One question from the first two tiers of Bloom's Taxonomy (Remember/Understanding)
Remember – **What does a write ahead log do?**
Answer: A write ahead log is a form of logging where the changes performed on a system are first
recorded in a log prior to being committed on the system. This ensures that small appends are done for
each update which can improve atomicity and performance compared to other logging methods like
shadow copies.

One question from the second two tiers of Bloom's Taxonomy (Apply/Analyze)
Apply – **System X uses shadow copies to support maintaining logs of specific files. However, the system needs to support better atomicity and have better performance. What type of logging can be introduced that addresses these specifications?**
Answer: A write ahead log can be introduced. Since small appends are done for each update operation
in a write ahead log, the atomicity and performance are better. This is because small appends are more
atomic and cheaper than copying/snapshotting an entire file which is done using a shadow copy.

One question from the last two tiers of Bloom's Taxonomy
In lecture, we saw two kinds of write ahead logs: Recovery on read, and recovery on failure. Recovery on failure had two other features to improve write and recovery speed, namely caching and log truncation. Consider a company developing customizable devices intended for crop sensing. The devices are solar powered and connect once a day over and LTE connection to a central database to aggregate information across different fields and different points in the same field. Users may write their own software to run custom sensors or use the off-the-shelf sensors also available from the company. Each device is expected to support more than one sensor, and expected to record measurements about every 15 minutes, though this can be modified by the user. The sensors can measure things like wind speed, humidity, soil moisture content, sunlight, etc. To conserve energy, a hardware module wakes and runs tasks at the specified frequency and expects tasks to run to completion to record a single measurement. This has the additional advantage of allowing a task to recover if it crashes on a particular measurement. The company expects this device to be a relatively low volume product and has allocated a relatively small team to bring it to market. Design a write-ahead logging data storage solution that would be appropriate for this application, evaluating each option to motivate your choice.
Possible answer:
In this case I would go with a recovery on read data storage solution, since reads could plausibly be the
least common action in this application. Since the devices are solar powered, and users can write

custom software to run on the device, the likelihood that the custom software contains errors, or the
device loses power at least once a day is quite high, and recovering on each failure could disrupt
monitoring. The device also likely will not read its own data, since it should only be recording
measurements, and sending the data to the central server for processing. The only reading would need
to be performed in the daily LTE upload. Additionally, we can add a simple truncation action after the

data has been uploaded, by clearing all data that has been sent to the central server. More frequent log
truncation would not be valuable to implement, since the read performance of the device is not critical,
and the additional programming effort could be better spent elsewhere in the project. Caching is also
not important for this device, since neither reading nor writing is highly time sensitive and relatively
infrequent.

%%
Remember:

What is SSE?

SSE stands for Server-Side-Events, it is an application protocol used to establish a TCP-based persistent
connection between apps and data centers. It is a unidirectional protocol that is used to achieve "at least
once" delivery via acknowledgments and retires.

Apply:

What is the use heartbeat mechanism and where else can we use it?

It is a mechanism used by the RAMEN protocol.

In this protocol, if a message isn't delivered (e.g., seq#3), the client reconnects using the last successful
seq number (e.g., seq=2), indicating non-delivery of seq#3. Then the server resends the non-delivered
message.

This is accomplished using the heartbeat mechanism in which the server sends a byte-sized heartbeat
message every 4 seconds to check for a connection.

A client reconnects if no heartbeat/message is received for 7 seconds and the data stream continues.

Another use of the heartbeat mechanism would be by a view server to determine if the primary server is

active or not. If no message is received for a certain period of time then that indicates that the primary is
down and the view server should appoint the backup as the primary. This would also allow us to retrace
the last step performed by the primary and determine how to resume services.
Evaluate:
Justify the decision to use SSE over Web sockets and HTTP/1.1 with long polling by the RAMEN protocol.
RAMEN was introduced to eliminate polling through push messages, so using long polling would have
been counterintuitive in a way. Moreover, since we wait for a certain period of time before sending a
response in long polling it would have caused unnecessary delays in operations. Additionally, there would
be a certain number of redundant open connections using long polling that would be waiting to receive or
send a response, since millions of requests are to be handled this would cause unnecessary overhead
and higher latency.
Websockets are powerful and introduce bidirectional streaming over one persistent connection, however,
they are more complicated to implement and introduce a greater overhead per connection over SSE.
They also have a higher binary size impact and would require additional infrastructure to implement at
Uber. Considering the server had to handle millions of requests, this overhead would be substantial and
cause delays.
SSE was the best choice for RAMEN because it had minimal overhead and impact on binary size;
simplified operations because it used existing HTTP + JSON API stack at Uber; and it also supports
mobile SDKs

