Towards Low-Jitter and Energy-Efficient Data Processing in Cyber-Physical Information Systems

Stefan Reif[1], Luis Gerhorst[2], Kilian Bender[2], and Timo Hönig[1]
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
[1]{reif, thoenig}@cs.fau.de [2]{luis.gerhorst, kilian.bender}@fau.de

Abstract

Cyber-physical systems build the backbone of today's information systems and implement, for example, complex control applications that strictly rely on sensor data. Thus, it is inherently important for cyber-physical systems to provide a reliable data path throughout the entire system: from the sensor nodes to the data post-processing infrastructure in networked environments (e.g., edge and cloud infrastructure).

This paper analyzes system-level aspects of the data path of cyber-physical systems (i.e., storage components and file systems) and reveals limitations of current technologies. To improve the current state of the art, we present the implementation of an embedded file system with low jitter which improves predictability characteristics of cyber-physical systems.

1. Introduction

Cyber-physical systems (CPS) build the backbone of today's information systems [1, 2]. The systems are employed to support and implement a wide variety of different applications [3] which, as of today, use countless Internet-of-Things (IoT) devices [4] that sense, collect, and pre-process data. The data of CPSs propagate to large-scale, distributed workloads that are handled by cloud-computing infrastructures [5, 6]. In such complex systems, it is crucial to actively protect the weakest system components to support the overall system. The base infrastructure consists of smallest-sized battery-powered IoT devices (e.g., [7]) that have modest compute and storage resources and communicate over wireless communication links.

The heterogeneity of individual components (i.e., network links, storage devices) makes it a challenge to design and build cyber-physical information systems (CPIS) under consideration of non-functional properties at system- and component-level. Such non-functional properties include, for example, timing characteristics and power demand. In particular, micro-sized systems [8] that are no larger than the tip of a grain of rice are operated at technological limits. Therefore, the systems' software must be closely adapted to individual hardware characteristics. For example, software-design considerations must include timing properties and power demand of operations at system level. Cross-layer approaches [9] that allow root-cause analyses of complex systems are necessary to achieve a holistic design for CPIS.

In CPIS, measured data often can only be sensed once, as environmental properties (e.g., atmospheric pressure, warm-up of a workpiece) that are captured by sensors, change rapidly. Thus, data that is sensed and (pre-)processed by the base infrastructure is extremely valuable to the overall system. Obtained data must be handled with due care and precautions to proactively avoid data losses and to reduce the number of costly recalculations and retransmissions. In addition, CPISs are fragile and sensitive to errors as to the large number of transmissions that occur across system boundaries and use unreliable links (e.g., wireless networks).

Recent research has explored different ways to establish protective schemes for the data path, for example, by reliable network protocols [10] and corresponding operating-system level support [11]. So far, however, it has not been considered that storage infrastructures in general and file systems in particular play a decisive role to protect the data path of CPISs. Our findings, that we present in this work, show that file systems which are used within CPISs have disadvantageous characteristics (i.e., high jitter) due to unpredictable runtime behavior that causes high jitter for file system operations. This is a serious threat to the operation of CPISs as sensor nodes at the base infrastructure suffer from low performance and high energy demand. This leads to low energy-efficiency and may result in runtime errors (deviation from target or actual) and can even entail failures (breakdown). To improve the current status quo, we present MPFS, a file system which is tailored to the requirements of CPISs (i.e., low jitter).

URI: https://hdl.handle.net/10125/60163
ISBN: 978-0-9981331-2-6

The contributions of this work are threefold. First, we present an in-depth analysis for file system operations with regard to their non-functional properties (e.g., latencies). Second, we present MPFS, a file system designed to improve the current state of the art by providing predictable, low-latency operations at file system level. Third, we evaluate MPFS and compare the results with an industry standard (i.e., EXT4). We further discuss the impact of MPFS for the system design and improved energy-efficiency (i.e., low-power operations). In sum, the paper improves the data path of CPISs at storage level and makes file system operations significantly more predictable (i.e., low jitter) compared to the current status quo.

