CLOUD COMPUTING

CAP Theorem

Dr. Prafullata Kiran Auradkar

Department of Computer Science and Engineering

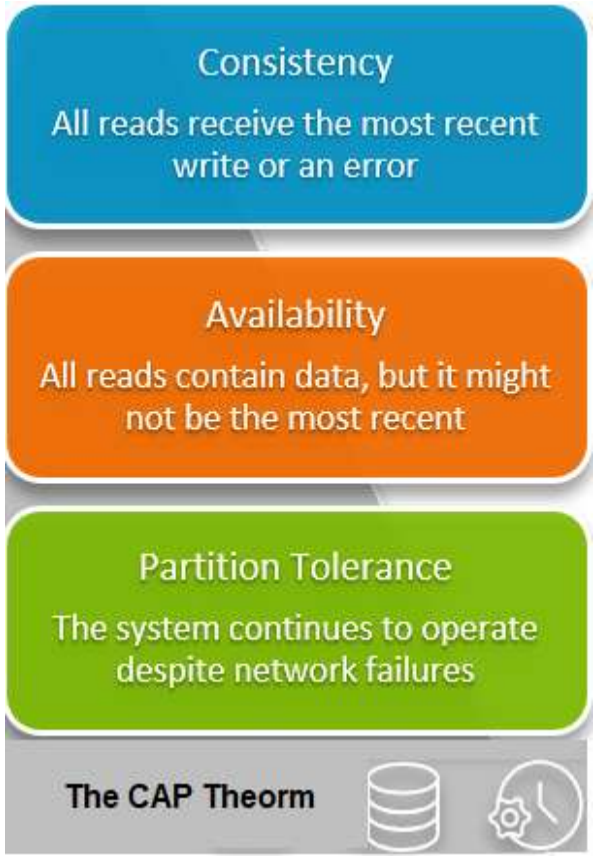
Acknowledgements:

Significant information in the slide deck presented through the Unit 3 of the course have been created by **Dr. H.L. Phalachandra** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for classroom presentation only.

