# An Introduction to the high-availability and high-consistency problem

**Ignacio Laguna, Fahad Arshad**
Dependable Computing Systems Laboratory
Purdue University

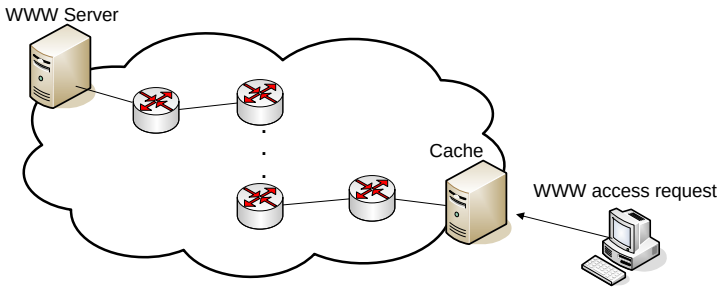
Aug 29, 2007

# Reasons for Replication

- Two primary reasons for replication: **reliability** and **performance**.
- **Increasing reliability**:
  - If a replica crashes, system can continue working by switching to other replicas.
  - Avoid corrupted data:
    - can protect against a single, failing write operation.
- **Improving performance**:
  - Important for distributed systems over large geographical areas.
  - Divide the work over a number of servers.
  - Place data in the proximity of clients.

# Replication helps, so what is the problem?

- There is a price to be paid when replicating: **to maintain consistency**.
- Whenever a copy is modified, it becomes different from the rest.






# Consistency models

- Consistency is discussed in the context of **read** and **write** operations.
- Operations are propagated over a shared **data object** or **data store**.
- A write operation is any operation that changes the data.
- A consistency model is a contract between processes and the data store.

# Examples of consistency models

- Continuous consistency [Yu and Vahdat, 2002].
  - Deviation in numerical values between replicas,
    - For example, number of updates applied to a given replica, but not seen by others.
    - Stock market applications: two copies should not deviate more than $0.02.
  - Deviation in staleness,
  - Deviation with respect to ordering of writes.
- Other models: client-centric models, consistent ordering models (sequential consistency, casual consistency)

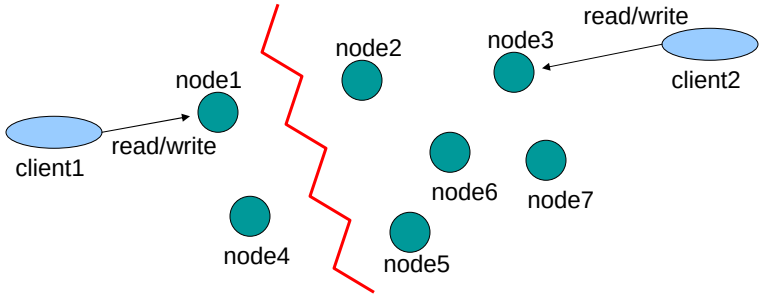


# Consistency protocols

- Primary-based protocols [Budhiraja, 1993]:
  - Each data item *x* in the data store has an associated primary (e.g. server).
  - The primary is responsible for coordinating write operations on *x*.
  - There exist blocking and non-blocking protocols.
- Quorum-based protocols [Gifford, 1979]:
  - Clients require permission of multiple servers for reading and writing.
- Continuous consistency [Yu and Vahdat, 2002]
  *(see next week…)*

# Availability-and-Consistency dilemma

- Systems cannot **simultaneously** achieve both high **C**onsistency and high **A**vailability if they are subject to network **P**artitions (CAP) [Davidson, 1985].






# The CAP principle

- **Strong consistency**: system should be able to provide updates.
- **High availability**: any consumer of data can always reach some replica.
- **Partition resilience**: the system can survive network partitions
- CAP: Strong **C**onsistency, High **A**vailability, **P**artition-resilience: Pick at most 2!

# Examples of the CAP principle

- Easier to find examples than to prove it.

Some examples:

  - **CA without P**: databases with distributed transactional semantics. They only works in the absence of network partitions.
  - **CP without A**: Some transactions may be blocked in the event of network partitions.
  - **AP without C**: Web caching of replicated documents. In a network partition, clients cannot verify freshness of documents.

- In practice, many applications can be described in terms of: *reduced* **consistency** or **availability**.



# Harvest and Yield

[Fox and Brewer, HotOS 1999]

- Proposed as a new research area in the paper.
- The goal was to **improve availability** at large scale by exploiting the tradeoffs of the CAP principle.
- **Assumptions:**
  - Clients make queries to servers.
  - Two metrics for correct behavior:

  **(1) Yield**: the probability of completing the request

  **(2) Harvest**: the completeness of the answer to the query

- Yield is the common metric, typically measured in "nines":
  - "four-nines availability" means P(completion) = 0.9999.

# Harvest and Yield

- Tradeoff between providing:
  - **no answer** (reducing yield)
  - **Imperfect answer** (maintaining yield, reducing harvest)
- Some applications do not tolerate harvest degradation:
  - For example: a sensor application that must provide a binary sensor reading.
- Other applications tolerates some harvest degradation:
  - For example, online aggregation, etc.



# Strategy 1: Trading Harvest for Yield

- This is called **probabilistic Availability**.
- Nearly all systems are probabilistic:
  - A system that is 100% available under single faults is probabilistically available overall (there is nonzero probability of multiple failures).
  - Internet servers depend on the best-effort Internet for true availability.
- Example:
  - Node faults in the Inktomi search engine remove a fraction of the search database.
  - In a cluster of 100 nodes, a single-node fault reduces harvest by 1% (during the duration of the fault).
  - It is assumed that data is placed randomly in the nodes of the search database.

# Strategy 2: Application Decomposition

- Some large applications can be decomposed into subsystems
- If a subsystem fails, yield is reduced. But independent failures allows the application to continue working (with reduced utility).
- The application as a whole is tolerant of harvest degradation.
- Actual benefit is the ability to providing strong consistency to the subsystems that need it, not for the entire application.



# Example for strategy 2

- Example: Consider an e-commerce site that has:
  - A read-only subsystem (user info),
  - A transactional subsystem (billing),
  - A subsystem that manages states over user sessions (shopping cart)
  - A subsystem that manages read-mostly / write-rarely states (user personalization profiles).
- Any subsystem can fail without rendering the whole service useless (except possibly billing):
  - If the user profile store fails, user may still browse merchandise (but without personalized presentation).
  - If the shopping cart subsystems fails, one-at-a-time purchases are still possible.

# Review

- There is a price for replicating, either for reliability or performance: maintaining consistency of data.
- Some applications require strong consistency, while others can tolerate deviated consistency.
- CAP dilemma: choose only 2.
- Trading harvest for yield, or use application decomposition (orthogonal programming).



# References


- Slides 1-6 taken from “Distributed Systems: Principles and Paradigms”, A. Tanenbaum, M. Van Steen, 2nd Edition, 2007.
- Slides 8-15 taken from A. Fox, and E. A. Brewer, "Harvest, Yield, and Scalable Tolerant Systems", in Proceedings of HotOS-VII, March 1999.
- **Other references:**
- [Yu and Vahdat, 2002]: H. Yu and A. Vahdat, “Design and Evaluation of a Conit-Based Continuous Consistency Model for Replicated Services,” ACM TOCS, Vol. 20, Issue 3, Aug 2002.
- [Budhiraja, 1993]: N. Budhiraja, K. Marzullo, F. B. Schneider, S. Toueg, "The primary-backup approach," In Distributed Systems, 2ed Edition, S. Mullender, editor, pp. 199--216, Addison-Wesley, 1993.
- [Gifford, 1979]: D. K. Gifford, "Weighted Voting for Replicated Data", 7th SOSP, December 1979
- [Davidson, 1985]: S. B. Davidson, H. Garcia-Molina, D. Skeen, "Consistency in a partitioned network: a survey," ACM Computing Surveys (CSUR), Volume 17 , Issue 3, September 1985.