Oct 16 – 20

* [What is a DNS Amplification Attack](https://www.f5.com/labs/articles/education/what-is-a-dns-amplification-attack-) by Debbie Walkowski at F5 labs
* [Window.localStorage](https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage) by Mozilla

**Purpose:**

* analyze which level of caching to use (client, server)
* explain how DoS attack works
* know how errors are handled and shared.
* trivial multithreading for independent requests
* explain how to improve performance on web servers/deployments

Likely we will move into DHTs this week.

**什么是DNS攻击：**  
DNS放大攻击是什么？

定义与目的：DNS放大攻击是一种分布式拒绝服务（DDoS）攻击，其目的是阻止用户访问网络系统、服务、网站、应用程序或其他资源。这种攻击通过使目标系统响应缓慢或完全失效来实现其目的。

攻击方式：攻击者利用域名系统（DNS）的正常运作，通过向开放的DNS解析器发送伪造的DNS查找请求来发起攻击。这些请求被设计成返回比原始请求大得多的响应，从而消耗目标网站的网络带宽。

DNS放大攻击的工作原理

利用开放DNS解析器：攻击者使用公开可访问的DNS解析器发送伪造的DNS请求。

请求放大：通过特殊构造的DNS请求（例如请求整个域的信息），使得响应的大小远远超过原始请求。

UDP协议的作用：DNS依赖于用户数据报协议（UDP），它是一种快速但不追踪对话的连接无关协议。攻击者通过伪造源IP地址，使DNS响应被发送到受害者而不是攻击者。

如何防御DNS放大攻击

出站安全：确保所有客户端使用本地内部DNS服务器，并配置为仅处理组织内部的DNS请求。

入站安全：阻止不是发往这些DNS服务器的任何DNS响应，并使用DNS感知的防火墙来允许仅从实际发送的请求返回的流量。

使用DNS Anycast：通过在多个位置的服务器上分配DNS流量来有效地平衡DNS流量，确保没有单个服务器过载。

与ISP合作：在网络连接饱和时，与互联网服务提供商合作阻止上游流量。

总结

DNS放大攻击是一种有效的DDoS攻击方式，通过放大DNS响应来消耗目标网站的资源。这种攻击利用了DNS的正常机制和UDP协议的特性。防御这种攻击需要在出站和入站安全上采取措施，并可能需要与ISP合作来减轻攻击的影响。

**Window.localStorage属性概述**

基本概念：localStorage是window接口的一个只读属性，允许访问文档源的Storage对象；存储的数据在浏览器会话间是持久的。

与sessionStorage的比较：localStorage与sessionStorage相似，但localStorage数据没有过期时间，而sessionStorage数据在页面会话结束时被清除（即页面关闭时）。在“隐私浏览”或“无痕”会话中加载的文档的localStorage数据在最后一个“隐私”标签页关闭时被清除。

localStorage的使用

存储数据：可以使用localStorage.setItem("key", "value")来添加数据项。

读取数据：使用localStorage.getItem("key")来读取数据项。

移除数据：使用localStorage.removeItem("key")来移除特定的数据项。

清除所有数据：使用localStorage.clear()来移除所有的数据项。

特点和注意事项

数据格式：存储在localStorage中的键和值始终以UTF-16字符串格式存储，整数键会自动转换为字符串。

协议特定性：localStorage数据特定于文档的协议。例如，通过HTTP加载的站点（如http://example.com）的localStorage与通过HTTPS加载的相应站点（如https://example.com）的localStorage返回不同的对象。

文件URL的行为：从file: URL加载的文档（即直接从用户本地文件系统中打开的文件）对于localStorage的行为没有明确的要求，可能在不同的浏览器中有所不同。

浏览器兼容性

localStorage在主流浏览器（如Chrome、Firefox、Safari、Edge等）中都得到了支持。

**1. Improving Performance on Web Servers/Deployments**

Improving the performance of web servers and deployments can be achieved through various strategies:

* **Load Balancing**: Distribute incoming network traffic across multiple servers to ensure no single server bears too much load.
* **Optimizing Content**: Minimize file sizes using compression techniques and optimize images.
* **Caching**: Implement caching at various levels (like client-side, server-side, or database caching) to reduce load on the server.
* **Content Delivery Network (CDN)**: Use CDNs to distribute the service spatially relative to end-users, which reduces latency.
* **Database Optimization**: Optimize queries and use efficient database indexing to reduce response times.
* **Server Configuration**: Tune server settings for optimal performance, including adjusting thread pool sizes and connection limits.
* **Monitoring and Scaling**: Continuously monitor server performance and scale resources (vertically or horizontally) as needed.

