Distributed Dependability

18-849b Dependable Embedded Systems

Robert Slater

February 11, 1999

Required Reading: Bates,I.J.and Burns A. "A dependable Architecture for a Safety Critical Hard Real-Time System." IEEE Half-day Colloquium on Hardware Systems for Dependable Applications, p.30 1/1-6

Kopetz, H. "The Time-Triggered Architecture." Proceedings of the First International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '98) p. xiv+485 22-9

Best Tutorial: Mullender, Sape ed. "Distributed Systems." ACM Press 1989

Authoritative Books: Chow, Randy and Theodore Johnson "Distributed Operating Systems and Algorithms." Addison-Wesley 1998

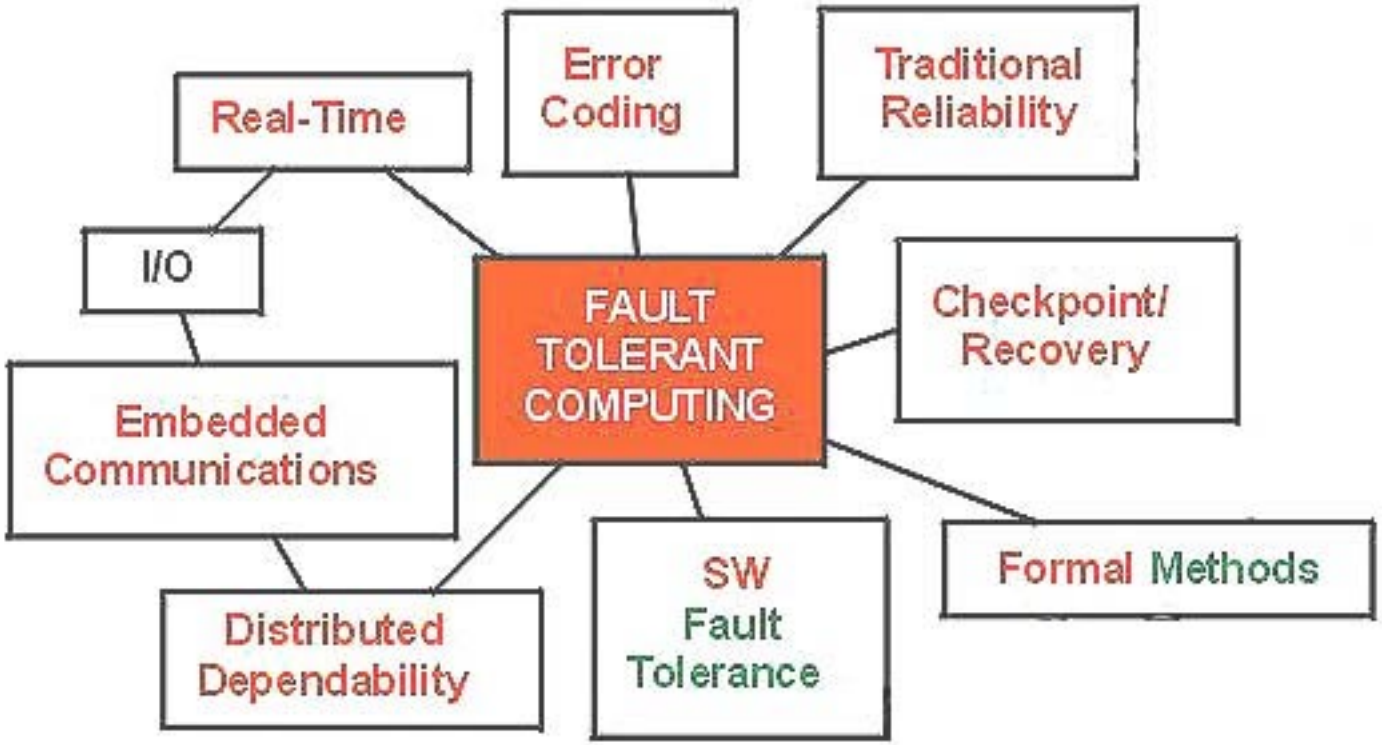
Kopetz, Hermann "Real Time Systems: Design Principles for Distributed Applications." Kluwer Academic Publishers 1997

Carnegie Mellon

Overview: Distributed Dependability

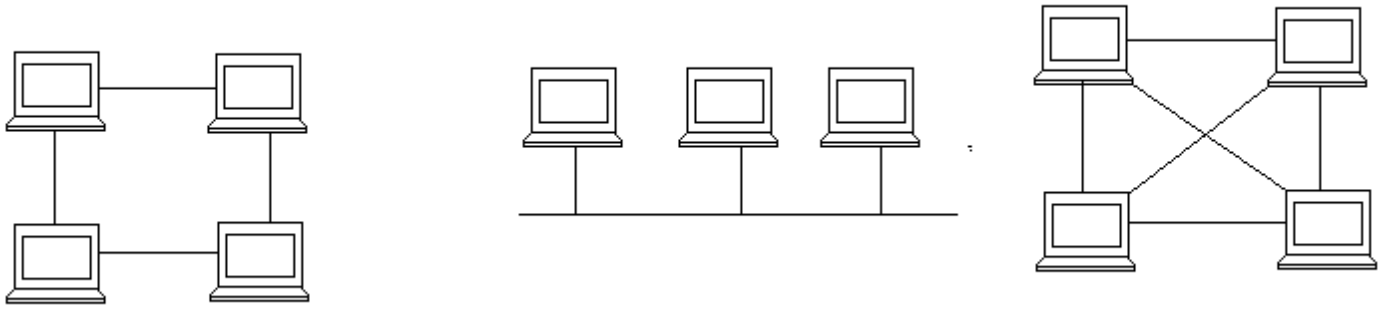
- **Introduction**
  - What is a distributed system?
- **Key concepts**
  - Reliability : synchronization & coordination
  - Availability : voting, the Byzantine generals, & checkpointing
  - Security: authentication & encryption
- **Tools**
  - Software tools/environments
- **Relationship to other topics**
  - Real-time, Embedded communication, Fault-tolerant computing
- **Conclusions & future work**

YOU ARE HERE MAP



What are Distributed Systems?

- Distributed systems coordinate computations between multiple processors/nodes



- Multiple Computational Nodes
- Connections Between Nodes
- Data/process Sharing
- Appears as a single node at any one node

Reliability - how do we make it work right?

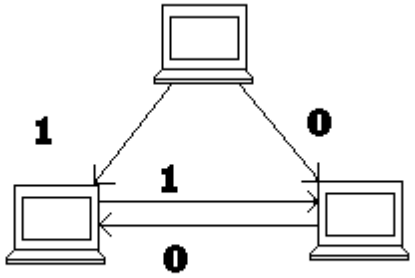
- **Synchronization is necessary for ordered operation**
  - Asynchronous systems must coordinate through system artifacts
  - Synchronous systems must provide a global clock
- **Coordination is necessary to avoid conflicts**
  - Mutual exclusion techniques ; semaphores, monitors, mutexes
  - Communication protocols ; time-shared, polled, arbitrated
  - Transactions ; double- and triple- commit
- **One-Copy Semantics make distribution transparent**
  - Provides view as if single copy of data
  - If backups exist, then all copies are consistent
- **Allocates nodes to processes as necessary to complete**
  - But scheduling becomes more complicated

Availability- how do we keep it working?

- **Depends on your failure mode**
  - If fail-silent, can use simple detection & masking to continue operation
  - If more complex failure mode (i.e. fail-consistent, fail-malicious) then requires complicated fault masking
  - The key is containment and masking
- **The simple stuff**
  - Timeout, re-try
  - Requires n+1 nodes to mask out n faults
- **Voting**
  - Use redundancy to mask out fail-consistent failures
  - Requires 2n+1 nodes to mask out n faults

Byzantine Algorithms

- **What about malicious failures?**



- **Byzantine algorithms compare messages received at all nodes**
  - Requires 3n+1 nodes & n+1 messages at each node to mask n faults if origin of messages not known
  - Requires n+1 nodes if origin of messages known
- **Always remember design redundancy!**

Security- how do we protect our system?

- **Keeps authorized users and their data safe**
- **Authentication**
  - Passwords, expiring tokens, and public keys
  - Always ways for this to become insecure
- **Protection**
  - Provide limited access to resources based on identity
  - Must insure secrecy, privacy, data authenticity, and data integrity
  - Must avoid intruder browsing, leaking, inferencing, or masquerading
  - Generally use access matrix to match identity/access level to rights
  - Depends upon identity to ensure protection, therefore insecure

Software Tools

- **CORBA**
  - Abstracts distributed system into objects and references
  - Supports process migration through naming service
  - Hides network details behind abstraction
  - Implementations for embedded systems underway
- **DCOM**
  - Allows objects to connect to other objects through interfaces
  - More of a desktop solution than an embedded one
  - It is Microsoft's solution, may be more common as Windows spreads

Relationship To Other Topic Areas

- **Embedded Communication**
  - Requires adaptation to the communication protocol
  - Embodies functionality above that of the network
- **Real-time**
  - Often used for real-time systems
  - Scheduling problems similar
- **Fault-Tolerant Computing**
  - Provides redundancy necessary to fault-tolerant computing
  - Shows the Reliability vs. Availability balance
- **Checkpoint/Recovery**
  - Often uses checkpointing in saving and recovering state

Conclusions & Future Work

- **Used to get cheap computation and bring it closer to where it is needed**
  - Centralized computing costs more for similar capabilities, and communication overhead is greater
- **High availability is a possible benefit, but must be worked into the system**
  - usually at the cost of more complicated reliability
- **Distributed systems are powerful for the right kinds of problems**
  - Make sure you need the availability and processing power enough to justify the overhead
- **Future work**
  - Better synchronization, fault tolerance, and security measures
  - Better matching of distributed solution to problems
  - Connecting to the Internet

Bate & Burns and Kopetz

- **One high-level and one design**
- **Bate and Burns**
  - Highlights the major design issues
  - But there's a lot of hand waving
- **Kopetz and the TTA**
  - Correlates with B&B on a lot of areas
  - Provides examples of design elements
  - Fault-tolerance built into every part of the system
  - Obviously embedded : No security discussion