CS4224/CS5424 Lecture 8
Replicated Data Consistency

Strong vs Eventual Consistency

- Values returned by read operations are not necessarily the latest values
- **Eventual consistency**: each replica eventually receives each write operation
  - If clients stopped updating data, then read operations would eventually return an object's latest value
- What does a read operation return?
  - Strong consistency: value that was last written
  - Eventual consistency: any value that was written in the past
- Systems that support both strong & eventual consistency levels:
  - Amazon's DynamoDB
  - Google App Engine Datastore
  - etc.

Other Data Consistency Levels

Strong Consistency ⟷ Eventual Consistency

- Other intermediate consistency levels
  - Consistent Prefix
  - Bounded Staleness
  - Monotonic Reads
  - Read My Writes
- Consistency level
  - dictates set of allowable return values
  - defined by set of previous writes whose results are visible to a read operation

Baseball Game

Figure 1. A simplified baseball game.

```
Write ("visitors", 0);
Write ("home", 0);
for inning = 1 .. 9
   outs = 0;
   while outs < 3
     visiting player bats;
     for each run scored
        score = Read ("visitors");
        Write ("visitors", score + 1);
   outs = 0;
   while outs < 3
      home player bats;
      for each run scored
         score = Read ("home");
         Write ("home", score + 1);
end game;
```

(D. Terry, 2013)

Database Updates for Sample Game

Figure 2. Sequence of writes for a sample game.

```
Write ("home", 1)
Write ("visitors", 1)
Write ("home", 2)
Write ("home", 3)
Write ("visitors", 2)
Write ("home", 4)
Write ("home", 5)
```

Figure 3. The line score for this sample game.

	1	2	3	4	5	6	7	8	9	RUNS
Visitors	0	0	1	0	1	0	0			2
Home	1	0	1	1	0	2				5

(D. Terry, 2013)

Example: Baseball Game Database

- Database with two records: "home" & "visitors"
- Initial record values are 0
- **Client1** performs the following updates:
  1. Write ("home", 1)
  2. Write ("visitors", 1)
  3. Write ("home", 2)
  4. Write ("home", 3)
  5. Write ("visitors", 2)
  6. Write ("home", 4)
  7. Write ("home", 5)
- **Client2** performs the following operations:
  - v = Read ("visitors")
  - h = Read ("home")
  - output v"-"h

Strong Consistency

- Read operation returns the value that was last written for object
- Read observes effects of all previously completed writes

1. Write ("home", 1)
2. Write ("visitors", 1)
3. Write ("home", 2)
4. Write ("home", 3)
5. Write ("visitors", 2)
6. Write ("home", 4)
7. Write ("home", 5)

```
v = Read ("visitors")
h = Read ("home")
Output v"-"h
```

1	2	3	4	5	6	7
0-1	1-1	1-2	1-3	2-3	2-4	2-5

- Possible output: 2-5

Eventual Consistency

- Read operation returns any value that was written in the past for object

1. Write ("home", 1)
2. Write ("visitors", 1)
3. Write ("home", 2)
4. Write ("home", 3)
5. Write ("visitors", 2)
6. Write ("home", 4)
7. Write ("home", 5)

v = Read ("visitors")
h = Read ("home")
Output v"-"h

1	**2**	**3**	**4**	**5**	**6**	**7**
0-1	1-1	1-2	1-3	2-3	2-4	2-5

- Possible output: v-h, where $v \in \{0, 1, 2\}$, $h \in \{0, 1, 2, 3, 4, 5\}$

Consistent Prefix

- Reader observes an ordered sequence of writes starting with the first write
- Reader observes all the writes up till (and including) the k^{th} write, for some $k \in \{0, 1, \cdots\}$

1. Write ("home", 1)
2. Write ("visitors", 1)
3. Write ("home", 2)
4. Write ("home", 3)
5. Write ("visitors", 2)
6. Write ("home", 4)
7. Write ("home", 5)

```
v = Read ("visitors")
h = Read ("home")
Output v"-"h
```