CLOUD COMPUTING

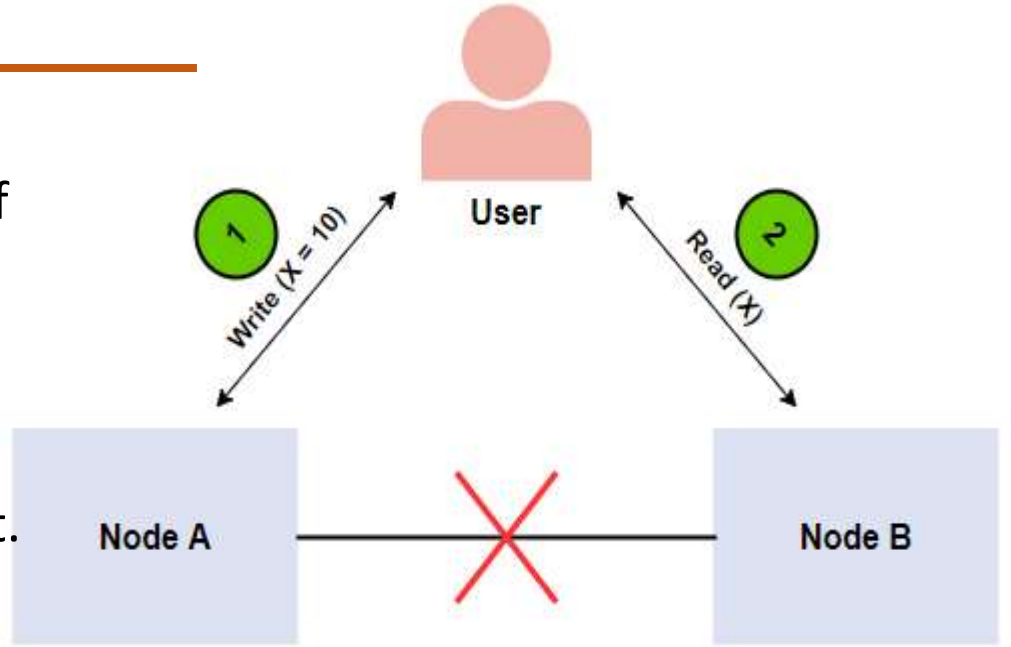
CAP Theorem

- The CAP theorem, originally introduced as the CAP principle, can be used to explain some of the competing requirements in a distributed system with replication.
- It is a tool used to make system designers aware of the trade-offs while designing “network shared-data systems” **or** A distributed system that stores data on more than one node (physical/virtual machines) at the same time
- The three letters in CAP refer to three desirable properties of distributed systems with replicated data:
  - **C - Consistency** (among replicated copies)
  - **A - Availability** (of the system for read and write operations)
  - **P - Partition tolerance (**in the face of the nodes in the system being partitioned by a network fault).
- The **CAP theorem** states that **it is not possible to guarantee all three of the desirable properties – consistency, availability, and partition tolerance at the same time in a distributed system with data replication**. We can strongly support only two of the three properties



CLOUD COMPUTING

CAP Theorem Example



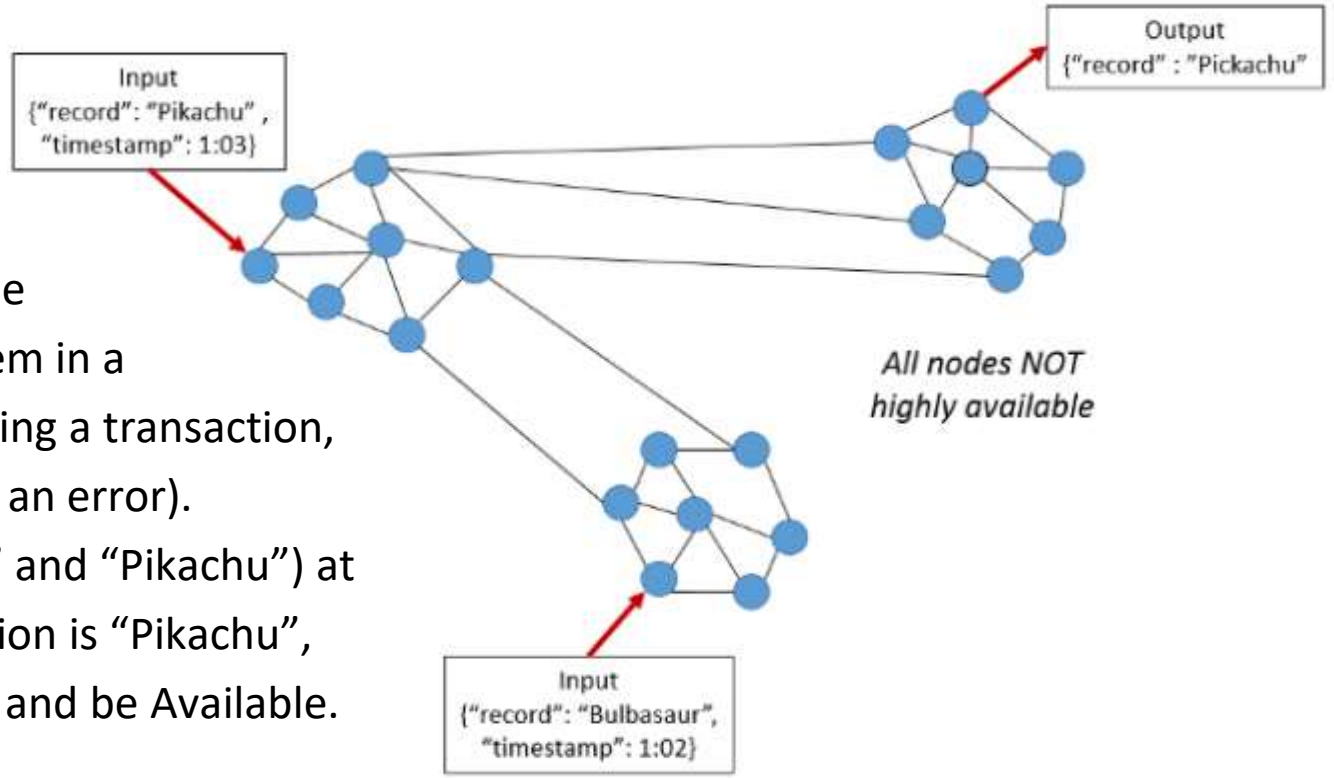
- Imagine a distributed system consisting of two nodes:
- The distributed system acts as a plain register with the value of variable X.
- There's a network failure that results in a network partition between the two nodes in the system.
- An end-user performs a write request, and then a read request.
- Let's examine a case where a different node of the system processes each request. In this case, our system has two options:
  - It can fail at one of the requests, breaking the system's availability
  - It can execute both requests, returning a stale value from the read request and breaking the system's consistency
- The system can't process both requests successfully while also ensuring that the read returns the latest value written by the write.
- This is because the results of the write operation can't be propagated from node A to node B because of the network partition.

