Distributed Systems
CPS 512
Spring 2020

Instructor: Jeff Chase

CPS 512 focuses on core concepts in distributed systems, using cloud platforms for megaservices as a motivation and driving example. Well-designed cloud applications are layered above common service platforms that handle the hard problems: tracking groups of participating servers (views), distributing state and functions across a group, coordinating control and ownership of data, storing data reliably, and recovering from server and network failures. The course focuses on the design of these service platforms and their abstractions.

Although the course covers the fundamentals, the emphasis is on practical technologies and their limitations with an important software technology component. Assigned projects are based on DSLabs from the University of Washington (by way of MIT).

The readings for the course include some tutorial and survey papers, with a handful of full-strength research papers, including systems papers from major cloud services companies (e.g., Amazon, Google, and Facebook). There is no textbook.

Here is an outline of topics and readings for the course. Each section corresponds to a course unit and each bullet corresponds roughly to one class. An offering of CPS 512 selects from these topics according to circumstances of the semester and class, and assigns a subset of the recommended readings below, and others.

Reliable communication and services in the client/server model

Structure of distributed systems: networking, naming and addressing, clients and servers, request/response (Web/HTTP) and Remote Procedure Call (RPC), multi-tier services, geo-distributed mega-services. Messaging, failure models, and the problem of network partitions. State and the data storage tier.

- **Course intro**. Nodes and networks. Services: from micro-services to mega-services; multi-tier services and the services stack. The problems of scale and resilience. Failure models (fail-stop vs. Byzantine) and defenses. State and recovery. Authority, control, and trust. Example: Web services and Content Distribution Networks (CDNs).
- **The Internet as a distributed system.** The network stack: messages and streams, sockets, reliable communication with Transmission Control Protocol (TCP). Classical IP-LANs as an example of naming/addressing: Dynamic Host Configuration Protocol (DHCP), Address Resolution Protocol (ARP), role of broadcast, address allocation and leases, ARP caching, cache staleness, and Time To Live (TTL).
- **Remote Procedure Call (RPC).** Clients, servers, ports, and connections. Request-reply model and server interfaces (APIs). Calling server methods with RPC. API stubs and marshalling. Classical Birrell-Nelson RPC: retransmissions, acknowledgments, duplicates and suppression, reply cache and At-Most-Once semantics, interaction of RPC with threads and concurrency. Application example: key-value store (DSLabs Lab1). *Reading*: OSTEP, Birrell-Nelson [5].
- **Network storage: the Network File System (NFS)**. Review file system abstraction: hierarchical name space, directories, files/inodes, virtual file systems (VFS) in the OS kernel. NFS as an RPC service: object-based RPC and NFS file handles. Failures and recovery in classic NFS: idempotent operations and "statelessness", implications of stateless model. Detecting session failure in NFSv3: reliable asynchronous writes and the commit operation. *Reading*: OSTEP, NFS.

Scalable services, reliability, and consistency

- **Scale and recovery for storage.** The problem of scale in NFS: NFSv4, and parallel NFS (pNFS). The Google File System (GFS) as an example of scalable storage based on a common design pattern: scalable data plane with a central control service (master). Sharding and replication in GFS: chunks, metadata, and the master. Recovery of the GFS master: primary-backup replication and views (DSLabs Lab 2). Reading: GFS [12].
- **Leases.** Importance of keep-alive pings and timeouts for failure detection. The problem of distinguishing failures from network partitions. Leases for caching and state assignments in scalable storage (GFS and pNFS/NFSv4). Single-copy consistency with lease-based caching. A first glance at the CAP theorem. *Optional reading*: *Practical Uses of Synchronized Clocks in Distributed Systems* [16] (Sections 1, 5, 8).
- **Linearizable RPC for a replicated storage service.** RPC for sharded/replicated services. Example: RAMcloud key-value store. Integration of reliable RPC with replication mechanisms: Birrell-Nelson RPC revisited. Linearizability. *Reading*: RAMCloud/RIFL: *Implementing Linearizability at Large Scale and Low Latency* [15].

Elastic services in the cloud

- **Managed services.** Clusters, virtualization, and cluster management. Google's Borg and Kubernetes. Service abstractions: images and instances, containers, pods, and services. Kubernetes policy model: labels and policy, elastic service models: ReplicaSet, Deployment, and StatefulSet. Basic request routing for elastic services: connection routing (L4) on a virtual IP address (VIP) for ReplicaSet, load balancing, sharding and StatefulSet.
- **Mega-services and auto-scaling.** Multi-tier services and managed request routing at Google. The problems of resource allocation and capacity provisioning. Performance measures: throughput and response time. Service-Level Objectives (SLOs) and Service-Level Agreements (SLAs). Auto-scaling using basic queuing models: service demand, capacity and utilization, Little's Law, bottlenecks. Tail latency and the importance of load balancing. Request fanout and stragglers. *Reading*: Google's *Site Reliability Engineering*, *The Tail at Scale* [9].
- **Request routing and load balancing: into the network.** Middleboxes and smart switches. Datacenter networks: multipath routing, modular scaling, ECMP, and flow hashing. Google's Maglev approach for L4 connection routing: flow hashing and the problem of reconfiguration. A quick look at network virtualization: end-to-end principle revisited, Network Function Virtualization (NFV) and Software-Defined Networking (SDN).
- **Auto-sharding and sharded request routing.** Statefulness vs. statelessness in service tiers. A standard design pattern for multi-tier request routing: L4 to stateless forward tier with a sharded data tier. Load balancing and rebalancing. Adapting to churn with hashed distributions: consistent hashing (ring hashing) in the Akamai CDN and Dynamo key-value store. Limitations of consistent hashing. Google's auto-sharding service: Slicer. *Reading*: Slicer [1], Dynamo [10]. Optional reading: A Little Riak Book.

Coordination, consistency, and consensus

We spend a few days discussing consensus in theory and in practice. A safe, live consensus algorithm is the cornerstone of reliable distributed systems. But it is impossible to build one! So this is a study of what works in practice.

- **Coordination and consensus-as-a-service.** The need for consistent assignments: GFS and Slicer revisited. The problem of a lock server: leases revisited. Coordination services: Google's Chubby and Apache Zookeeper. Chubby abstractions: names, locks, events and notifications. Chubby's refinement of leases. Chubby's semantic guarantees and the concept of "jeopardy". *Optional reading*: Chubby [7], Zookeeper.

- **Foundations of consensus.** Consensus abstraction. Safety and liveness properties. The Fischer-Lynch-Patterson (FLP) result on impossibility of consensus in asynchronous networks (e.g., IP networks). Brewer's conjecture and its restatement of FLP result as the CAP Theorem. The meaning of Consistency, Availability, and Partition-resilience. Introduction to replicated state machines (RSM) also known as state machine replication (SMR). The partition decision and the quorum rule. Reading: *CAP Revisited* [6].

- **Consensus replication.** The asynchronous RSM fails-stop consensus algorithm in three flavors: Viewstamped Replication (VR), Paxos, and Raft. Basic consensus replication with a stable view. The problem of view changes: leader failures and partitions. View quorums and viewstamps (terms and ballots) as the key to safe consensus. *Reading*: *From Viewstamped Replication to Byzantine Fault Tolerance* [17]. Optional: RAFT [18], *How to Build a Highly Available System Using Consensus* [14].

- **What's the problem with Paxos?** Why Paxos confuses the hell out of people and how to keep it simple. Paxos in DSLabs Lab 3: maintaining a stable leader, leader election, preparing to start a new view. Alternative leader selection approaches of VR and Raft. *Reading*: *Paxos Made Moderately Complex* [21] (see paxos.systems).

Distributed transactions

Atomic transactions are fundamental for managing complex shared state. Early datacenter stacks avoided ACID transactions in favor of BASE key-value stores such as Dynamo. Modern key-value stores incorporate client/server transactions following the Thor model, with many variations. DSLabs Lab 4 features a transactional key-value store with sharding and replication, following many elements of RAMCloud/RIFL.

- **Serializable transactions**. Client/server transactions. Serializability and conflict-serializability. Transactional concurrency control: two-phase locking (2PL) and simple optimistic concurrency control as in RamCloud/RIFL. basic logging and snapshots, write-ahead logging. Reading: *Concurrency Control and Recovery* [11] (through 3.1.1; but we won't talk much about buffering as in 2.2.2), Birrell-Wobber small databases, Revisit RAMCloud/RIFL.

- **Transaction commit and recovery**. Two-phase commit (2PC). Handling prepared transactions. Failure and the blocking problem for 2PC. Are replicated shards sufficient to avoid blocking? Recovery by cooperative termination protocol (CTP). Role of linearizable RPC in 2PC recovery in RAMCloud/RIFL.

- **Transactions and time**. As time permits we discuss: timestamped optimistic concurrency control in Thor and successors; synchronized clocks; snapshot transactions and multi-version concurrency control (MVCC); role of TrueTime timestamps in Spanner; external consistency in Thor and Spanner; Spanner vs. CAP.

Causality and eventual consistency

CAP tells us that systems built for high availability in demanding environments (unreliable networks) must compromise consistency. But how? How to build "AP" systems and applications that function correctly even without synchronous communication with a quorum-approved master? We need a weaker notion of consistency based on causal orderings and a deeper understanding of concurrency and update conflicts.

- **Asynchronous replication.** Strong and weak consistency models: review of CAP, the partition decision, and the choice for AP. Replica convergence without linearizability: forks in the timeline and healing/repair. History of eventual consistency: the Grapevine name service, epidemic replication, gossip and anti-entropy, ACID vs. BASE revisited. The BASE view of consistency, commitment, and conflicts. Examples of causal inconsistencies. Overview of Bayou.
- **Concurrency and logical time**. Logical/Lamport clocks, happened-before and causality, causal multicast. The origins of state-machine replication: Lamport's decentralized lock service. Limitations of Lamport causal multicast: the problem of detecting when an update is stable under failure. Reading: *Time, Clocks, and the Ordering of Events in a Distributed System* [13]; *Why Logical Clocks are Easy* [4].
- **Vector clocks and causal ordering**. States of a distributed system: cuts and consistent cuts. Vector clocks: representing cuts, capturing happened-before, detecting concurrency/conflicts. Using vector clocks for replica reconciliation by anti-entropy exchanges in Bayou. Bayou's causal ordering based on logical clocks (Lamport causal multicast). Update conflicts and reordering in Bayou: tentative vs. committed updates. Bayou's use of primary commit to stabilize updates. *Reading*: Bayou [19]. Optional reading: Bayou's flexible update propagation [20].
- **RWN replication: Dynamo and Probabilistic Bounded Staleness (PBS)**. The need for eventual consistency in Dynamo: lack of consistent shard assignments, hashing with sloppy views. Background: quorum replication and weighted quorum voting. Dynamo's weak quorum replication: sloppy quorum and RWN replication. Dynamo's use of vector clocks to detect the resulting update conflicts. RWN replication as a means to reduce tail latency: PBS. The shopping cart example and a peek at Consistency as Logical Monotonicity (CALM) and Conflict-Free Replicated Data Types (CRDTs). CALM as defining the subset of applications that can sidestep CAP. Reading: *Eventual Consistency Today* [2]. Optional reading: *Quantifying Eventual Consistency with PBS* [3].

Trust in networked systems

We shift focus to secure Internet-scale systems with multiple identities, multiple trust domains, and federation. Multi-domain systems integrate cryptographic primitives to defend against subversion and unfaithful behavior by malicious participants. They include defenses against Byzantine faults, in which failed nodes may lie or cheat.

- **Domain Name Service: DNS and DNSSEC**. Scalable delegation and governance in Internet naming systems. Introducing authority into a hierarchical name service. The DNS roots, top-level domains, and subdomains. Threats to DNS: denial of service, resolvers and privacy, root address hijacking, name spoofing as an enabler for man-in-the-middle attacks. Crypto primitives to defend against name spoofing: digital signatures. DNSSEC and its deployment.
- **Blockchains**. Cryptocurrency and smart contracts as replicated state machine (RSM) applications. Extending consensus replication to a "trustless" system with client anonymity and no central point of trust: identity as self-generated public key hashes, signed operations, blockchain and Merkle tree as authenticated data structures. Randomized leader selection using cryptopuzzles and Proof of Work (PoW). Nakamoto consensus in Bitcoin. Blockchain forks, rollback and repair, and replica convergence. Limitations of public blockchains: throughput, energy cost, and the 51% attack.
- **Byzantine consensus.** Permissioned blockchains and the consensus core. The Byzantine Generals problem, agreement, and the Lamport-Pease result. Practical Byzantine Fault Tolerance (PBFT) protocol.
- **Trust logic and applications.** Assertions spoken by authenticated principals. Authority and delegation using logic. Validating assertion chains with logical rules. Applications to naming, routing, public key infrastructure, single sign-on identity services, and authorization. Logic of authentication. *Reading*: BAN Logic [8].

References

[1] A. Adya, D. Myers, J. Howell, J. Elson, C. Meek, V. Khemani, S. Fulger, P. Gu, L. Bhuvanagiri, J. Hunter, et al. Slicer: Auto-sharding for datacenter applications. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 739–753, 2016.

[2] P. Bailis and A. Ghodsi. Eventual consistency today: Limitations, extensions, and beyond. *Commun. ACM*, 56(5):55–63, May 2013. [local pdf].

[3] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica. Quantifying eventual consistency with PBS. *Communications of the ACM*, 57(8):93–102, Aug. 2014. [local pdf].

[4] C. Baquero and N. Preguiça. Why logical clocks are easy. *Communications of the ACM*, 59(4):43–47, 2016. [local pdf].

[5] A. Birrell and B. Nelson. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems (TOCS)*, 2(1):39–59, 1984. [local pdf].

[6] E. Brewer. CAP twelve years later: How the "rules" have changed. *Computer*, 45(2):23–29, February 2012. [local pdf].

[7] M. Burrows. The Chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 335–350. USENIX Association, May 2006. [local pdf].

[8] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computing Systems (TOCS)*, 8(1):18–36, 1990. [local pdf].

[9] J. Dean and L. A. Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[10] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *SOSP '07: Proceedings of 21st ACM SIGOPS Symposium on Operating systems Principles*, pages 205–220, New York, NY, USA, 2007. ACM. [local pdf].

[11] M. J. Franklin. Concurrency control and recovery, 1997. [local pdf].

[12] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 29–43, New York, NY, USA, October 2003. ACM. [local pdf].

[13] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978. [local pdf].

[14] B. W. Lampson. How to build a highly available system using consensus. In *Distributed Algorithms*, pages 1–17. Springer, 1996. [local pdf].

[15] C. Lee, S. J. Park, A. Kejriwal, S. Matsushita, and J. Ousterhout. Implementing linearizability at large scale and low latency. In *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP)*, pages 71–86, 2015.

[16] B. Liskov. Practical uses of synchronized clocks in distributed systems. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '91, pages 1–9, New York, NY, USA, 1991. ACM. [local pdf].

[17] B. Liskov. From Viewstamped Replication to Byzantine Fault Tolerance. In *Replication*, pages 121–149. Springer, 2010. [local pdf].

[18] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the USENIX Annual Technical Conference*, pages 305–320, June 2014. [local pdf].

[19] K. Petersen, M. Spreitzer, D. Terry, and M. Theimer. Bayou: Replicated database services for worldwide applications. In *Proceedings of the 7th ACM SIGOPS European Workshop: Systems Support for Worldwide Applications*, pages 275–280. ACM, September 1996. [local pdf].

[20] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. Flexible update propagation for weakly consistent replication. In *SOSP '97: Proceedings of the Sixteenth ACM Symposium on Operating systems Principles*, pages 288–301, New York, NY, USA, October 1997. ACM. [local pdf].

[21] R. Van Renesse and D. Altinbuken. Paxos made moderately complex. *ACM Computing Surveys*, 47(3):42:1–42:36, Feb. 2015. [local pdf].