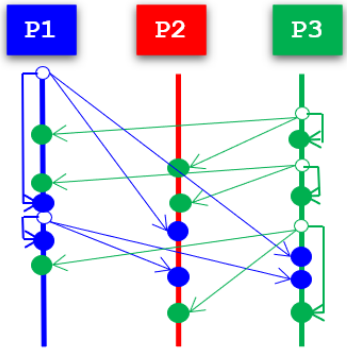
分布式理论作业

Question 1

> Given the following operations of three processes (P1, P2, P3). They belong to( )
>
> 
>
>
> (a) Total Ordering
> (b) Sequential Ordering
> (c) Causal Ordering
> (d) Data-centric ordering

Answer

- **(a) (b) (d)**

Reason

(a) Total Ordering (全序)

- **定义**：所有进程接收消息的顺序在全局范围内是一致的。如果消息 m1 在进程 A 中先于消息 m2，则所有进程都必须以相同的顺序接收 m1 和 m2。
- 从图中推导：
  - 图中展示了所有消息或事件的执行顺序，箭头的方向清晰地表明消息在各进程间传递的顺序完全一致。无论是进程 P1、P2 还是 P3，所有消息严格按照相同的顺序接收和执行。
  - 例如，若 P1 发送的消息 m1 和 P2 发送的消息 m2 被其他进程接收，接收顺序在所有进程中都相同。

(b) Sequential Ordering (顺序)

- **定义**：对于每个发送者，其发送的消息在所有接收者中必须按照发送顺序被接收。
- 从图中推导：
  - 在每个单独的进程中(P1、P2、P3)，消息的发送顺序与接收顺序一致。
  - 例如，P1 先发送某一条消息 m1，再发送消息 m2，无论其他进程如何接收消息，所有进程都必须先接收 m1，再接收 m2。这表明消息的顺序性在发送者的范围内得到了保证。

(d) Data-centric Ordering (数据中心顺序)

- **定义**：对于任意一个数据对象，所有进程对该数据对象的操作顺序必须一致。如果某进程对数据对象进行了操作 o1，另一进程对数据对象进行了操作 o2，则所有进程都必须按照相同的顺序接收这两个操作。

- 从图中推导：
    - 全序(Total Ordering)的性质直接包含了数据中心顺序。因为全序要求所有进程对任何消息的接收顺序是全局一致的，而数据中心顺序是全序在单一数据对象上的一个特例。
    - 图中并没有表明数据操作的顺序会出现冲突，所有操作显然都是按照全局一致的顺序进行。因此，对于任意数据对象的操作顺序也始终保持一致。

为什么不选 C (Causal Ordering)?

- **定义**：如果消息 m1 的发送导致了消息 m2 的发送，则所有进程都必须先接收 m1，再接收 m2。这要求消息之间存在因果关系。
- 从图中分析：
    - 图中没有表示因果关系的标志。例如，某条消息的发送直接引起了另一条消息的发送或依赖性，没有任何具体的因果信息表明 m1 导致了 m2。
    - 图中仅显示了消息的发送和接收顺序，而非因果依赖，因此不满足因果顺序。

Question 2

> Explain in your own words what the main reason is for actually considering weak consistency models.

Answer

弱一致性模型之所以广泛应用于分布式系统，主要在于它在性能、可用性和扩展性之间实现了有效的平衡。与强一致性相比，弱一致性允许操作在局部节点快速完成，而无需全局同步，从而显著降低系统的延迟，提升了响应速度。

在分布式系统中，网络分区和节点故障是无法避免的，弱一致性通过允许局部处理操作，即使在网络分区或部分节点不可用的情况下，系统依然可以继续运行，从而保障了高可用性。同时，随着系统规模的扩大，强一致性需要更高的通信和计算成本，而弱一致性减少了这些开销，支持系统轻松扩展到更大的规模。此外，许多实际应用场景并不需要严格一致性，例如电商购物车或社交媒体动态中，短暂的不一致对用户体验的影响微乎其微，而性能的提升则更为重要。弱一致性还通过减少全局同步需求，大幅降低了资源消耗，包括计算、存储和网络带宽成本。

总体而言，弱一致性模型的核心价值在于用有限的不一致性换取系统性能、可用性和扩展性的提升，为分布式系统带来了更高的效率、灵活性和经济性，特别适用于高并发、跨地域和用户体验优先的应用场景。这种模型通过在一致性要求上的让步，实现了分布式系统在实际运行中的高效性和灵活性。

Question 3

> Why is computer clock synchronization necessary? Describe the design requirements for a system to synchronize the clocks in a distributed system.

Answer

为什么需要计算机时钟同步?

在分布式系统中，不同计算机通常具有独立的时钟，这些时钟可能由于硬件特性、环境变化等因素导致偏差，甚至产生“时钟漂移”。计算机时钟同步的必要性体现在以下几个方面：

1. **事件协调**：在分布式环境中，任务可能需要在不同节点间协调。例如，在分布式文件系统中，多个客户端同时写入文件时需要准确记录操作顺序以避免冲突。
2. **日志和故障排查**：系统日志依赖于时间戳来标记事件发生的时间。若时钟不同步，可能会导致事件顺序错误，增加问题定位的复杂性。
3. **安全性**：许多安全协议（如认证和加密）依赖于时间。例如，防止重放攻击的时间戳必须是准确的。
4. **协同工作**：在协作环境（如多人文档编辑或实时通信系统）中，时间同步确保了操作顺序的一致性，避免数据冲突。
5. **避免数据不一致**：数据库事务需要基于一致的时间进行排序。若时间不同步，可能会破坏事务的原子性或一致性。
6. **支持时间敏感任务**：如交通管理、城市应急系统（例如分布式相机追踪车辆），需要准确记录事件时间。
7. **提升系统可靠性**：通过同步时间，各节点能够更高效地检测异常时钟、实现备份恢复或故障切换。

分布式系统时钟同步的设计要求

1. **一致性**：保证所有节点的时间在一定误差范围内保持一致，支持逻辑或物理时间的同步。
2. **精度**：同步时间的误差应足够小，以满足特定应用的需求。例如，在全球导航系统中，误差应低于纳秒级。
3. **容错性**：系统应能够容忍部分节点或通信路径故障。例如，NTP（网络时间协议）支持多路径冗余，能够应对网络不稳定或时间服务器故障。
4. **可扩展性**：时钟同步系统应支持大规模节点同步，能够高效扩展至全球范围，例如NTP的层级设计（Stratum模型）。
5. **安全性**：防止伪造时间或恶意节点的影响。例如，NTP通过验证消息来源的真实性来增强安全性。
6. **实时性**：系统需要在一定时间内完成同步，并能够快速响应时钟偏差的调整。
7. **低开销**：同步算法的计算复杂度和网络通信开销应尽量降低，以减少对系统性能的影响。

Question 4

> Given a certain merge sort algorithm, we assume that the sequential part accounts for 60% running time and the remaining parallel part 40% time. Moreover a 6-core computer can speed up the parallel part by 4×, 10-core computer by 5×, and yet 20-core computer by 8×, could you approximate the sweet point of the merge sort algorithm?

Answer

要找到归并排序算法的最佳性能点，我们需要根据阿姆达尔定律计算不同核心数下的总加速比，并确定能够最小化运行时间的核心数。最佳性能点出现在并行化收益与多核心开销达到平衡的地方。

阿姆达尔定律的公式为：$S_{all} = \frac{1}{(1-P)+\frac{P}{S_{part}}}$

其中：

- $P = 0.4$（并行部分比例）
- S_{part}

  为不同核心数下的并行部分加速比：
  - $S_{part} = 4$（6核计算机）
  - $S_{part} = 5$（10核计算机）
  - $S_{part} = 8$（20核计算机）

接下来计算不同核心数下的总加速比 S_{all}：

1. **6核计算机 ($S_{part} = 4$)**：

   $$S_{all} = \frac{1}{(1-0.4)+\frac{0.4}{4}} = \frac{1}{0.6+0.1} = \frac{1}{0.7} \approx 1.428$$

2. **10核计算机 ($S_{part} = 5$)**：

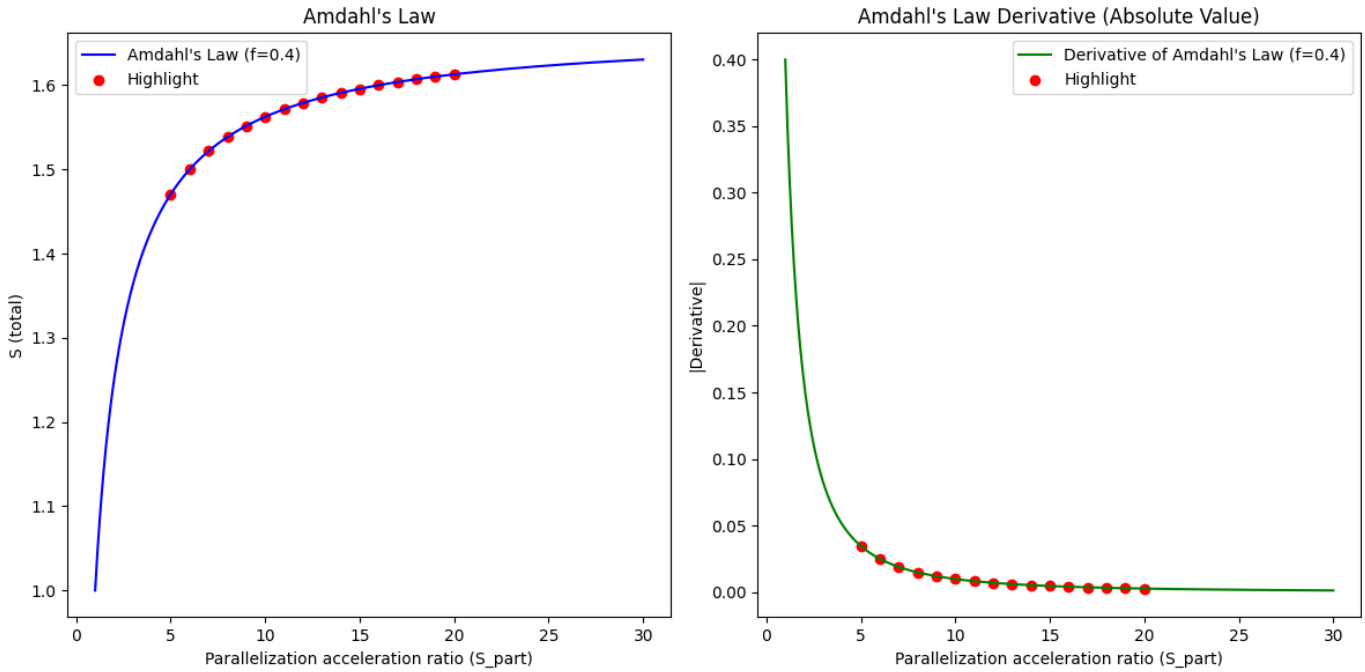
   $$S_{all} = \frac{1}{(1-0.4)+\frac{0.4}{5}} = \frac{1}{0.6+0.08} = \frac{1}{0.68} \approx 1.470$$

3. **20核计算机 ($S_{part} = 8$)**：

   $$S_{all} = \frac{1}{(1-0.4)+\frac{0.4}{8}} = \frac{1}{0.6+0.05} = \frac{1}{0.65} \approx 1.538$$

分析：

- 总加速比 S_{all} 随核心数增加而提升，但提升幅度逐渐减小（递减收益）。
- 6核、10核、20核的总加速比分别为 $1.428, 1.470, 1.538$。
- 尽管20核提供最高的总加速比，考虑实际多核开销（例如通信、同步、内存带宽限制等），**10核计算机可能是性能和成本的"甜点"点**，但是不可忽略20核加速比8倍所带来的总加速比的提升，在实际中需要考虑实际开销确定"甜点"点



Question 5

> A client attempts to synchronize with a time server. It records the round-trip times and timestamps returned by the server in the table below.Which of these times should it use to set its clock? To what time should it set it? Estimate the accuracy of the setting with respect to the server's clock. If it is known that the time between sending and receiving a message in the system concerned is at least 8 ms, do your answers change?

Round-trip (ms)	*Time (hr:min:sec)*
22	*10:54:23.674*
25	*10:54:25.450*
20	*10:54:28.342*

Answer

客户端应当使用第三个记录（RTT = 20 毫秒，服务器时间 = 10:54:28.342）来设置其时钟，因为它具有最小的往返时间（RTT），这意味着网络延迟最小，时间同步会更准确。

设置时间：

假设网络延迟是对称的，那么从服务器发送时间到客户端接收时间的延迟是 RTT 的一半，即 10 毫秒。因此，当客户端收到服务器时间戳 10:54:28.342 时，实际的服务器时间已经是：

```
10:54:28.342 + 10 毫秒 = 10:54:28.352
```

因此，客户端应将其时钟设置为 **10:54:28.352**。

估计准确性：

由于网络延迟可能存在不对称性，最大误差可达 RTT 的一半，即 ±10 毫秒。因此，客户端的时钟相对于服务器时钟的准确性估计为 ±10 毫秒。

已知最小消息传输时间为 8 毫秒的情况下：

如果已知系统中发送和接收消息的最小时间为 8 毫秒，那么最小的单程延迟为 8 毫秒，最小的 RTT 为 16 毫秒。对于记录的 RTT（20 毫秒），超过最小 RTT 的部分是可变延迟，即：

```
20 毫秒 - 16 毫秒 = 4 毫秒
```

可变延迟的单程为 2 毫秒（4 毫秒 ÷ 2）。因此，总的单程延迟为：

```
8 毫秒（最小延迟）+ 2 毫秒（可变延迟）= 10 毫秒
```

这与之前的计算一致，但现在我们可以更准确地估计误差范围为 ±2 毫秒（可变延迟的单程）。因此，客户端仍应将时钟设置为 **10:54:28.352**，但相对于服务器时钟的准确性提高到 **±2 毫秒**。