1	2	3	4	5	6	7
0-1	1-1	1-2	1-3	2-3	2-4	2-5

- Possible output:
  0-0, // $k = 0$
  0-1, // $k = 1$
  1-1, // $k = 2$
  1-2, // $k = 3$
  1-3, // $k = 4$
  2-3, // $k = 5$
  2-4, // $k = 6$
  2-5 // $k = 7$

Bounded Staleness

- Ensures read results are not too out of date
- Staleness defined by a time period T (assume unit of time is minutes).
- Guarantees that the value returned by a read operation is no more than T minutes out-of-date
- Assume read operation issued at time t_{read}
- Writes performed at time $\leq t_{read} - T$ are visible
- Writes performed at time $> t_{read} - T$ may or may not be visible

Bounded Staleness (cont.)

1. Write ("home", 1)
2. Write ("visitors", 1)
3. Write ("home", 2)
4. Write ("home", 3)
5. Write ("visitors", 2)
6. Write ("home", 4)
7. Write ("home", 5)

```
v = Read ("visitors")
h = Read ("home")
Output v"-"h
```

1	2	3	4	5	6	7
0-1	1-1	1-2	1-3	2-3	2-4	2-5

- Assume that the last update performed *T* minutes ago was Write("visitors",2)
- Possible output: 2-3, 2-4, 2-5

Monotonic Reads

- Property that applies to a sequence of reads by same client
- If client issues a sequence of two read operations for same object,
  - either both read operations return same value,
  - or second read operation returns a more recent value than the one returned by the first read

1. Write ("home", 1)
2. Write ("visitors", 1)
3. Write ("home", 2)
4. Write ("home", 3)
5. Write ("visitors", 2)
6. Write ("home", 4)
7. Write ("home", 5)

v_1 = Read ("visitors")
h_1 = Read ("home")
Output v_1"-"h_1
v_2 = Read ("visitors")
h_2 = Read ("home")
Output v_2"-"h_2

1	2	3	4	5	6	7
0-1	1-1	1-2	1-3	2-3	2-4	2-5

- If v_1-h_1 is 1-3,
  possible output for v_2-h_2 is
  1-3, 1-4, 1-5,
  2-3, 2-4, 2-5

Read My Writes

- Effects of all writes performed by a client on an object are visible to client's subsequent reads of object
- Same as eventual consistency if client has issued no writes on object

1. Write ("home", 1)
2. Write ("visitors", 1)
3. Write ("home", 2)
4. Write ("home", 3)
5. Write ("visitors", 2)
6. Write ("home", 4)
7. Write ("home", 5)

v = Read ("visitors")
h = Read ("home")
Output v"-"h

1	2	3	4	5	6	7
0-1	1-1	1-2	1-3	2-3	2-4	2-5

- Possible output for **writer**: 2-5
- Possible output for **non-writer**: v-h, where $v \in \{0, 1, 2\}$, $h \in \{0, 1, 2, 3, 4, 5\}$

Consistency Levels

Table 1. Six consistency guarantees.

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See initial sequence of writes.
Bounded Staleness	See all "old" writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.

(D. Terry, 2013)

Strength of Consistency Level

- Strength of a consistency level is defined by the size of set of allowable results for a read operation
- Smaller sets of possible read results indicate strong consistency

Table 3. Possible scores read for each consistency guarantee.

Strong Consistency	2-5
Eventual Consistency	0-0, 0-1, 0-2, 0-3, 0-4, 0-5, 1-0, 1-1, 1-2, 1-3, 1-4, 1-5, 2-0, 2-1, 2-2, 2-3, 2-4, 2-5
Consistent Prefix	0-0, 0-1, 1-1, 1-2, 1-3, 2-3, 2-4, 2-5
Bounded Staleness	scores that are at most one inning out-of-date: 2-3, 2-4, 2-5
Monotonic Reads	after reading 1-3: 1-3, 1-4, 1-5, 2-3, 2-4, 2-5
Read My Writes	for the writer: 2-5 for anyone other than the writer: 0-0, 0-1, 0-2, 0-3, 0-4, 0-5, 1-0, 1-1, 1-2, 1-3, 1-4, 1-5, 2-0, 2-1, 2-2, 2-3, 2-4, 2-5

(D. Terry, 2013)

Tradeoffs

- **Consistency** = strength of consistency level
- **Performance** = read latency
- **Availability** = likelihood of successful read operation in the presence of server failures

Table 2. Consistency, performance, and valuability trade-offs.

Guarantee	Consistency	Performance	Availability
Strong Consistency	excellent	poor	poor
Eventual Consistency	poor	excellent	excellent
Consistent Prefix	okay	good	excellent
Bounded Staleness	good	okay	poor
Monotonic Reads	okay	good	good
Read My Writes	okay	okay	okay

(D. Terry, 2013)

Pileus

- Replicated key-value cloud storage system
- Developed at Microsoft Research Silicon Valley
  - Microsoft Azure Cosmo DB (previously known as DocumentDB) has similar tunable consistency levels
- Each **object** stored in a **table** has a unique string-valued **key** and a byte-sequence **value**
- Tables are **range-partitioned** by key values into **tablets** & replicated
- Uses **lazy centralized replication protocol**
  - Each storage site is classified as primary or secondary site

Pileus (cont.)

- **Primary sites**
  - Primary sites store master copies of objects
  - All updates are performed at primary sites & ordered by **commit timestamps**
- **Secondary sites**
  - Updates are propagated asynchronously to secondary sites
  - Updates are transmitted & received in order of commit timestamps
- Assume each site consists of a single server

Pileus (cont.)

- Uses **distributed snapshot isolation protocol** for concurrency control
  - **readTS(T)** = read timestamp of Xact *T*
  - **commitTS(T)** = commit timestamp of Xact *T*
- Allows **tunable consistency levels** for transaction executions
  - strong
  - causal
  - bounded staleness
  - read my writes
  - monotonic reads
  - eventual
- Allows applications to specify consistency/latency preferences using **Service Level Agreements (SLAs)**

Pileus API

- **Put**(key, value)
- **Get**(key)
- **BeginSession**()
  - Each session consists of one or more transactions
  - Session defines the scope for some consistency levels (e.g., read-my-writes, monotonic reads)
- **BeginTx**(consistency, key-set)
  - **consistency**: Each transaction specifies a consistency level
  - **key-set**: Each transaction can optionally specify the keys of objects that will be read
  - All Gets within a transaction access the same snapshot
- **EndTx**()
  - Ends the transaction & attempts to commit its Puts
- **EndSession**()

Pileus: Consistency Guarantees

What are the properties of snapshot accessed by Xact T?

- **Strong**: Snapshot contains the results of all Xacts that committed before the start of T
- **Eventual**: Snapshot contains the results of an arbitrary prefix of the sequence of committed Xacts. This is actually consistent prefix consistency.
- **Read-my-writes**: Snapshot contains the results of all previous update Xacts in the same session as well as the previous Puts in T
- **Monotonic reads**: Snapshot contains at least the results of all previous snapshots that were read in the same session
- **Bounded(t)**: Snapshot contains the results of all Xacts committed more than t seconds before the start of T
- **Causal consistency**: Snapshot contains the results of all Xacts that causally precede T

Pileus: Causal Consistency

- Given two Xacts T_1 & T_2, T_1 causally precedes T_2 (denoted by $T_1 < T_2$) if one of the following conditions hold:
  (1) T_2 is executed after T_1 in the same session,
  (2) T_2 reads some object written by T_1,
  (3) T_1 & T_2 both performed a Put on the same object, and T_2 commits after T_1, or
  (4) there is some Xact T_3 where $T_1 < T_3$ and $T_3 < T_2$
- **Causal consistency**: Snapshot contains the results of all Xacts that causally precede T
- Pileus ensures that if $T_1 < T_2$, then $commitTS(T_1) < commitTS(T_2)$

Causal Consistency: Example

Client A: $R_1(x_0), W_1(y_1), C_1$
Client B: $R_2(z_0), C_2, R_3(x_0), W_3(y_3), C_3$
Client C: $R_4(y_3), W_4(z_4), C_4$

Pileus: Multiversion Storage

- Each server maintains the following:
  - **key-range** = range of keys managed by server
  - **store** = set of (key, value, timestamp) tuples
  - **highTS** = commit timestamp of the latest transaction that has been processed by server
  - **lowTS** = timestamp of server's most recent pruning operation
- Each primary server additionally maintains the following:
  - **logical clock** for assigning commit timestamps
  - **pending** = list of (Put-set, proposed timestamp) pairs for transactions that are in the process of being committed
  - **propagating** = queue of (Put-set, commit timestamp) pairs for recently committed transactions to be sent to secondary replicas

Pruning Old Data Versions

- Servers prune old versions of data objects to reduce storage usage by increasing their low timestamp **lowTS**
- For each data object O at server S, all versions of O with commitTS $\leq$ S.lowTS are pruned except for the latest version
  - The pruning retains all versions with commitTS > S.lowTS & the latest version with commitTS $\leq$ S.lowTS
- **Example**
  - Let O_t denote the version of object O with commitTS = t
  - Consider a server with lowTS = 25 & five versions of O: $\{O_{20}, O_{40}, O_{50}, O_{100}, O_{160}\}$
  - Increasing lowTS to 125 will prune O_{20}, O_{40} & O_{50}

Client's State

- Applications access servers through a **client library**
  - Routes Get & Put operations to appropriate servers
  - For convenience, "client" refers to the client library
- Each client maintains the following information for **each server** *S*
  - **key-range[S]** = key-range of *S*
  - **latency[S]** = round-trip latency to *S*
  - **highTS[S]** = high timestamp of *S*
- Each client maintains the following information for its **current session**:
  - Commit timestamps of previous Puts in the session
  - Commit timestamps of versions return by previous Gets in the session

Put(key,value)

- Put operations by Xact T are buffered at client
- New object versions created by T are visible to Gets within T but are not visible to other Xacts until T commits

Get(key)

- Get(key) by Xact T is processed by sending **Get(key, readTS(T))** to a server S
  - readTS(T) is determined by client when processing BeginTx(consistency, key-set)
  - Server S is chosen by client
- Server S processes **Get(key, t)** as follows:
  - If S is the primary server for *key*, then S accepts the request if $t \geq S$.lowTS
    - S updates its logical clock to max(local clock timestamp, t)
  - If S is a secondary server for *key*, then S accepts the request if $t \in [S.\text{lowTS}, S.\text{highTS}]$
  - If S accepts the request, then S returns (v, v.commitTS, S.highTS) where v is the most recent version of *key* in S with $v.commitTS \leq t$, and $v.commitTS$ is the commit timstamp of v
  - Otherwise, S rejects the Get request

BeginTx(consistency, key-set)

- Client chooses **readTS(T)** for new Xact T
  - readTS(T) determines the snapshot that T access for all its Get operations
- First, client determines $MARTS(T)$
  - MARTS(T) = minimum acceptable read timestamp for T
  - if $readTS(T) \geq MARTS(T)$, then the snapshot for T satisfies the requested consistency guarantee
- Next, for each key k_i in key-set, client selects the closest server S_i that is sufficiently up-to-date
  - $k_i \in$ key-range[S_i]
  - highTS[S_i] $\geq$ MARTS(T) if S_i is a secondary server
  - for every server S_j that also satisfies the above two conditions, latency[S_j] $\geq$ latency[S_i]
  - If there's still a tie (i.e., there's some server S_j with latency[S_j] $=$ latency[S_i]), then highTS[S_i] $\geq$ highTS[S_j]
- $readTS(T) = \min \{\text{highTS}[S_i] \mid k_i \in \text{key-set}\}$

Example 1: Client B's BeginTx(MR, $\{x\}$)

Client A: $W_1(x_1), C_1, W_4(x_4), C_4$
Client B: $R_2(x_1), C_2,$
Client C: $W_3(x_3), C_3$

Servers' State

Server	Store
S1	$(x_0, 0)$, $(x_1, 20)$, $(x_3, 80)$, $(x_4, 120)$
S2	$(x_0, 0)$
S3	$(x_0, 0)$, $(x_1, 20)$
S4	$(x_0, 0)$, $(x_1, 20)$, $(x_3, 80)$

Client B's State

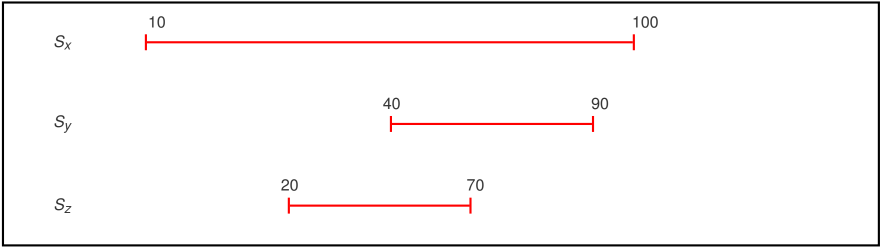
Server	highTS	latency
S1	120	30
S2	0	10
S3	20	40
S4	80	20

- MR: abbreviation for Monotonic Reads
- (x_i, t) denote the version of object x created by Xact T_i with a commit timestamp of t
- Assume that the initial database state was created by T_0

Example 2: BeginTx(ℓ, $\{x, y, z\}$) for Xact T

- Assume MARTS(T) = 60
- S_k = closest server for key k

(lowTS, highTS) for selected servers



- readTS(T) = $\min\{100, 90, 70\} = 70$

MARTS for Read-My-Writes Consistency

- MARTS(T) = maximum timestamp of all previously committed Puts in current session for objects accessed by T
- **Example**: BeginTx(RMW,$\{x\}$) for T_5 at client A

Client A: $W_1(x_1), C_1, W_4(x_4), C_4$
Client B: $R_2(x_4), W_2(x_2), C_2,$
Client C: $W_3(x_3), C_3$

Servers' State

Server	Store
S1	$(x_0, 0)$, $(x_1, 20)$, $(x_3, 80)$, $(x_4, 120)$, $(x_2, 300)$
S2	$(x_0, 0)$, $(x_1, 20)$
S3	$(x_0, 0)$, $(x_1, 20)$, $(x_3, 80)$
S4	$(x_0, 0)$, $(x_1, 20)$, $(x_3, 80)$, $(x_4, 120)$

MARTS for Monotonic Reads Consistency

- MARTS(T) = maximum timestamp of all previous Gets in current session
- **Example**: BeginTx(MR,$\{x\}$) for T_5 at client B

Client A: $W_1(x_1), C_1, W_4(x_4), C_4$
Client B: $R_2(x_4), W_2(x_2), C_2,$
Client C: $W_3(x_3), C_3$

Servers' State

Server	Store
S1	$(x_0, 0), (x_1, 20), (x_3, 80), (x_4, 120), (x_2, 300)$
S2	$(x_0, 0), (x_1, 20)$
S3	$(x_0, 0), (x_1, 20), (x_3, 80)$
S4	$(x_0, 0), (x_1, 20), (x_3, 80), (x_4, 120)$

MARTS for Causal Consistency

- MARTS(T) = maximum timestamp of all previous Gets & Puts in current session
- **Example**: BeginTx(CS,$\{x\}$) for T_5 at client A

Client A: $W_1(x_1), C_1, R_4(x_3), W_4(x_4), C_4$
Client B: $R_2(x_4), W_2(x_2), C_2,$
Client C: $W_3(x_3), C_3$

Servers' State

Server	Store
S1	$(x_0, 0), (x_1, 20), (x_3, 80), (x_4, 120), (x_2, 300)$
S2	$(x_0, 0), (x_1, 20)$
S3	$(x_0, 0), (x_1, 20), (x_3, 80)$
S4	$(x_0, 0), (x_1, 20), (x_3, 80), (x_4, 120)$

MARTS for Eventual, Bounded Staleness & Strong Consistencies

- Eventual Consistency
  - MARTS(T) = 0
- Bounded(t) Consistency
  - Client maintains mapping from real time to each primary server's logical clock
  - MARTS(T) = **realTimeToLogicalTime**(client's clock time - t)
- Strong Consistency
  - Let $maxTS(k_i)$ denote the maximum timestamp among all versions of key k_i in the primary server for k_i
  - MARTS(T) = $\max \{maxTS(k_i) \mid k_i \in \text{key-set}\}$

EndTx

- Client selects a commit coordinator (CC) to process EndTx for Xact T
  - Only primary servers with data updated by T will be participants in T's commit process
  - CC is one the of participants
- Client sends a commit request to CC with the following information:
  - **readTS(T)**
  - set of Puts for T (known as **Put-set**)
  - largest commit timestamp among all Gets/Puts in the session (**LCT**)
- Let $\{P_1, \cdots, P_n\}$ be the set of participants involved in the commit process
- CC partitions Put-set into $PS_1 \cup \cdots \cup PS_n$
  - Participant P_i is the primary server for keys in PS_i

EndTx (cont.)

- On receiving a commit request from client,
  - CC updates its local clock timestamp to max(local clock timestamp, LCT+1)
  - CC sends a prepare-commit request to each participant P_i
    - prepare-commit request contains PS_i
- On receiving prepare-commit from CC, each participant P_i performs the following
  - proposedTimestamp = local clock timestamp
  - increments its local clock timestamp
  - appends (PS_i, proposedTimestamp) to its `pending` list
  - replies to CC with proposedTimestamp
- From all the proposedTimstamps received,
  - CC selects the maximum as commitTS(T)
  - CC sends commitTS(T) to all participants

EndTx (cont.)

- On receiving commitTS(T) from CC, each participant performs the following
  - updates its local clock timestamp to max(local clock timestamp, commitTS(T)+1)
  - validates whether it can commit *T*
  - sends commit/abort reply to CC
- If all participants voted to commit, CC commits *T* as follows
  - writes a commit log record to stable storage
    - Commit timestamp
    - Put-set of *T*
  - informs client that *T* has committed
  - informs participants of the commit decision

EndTx (cont.)

- On receiving commit decision, each participant P_i
  - processes PS_i by creating new object versions using commitTS(T)
  - appends PS_i to its `propagating` queue
- When a participant P_i has processed PS_i for T
  - P sends notifies CC that P has completed T
  - P removes T's entry from its `pending` list
- Each P_i asynchronously sends PS_i to secondary servers from `propagating` queue

Pending Transactions

- Certain operations at a primary server P might be blocked by a pending transaction T'
- Get(k) request for Xact T
  - If T' has updated the same key k & T'.proposedTime $\leq$ readTS(T)
    - Possible for commitTS(T') $\leq$ readTS(T)
- Validation request for Xact T
  - If T & T' have updated some common key & T'.proposedTime $\leq$ commitTS(T)
    - Possible for commitTS(T') $\leq$ commitTS(T)
- Replicating updates for Xact T
  - If T'.proposedTime $\leq$ commitTS(T)
    - Possible for commitTS(T') $\leq$ commitTS(T)

Azure Cosmos DB

- Supports five consistency levels:
  - Strong consistency
  - Bounded staleness consistency
  - Session consistency
    - Consistent Prefix + Monotonic reads + Read my writes + Monotonic writes + Writes follow reads
  - Consistent prefix consistency
  - Eventual consistency

References

- D. Terry, *Replicated data consistency explained through baseball*, CACM 56(12), 82-89, 2013
- D. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, M. Aguilera, *Transactions with consistency choices on geo-replicated cloud storage*, MSR-TR-2013-82, September 2013
- Consistency levels in Azure Cosmos DB

  https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels