groups, according to their influence on the CB design. The Structural comprises the options which affect the structure in terms of implementation and execution. The Behavioural includes those related to the behavior at run-time. The last is related to communication.

2.1 Structural

A critical decision to be made in IoT scenarios is the deployment location where the CB will execute, i.e., in which physical node it will be hosted. Because every message must pass through it, its availability, security, and performance have to fit the application needs. In IoT applications, these aspects include the reliability and the capacity of the hardware, the deployment location, the energy supply, the connectivity, and also the other process/services competing in the same node. In essence, it can execute in the caller, service, or as an additional node.

The management of CB instances is another critical decision. Simple solutions can adopt a single instance option running in the caller, service, or proxy. However, in some applications, redundancy strategies involving the use of multiple instances should be considered also to improve its availability. Also, the way how the instances interact with the caller or the service (wrapping) may also be taken into consideration. It can share the same process (In-process), or it can run in a separate process (Inter-process).

2.2 Behavioural

Several options exist, but they can be grouped in the following categories: Masking, Fail-fast, and Retry. While the masking strategy seeks to hide the failure to the caller, the fail-fast allows to notice it as fast as possible. The most common masking strategies are: return a default value; return the last valid result (e.g., from a cache); or return a calculated value (e.g., forecasting based on historical data). Among the fail-fast strategies, the most common are: return an error code; return the original error to the caller. Finally, the Retry strategy seeks to try the failed operation again some times, hoping it can be successful. It can also retry in a surrogate service.

A fundamental ability of the CB is the failure detection on the target service in order to trip the circuit. In the same way, when the circuit is open, it must monitor the service health to close the circuit safely. The main strategies to do this are: (a) On-demand – it periodically transits to the state of half-open and seize a real request to test the service; (b) Passive – it waits for periodic health signals from the service (e.g., heartbeats, keepalives) to confirm if the service is available or unavailable; (c) Active – it regularly checks the service health independently of the caller requests.

Finally, several parameters should be adjusted appropriately, considering the application needs, to attain the CB benefits. One of them is the *timeout*, which is used to establish a limit of how long the CB should wait before assuming an omission failure. The *concurrency* configuration includes parameters such as the maximum of concurrent requests, requests queue size, and maximum throughput. The *checking interval* establishes the frequency of health checking. To differ intermittent and permanent failures, the CB uses the *error threshold* which indicates the limit of failures it can tolerate.

2.3 Communication

IoT solutions are strongly based on communication between things. Currently, a vast number of communication protocols and technologies coexist. As an intermediary, the CB must follow the same communication model adopted by the peers. Despite the several options, we can abstractly classify them in four models.

The Request-response is the model which the caller sends requests, and the service responds. This model is usually synchronous, as the caller has to wait for the response to continue its task. It is also stateless as no information about the caller is kept between requests. Even in IoT systems, RESTful HTTP solutions are commonly found [3]. However, for constrained devices and networks, other REST derivatives options are frequently mentioned, e.g., the Devices Profile for Web Services (DPWS) and the Constrained Application Protocol (CoAP).

The Asynchronous messaging allows the systems to send messages to each other asynchronously. Commons examples of this model in IoT are WebSocket protocol and Reactive Streams. One variation of that model is the Publish-Subscribe. In this model, the sender of a message (publisher) does not send it directly to the receiver (subscriber). Alternatively, they use an intermediate (i.e., message broker) to asynchronously delivered the messages. One of the most common examples of this model in IoT solutions is the MQTT (Message Queuing Telemetry Transport). Finally, Exclusive-Pair is a simple model and considered a low-level option. It is a bidirectional and fully duplex communication model which uses a persistent connection between a pair of elements.

3 Experimental Study

To demonstrate the suitability of the CB to address the dependability, we developed a prototype[3] that simulates a collaborative traffic light system. It is composed of several micro-controllers, each one placed in a traffic junction to control a traffic light group. Each runs an instance of a Traffic Junction System (TJS) and are wirelessly interconnected. The TJS performs a periodic task of monitoring the traffic density on the roads its controls and also requests data from adjacent TJSs to make a more accurate decision. In this study, we sought to evaluate the consequences of intermittent nodes for the whole solution.

This scenario aims to demonstrate the SoS paradigm applied to IoT. A critical issue in such systems is that failures in one part can induce the dependent systems to fail too, potentially triggering the cascading effect. In order to examine this issue, we ran five TJS instances (TJ0...TJ4) and simulated omission failures in two of them according to the following configuration (*node, start, duration*): {*(TJ1, 30s, 30s), (TJ1, 90s, 10s), (TJ3, 40s, 20s)*}. This kind of failure happens when the service omits to respond to a request. The duration of the experiment was 120s, with task periods of 2s. For the physical nodes, we used Raspberry Pi 3 Model B, interconnected in a network of 100 Mbps. Concerning the CB design structural decisions, we adopted the following strategies:

[3] https://github.com/labcomu/smart-traffic-prototype

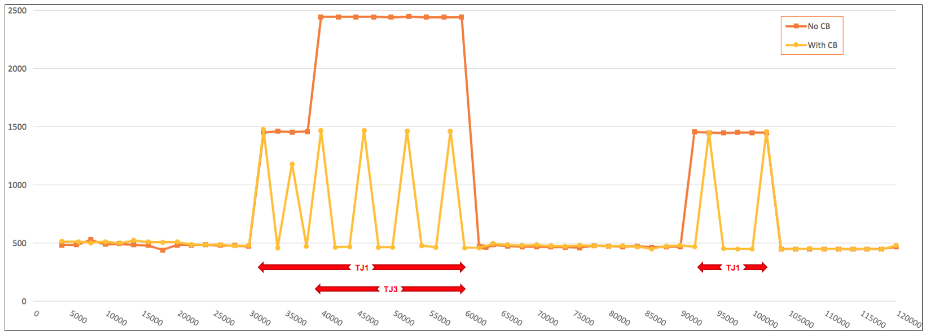

Fig. 2. Task time over the experiment execution.

caller deployment, single instance, and in-process wrapping. Such choices aimed to create a configuration suitable to the typical constraint of traffic junction infrastructures. Concerning the behavior, we implemented the fail-fast strategy, on-demand health checking, and configuration parameters of {*timeout=1s, error threshold=1, no concurrency*}. These choices sought to achieve more accurately the requirements of the TJS.

3.1 Experimental results

In our study, we extended the traditional availability measurement by introducing the metric *availability to collaborate* (AC). It is defined as the fraction of time the TJS is active responding requests from its peers. As the solution's key point is the collaboration ability, the time it is available to provide information to the adjacent nodes is a relevant quality attribute. Table 1 shows that the AC increased with the use of the CB. It means the solution with CB allowed the TJS to dedicate more time collaborating with its peers and consequently improving the accuracy of its decision.

Table 1. Results of the experiment execution collected on the TJ0 node.

	No CB	With CB
Executed Tasks	60	60
Complete Tasks	41	41
Partial Tasks	10	19
Aborted Tasks	9	0

	No CB	With CB
Task time (mean)	929 ms	600 ms
Task time (std)	737 ms	329 ms
Availability to collaborate	53.6%	70.0%

This effect is influenced by the decrease of the *Task time*, which is the time to the TJS completes one task. Accordingly, the use of the CB improved the performance and also the stability. Without the CB, the task time becomes very high during the failures (Fig. 2). Because the TJS needs to wait for the adjacent nodes response to complete its task, its performance is strongly affected if some of the peers last to respond.

Finally, we also evaluated the completion of the tasks (Table 1). The tasks were classified as follows: *Complete* – executed considering all peers response and completed before the task cycle expires; *Partial* – completed before the task cycle expires, but could not use information from all peers; *Aborted* – did not complete before the task cycle expires. The results showed a complete reduction in the aborted tasks with the CB. It means the TJS was able to make more accurate decisions.

4 Conclusions

This paper sought to examine the circuit breaker pattern in the context of IoT. We seized the growing belief that some MSA practices are promising to IoT, particularly considering the expected complexity of future applications. Our main contribution was the definition of a taxonomy, based on the compilation of several aspects concerning the design and implementation of the CB tailored to IoT applications. Also, we conducted an experimental validation to compare the benefits of this pattern in a prototype of a traffic light system. The results showed several advantages for this specific application. In particular, it demonstrated the CB ability to improve performance, availability, and accuracy significantly.

Acknowledgements

This research was partially funded by CAPES - Finance Code 001, INES 2.0, CNPq grant 465614/2014-0, FACEPE grant APQ-0399-1.03/17, and CAPES grant 88887.136410/2017-00.

References

1. Ackoff, R.L.: Towards a system of systems concepts. Management science **17**(11), 661–671 (1971)
2. Bello, O., Zeadally, S.: Intelligent device-to-device communication in the internet of things. IEEE Systems Journal **10**(3), 1172–1182 (2016)
3. Butzin, B., Golatowski, F., Timmermann, D.: Microservices approach for the internet of things. In: 21st International Conference on Emerging Technologies and Factory Automation (ETFA). pp. 1–6. IEEE (2016)
4. Delicato, F.C., Pires, P.F., Batista, T., Cavalcante, E., Costa, B., Barros, T.: Towards an iot ecosystem. In: Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems. pp. 25–28. ACM (2013)
5. Hahm, O., Baccelli, E., Petersen, H., Tsiftes, N.: Operating systems for low-end devices in the internet of things: a survey. IEEE Internet of Things Journal **3**(5), 720–734 (2016)
6. Hammoudi, S., Aliouat, Z., Harous, S.: Challenges and research directions for internet of things. Telecommunication Systems **67**(2), 367–385 (2018)
7. Santana, C., Alencar, B., Prazeres, C.: Microservices: A mapping study for internet of things solutions. In: 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA). pp. 1–4. IEEE (2018)
8. Zikria, Y.B., Yu, H., Afzal, M.K., Rehmani, M.H., Hahm, O.: Internet of things (iot): Operating system, applications and protocols design, and validation techniques. Future Generation Computer Systems **88**, 699 – 706 (2018)