Achal: Building Highly Reliable Networked Control Systems

Arpan Gujarati
MPI-SWS
Germany
arpanbg@mpi-sws.org

Malte Appel
Saarland University
Germany
s9maappe@stud.uni-saarland.de

Björn B. Brandenburg
MPI-SWS
Germany
bbb@mpi-sws.org

ABSTRACT

In a highly reliable networked control system, active replication of critical system components is necessary for instantaneous recovery from crash or a network partition failure. However, due to *Byzantine* errors, the replicas can diverge and produce incorrect outputs. In this work, we target the specific problem of replica coordination in presence of environmentally-induced Byzantine errors, while addressing challenges and constraints specific to the CPS domain.

ACM Reference Format:
Arpan Gujarati, Malte Appel, and Björn B. Brandenburg. 2019. Achal: Building Highly Reliable Networked Control Systems. In *2019 International Conference on Embedded Software Companion (EMSOFT'19 Companion), October 13–18, 2019, New York, NY, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3349568.3351545

1 INTRODUCTION

Commercial aircraft systems are designed to be extremely reliable, typically to the order of less than 10^{-10} failures per hour. Such high levels of reliability are achieved using expensive custom hardware components with built-in redundancy, *e.g.*, [9]. In contrast, cyber-physical systems (CPS) in other domains such as autonomous vehicles, drones, and robots are not engineered as rigorously and, therefore, are not as reliable. In particular, strong economic and time-to-market pressures drive the adoption of cheap commercial off-the-shelf (COTS) components whenever possible, which creates a need for efficient software reliability solutions that work well on cost-efficient commodity embedded platforms. In this regard, we explore the problem of designing an effective middleware for building highly reliable networked control systems (NCS) over COTS embedded platforms (*e.g.*, Raspberry Pi's connected over Ethernet).

An ideal solution must tackle four main challenges. **(1)** Software fault tolerance for NCSs must adhere to *hard real-time requirements*, since any deviation from the assumed temporal properties may negatively impact the quality of control. **(2)** Active replication ensures instantaneous recovery from a crash or a network partition failure, but the replica states may diverge and produce incorrect outputs due to complex *Byzantine* error scenarios (inconsistent broadcasts) [8]. *Replica determinism in presence of Byzantine errors* is thus an important concern. **(3)** Embedded platforms also have *size, weight, and power constraints*, which must be strictly respected to not adversely affect system performance; deploy more powerful hardware is not usually possible. **(4)** Finally, control engineers are not experts on fault tolerance and vice versa. Thus, separation of concerns, with a *minimal programmer-friendly interface* between the application and the fault tolerance logic, is necessary for ease of adoption.

Prior work fails to tackle all of the aforementioned challenges together. Platforms for building highly reliable flight control systems, such as FTMP [9] achieved high reliability by employing specially-designed hardware, whereas our objective is to obtain high reliability using COTS platforms. More recently, middleware such as RT-CORBA [1] and DDS [2] were designed targeting distributed CPS with real-time requirements but do not consider Byzantine errors. General-purpose Byzantine fault tolerance (BFT) systems, such as RBFT [5], aim to improve the throughput of large-scale distributed systems, but are not well-suited for CPS. These solutions rely on leader-based protocols, but faulty leaders can become performance bottlenecks [4] and violate the hard real-time guarantees.

In contrast to prior work, we take a CPS-centric view towards Byzantine fault tolerance. For the specific problem of replica coordination, we propose to use a synchronous BFT protocol, interface it with NCS applications using a minimal time-aware API, and realize this framework using a hard real-time implementation. Our hypothesis is that such a framework can assist in building highly reliable actively replicated NCS despite the presence of environmentally-induced Byzantine errors. Next, we explain the proposed system, *Achal*, in detail along with the rationale behind our design choices.