The paper is structured as follows. Section 2 presents the system model for cyber-physical information systems that are subject to the discussion. The challenges outlined in Section 3 are addressed by our proposed file system MPFS. Section 4 presents the design and implementation of MPFS and we evaluate the file system with benchmarks and compare it against an industry standard (i.e., EXT4) in Section 5. Section 6 presents related work and Section 7 concludes.

2. System Model

For the work presented in this paper we consider the following system model of a cyber-physical information system. Our system model assumes concurrent activities that are spread across several networked system components (i.e., IoT devices, edge servers, and cloud infrastructure). The individual components concurrently execute threads which communicate with each other and interact in a reactive manner with the physical world. For this interaction, a system-wide feedback and control loop is deployed to the overall cyber-physical system. Figure 1 shows an overview of the system model. Our system model assumes an indirect coupling of IoT devices with a cloud infrastructure by means of an edge server.

2.1. System Properties

The system properties of the targeted systems are grouped into three aspects. First, we assume that concurrent activities in the system jointly implement the necessary functionality of the overall CPIS in a cooperative manner. Second, the interactions of the system components use unreliable network links that may slow down or break during operation. Third, we assume that sensors and actuators implement feedback loops and control applications at different scopes of the system. On the one hand, we consider fast-path operations that operate on a local scope (i.e., component scope) and slow-path operations that have to operate across component borders on a global scope (i.e., overall CPIS scope), and thus, must utilize communication links that are susceptible to transmission failures.

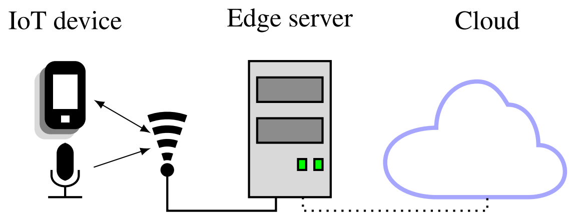


Figure 1: Abstract system architecture of a cyber-physical system that connects networked sensor nodes (i.e., IoT devices) to the cloud via an edge server.

Concurrency The system model assumes concurrency at thread- and component-level of the system nodes (i.e., IoT devices, edge server, cloud infrastructure) that operate within the CPS. As different system components must cooperatively implement higher level application logics, it is assumed that individual nodes do not operate on shared memory and cache coherency is not given. Instead, the individual system nodes use message passing for coordination purposes (i.e., control flow) that implement the application logic, on the one hand. On the other hand, the system model expects that data delivery (i.e., data flow) uses a network infrastructure that may be prone to data transmission delays and failures.

Interaction The cooperative work of the CPS leads to interactions that impose interferences in different areas of the CPS. For example, concurrent threads execute different subtasks and operate on finite system resources of the CPS (i.e., limited CPU resources and I/O bandwidth). Thus, the operations influence each other and cause non-deterministic delays when resources exhaust or are temporarily unavailable. On network links, as another example, interference between system components lead to congestion and retransmissions. Such events entail an increased resource demand (i.e., time and energy demand). CPS interactions inherently depend on physical aspects of the environment and often lead to unforeseeable and unpredictable situations that must be handled gracefully.

Feedback and Control The degree of concurrency and interferences of the overall system impact the quality of the feedback and control loop of the CPS. Thus, our system model assumes that the control quality directly depends on the predictability of individual system components. For example, when the resource demand of specific CPS operations (i.e., transmission of datagrams, storage of data blocks) is predictable, the system can adapt to such system properties to improve non-functional characteristics (i.e., performance, latencies, energy demand) of the overall CPS.

For our system model we identify and discuss distinct cyber-physical information systems (CPIS) in the following section.

2.2. Cyber-Physical Information Systems

We consider a cyber-physical information system (CPIS) to consist of one or more networked IoT devices that implement sensors and actuators of a cyber-physical system (CPS) and implement the base functionality of a information system (IS) that operates across edge and cloud infrastructure at a higher abstraction level. Both of CPISs presented in this section share the same system model as discussed in the previous section. We discuss two different types of CPISs and focus on open research questions, in particular, regarding the data path.

Predictive Maintenance Systems With the support of CPSs, it is feasible to build predictive maintenance systems that implement CPISs which uses sensor values to detect the imminent breakdown or failure of specific system components. As to the proactive characteristics of predictive maintenance systems, it is possible to minimize down times as maintenance can be applied ahead of failure of the affected system component(s). Anomalies of system components are detected in such systems, for example, to recognize a failing solar cell in a batch of identically constructed cells in a neighboring environment [12]. Sensed data in predictive maintenance systems includes operating values or their average (instead of detailed time series data). To guarantee a reliable storage of the sensed data, predictive maintenance systems must ensure that sensed data is reliably stored and aggregated even in case of network failures. Besides storing capabilities this also entails a certain amount of compute power that is necessary for the individual IoT devices that are equipped with the sensors.

Dynamic Distributed Wireless Systems Dynamic distributed wireless systems build the base infrastructure for CPISs that have a variable number of subsystems. For example, during the operation of the dynamic distributed wireless system, individual system nodes (i.e., IoT devices) join or leave the overall system. The application logic at a high level of abstraction, however, is designed to handle the dynamic system structure. Interdisciplinary research, for example, relies on such dynamic distributed wireless system to monitor the behavior of wild animals (e.g., bats [11]). Although the higher-level application logic is prepared to tolerate the sporadic vanishing of individual nodes, such situation increase the importance of reliable data sensing and storage at the level of the IoT devices: when network connections are lost or become available, the system must reconfigure and adapt to the new environmental conditions. Data must be pre-processed and reliably stored. In particular, the storage access must be predictable with regard to time and energy demand in order to respect battery capacities of the sensor nodes within the CPIS.

For both of the discussed CPISs it is important to stress the dynamic operation in dependence of the availability of network links. In cases when network links are congested or unavailable, it is necessary to dynamically switch to an operation mode during which data is (i) pre-processed and (ii) stored at system components that have only mediocre amounts of resources (i.e., IoT devices). From the discussed CPISs we extract distinct challenges in the next section.

3. Challenges

Aligned to the system model and the concrete CPISs discussed in the previous section, we extract challenges that arise when CPS are integrated into IS.

Real-time Requirements Within CPSs, strict timeliness is required to ensure stability of controllers. If a computation is delayed too long, control decisions are potentially invalidated by concurrent events in the physical system. The result is an instability of the controlled system, which can lead to system errors and, in the worst case, a breakdown of the system.

Similarly, the IS that accumulates and processes data on a higher logical level has to meet timing requirements, because otherwise, decisions made by the system can be outdated and consequently invalid. Therefore, the information processing chain has to consider data age during operation.

Cost and Size CPSs often employ smallest-size embedded computing systems [13, 8] in very large numbers [14]. It is therefore mandatory that the hardware comes at low cost. In consequence, these systems are severely restricted in terms of computing power and energy supply. Therefore, using the available resources efficiently is vital for CPISs. However, these systems still have to work reliably, even under rough environmental influences [15].

Energy Awareness CPSs and embedded systems [16] typically operate under energy and power constraints. Data processing and data storage contribute significantly to the whole-system energy demand [17]. Therefore, energy efficiency in the average-case and in the worst-case [18] are important aspects.

Data Management To integrate a CPS into an IS, the data management has to be considered carefully. The CPS can process data locally, transmit it via network, or save it locally on persistent storage. The optimal trade-off depends on the network parameters, local storage properties, and timing and energy requirements.

4. Design and Implementation

Persistent storage is a critical core element for CPIS. However, modern file systems are designed to balance a trade-off between a variety of functional and non-functional requirement of numerous use-cases. For instance, they focus on high throughput rather than worst-case scenarios. Therefore, current state-of-the-art file systems do not satisfy the challenges presented in Section 3. This section presents the *embedded predictable file system* (MPFS), a file system prototype tailored for CPIS.

4.1. Objectives

Goal of the MPFS file system is to provide typical file operations for CPISs, such as read, write, append, create, and delete, with ACID properties (atomicity, consistency, isolation, and durability) in a predictable and energy-efficient manner. We assume that CPISs use flash storage for cost, size and robustness reasons. The key source of unpredictability, and also a major energy consumer, are block-layer operations on the flash device [17]. Therefore, the main design goal of MPFS is to keep the number of block-layer operations small and analyzable, even in the worst case. To achieve this, MPFS dispenses with common file-system features that are not necessary in embedded systems. In particular, MPFS purposefully does not implement file system features such as transparent compression or sparse files.