**2. Analyzing Which Level of Caching to Use**

The level of caching to use depends on the specific needs of your application:

* **Client-Side Caching**: Stores data on the client’s browser. It’s best for static, unchanging resources.
* **Server-Side Caching**: Involves storing data on the server. It’s useful for data that changes infrequently and is requested often.
* **Database Caching**: Caches query results to speed up read operations. Ideal for applications with heavy read operations.
* **Distributed Caching**: Useful for high-traffic sites to reduce database load, often implemented using in-memory data stores like Redis or Memcached.

**3. How DoS Attacks Work**

A Denial of Service (DoS) attack aims to make a machine or network resource unavailable to its intended users. This is typically achieved by overwhelming the target with a flood of internet traffic. DoS attacks can be executed in various ways, such as:

* **Volume-Based Attacks**: Overwhelm the bandwidth of the target.
* **Protocol Attacks**: Exploit server resources or intermediate communication equipment like firewalls and load balancers.
* **Application Layer Attacks**: Target the web application layer with seemingly legitimate requests.

**4. Handling and Sharing Errors**

Error handling and sharing in web applications involve:

* **Consistent Error Handling**: Implement a global error handling mechanism to catch and log errors.
* **User-Friendly Messages**: Display user-friendly error messages to the end-users.
* **Monitoring and Alerting**: Use monitoring tools to track errors and alert the development team.
* **Logging**: Log errors for further analysis and debugging.
* **Error Sharing**: In a microservices architecture, use centralized logging or error tracking services to aggregate errors from different services.

**5. Trivial Multithreading for Independent Requests**

Multithreading can be used in web servers to handle independent requests simultaneously, improving throughput and performance. Here’s how it works:

* **Thread Pool**: Use a pool of worker threads to handle incoming requests. Each thread handles one request at a time.
* **Non-blocking I/O**: Implement non-blocking I/O operations to keep threads from being idle.
* **Concurrency Control**: Manage access to shared resources to prevent race conditions and ensure thread safety.
* **Scalability**: Design the application to efficiently handle an increasing number of threads or requests.

回答问题：

1. **Why Distributed Data Structures Exist**:
   * **Scalability**: Distributed data structures allow data to be stored across multiple machines, which can handle more data and users than a single machine.
   * **Fault Tolerance**: If one machine fails, the system can still operate, as the data is replicated across multiple machines.
   * **Availability**: Data can be accessed from multiple locations, improving the speed and reliability of access.
   * **Load Balancing**: Distributing data helps in balancing the load across different servers, preventing any single server from becoming a bottleneck.
2. **Adding Hosts to a Distributed Hash Table (DHT)**:
   * **Initialization**: When a new host joins, it initializes its own data structure for the hash table.
   * **Integration**: The new host communicates with existing hosts to understand the current state of the DHT.
   * **Data Transfer**: Depending on the DHT's design, some data from existing hosts may be transferred to the new host to maintain a balanced distribution of data.
   * **Update Routing Information**: Other hosts in the DHT update their routing tables to include the new host.
3. **Detecting Host Failures or Inconsistencies in DHTs**:
   * **Regular Health Checks**: Hosts can periodically send messages to each other to check if they are still operational.
   * **Replication**: Data is often replicated across multiple hosts. If one host fails, the data can still be accessed from another host.
   * **Consistency Checks**: Regular checks can be performed to ensure data consistency across different hosts.
4. **Understanding How a Blockchain is a Distributed Data Structure**:
   * **Decentralization**: Blockchain stores data across a network of nodes, making it a distributed system.
   * **Immutability and Consensus**: Once data (or a block) is added to the blockchain, it cannot be changed. This is ensured through consensus algorithms.
   * **Transparency and Trust**: Every node has a copy of the entire blockchain, which ensures transparency and builds trust among participants.
5. **Simulating Ring Election and Bully Algorithm**:
   * **Ring Election Simulation**:
     + Arrange a set of nodes in a logical ring.
     + Each node knows its successor in the ring.
     + When an election is initiated, a node sends an election message to its successor, and the message circulates until the node with the highest identifier is found.
   * **Bully Algorithm Simulation**:
     + Start with a set of nodes, each with a unique identifier.
     + When a node initiates an election, it sends a message to all nodes with higher identifiers.
     + If no node responds, the initiating node declares itself as the leader.
     + If a node with a higher identifier responds, it takes over the election process.

Oct 23-27 (week 7)

* [Two phase commit (2PC)](http://book.mixu.net/distsys/replication.html) section in Chapter 4 of Distributed systems for fun and profit
* [Up and down the level of abstraction](http://book.mixu.net/distsys/single-page.html#abstractions) in Distributed systems for fun and profit up to “Strong consistency vs. other consistency models”
* [RPC Fundamentals](https://book.systemsapproach.org/e2e/rpc.html#rpc-fundamentals) in Computer Networks: A Systems Approach

**分布式系统中的复制问题**

* **复制问题的重要性**：在分布式系统中，复制问题是众多问题之一。作者选择关注复制问题，因为它通常是人们最感兴趣的部分。复制问题为许多子问题（如领导者选举、故障检测、共识和原子广播）提供了上下文。
* **复制作为群体通信问题**：复制涉及到的关键问题是如何安排和通信模式，以获得所需的性能和可用性特性。如何在网络分区和节点同时故障的情况下确保容错、持久性和非分歧性？

**复制的高级模式**

* **复制的定义和阶段**：假设有一个初始数据库，客户端发出改变数据库状态的请求。复制的安排和通信模式可以分为几个阶段：客户端发送请求、同步复制部分、返回响应给客户端、异步复制部分。
* **同步复制与异步复制**：同步复制（也称为主动复制或急切复制）涉及客户端在系统回复前被阻塞的情况。异步复制（也称为被动复制或懒惰复制）则是主节点立即向客户端回复，稍后再进行更新传播。

**主要复制方法概述**

* **防止分歧的复制方法**：这类方法确保复制副本始终一致，被称为共识问题。共识算法包括1n消息（异步主/备份）、2n消息（同步主/备份）、4n消息（两阶段提交、Multi-Paxos）和6n消息（三阶段提交、Paxos重复领导者选举）。
* **容忍分歧的复制方法**：这类方法允许在多个主节点上接受写入，但存在分歧风险。例如，客户端一致性模型、CRDTs（收敛和交换复制数据类型）、流言协议和部分法定人数系统。

**复制算法的实现**

* **主/备份复制**：这是最常见的复制方法，所有更新在主节点上执行，然后通过网络传输到备份副本。
* **两阶段提交（2PC）**：这是许多经典关系数据库中使用的协议，涉及两个阶段：投票和决策。
* **分区容忍共识算法**：这类算法能够在网络分区的情况下维持单一副本的一致性，例如Paxos和Raft算法。