2 DESIGN

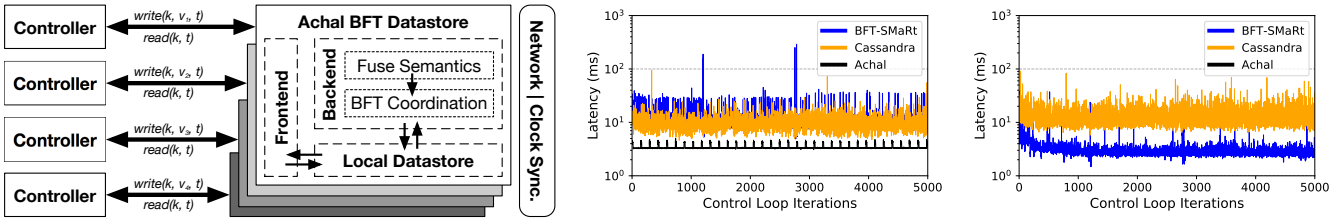
In a nutshell, Achal consists of a local datastore on each physical host. The local datastore interfaces via its frontend with the application replicas deployed on that host, and coordinates via its backend with the datastores local to other hosts, as shown in Fig. 1(a).

To ensure easy and transparent integration with the application, and to allow specification of a variety of temporal constraints on data (which are common in the CPS domain), Achal's frontend exposes a simple time-aware key-value API with the usual read/write semantics of a key-value store, but enhanced with an absolute time parameter *t*. The *write(k, v, t)* API allows programmers to write any value *v* (corresponding to any replica state) to key *k* with an absolute *publishing time t*, which means that the value becomes visible to other application replicas only at time *t*. The *read(k, t)* API returns the latest value *v* with publishing time *no earlier than t* for which consensus has been achieved among the replicas.

In other words, a periodic control loop during each of its iterations can read the global state in the beginning using the read API and write its global state in the end using the write API. The absolute time in case of the read API may refer to the time of the previous control loop iteration, and in case of the write API, it may refer to the time of the next control loop iteration. Thus, if Achal

EMSOFT'19 Companion, October 13–18, 2019, New York, NY, USA

ACM ISBN 978-1-4503-6924-4/19/10…$15.00
https://doi.org/10.1145/3349568.3351545



(a) Achal overview (b) Write latency (c) Read latency

Figure 1: (a) An overview of Achal's architecture. The controllers are actively replicated on independent physical hosts. (b), (c) Latency for writing and reading back a single key. Achal 's read latency was under 100 microseconds, hence not shown.

backends guarantee that all write requests are propagated reliably within their respective publishing times, all application replicas can essentially function in a synchronous manner on identical states.

However, ensuring timely and correct propagation of all write requests despite Byzantine errors is not trivial. For this purpose, Achal's backends rely on a classic leaderless BFT protocol for synchronous networks [11]. A *synchronous network* model, which assumes reliable and timely delivery of messages, is a suitable choice since safety-critical CPS typically use predictable networking standards and clock synchronization protocols. In practice, although messages sent during fault-induced phases of asynchrony would be lost under this assumption, the extent of such message losses can be quantified *a priori* to verify that such cases are extremely rare, *e.g.*, using a probabilistic analysis [12].

In conclusion, while Achal's frontend seeks to provide a generic middleware on which existing NCS applications could be easily and transparently integrated, its backends are designed with a focus on predictable timing properties, which is needed for safety analysis.

3 EVALUATION

We evaluated Achal on a cluster of four Raspberry Pi's (1.4 GHz Cortex-A53 quad-core processor, 1 GB of memory, and Linux kernel 4.14.27). The Pi's were connected over IEEE 802.3ab Gigabit Ethernet using a 1 Gbps Ethernet connection, although the effective throughput was limited to 300 Mbps since the Ethernet controller is internally connected via USB 2.0.

Achal's key-value store was realized using a set of periodic lightweight POSIX processes matching the periodic real-time task model of Liu and Layland [10], which makes it amenable to real-time analysis and validation. In particular, the processes executed different rounds of the synchronous BFT protocol and were scheduled using partitioned fixed-priority scheduling [7]. We compared Achal's performance against BFT-SMaRt [6] and Cassandra [3] (with BFT quorums), both state-of-the-art systems representing the general classes of state-machine replication and BFT quorum-based solutions, respectively. Cassandra and BFT-SMaRt were enhanced to provide equivalent fault-tolerance and time-aware semantics as Achal.

In this initial evaluation, we measure the latency of writing and reading back a single key (see Figs. 1(b) and 1(c), respectively). Achal, BFT-SMaRt, and Cassandra's read and write latency distributions each follow a unique pattern. The write latency of Achal always remains between 3 ms and 5 ms, since it is upper-bounded by the time period of Achal tasks (which was 5 ms in this case). In contrast, the write latency of Cassandra and BFT hovers between 10 ms and 30 ms for a majority of iterations, but exceeds 100 ms occasionally. The read latency of Achal (not shown in the figure) was consistently under 100 μs, since Achal's read operation reads the key from the local datastore and does not require any coordination. The read latency of BFT-SMaRt is also quite low (a couple of milliseconds). In contrast, Cassandra 's read latency is significant, averaging in excess of 10 ms, which we attribute to its QUORUM consistency level.

In summary, initial results indicate that on embedded platforms with limited CPU, memory, and network resources, Achal is more efficient than the state-of-the-art server-scale systems BFT-SMaRt and Cassandra. Most importantly, Achal's low latency and variance helps in validating temporal constraints prior to deployment, which in turn would be helpful when targeting high-frequency applications with strict timing constraints.

4 FUTURE WORK

We plan to validate Achal's design with further experiments. We plan to evaluate Achal's scalability with respect to the number of keys, size of keys, and application frequency. We also plan to investigate the challenges involved in porting an existing NCS application to Achal, and in enhancing the reliability of an existing standalone control system with active replication using Achal.

REFERENCES

[1] 2005. Real-time CORBA Specification. https://www.omg.org/spec/RT/1.2/
[2] 2015. Data Distribution Service Specification. https://www.omg.org/spec/DDS/
[3] 2018. Apache Cassandra. https://cassandra.apache.org/
[4] Y. Amir, B. Coan, J. Kirsch, and J. Lane. 2008. Byzantine replication under attack. In *DSN 2008*.
[5] P. Aublin, S. B. Mokhtar, and V. Quema. 2013. RBFT: Redundant Byzantine Fault Tolerance. In *ICDCS 2013*.
[6] Alysson Bessani, Joao Sousa, and Eduardo E.P. Alchieri. 2014. State Machine Replication for the Masses with BFT-SMART. In *DSN 2014*.
[7] R. I. Davis and A. Burns. 2011. A survey of hard real-time scheduling for multiprocessor systems. *Comput. Surveys* 43, 4 (2011), 1–44.
[8] K. Driscoll, B. Hall, H. Sivencrona, and P. Zumsteg. 2003. Byzantine Fault Tolerance, from Theory to Reality. In *Computer Safety, Reliability, and Security*. Vol. 2788. Springer Berlin Heidelberg, 235–248.
[9] A.L. Hopkins, T.B. Smith, and J.H. Lala. 1978. FTMP—A highly reliable fault-tolerant multiprocess for aircraft. *Proc. IEEE* 66, 10 (1978), 1221–1239.
[10] C. L. Liu and James W. Layland. 1973. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM* 20, 1 (1973), 46–61.
[11] M. Pease, R. Shostak, and L. Lamport. 1980. Reaching Agreement in the Presence of Faults. *J. ACM* 27, 2 (1980), 228–234.
[12] F. Smirnov, M. Glaß, F. Reimann, and J. Teich. 2016. Formal reliability analysis of switched ethernet automotive networks under transient transmission errors. In *DAC 2016*. Austin, Texas.