4.2. Design

Tailoring a file system specifically to the needs of CPISs requires the separation between necessary features and unnecessary features of file systems.

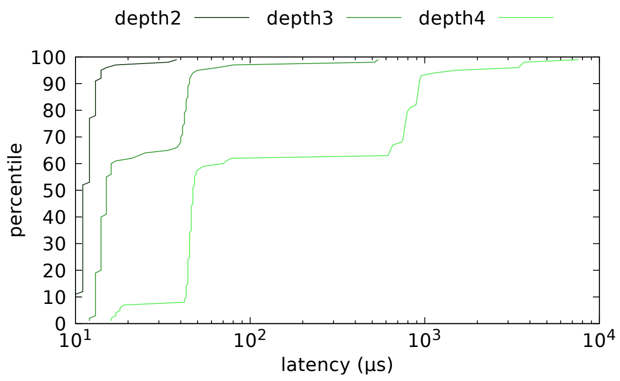


Figure 2: Latency of EXT4 for opening random files depending on directory-nesting depth

Therefore, MPFS intentionally lacks common but unnecessary file system features that interfere with predictability properties and energy efficiency.

Flat File-System Structure The key to predictable execution time and energy demand is the minimization of complexity. As a motivating example, we have evaluated 10,000 `fopen` operations on an EXT4 file system on a Raspberry Pi [19]. In this experiment, we place these files into directories with different levels of nesting depth. The cumulative latency distributions are shown in Figure 2. Thereby, the bottom end of each line shows the best-case latency, and the top end shows the worst-case latency. The results demonstrate that, with increasing nesting depth, both latency and variance grow. The reason for this effect is that the path resolution algorithm has to access more storage blocks when nesting directories. In summary, this experiment shows that directory nesting causes unpredictability in file systems. We therefore opt for an unconventional approach in MPFS: it supports no directories except for the file system root and therefore provides a flat file-system structure, only.

Instead of a hierarchical directory structure, MPFS stores all file-system entries in a hash table of configurable size. Thus, it enables path resolution with a constant and small number of block operations. The designer of a CPIS is responsible to configure the hash table size appropriately. This is a typical approach in CPS: system configuration is decided statically at design-time in order to reduce complexity at run-time.

Memory Management To allow block allocation and deallocation with a constant number of block-layer operations, MPFS manages information about available storage space in a linked list (i.e. list free blocks). On the one hand, this data structure increases predictability because allocation and deallocation have a constant cost in terms of block-layer operations. On the other hand, it does not implement wear leveling, as this is commonly implemented in hardware, below the block-layer level. This memory-management strategy of MPFS improves the worst-case behavior with respect to latency and energy demand.

Block-Layer File Structure Allocated data that is managed within files at logical level are stored in individual blocks of the flash storage device. Used blocks are managed on a per-file basis and in linked lists. This data structure allows simple and deterministic append operations to the file contents.

Consistency and Durability We assume that unexpected power losses during complex operations (i.e., operations that span across several block-layer operations) may lead to corruptions and errors. We use appropriate counter measures to avoid a propagation of errors. Instead of transaction-based operations, MPFS solely relies on atomic block-updates to ensure consistency. The basic idea is that every block update maintains a logically consistent state. Besides, the order of block operations is enforced using barriers and `sync` operations. Since block operations in modern flash storage are atomic, the file system always remains in a consistent state.

Compared to transaction-oriented approaches (e.g. journaling), the MPFS approach to data consistency has better worst-case behavior, since it has less overhead. Furthermore, all data is written immediately to the disk, rather than caches. This results in performance degradation for the average-case, but it ensures that all written data is immediately in persistent storage. Furthermore, it has lower run-time interference than asynchronous cache write-back operations.

Consistency in MPFS is, however, not a contradiction to transaction-based consistency. Instead, a CPIS can implement transactions on a higher logical level, for instance, using a database system (i.e., SQLite [20]). This combined approach enables complex transactions that are not atomic in the current implementation of MPFS.