CAP Theorem

1. **Consistency (C)** Consistency means that the nodes will have the same copies of a replicated data item visible for various transactions. It's the guarantee that every node in a distributed cluster returns the same, most recently updated value of a successful write at any logical time.

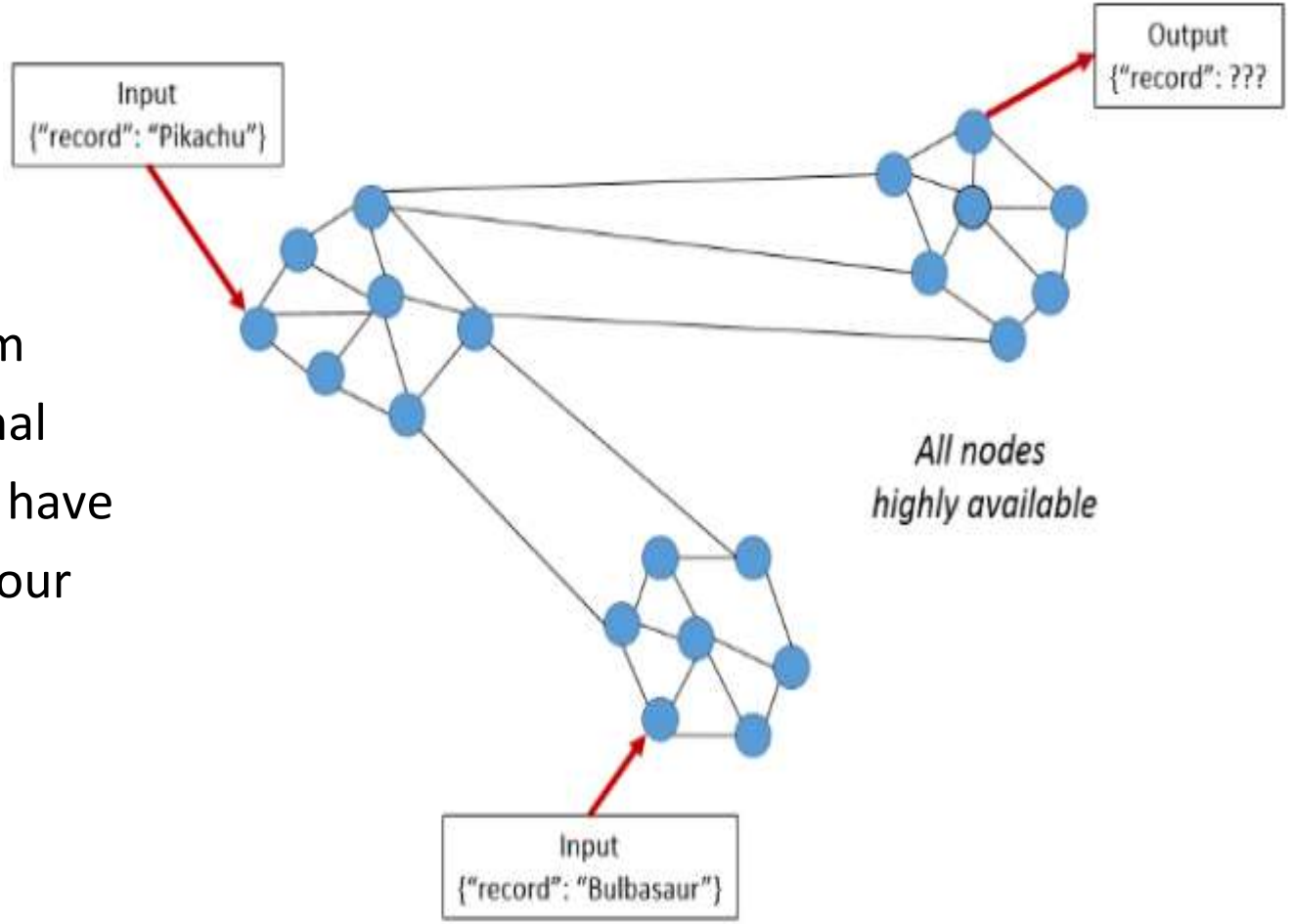
   In CAP, the term consistency refers to the consistency of the values in different copies of the same data item in a replicated distributed system. This can be verified if all reads initiated after a successful write return the same and latest value at any given logical time.