**分布式系统中的抽象概念**

1. **基础概念**：分布式编程涉及使用多台计算机解决单台计算机可以解决的问题。这涉及存储和计算两个基本任务。分布式系统的使用通常是因为问题规模超出了单台计算机的处理能力。
2. **性能和可用性**：分布式系统的设计目标包括可扩展性、可用性、性能、延迟和容错能力。这些目标难以实现，需要通过抽象和模型以及分区和复制来解决。
3. **抽象和模型**：抽象通过去除解决问题时不相关的现实世界方面来简化问题。模型则精确描述分布式系统的关键属性。讨论了系统模型（异步/同步）、故障模型（崩溃故障、分区、拜占庭）和一致性模型（强一致性、最终一致性）。
4. **CAP定理和FLP不可能性结果**：探讨了CAP定理及其对一致性模型的影响，以及FLP不可能性结果的总结。
5. **时间和顺序**：理解分布式系统的关键在于理解时间和顺序。讨论了时间、顺序和时钟的使用，例如向量时钟和故障检测器。
6. **复制：防止分歧**：介绍了复制问题及其两种基本方式。讨论了维持单一副本一致性的复制方法，从最不容错的（两阶段提交）到Paxos。
7. **复制：接受分歧**：讨论了具有弱一致性保证的复制。介绍了基本的协调场景，例如分区副本尝试达成一致，以及Amazon的Dynamo作为具有弱一致性保证的系统设计示例。
8. **设计技术：分区和复制**：讨论了数据集在多个节点之间的分布方式。分区是将数据集划分为较小的独立集合，以允许更多的并行处理。复制或缓存数据到不同节点可以减少客户端和服务器之间的距离，增加容错能力。

**远程过程调用（RPC）**

1. **RPC基础**：RPC不是一个协议，而是一种用于构建分布式系统的机制。它允许应用程序调用过程，就像它们是本地的一样，即使它们是远程的。这种简化是通过使用处理请求和响应消息的存根来实现的，使远程调用看起来像本地调用。
2. **RPC的挑战和组成部分**：由于网络复杂性和不同的计算机架构，RPC面临独特的挑战。它涉及两个主要组件：一个用于管理客户端和服务器之间消息的协议，以及用于参数打包和翻译的语言/编译器支持。
3. **RPC过程**：RPC过程涉及客户端调用本地存根，然后将调用转换为请求消息。这个消息被发送到服务器，在那里它被转换回过程参数。执行后，服务器将响应发送回客户端。
4. **RPC中的标识符和消息匹配**：RPC需要过程的唯一命名系统和将响应与请求匹配的方法。这可以通过扁平或分层的命名空间以及使用消息ID来实现。
5. **克服网络限制**：RPC协议通常包括可靠消息传递的机制和通过分段和重组支持大消息的机制。这可以建立在现有的协议（如TCP）之上，或在RPC协议本身内实现。
6. **同步与异步协议**：RPC协议通常是同步的，意味着客户端在发送请求后等待响应。这与异步协议形成对比，在异步协议中，发送者不等待响应。
7. **RPC实现**：该部分还讨论了各种RPC实现，如SunRPC、DCE-RPC和gRPC，每种都有其设计选择和特点。

**Elections:**

**Bully Algorithm(霸凌算法)**

The Bully Algorithm is a method used in distributed systems to elect a coordinator or leader among a group of machines (nodes). It's particularly used in systems where each node can independently start an election for a new coordinator if it believes the current coordinator has failed. Here's a basic overview of how the Bully Algorithm works:

1. **Initiating Election**: When a node suspects that the coordinator is not functioning, it initiates an election. This can also happen if a node that was previously down comes back online and wants to assert its eligibility to be the coordinator.
2. **Election Message**: The initiating node sends an election message to all nodes with higher identifiers (IDs) than itself. The ID can be based on various criteria like node number, processing power, etc.
3. **Response**: If a node with a higher ID receives the election message, it responds to the initiating node, indicating that it will take over the election process. If the initiating node doesn't receive a response from any node with a higher ID, it wins the election and becomes the coordinator.
4. **Taking Charge**: A node that takes charge (either because it didn't receive any response to its election message or because it has the highest ID among the currently active nodes) declares itself as the coordinator to all nodes with lower IDs.
5. **Acknowledgment**: The other nodes acknowledge the new coordinator.

The name "Bully" comes from the fact that a node with a higher ID "bullies" its way to becoming the coordinator. This algorithm ensures that the node with the highest ID will eventually become the coordinator, assuming it's operational.

The Bully Algorithm is simple and effective in certain scenarios but can be inefficient in large networks due to the number of messages required. It also assumes that all communication is reliable and that each node has knowledge of every other node in the system.

想象一下，有一群孩子在玩游戏，他们需要选一个队长。这个游戏的规则是这样的：

1. **开始选举**：如果有一个孩子认为现在的队长不行了（比如队长走开了），他就可以说：“我们需要选一个新的队长！”
2. **挑战**：这个孩子会向所有比他号码大的孩子发出挑战，意思是“我想成为队长，你们有谁敢反对吗？”
3. **回应**：如果一个号码比他大的孩子回应了，就意味着他接受挑战，并且会自己开始一个新的选举。
4. **没有回应**：如果没有比他号码大的孩子回应他，那么他就成为新的队长。
5. **宣布胜利**：一旦一个孩子确定没有比他号码大的孩子挑战他，他就会告诉所有人他是新的队长。

这个“霸凌算法”就是这样工作的。在计算机的世界里，每台计算机就像是这些孩子，它们通过发送消息来决定谁是“队长”（在这里指的是协调器或领导者）。最终，号码最大的那台计算机（如果它运行正常的话）会成为队长。这个算法的名字来源于这样一个事实：号码大的计算机好像在“霸凌”号码小的计算机，让自己成为领导者。

Oct 30 – Nov 3 (week 8)

* [CAP Theorem](http://book.mixu.net/distsys/abstractions.html) section of chapter 2, up to “Strong consistency”

**CAP：**

1. **分布式系统的抽象层次**：本章讨论了在不同抽象层次上的分布式系统，包括CAP和FLP这两个不可能性结果。在分布式编程中，处理分布的后果是一个主要挑战，即在多节点的现实和期望系统“像单一系统一样工作”之间找到平衡。
2. **系统模型**：分布式系统的关键特性是分布性。这意味着程序在独立节点上并发运行，通过可能引入不确定性和消息丢失的网络连接，且没有共享内存或共享时钟。系统模型列举了与特定系统设计相关的许多假设。
3. **同步与异步系统模型**：同步系统模型假设进程以锁步方式执行，消息传输延迟有已知上限，每个进程都有准确的时钟。而异步系统模型则没有这些时序假设。
4. **共识问题**：共识问题是分布式系统的核心问题。它要求多个计算机（或节点）在某个值上达成一致。共识问题的解决对于实现分布式系统的可靠性和性能至关重要。
5. **不可能性结果（FLP和CAP）**：FLP不可能性结果表明，在异步系统中，即使消息永远不会丢失，最多只有一个进程可能失败，并且只能通过崩溃（停止执行）来失败，也不存在解决共识问题的（确定性）算法。CAP定理则表明，在一致性、可用性和分区容忍性这三个属性中，只能同时满足两个。
6. **CAP定理**：CAP定理是计算机科学家Eric Brewer提出的，它指出在一致性（所有节点同时看到相同数据）、可用性（节点故障不阻止幸存节点继续操作）和分区容忍性（系统即使在网络和/或节点故障导致消息丢失的情况下也能继续运行）这三个属性中，任何时候只能同时满足两个。
7. **强一致性与其他一致性模型**：一致性模型可以分为强一致性和弱一致性模型。强一致性模型保证更新的表现顺序和可见性等同于非复制系统。而弱一致性模型则不作此类保证。

CAP问题：

CAP 定理，也称为布鲁尔定理，是分布式计算中的一个基本原理，它指出分布式数据存储不可能同时提供以下三个保证中的两个以上：

1. **一致性**：每次读取都会收到最近的写入或错误。
2. **可用性**：每个请求都会收到一个（非错误）响应，但不保证它包含最新的写入。
3. **分区容错性**：尽管节点之间的网络丢弃（或延迟）任意数量的消息，系统仍继续运行。

将 CAP 定理应用于分布式系统中的问题时，您需要确定其中哪两个保证对于您的应用程序最重要，并相应地设计您的系统。应用方法如下：

1. **确定系统要求**：
   * 如果您的系统需要强一致性（例如必须准确记录交易的银行系统），您可能会优先考虑一致性和分区容错性（CP）。这可能意味着牺牲一些可用性（即，在网络分区期间，某些操作可能无法完成）。
   * 如果您的系统需要始终可用（例如社交媒体平台，数据更新存在一些滞后是可以接受的），您可能会优先考虑可用性和分区容错性 (AP)。在这种情况下，您可能必须处理最终一致性。
2. **分区容差设计**：
   * 由于分布式系统中网络故障是不可避免的，因此分区容错性通常是不可协商的。因此，真正的权衡往往是在一致性和可用性之间。
   * 实现处理分区的机制，例如重试、超时以及使用仲裁进行读写操作。
3. **处理一致性与可用性**：
   * 对于CP（一致性和分区容错）系统，可以考虑使用Paxos或Raft等共识协议来确保一致性。请做好准备，在分区期间，系统的某些部分可能不可用。
   * 对于 AP（可用性和分区容错性）系统，请使用最终一致性等技术，其中更新通过系统异步传播。整个系统中的数据可能并不总是一致的，但每个节点最终都会获得最新的更新。
4. **测试和监控**：
   * 定期测试系统在网络分区场景下的行为，以确保其行为符合预期。
   * 监视系统以检测分区和其他故障并对其做出响应。
5. **教育利益相关者**：
   * 确保利益相关者了解系统设计中所做的权衡。例如，在 AP 系统中，他们应该意识到他们可能偶尔会看到过时的数据。

Nov 6 – 10 (week 9)

* [Time and Order](http://book.mixu.net/distsys/time.html) Chapter 3 of Distributed Systems for fun and profit
* [Open-sourcing a more precise time appliance](https://engineering.fb.com/2021/08/11/open-source/time-appliance/) by Ahmad Byagowi, Oleg Obleukhov

**Time And Order：**

1. **顺序的重要性**：文章首先讨论了顺序在分布式系统中的重要性。它解释说，在单一系统中，操作有一个完整的顺序，但在分布式系统中，这种顺序并不自然存在，需要建立。顺序的概念是定义分布式系统正确性的核心，通常等同于单机上的行为。
2. **完全顺序与部分顺序**：文章区分了完全顺序和部分顺序。在分布式系统中，自然状态是部分顺序，不同节点间操作的相对顺序并不保证。而完全顺序则是为一组元素中的每个元素定义顺序的二元关系。
3. **时间的作用**：时间被呈现为分布式系统中的顺序来源。它允许对操作进行排序，并具有顺序、持续时间和解释等多种解释。文章讨论了时间戳的使用和逻辑时间的概念，后者不依赖于物理时钟，而是依赖于计数器和通信。
4. **时间假设**：文章探讨了分布式系统中关于时间的不同假设：全局时钟（时间在任何地方以相同的速率推进）、局部时钟（每台机器都有自己的时钟）和无时钟（逻辑时间，无需物理时钟即可确定顺序）。
5. **向量时钟和Lamport时钟**：文章介绍了Lamport时钟和向量时钟，这些时钟用于在分布式系统中跟踪顺序和因果关系，而不依赖于物理时钟。
6. **故障检测器**：讨论了分布式系统中故障检测器的使用，这些检测器有助于区分高延迟和服务器或网络故障。文章解释了怀疑级别的概念以及使用累积故障检测器。
7. **时间、顺序和性能**：最后，文章触及了时间、顺序和分布式系统中性能之间的关系，强调了其中的权衡。
8. **时间卡的创新**：Facebook工程师开发了一种名为“时间卡”的PCI Express（PCIe）卡，它可以将几乎任何普通服务器转变为时间设备。
9. **开源时间服务器**：为了实现这一目标，他们成立了开放计算时间设备项目（Open Compute Time Appliance Project），并开源了开放时间服务器（Open Time Server）的所有方面。
10. **基于NTP的新时间服务**：Facebook在2020年3月宣布，他们正在将数据中心的服务器（以及消费者产品）转换为基于网络时间协议（NTP）的新时间服务。这项内部构建并稍后开源的新服务，提高了Facebook基础设施的时间保持精度，从10毫秒提高到100微秒。
11. **时间设备的构建**：为了消除对公共NTP池的依赖，Facebook构建了一种新的硬件——时间设备，它包括一个全球导航卫星系统（GNSS）接收器和一个微型原子钟（MAC）。时间设备的用户即使在失去GNSS连接的情况下也能保持准确的时间。
12. **时间卡的设计**：时间卡使用了一个板载MAC、一个多频段GNSS接收器和一个现场可编程门阵列（FPGA）来实现时间引擎。时间引擎的工作是在连续的PPS信号之间以纳秒级粒度进行插值。
13. **开源时间设备的设计**：Facebook的目标是构建一个精确、经济且不受供应商限制的设备，并对整个行业产生更大的影响。他们与开放计算项目（OCP）合作，在时间设备项目（TAP）的GitHub仓库中开源了设计，包括规格、原理图、机械、BOM和源代码。
14. **如何从分布式系统的故障中恢复**：
    * **数据备份**：定期备份数据，以便在系统崩溃时可以恢复。
    * **冗余**：在多个节点上存储数据的副本，以便在一个节点失败时，其他节点可以接管。
    * **故障检测和自动恢复**：系统应能自动检测节点故障，并将任务迁移到健康的节点上。
    * **分布式事务**：确保即使在故障发生时，也能保持数据的一致性和完整性。
    * **状态检查点**：定期保存系统状态的快照，以便在发生故障时可以从最近的检查点恢复。
15. **理解“Happened Before”关系**：
    * “Happened Before”是分布式系统中用于描述事件顺序的一种关系。
    * 如果事件A发生在事件B之前，并且A的结果以某种方式影响了B，那么我们说“A happened before B”（A在B之前发生）。
    * 这个概念对于理解和设计分布式系统中的数据一致性和同步机制非常重要。
16. **描述网络时间协议（Network Time Protocol，NTP）**：
    * NTP是一种用于在计算机网络中同步时间的协议。
    * 它通过网络将计算机的时钟与世界标准时间（如UTC）对齐。
    * NTP客户端会从一个或多个NTP服务器请求时间信息，然后根据这些信息调整本地时钟。
    * NTP可以减少时间漂移，确保分布式系统中的计算机具有一致的时间观念，这对于日志记录、事务处理和其他时间敏感的操作至关重要。

Nov 20-24 (week 10)

* simulate how Raft leader election process works
* simulate how Raft log replication is executed

**Raft 领导者选举过程模拟**

1. **初始状态**：在 Raft 算法中，每个节点初始状态为跟随者（Follower）。
2. **选举超时**：如果一个跟随者在选举超时时间内没有收到来自领导者（Leader）或候选者（Candidate）的消息，它会变成候选者并开始一次选举。
3. **开始选举**：
   * 候选者增加自己的任期号（Term）。
   * 给自己投票。
   * 向其他节点发送请求投票的消息。
4. **投票**：
   * 其他节点在收到投票请求时，如果在该任期内尚未投票且候选者的日志至少与自己的一样新，就会投票给该候选者。
   * 如果已经投票或候选者的日志较旧，则拒绝投票。
5. **选举结果**：
   * 如果候选者获得了大多数节点的投票，它就成为新的领导者。
   * 如果没有候选者获得大多数票，选举超时后将开始新一轮的选举。

**Raft 日志复制过程模拟**

1. **客户端请求**：客户端向领导者发送一个操作请求。
2. **日志条目创建**：领导者将该操作作为新的日志条目追加到自己的日志中。
3. **日志复制**：
   * 领导者向所有跟随者并行发送包含新日志条目的追加日志请求。
   * 跟随者将该日志条目追加到自己的日志中，并向领导者发送响应。
4. **日志提交**：
   * 一旦领导者收到了来自大多数节点的确认，它就将该日志条目标记为已提交。
   * 领导者随后通知跟随者该日志条目已被提交。
5. **应用到状态机**：
   * 一旦日志条目被提交，领导者和跟随者将该操作应用到自己的状态机。
   * 领导者向客户端返回操作结果。