4.3. Implementation

We have implemented MPFS as a module for the Linux kernel, based on the Linux virtual file system (VFS). This kernel subsystem offers necessary abstractions which increase portability with relatively low overhead.

5. Experimental Evaluation

We evaluate MPFS on a Raspberry Pi 3 Model B [19] with a 16 GiB SD card, using Raspbian 9 (Linux Kernel v4.9). The system features a quad-core processor running at 1.2 GHz and 1 GiB of main memory. In the evaluation we compare MPFS against the industry-standard general-purpose file system EXT4 [21].

5.1. Evaluation

For each experiment, we first drain all file system caches, and then execute 10,000 repetitions. We measure the latency using the function `clock_gettime`, store measured latency values in the volatile RAM during the experiment, and make the results persistent by writing them onto the SD card after the experiments' completion. This minimizes interference on the file system and caches during the experiment.

For each experiment, we prepare the file system specifically so that, for instance, all necessary files exist. Figure 3 summarizes the evaluation results for the following four benchmark scenarios:

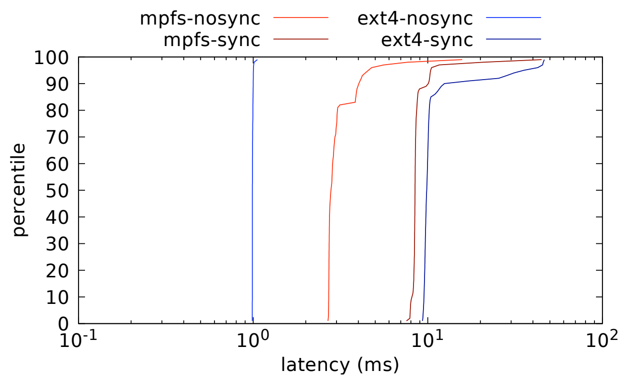
Scenario / Description	
append	Append a block of 4096 bytes to an existing file.
read	Read a block of 4096 bytes from an existing, non-empty file.
open-new	Open a non-existing file, which implicitly creates the file.
unlink-empty	Delete an empty file.

We evaluate all scenarios with the MPFS and EXT4 file systems, both in a *nosync* variant that uses volatile caches, and a *sync* variant that enforces data persistence.

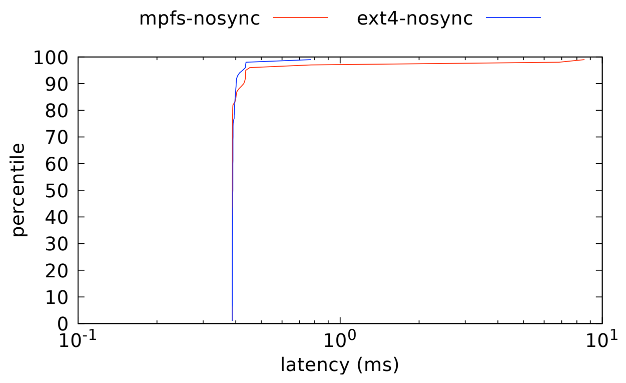
5.2. Analysis

Our evaluation of the four benchmark scenarios shows the results visualized by Figure 3. In the following we discuss the results which show several different trends.

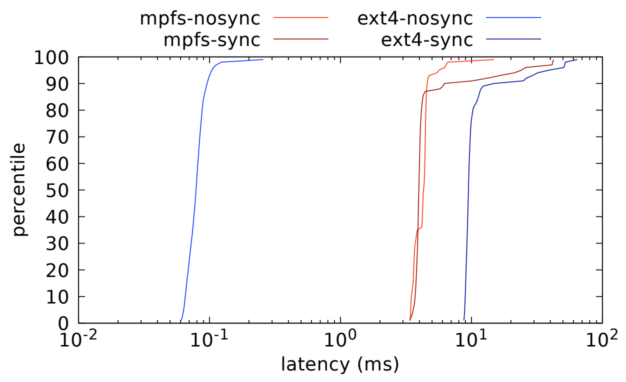
- EXT4-nosync performs best but does not guarantee data persistence, because data is written to volatile caches, only. Thus, EXT4-nosync avoids the latency and the jitter of operations to the persistent flash storage. However, this approach is not useful for CPISs,