   - Performing a read operation will return the value of the most recent write operation causing all nodes to return the same data.
   - A system has consistency if a transaction starts with the system in a consistent state, and ends with the system in a consistent state (may have an inconsistent state during a transaction, but the entire transaction gets rolled back if there is an error).
   - In the image, we have 2 different records ("Bulbasaur" and "Pikachu") at different timestamps. The output on the third partition is "Pikachu", the latest input although it will need time to update and be Available.

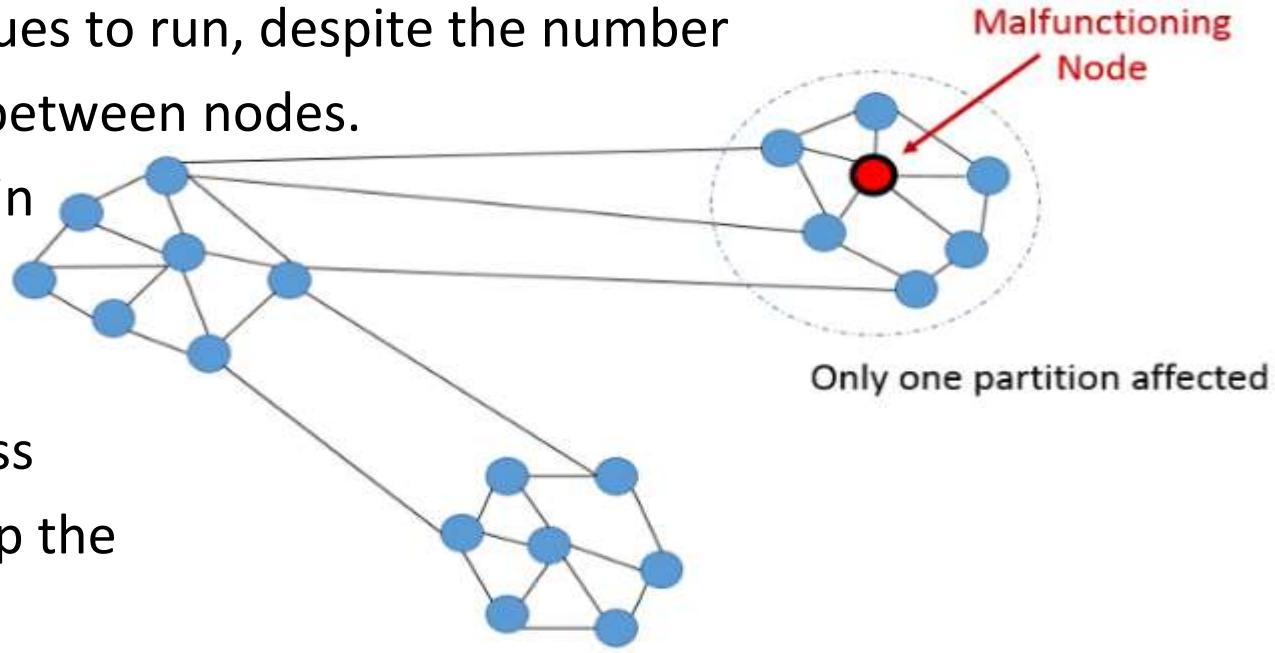


2. **Availability (A)** Availability means that each read or write request for a data item from a client to a node will either be processed successfully or will receive a message that the operation cannot be completed (if not in failed state).

- All working nodes in the distributed system return a valid response for any request, without exception
- Achieving availability in a distributed system requires that the system remains operational 100% of the time, which may need that we have "x" servers beyond the "n" servers serving our application



3. **Partition Tolerance** (P) Partition tolerance requires that a system be able to re-route a communication when there are temporary breaks or failures in the network (network partitions).
   - It means that the system continues to function and upholds its consistency guarantees in spite of network partitions.
   - Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.
   - This condition states that the system continues to run, despite the number of messages being delayed by the network between nodes.
   - A system that is partition-tolerant can sustain any amount of network failure that doesn't result in a failure of the entire network.
   - Data records are sufficiently replicated across combinations of nodes and networks to keep the system up through intermittent outages.



