Module 5 Capstone Milestone: Fault-tolerant Software

Alexander Ricciardi

Colorado State University Global

CSC480: Capstone Computer Science

Dr. Shaher Daoud

July 13, 2025

© 2025 Alexander Samuel Ricciardi — CC BY 4.0
SPDX-License-Identifier: Apache-2.0

Module 5 Capstone Milestone: Fault-tolerant Software

As early as 1984, a study by Slivinski et al. (1984) identified software faults as the most significant issue in software systems. Today, designing fault-tolerant software is essential, as software applications have become larger and more complex, and consequently more prone to faults. As a result, fault tolerance is now an integral part of software design, making software systems more reliable and resilient. This paper examines how fault tolerance is implemented in modern software and describes how it is implemented with the Mining Regulatory Compliance Assistant (MRCA) project, within its microservices architecture and Advanced Parallel HybridRAG (APH) system.

Fault-Tolerant Software

Fault-tolerant software is software designed to detect and recover from faults that are happening or have already happened in either the software or hardware (Inacio, 1998). In other words, the primary goal of fault-tolerance systems is to increase a software application's dependability despite internal faults or failures (Solouki et al., 2024). On the other hand, fault-tolerant software design is a set of principles, strategies, and architectural patterns used to build software systems that can detect and recover from faults or failures (Andersen, 2024). The primary objective of fault-tolerant software design is to design a reliable and resilient software system. The key fault-tolerant software principles are resilience, redundancy, error detection and correction, checkpointing (periodically saving system state), and continuous operation (Lee, 2025; Gopinath, 1986; GeeksforGeeks, 2024). The key fault-tolerant software design techniques are leveraging microservices architecture for improved resilience; using graceful degradation components for continuous operation after faults or failures; adding redundancy by having additional components or systems that can take over in case of a failure; load balancing for

© 2025 Alexander Samuel Ricciardi — CC BY 4.0
SPDX-License-Identifier: Apache-2.0

distributing incoming network traffic across multiple servers; implementing health check mechanisms for crucial components; using foundation patterns such as publish-subscribe pattern or service-oriented architecture pattern to promote testing and interoperability; and implementing a circuit breaker system to prevent cascading failures (ByteByteGo, 2024; Keeling, 2027; Andersen, 2024).

Fault Tolerance in MRCA

MRCA employed various fault-tolerant techniques that are woven into its microservices architecture. To start with, a circuit breaker system is implemented through the `CircuitBreaker` class found in the `circuit_breaker.py` module. The class is used to monitor the health of external dependencies, which include Neo4j database connections, OpenAI API calls, and Google Gemini API calls. It handles failure thresholds, timeouts, and recovery-testing mechanisms. MRCA has an integrated health monitoring system that functions across multiple layers. This system is implemented using FastAPI framework, which handles the backend and provides both basic health checks for the overall system, `/health`, and specific core components like the APH core components (fusion, and parallel engine), `parallel_hybrid/health`. The RAG tools (vector search, cypher queries, and general guidance) have their own dedicated health check functions. Moreover, the Docker containers (frontend and backend services) include their own health check that enables automatic restart if a service fails.

In addition to the circuit breaker and health monitoring systems, MRCA has a safe tool getter system where each RAG tool comes with a fallback functionality option when the primary tool is unavailable. The `get_vector_tool_safe()` and `get_cypher_tool_safe()` functions return fallback functions with informative error messages rather than allowing the system to crash, ensuring that the frontend remains functional even when backend tools fail. The frontend,

© 2025 Alexander Samuel Ricciardi — CC BY 4.0
SPDX-License-Identifier: Apache-2.0

through Streamlit implements a persistent session management module that remembers conversation history and settings even if API requests fail. Additionally, API errors are handled by `call_parallel_hybrid_api()` function in the `frontend/bot.py`. Furthermore, MRCA uses the Pydantic Python library to (data) request and reject malformed requests. Moreover, the `ParallelHybridRequest` class in the `backend/main.py` module is used to validate user inputs, session identifiers, fusion strategies, and template types. This prevents invalid data from propagating through and being used by the APH system.

Conclusion

Fault-Tolerant Software is reliable and resilient software that is designed to withstand faults and failures, whether they are caused by software bugs, hardware malfunctions, or external factors. Such software implements redundancy, error detection and correction, checkpointing, and continuous operation within its architecture and implementation. This software is designed using techniques, patterns, and architecture such as microservices, graceful degradation, load balancing, health monitoring, foundational patterns, and circuit breakers. MRCA implements most of these strategies and techniques deep within its design and implementation. In essence, MRCA integrates a fault-tolerance design and techniques from the ground up; fault-tolerance is woven into its microservices architecture and implementation, ensuring that the system is reliable and resilient.

© 2025 Alexander Samuel Ricciardi — CC BY 4.0
SPDX-License-Identifier: Apache-2.0

References

Andersen, G. (2024, February 8). *Building resilient systems – A guide to fault-tolerant software architecture.* MoldStud. https://moldstud.com/articles/p-building-resilient-systems-through-fault-tolerant-software-architecture

ByteByteGo. (2024, February 14). *A cheat sheet for designing fault-tolerant systems*. ByteByteGo. https://bytebytego.com/guides/a-cheat-sheet-for-designing-fault-tolerant-systems/

GeeksforGeeks. (2024, June 19). *Basic fault-tolerant software techniques*. GeeksforGeeks. https://www.geeksforgeeks.org/software-engineering/basic-fault-tolerant-software-techniques/

Gopinath, D. (1986). *A tutorial on the principles of fault tolerance*. Sādhanā, 11(1–2), 7–22. https://www.ias.ac.in/article/fulltext/sadh/011/01-02/0007-0022

Keeling, M. (2017). Chapter 7: Create a Foundation with Patterns. *Design it! From programmer to software architect*. Pragmatic Bookshelf. ISBN-13: 978-1-680-50209-1

Lee, S. (2025, June 11). *Fault Tolerance 101: A Beginner's Guide*. Number Analytics. https://www.numberanalytics.com/blog/fault-tolerance-101

Inacio, C. (1998). *Software fault tolerance - Electrical and computer engineering* [Course]. Carnegie Mellon University. https://users.ece.cmu.edu/~koopman/des_s99/sw_fault_tolerance/

Slivinski, T., Broglio, C., Wild, C., Goldberg, J., Levitt, K., Hitt, E., & Webb, J. (1984). *Study of fault-tolerant software technology* (NASA Contractor Report 172385). National Aeronautics and Space Administration, Langley Research Center. https://ntrs.nasa.gov/api/citations/19870002074/downloads/19870002074.pdf

© 2025 Alexander Samuel Ricciardi — CC BY 4.0
SPDX-License-Identifier: Apache-2.0

Solouki, M. A., Angizi, S., & Violante, M. (2024, April 16). *Dependability in embedded systems: A survey of fault tolerance methods and software-based mitigation techniques* (arXiv:2404.10509v1) [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2404.10509

© 2025 Alexander Samuel Ricciardi — CC BY 4.0
SPDX-License-Identifier: Apache-2.0