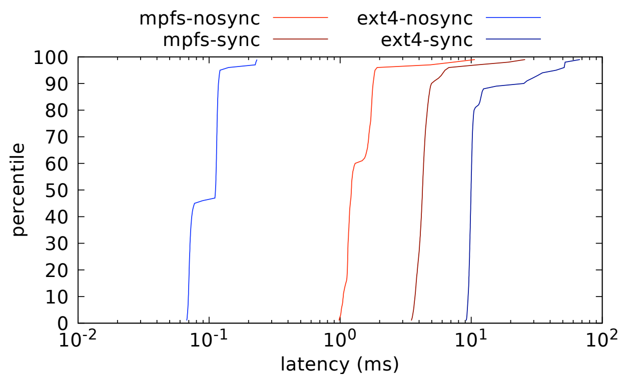
(a) 4096 bytes append latency



(b) 4096 bytes read latency



(c) Open new file latency



(d) Unlink empty file latency

Figure 3: Cumulative latency distributions of file-system operations

because data that was sensed shortly before a system failure is likely to be very important and may not be recovered otherwise.

- EXT4-sync has the highest latency of all evaluated file system variants, because the EXT4 implementation of Linux assumes that fast caches in volatile memory are available. Therefore, the implementation is not tailored to the system model of CPIS. Since EXT4 is a general-purpose file system, it targets a pareto-optimal balance between functional (i.e., features) and non-functional (i.e., performance, energy) system requirements. In consequence, the file system is more complex, resulting in higher worst-case latency and jitter compared to the persistent MPFS variant.
- MPFS-nosync is slower than EXT4-nosync because the current prototype does not use memory caches as efficiently as EXT4. However, MPFS is not designed to be efficient in this non-persistent operation mode.
- MPFS-sync is faster than EXT4-sync because it is specifically tailored to achieve persistence with a predictably small number of flash operations. In particular, the worst-case behavior latency is lower. This result show that MPFS achieves better predictability than EXT4.
- When reading data, MPFS is slower than EXT4 because the latter can use buffers in volatile memory more efficiently.
- For all persistent variants, the worst-case latency is significantly higher than the average-case because the required flash operations introduce latency and jitter.

In summary, our MPFS prototype offers persistent and consistent data storage with better predictability than EXT4. However, EXT4 uses caches more efficiently. In summary, EXT4 is well-suited for various use cases, but MPFS provides better predictability which is the most important factor for the reliable operation of CPISs.

6. Related Work

Molano et al. propose a real-time file system [22] for the RT-Mach kernel. It uses hard disk drives as storage back-end and bandwidth reservation to provide latency guarantees. However, storage technology has evolved over the years and, today, hard disks contain complex controllers that do not guarantee operation latencies. Besides, hard disk drives are large-sized and sensitive to physical stress, which makes them unsuitable for CPSs that operate under rough environmental conditions. Therefore, MPFS uses flash devices for storage.

Over the last years, multiple file systems have been developed specifically for flash storage [23, 24, 25, 26]. These file systems consider problems specific to this storage technology, such as wear leveling. However, their performance focus remains on average-case performance rather than predictable latencies. For instance, flash file systems typically apply garbage-collection, which increase throughput when inactive, but it sporadically causes large jitter when active. One reason is that popular file system benchmarks, such as bonnie++ [27], evaluate the file system throughput.

The paradigm shift from throughput to predictable latencies has reached various fields of research related to cyber-physical information systems. For instance, latency outliers in data centers can harm the whole-system performance significantly [28, 29, 30], demanding jitter reduction and mitigation strategies at large scale. Similarly, operating systems aim at predictably low operation latency [31]. They thus provide a reliable and predictable infrastructure for CPS and CPIS.

Predictable operating systems form the basis for IoT devices, as well as the fog and edge computing paradigms [14, 32]. CPISs can therefore utilize system components tailored for predictable operating systems. However, the information management remains a challenge specifically for CPISs that demands for application-specific [13] approaches.

7. Conclusion

Modern information systems are connected to the real-world and use cyber-physical systems (CPSs) that sense and control the real world. The CPSs provide necessary data for further information processing, and enable novel use-cases that observe, evaluate, and predict the behavior of physical systems at fine detail and, simultaneously, at large scale.

When integrating CPSs into large-scale cyber-physical information systems (CPISs), efficient data management is the key challenge. Data can be (pre-) processed locally, it can be transmitted via different network links, and it can also be stored locally, while considering timing, energy, cost and further constraints. The optimal trade-off depends on the individual use case, network parameters, the properties of the local storage, and on functional and non-functional (i.e., timing and energy) constraints.

Local information processing in CPSs has been a research area for years, in terms of embedded and real-time systems. Similarly, research on real-time networks and cyber-physical networks integrates CPSs into networks. However, there is a distinct lack in research on file systems with predictable latency.

This paper has discussed shortcomings of existing file systems in the context of CPISs. The consequence is that, for CPISs, dedicated file systems are required. Therefore, this paper proposes MPFS, a file system specifically tailored to the needs of CPISs. The evaluation of the MPFS prototype shows a significant improvement in terms of latency and jitter, compared to the industry-grade EXT4 file system running on a Linux system.

8. Acknowledgments

This work was supported by the German Research Council (DFG) under grant no. SFB/TR 89 ("InvasIC", Project C1), grant no. SCHR 603/13-1 ("PAX"), and grant no. SCHR 603/15-1 ("LARN").

References

[1] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: the next computing revolution," in *Proceedings of the 47th ACM/IEEE Design Automation Conference (DAC'10)*, pp. 731–736, IEEE, 2010.

[2] S. Kartakis and J. A. McCann, "Real-time edge analytics for cyber physical systems using compression rates," in *Proceedings of the 11th International Conference on Autonomic Computing (ICAC'14)*, vol. 14, pp. 153–159, 2014.

[3] J. Lee and B. Bagheri, "Cyber-physical systems in future maintenance," in *Proceedings of the 9th World Congress on Engineering Asset Management (WCEAM)*, pp. 299–305, 2015.

[4] L. Atzori, A. Iera, and G. Morabito, "The Internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787 – 2805, 2010.

[5] B. Zhang, N. Mor, J. Kolb, D. S. Chan, N. Goyal, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz, "The cloud is not enough: Saving IoT from the cloud," in *Proceedings of the 7th USENIX Conference on Hot Topics in Cloud Computing (HotCloud'15)*, pp. 1–7, 2015.

[6] S. D'Souza and R. Rajkumar, "Time-based coordination in geo-distributed cyber-physical systems," in

Proceedings of the 9th USENIX Conference on Hot Topics in Cloud Computing (HotCloud'17), pp. 1–9, 2017.

[7] D. Blaauw, D. Sylvester, P. Dutta, Y. Lee, I. Lee, S. Bang, Y. Kim, G. Kim, P. Pannuto, Y. Kuo, *et al.*, "IoT design space challenges: Circuits and systems," in *Proceeding of 2014 IEEE Symposia on VLSI Technology and Circuits*, pp. 1–2, 2014.

[8] X. Wu, I. Lee, Q. Dong, K. Yang, D. Kim, J. Wang, Y. Peng, Y. Zhang, M. Saligane, M. Yasuda, K. Kumeno, F. Ohno, S. Miyoshi, M. Kawaminami, D. Sylvester, and D. Blaauw, "A 0.04mm3 16nw wireless and batteryless sensor system with integrated Cortex-M0+ processor and optical communication for cellular temperature measurement," in *Proceedings of the 2018 IEEE Symposium on VLSI Circuits (VLSI '18)*, IEEE, 2018.

[9] S. Reif, A. Schmidt, T. Hönig, T. Herfet, and W. Schröder-Preikschat, "X-Lap: A systems approach for cross-layer profiling and latency analysis for cyber-physical networks," in *Proceedings of the 15th International Workshop on Real-Time Networks (RTN'17)*, pp. 1–6, 2017.

[10] M. Gorius, *Adaptive delay-constrained internet media transport*. PhD thesis, Saarland University, 2012.

[11] F. Dressler, M. Mutschlechner, B. Li, R. Kapitza, S. Ripperger, C. Eibel, B. Herzog, T. Hönig, and W. Schröder-Preikschat, "Monitoring bats in the wild: On using erasure codes for energy-efficient wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 12, pp. 7:1–7:29, Feb. 2016.

[12] S. Iyengar, S. Lee, D. Sheldon, and P. Shenoy, "Solarclique: Detecting anomalies in residential solar arrays," in *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*, pp. 1–10, 2018.

[13] C. Eibel, S. Herbst, B. Cassens, T. Hönig, P. Wägemann, H. Janker, R. Kapitza, K. Meyer-Wegener, and W. Schröder-Preikschat, "A flexible, adaptive system for data-stream processing in energy-constrained ad-hoc networks," Tech. Rep. CS-2015-04, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2015.

[14] W. Shi and S. Dustdar, "The promise of edge computing," *IEEE Computer*, vol. 49, pp. 78–81, May 2016.

[15] E. Lee, "Cyber physical systems: Design challenges," in *Proceedings of the 11th International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'08)*, pp. 363–369, 2008.

[16] T. Mudge, "Power: A first-class architectural design constraint," *IEEE Computer*, vol. 34, pp. 52–58, Apr. 2001.

[17] J. Mohan, D. Purohith, M. Halpern, and V. C. V. J. Reddi, "Storage on your smartphone uses more energy than you think," in *Proceedings of the 9th USENIX Conference on Hot Topics in Storage (HotStorage'17)*, pp. 1–6, USENIX, 2017.

[18] P. Wägemann, C. Dietrich, T. Distler, P. Ulbrich, and W. Schröder-Preikschat, "Whole-system worst-case energy-consumption analysis for energy-constrained real-time systems," in *Proceedings of the 30th Euromicro Conference on Real-Time Systems (ECRTS'18)*, pp. 24:1–24:25, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018.

[19] Raspberry Pi Foundation, "Raspberry Pi 3 Model B." `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/`, 2016.

[20] "SQLite." `https://sqlite.org/index.html`.

[21] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, "The new ext4 filesystem: Current status and future plans," in *Proceedings of the Ottawa Linux symposium (OLS'07)*, pp. 21–33, 2007.

[22] A. Molano, K. Juvva, and R. Rajkumar, "Real-time filesystems. guaranteeing timing constraints for disk accesses in rt-mach," in *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS'97)*, pp. 155–165, IEEE, 1997.

[23] D. Woodhouse, "Jffs: The journalling flash file system," in *Proceedings of the Ottawa Linux Symposium (OLS'01)*, pp. 1–12, 2001.

[24] "Yet another flash file system (YAFFS)." `https://yaffs.net/`, 2018.

[25] J. Kim, H. Shim, S.-Y. Park, S. Maeng, and J.-S. Kim, "FlashLight: A lightweight flash file system for embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 11S, pp. 18:1–18:23, June 2012.

[26] C. Lee, D. Sim, J. Y. Hwang, and S. Cho, "F2FS: A new file system for flash storage," in *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST'15)*, pp. 273–286, USENIX, 2015.

[27] T. Bray and R. Coker, "Bonnie++." `https://www.coker.com.au/bonnie++/`.

[28] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, pp. 74–80, Feb. 2013.

[29] L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, "Attack of the killer microseconds," *Communications of the ACM*, vol. 60, pp. 48–54, Mar. 2017.

[30] D. Tsafrir, Y. Etsion, D. Feitelson, and S. Kirkpatrick, "System noise, os clock ticks, and fine-grained parallel applications," in *Proceedings of the 19th Annual International Conference on Supercomputing (ICS'05)*, pp. 303–312, ACM, 2005.

[31] S. Rumble, D. Ongaro, R. Stutsman, M. Rosenblum, and J. Ousterhout, "It's time for low latency," in *Proceedings of the 13th Conference on Hot Topics in Operating Systems (HotOS'11)*, pp. 1–5, 2011.

[32] M. Satyanarayanan, "The emergence of edge computing," *IEEE Computer*, vol. 50, pp. 30–39, Jan. 2017.