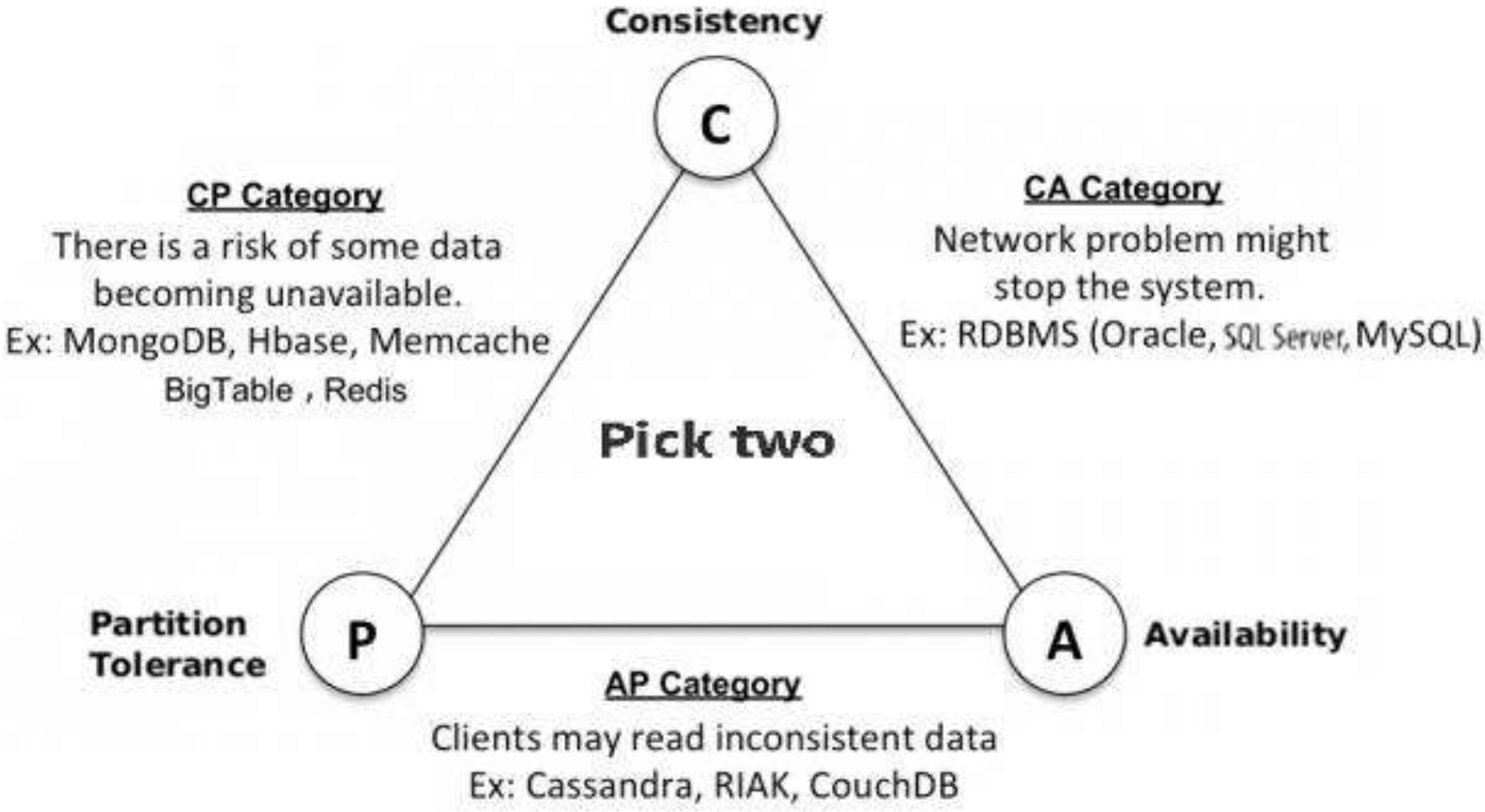
CAP Theorem : Tradeoffs between Requirements (Illustrations)

1. ***Availability and Partition-Tolerant*** ***(Compromised Consistency)***: Say you have two nodes and the link between the two is severed. Since both nodes are up, you can design the system to accept requests on each of the nodes, which will make the system available despite the network being partitioned. However, each node will issue its own results, so by providing high availability and partition tolerance you'll compromise *consistency*.

2. ***Consistent and Partition-Tolerant*** ***(Compromised Availability)***: Say you have three nodes and one node loses its link with the other two. You can create a rule that, a result will be returned only when a majority of nodes agree. So In-spite of having a partition, the system will return a consistent result, but since the separated node won't be able to reach consensus it won't be *available* even though it's up.

3. ***Consistent and Available*** ***(Compromised on a Partition-Tolerance)***: Although, a system can be both consistent and available, but it may have to block on a *partition*.

CAP Theorem : Illustrative Examples of products/Application categories to choose



CAP Theorem

- The “Pick Two” expression of CAP opened the minds of designers to a wider range of systems and tradeoffs
- CAP is **NOT** a choice “at all times” as to “which one of the three guarantees to abandon”. In fact, the choice is between consistency and availability only when a network partition or failure happens. When there is no network failure, both availability and consistency can be satisfied
- Products like DB e.g. Casandra supports AP but provides ***eventual consistency*** by allowing clients to write to any nodes at any time and reconciling inconsistencies as quickly as possible
- Reference: https://www.youtube.com/watch?v=9uCP3qHNbWw

CAP Theorem and some of the criticisms

- Although the Theorem doesn't specify an upper bound on response time for availability, in practice, there's exists a timeout. CAP Theorem ignores latency, which is an important consideration in practice. Timeouts are often implemented in services. During a partition, if we cancel a request, we maintain consistency but forfeit availability. In fact, latency can be seen as another word for availability.
- In NoSQL distributed databases, CAP Theorem has led to the belief that eventual consistency provides better availability than strong consistency. It could be considered as an outdated notion. It's better to factor in sensitivity to network delays.
- CAP Theorem suggests a binary decision. In reality, it's a continuum. It’s more of a trade-off. There are different degrees of consistency implemented via "read your writes, monotonic reads and causal consistency".

CLOUD COMPUTING

CAP Theorem and how do you use them when making decisions

- CAP Theorem can feel quite abstract, but both from a technical and business perspective the trade-offs will lead to some very important questions which has practical, real-world consequences.
- These questions would be
  - Is it important to avoid throwing up errors to the client?
  - Or are we willing to sacrifice the visible user experience to ensure consistency?
  - Is consistency an actually important part of the user's experience
  - Or can we actually do what we want with a relational database and avoid the need for partition tolerance altogether?
- All of these are ultimately leading to user experience questions.
- This will need understanding of the overall goals of the project, and context in which your database solution is operating. (E.g. Is it powering an internal analytics dashboard? Or is it supporting a widely used external-facing website or application?)

CAP Theorem and its relationship to Cloud Computing

- Microservices architecture for applications is popular and prevalent on both cloud servers and on-premises data centers
- We have also seen that microservices are loosely coupled, independently deployable application components that incorporate their own stack, including their own database and database model, and communicate with each other over a network.
- Understanding the CAP theorem can help you choose the best database when designing a microservices-based application running from multiple locations.
  E.g. If the ability to quickly iterate the data model and scale horizontally is essential to your application, but you can tolerate eventual (as opposed to strict) consistency, an database like Cassandra or Apache CouchDB (supporting AP - Availability and Partitioning) can meet your requirements and simplify your deployment. On the other hand, if your application depends heavily on data consistency—as in an eCommerce application or a payment service—you might opt for a relational database like PostgreSQL.

CAP Theorem Conclusion

- Distributed systems allow us to achieve a level of computing power and availability that were simply not available in the past.
- Our systems have higher performance, lower latency, and near 100% up-time in data centers that span the entire globe.
- These systems of today are run on commodity hardware that is easily obtainable and configurable at affordable costs.
- The disadvantage to it though is Distributed systems are more complex than their single-network counterparts.
- Understanding the complexity incurred in distributed systems, making the appropriate trade-offs for the task at hand (CAP), and selecting the right tool for the job is necessary with horizontal scaling.

THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu