Byzantine fault tolerance

From Wikipedia, the free encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
العربية
Deutsch
فارسی
Français
한국어
Italiano
日本語
Polski
Русский
Svenska
Українська
中文
Edit links

> *"Byzantine generals" redirects here. For actual military generals of the Byzantine empire, see Category:Byzantine generals.*

In fault-tolerant computer systems, and in particular distributed computing systems, **Byzantine fault tolerance** is the characteristic of a system that tolerates the class of failures known as the Byzantine Generals' Problem,[1] which is a generalized version of the Two Generals' Problem. The phrases *interactive consistency* or *source congruency* have been used to refer to **Byzantine fault tolerance**, particularly among the members of some early implementation teams.[2]

The objective of Byzantine fault tolerance is to be able to defend against *Byzantine failures*, in which components of a system fail with symptoms that prevent some components of the system from reaching agreement among themselves, where such agreement is needed for the correct operation of the system. Correctly functioning components of a Byzantine fault tolerant system will be able to provide the system's service assuming there are not too many faulty components.

The following practical, concise definitions are helpful in understanding Byzantine fault tolerance:[3] [4]

Byzantine fault

Any fault presenting different symptoms to different observers

Byzantine failure

The loss of a system service due to a Byzantine fault in systems that require consensus

The terms *fault* and *failure* are used here according to the standard definitions[5] originally created by a joint committee on "Fundamental Concepts and Terminology" formed by the IEEE Computer Society's Technical Committee on Dependable Computing and Fault-Tolerance and IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance.[6] A version of these definitions is also described in the Dependability Wikipedia page.

Note that the type of system services which Byzantine faults affect are agreement (a.k.a consensus) services.

Contents [hide]

Origin [edit]

Byzantine refers to the Byzantine Generals' Problem, an agreement problem (described by Leslie Lamport, Robert Shostak and Marshall Pease in their 1982 paper, "The Byzantine Generals Problem")[1] in which a group of generals, each commanding a portion of the Byzantine army, encircle a city. These generals wish to formulate a plan for attacking the city. In its simplest form, the generals must only decide whether to attack or retreat. Some generals may prefer to attack, while others prefer to retreat. The important thing is that every general agrees on a common decision, for a halfhearted attack by a few generals would become a rout and be worse than a coordinated attack or a coordinated retreat.

The problem is complicated by the presence of traitorous generals who may not only cast a vote for a suboptimal strategy, they may do so selectively. For instance, if nine generals are voting, four of whom support attacking while four others are in favor of retreat, the ninth general may send a vote of retreat to those generals in favor of retreat, and a vote of attack to the rest. Those who received a retreat vote from the ninth general will retreat, while the rest will attack (which may not go well for the attackers). The problem is complicated further by

the generals being physically separated and must send their votes via messengers who may fail to deliver votes or may forge false votes.

Byzantine fault tolerance can be achieved if the loyal (non-faulty) generals have a unanimous agreement on their strategy. Note that there can be a default vote value given to missing messages. For example, missing messages can be given the value <Null>. Further, if the agreement is that the <Null> votes are in the majority, a pre-assigned default strategy can be used (e.g., retreat).

The typical mapping of this story on to computer systems is that the computers are the generals and their digital communication system links are the messengers.

Known examples of Byzantine failures [edit]

Several examples of Byzantine failures that have occurred are given in two equivalent journal papers.[3][4] These and other examples are described on the NASA DASHlink web pages.[7] These web pages also describe some phenomenology that can cause Byzantine faults.

Byzantine errors were observed infrequently and at irregular points during endurance testing for the New Virginia Class submarine.[8]

Early solutions [edit]

Several solutions were described by Lamport, Shostak, and Pease in 1982.[1] They began by noting that the Generals' Problem can be reduced to solving a "Commander and Lieutenants" problem where Loyal Lieutenants must all act in unison and that their action must correspond to what the Commander ordered in the case that the Commander is Loyal.

- One solution considers scenarios in which messages may be forged, but which will be *Byzantine-fault-tolerant* as long as the number of traitorous generals does not equal or exceed one third of the generals. The impossibility of dealing with one-third or more traitors ultimately reduces to proving that the one Commander and two Lieutenants problem cannot be solved, if the Commander is traitorous. To see this, suppose we have a traitorous Commander A, and two Lieutenants, B and C: when A tells B to attack and C to retreat, and B and C send messages to each other, forwarding A's message, neither B nor C can figure out who is the traitor, since it is not necessarily A—another Commander could have forged the message purportedly from A. It can be shown that if n is the number of generals in total, and t is the number of traitors in that n, then there are solutions to the problem only when $n > 3t$ and the communication is synchronous (bounded delay).[9]
- A second solution requires unforgeable message signatures. For security-critical systems, digital signatures (in modern computer systems, this may be achieved in practice using public-key cryptography) can provide Byzantine fault tolerance in the presence of an arbitrary number of traitorous generals. However, for safety-critical systems, simple error detecting codes, such as CRCs, provide the same or better coverage at a much lower cost. This is true for both Byzantine and non-Byzantine faults. Thus, cryptographic digital signature methods are not a good choice for safety-critical systems, unless there is also a specific security threat as well.[10] While error detecting codes, such as CRCs, are better than cryptographic techniques, neither provide adequate coverage for active electronics in safety-critical systems. This is illustrated by the *Schrödinger CRC* scenario where a CRC-protected message with a single Byzantine faulty bit presents different data to different observers and each observer sees a valid CRC.[3][4]
- Also presented is a variation on the first two solutions allowing Byzantine-fault-tolerant behavior in some situations where not all generals can communicate directly with each other.

Several system architectures were designed c. 1980 that implemented Byzantine fault tolerance. These include: Draper's FTMP,[11] Honeywell's MMFCS,[12] and SRI's SIFT.[13]

Practical Byzantine fault tolerance [edit]

In 1999, Miguel Castro and Barbara Liskov introduced the "Practical Byzantine Fault Tolerance" (PBFT) algorithm,[14] which provides high-performance Byzantine state machine replication, processing thousands of requests per second with sub-millisecond increases in latency.

PBFT triggered a renaissance in Byzantine fault tolerant replication research[*citation needed*], with protocols like Q/U,[15] HQ,[16] Zyzzyva,[17] and ABsTRACTs [18] working to lower costs and improve performance and protocols like Aardvark[19] and RBFT[20] working to improve robustness.

Byzantine fault tolerance software [edit]

UpRight[21] is an open source library for constructing services that tolerate both crashes ("up") and Byzantine behaviors ("right") that incorporates many of these protocols' innovations.

In addition to PBFT and Upright, there is the BFT-SMaRt library,[22] a high-performance Byzantine fault-tolerant state machine replication library developed in Java. This library implements a protocol very similar to PBFT's, plus complementary protocols which offer state transfer and on-the-fly reconfiguration of hosts. BFT-SMaRt is the most recent effort to implement state machine replication, still being actively maintained.

Archistar[23] utilizes a slim BFT layer[24] for communication. It prototypes a secure multi-cloud storage system using Java licensed under LGPLv2. Focus lies on simplicity and readability, it aims to be the foundation for further research projects.

Byzantine fault tolerance in practice [edit]

One example of BFT in use is bitcoin, a peer-to-peer digital currency system. The bitcoin network works in parallel to generate a chain of Hashcash style proof-of-work. The proof-of-work chain is the key to overcome Byzantine failures and to reach a coherent global view of the system state.

Some aircraft systems, such as the Boeing 777 Aircraft Information Management System (via its ARINC 659 SAFEbus® network),[25] [26] the Boeing 777 flight control system,[27] and the Boeing 787 flight control systems, use Byzantine fault tolerance. Because these are real-time systems, their Byzantine fault tolerance solutions must have very low latency. For example, SAFEbus can achieve Byzantine fault tolerance with on the order of a microsecond of added latency.

Some spacecraft such as the SpaceX Dragon flight system [1] and the NASA Crew Exploration Vehicle [2] consider Byzantine fault tolerance in their design.

Byzantine fault tolerance mechanisms use components that repeat an incoming message (or just its signature) to other recipients of that incoming message. All these mechanisms make the assumption that the act of repeating a message blocks the propagation of Byzantine symptoms. For systems that have a high degree of safety or security criticality, these assumptions must be proven to be true to an acceptable level of fault coverage. When providing proof through testing, one difficulty is creating a sufficiently wide range of signals with Byzantine symptoms.[28] Such testing likely will require specialized fault injectors.[29][30]

See also [edit]

- Atomic commit
- Brooks–Iyengar algorithm
- Byzantine Paxos
- Consensus (computer science)
- Quantum Byzantine agreement

References [edit]

1. ^ a b c Lamport, L.; Shostak, R.; Pease, M. (1982). "The Byzantine Generals Problem" (PDF). *ACM Transactions on Programming Languages and Systems* **4** (3): 382–401. doi:10.1145/357172.357176.
2. ^ Kirrmann, Hubert (n.d.). "Fault Tolerant Computing in Industrial Automation" (PDF). CH-5405 Baden, Switzerland: ABB Research Center. p. 94. Retrieved 2015-03-02.
3. ^ a b c Driscoll, K.; Hall, B.; Paulitsch, M.; Zumsteg, P.; Sivencrona, H. (2004). "The Real Byzantine Generals". pp. 6.D.4–61–11. doi:10.1109/DASC.2004.1390734.
4. ^ a b c Driscoll, Kevin; Hall, Brendan; Sivencrona, Håkan; Zumsteg, Phil (2003). "Byzantine Fault Tolerance, from Theory to Reality" **2788**. pp. 235–248. doi:10.1007/978-3-540-39878-3_19. ISSN 0302-9743.
5. ^ Avizienis, A.; Laprie, J.-C.; Randell, B.; Landwehr, C. (2004). "Basic concepts and taxonomy of dependable and secure computing". *IEEE Transactions on Dependable and Secure Computing* **1** (1): 11–33. doi:10.1109/TDSC.2004.2. ISSN 1545-5971.
6. ^ "Dependable Computing and Fault Tolerance". Retrieved 2015-03-02.
7. ^ Driscoll, Kevin (2012-12-11). "Real System Failures". *DASHlink*. NASA. Retrieved 2015-03-02.
8. ^ Walter, C.; Ellis, P.; LaValley, B. (2005). "The Reliable Platform Service: A Property-Based Fault Tolerant Service Architecture". pp. 34–43. doi:10.1109/HASE.2005.23.
9. ^ Feldman, P.; Micali, S. (1997). "An optimal probabilistic protocol for synchronous Byzantine agreement" (PDF). *SIAM J. Computing* **26** (4): 873–933. doi:10.1137/s0097539790187084.
10. ^ Paulitsch, M.; Morris, J.; Hall, B.; Driscoll, K.; Latronico, E.; Koopman, P. (2005). "Coverage and the Use of Cyclic Redundancy Codes in Ultra-Dependable Systems". pp. 346–355. doi:10.1109/DSN.2005.31.

11. ^ Hopkins, Albert L.; Lala, Jaynarayan H.; Smith, T. Basil (1987). "The Evolution of Fault Tolerant Computing at the Charles Stark Draper Laboratory, 1955–85" **1**. pp. 121–140. doi:10.1007/978-3-7091-8871-2_6. ISSN 0932-5581.
12. ^ Driscoll, Kevin; Papadopoulos, Gregory; Nelson, Scott; Hartmann, Gary; Ramohalli, Gautham (1984), *Multi-Microprocessor Flight Control System* (Technical Report), Wright-Patterson Air Force Base, OH 45433, USA: AFWAL/FIGL U.S. Air Force Systems Command, AFWAL-TR-84-3076
13. ^ "SIFT: design and analysis of a fault-tolerant computer for aircraft control". *Microelectronics Reliability* **19** (3): 190. 1979. doi:10.1016/0026-2714(79)90211-7. ISSN 0026-2714.
14. ^ Castro, M.; Liskov, B. (2002). "Practical Byzantine Fault Tolerance and Proactive Recovery". *ACM Transactions on Computer Systems* (Association for Computing Machinery) **20** (4): 398–461. doi:10.1145/571637.571640. CiteSeerX: 10.1.1.127.6130.
15. ^ Abd-El-Malek; Ganger, G.; Goodson, G.; Reiter, M.; Wylie, J. (2005). "Fault-scalable Byzantine Fault-Tolerant Services". Association for Computing Machinery. doi:10.1145/1095809.1095817.
16. ^ Cowling, James; Myers, Daniel; Liskov, Barbara; Rodrigues, Rodrigo; Shrira, Liuba (2006). *HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance*. Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation. pp. 177–190. ISBN 1-931971-47-1.
17. ^ Kotla, Ramakrishna; Alvisi, Lorenzo; Dahlin, Mike; Clement, Allen; Wong, Edmund (December 2009). "Zyzzyva: Speculative Byzantine Fault Tolerance". *ACM Transactions on Computer Systems* (Association for Computing Machinery) **27** (4). doi:10.1145/1658357.1658358.
18. ^ Guerraoui, Rachid; Kneževic, Nikola; Vukolic, Marko; Quéma, Vivien (2010). *The Next 700 BFT Protocols*. Proceedings of the 5th European conference on Computer systems. EuroSys.
19. ^ Clement, A.; Wong, E.; Alvisi, L.; Dahlin, M.; Marchetti, M. (April 22–24, 2009). *Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults* (PDF). Symposium on Networked Systems Design and Implementation. USENIX.
20. ^ Aublin, P.-L.; Ben Mokhtar, S.; Quéma, V. (July 8–11, 2013). *RBFT: Redundant Byzantine Fault Tolerance*. 33rd IEEE International Conference on Distributed Computing Systems. International Conference on Distributed Computing Systems.
21. ^ UpRight. Google Code repository for the UpRight replication library.
22. ^ BFT-SMaRt. Google Code repository for the BFT-SMaRt replication library.
23. ^ Archistar. github repository for the Archistar project.
24. ^ Archistar-bft BFT state-machine. github repository for the Archistar project.
25. ^ M., Paulitsch; Driscoll, K. (9 January 2015). "Chapter 48:SAFEbus". In Zurawski, Richard. *Industrial Communication Technology Handbook, Second Edition*. CRC Press. pp. 48–1–48–26. ISBN 978-1-4822-0733-0.
26. ^ Thomas A. Henzinger; Christoph M. Kirsch (26 September 2001). *Embedded Software: First International Workshop, EMSOFT 2001, Tahoe City, CA, USA, October 8-10, 2001. Proceedings* (PDF). Springer Science & Business Media. pp. 307–. ISBN 978-3-540-42673-8.
27. ^ Yeh, Y.C. (2001). "Safety critical avionics for the 777 primary flight controls system" **1**. pp. 1C2/1–1C2/11. doi:10.1109/DASC.2001.963311.
28. ^ Nanya, T.; Goosen, H.A. (1989). "The Byzantine hardware fault model". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **8** (11): 1226–1231. doi:10.1109/43.41508. ISSN 0278-0070.
29. ^ Martins, Rolando; Gandhi, Rajeev; Narasimhan, Priya; Pertet, Soila; Casimiro, António; Kreutz, Diego; Veríssimo, Paulo (2013). "Experiences with Fault-Injection in a Byzantine Fault-Tolerant Protocol" **8275**. pp. 41–61. doi:10.1007/978-3-642-45065-5_3. ISSN 0302-9743.
30. ^ US patent 7475318, Kevin R. Driscoll, "Method for testing the sensitive input range of Byzantine filters", issued 2009-01-06, assigned to Honeywell International Inc.

External links [edit]

- Ocean Store replicates data with a Byzantine fault tolerant commit protocol.
- Practical Byzantine Fault Tolerance
- Byzantine Fault Tolerance in the RKBExplorer
- UpRight is an open source library for Crash-tolerant and Byzantine-tolerant state machine replication.

Categories: Public-key cryptography | Distributed computing problems | Fault tolerance | Fault-tolerant computer systems | Failure | Theory of computation