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Study Activity 5
(Remember)
Question: **What is the main benefit RAMEN (Real-time Asynchronous Messaging Network) provides?**
Answer: Goal: Eliminate polling through push messaging

(Analyze)
Question: **Compare between RAMEN and Polling, what are some scenarios where we would**

chosen RAMEN or Polling over the other?
Answer:
Polling — It can be used in scenarios where real-time updates are not critical, and a slight delay is acceptable.
RAMEN — Use for services that require short latency between the event and updates shown to the user.

(Create)
Question: **Your company makes use of RAMEN for their food-delivery platform. Give 3 kinds of messages would you think your application needs and rank their priority levels?**
Answer:
1. Your food has arrived — High Priority
2. Rider's location status — Medium Priority
3. Your food is still being prepared — Low Priority
4. Your order is processing — Low Priority

%%
Create:
Describe a real-word software scenario in which Kafka would be useful. List some system characteristics that Kafka would be useful to implement.

For example, you could use Kafka to manage real time messaging in a new dating app. You would want Kafka's high-throughput data ingestion from many sources since there are many senders and many receivers through your app, and it's important that everyone gets their messages on time.

Remember
What is Leader Election in Kafka, and why is it important?
Leader election is the process of selecting one broker as the "leader" for one particular partition of a topic. Kafka will distribute data over multiple brokers, and each partition of a topic can be present on the multiple brokers for fault tolerance and scalability. Leader election is important, as it is key for fault tolerance and high availability of Kafka clusters.

Analyze:
Using your knowledge of Kafka Architecture, explain how Kafka is able to achieve parallelism?
Kafka consists of partitions in the form of an ordered sequence of data records, which enables parallel processing. It allows stream tasks to process both parallel and independently. The maximum parallelism is limited by the number of stream tasks, determined by the total number of partitions that the input topic has.

%%
Group 4
Remember:

What are the key steps of MapReduce:
Answer:
Input splitting, Map phase, Shuffle and sort, Reduce phase
Apply:
A company needs to setup a monitoring system capable of performing real time data analysis. What should the company use to achieve these objects and why?
Answer:
The company should use stream processing because it has low latency and enables immediate insights and rapid decision making.
Evaluate:
A climate research team has produced massive datasets that must be processed. Argue why MapReduce should probably be used as opposed to Batch Processing.
Answer:
Unlike Batch Processing, MapReduce has a Reduce Phase that combines all the results into a final output, which is useful when trying to draw conclusions in a study.
Also, to process such a massive dataset, parallel processing may be required. Batch processing usually processes the data chunks in a sequential order, so is ill suited for this task.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Remember/Understanding
How does a view server discover failures?
Ans: It gets pinged by the primary / backup servers when there is a failure
Apply/Analyze:
You are designing a performance-critical system with multiple replica servers. Implement a view server and explain how the system would handle primary failure.
Ans: A view server would go in between all coordinators and servers, determining a primary replica server and pointing coordinators towards it. Primary failure would be handled when the view server receives a lack of pings from the primary, and appoints a secondary to become the new primary. The view server would then point coordinators towards the new primary while the original is in recovery.

Evaluate/Create:
Evaluate the impact of primary failures on view servers in a database replication system.
Ans:
In a database replication system using Replicated State Machines (RSMs), the primary-backup mechanism is essential for keeping data consistent and available. The view server's main role is to make sure only one replica serves as the primary. If the primary server fails, this could lead to issues with data consistency or availability, especially if shifting to a backup server is slow or not well-executed. Additionally, the effectiveness of the view server in managing these roles is crucial, but it can be hindered if the view server itself encounters problems, or if the switch from the primary to a backup server isn't smooth.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

o Remember:
▪ Q: **What are the three tradeoff problems data replication can introduce?**
▪ A: consistency, availability, and partition tolerance.
•
o

•
•

•

List type question from Bloom's Taxonomy

Understand:
▪ Q: **Maria and Bailey are conducting actions concurrently on a distributed database. Maria does (A, B, C) and Bailey does (X, Y, Z). Bailey sees B from Maria before A. Does the principle of linearizability apply to this distributed system?**
▪ A: No, because while each actor knows the order of their own actions, this gets contradicted in the overall system. A linearized system needs to support monotonic reads, monotonic writes, and "read your writes".
Classify type question from Bloom's Taxonomy

o Evaluate/Create:
▪ Q: **Compare and contrast the approach to quorum levels in CassandraDB and other NoSQL databases. Discuss the advantages and potential challenges of allowing programmers to choose the quorum level for each read/write in CassandraDB compared to the single read/write strategy in some other NoSQL databases.**
▪ A: Cassandra DB provides flexibility by allowing programmers to choose the quorum level for each read/write operation. This flexibility empowers developers to tailor consistency and availability based on specific use cases. In contrast, some other NoSQL databases follow a single read/write strategy, which simplifies configuration but may not offer the same level of fine-tuning. The advantage of Cassandra's approach is increased control, but the challenge lies in the complexity of managing different quorum levels and potential inconsistency if not chosen wisely.

o Apply/Analyze
▪ Q: **We've learnt that we can have varying thresholds for our write and read quorums to ensure database consistency. When would we want to have a write quorum requiring all replicas and a read quorum only requiring a single replica?**
▪ A: When we want to optimize for fast reads. Writing will require acknowledgements from all replicas, but reads will require only one acknowledgement making it faster.

%%

One question from the first two tiers of Bloom's Taxonomy (Remember/Understanding),
What is a linearizable distributed system?
A distributed system is linearizable if it preserves the partial ordering of distributed actions. In other words, it provides a consistency guarantee for concurrent operations ensuring that the outcome of any set of operations in the distributed system is equivalent to some sequential execution of those operations. For example, if distributed actor A does task 1, task 2, and task 3 in
that order, and distributed actor B does task 4, task 5, and task 6 in that order, then no one should
see task 2 before task 1, or task 6 before task 4. However, task 4 may be completed before task 1, as
they occur on different distributed actors and thus have no ordering guarantee.
Ethan:
One question from the second two tiers of Bloom's Taxonomy (Apply/Analyze),
Describe how "linearizability" (Montonic reads, monotonic writes, read your writes) is important to a shopping application in an e-commerce system.
Answer:
Consider a shopping cart application in a distributed e-commerce system. A user adds items to their
cart and proceeds to checkout. Monotonic reads ensure that the user, during the checkout process,
consistently sees an up-to-date view of their cart. Monotonic writes ensure that the order of adding
items to the cart is preserved. Read-your-writes consistency guarantees that once the user receives
confirmation of the order placement, any subsequent attempt to view the order details reflects the
most recent changes.

One question from the last two tiers of Bloom's Taxonomy (Evaluate/Create),
A senior engineer at your company believes that replication of data is the best option to not lose data without having any tradeoffs. Evaluate why he might be incorrect.
While data replication can be a valuable strategy for ensuring data availability and resilience, it's not without its challenges and potential tradeoffs. Here are some reasons why the senior engineer's belief might be incorrect:

1. Increased Storage Costs: Replicating data requires additional storage resources. Depending

2.

3.

4.

5.

6.

on the scale of replication (e.g., full copies or partial copies), this can significantly increase storage costs.
Complexity and Maintenance: Managing and maintaining multiple copies of data can introduce complexity. It requires careful synchronization mechanisms and robust maintenance processes to ensure consistency across all replicas.
Consistency Challenges: Achieving and maintaining consistency across replicated data can be challenging. In distributed systems, ensuring that all replicas are up to date and consistent with each other may require sophisticated coordination mechanisms, which can impact overall system performance.
Operational Overhead: Managing a system with data replication introduces operational overhead. This includes monitoring, troubleshooting, and implementing disaster recovery plans. It requires skilled personnel and ongoing effort to ensure the system functions as intended.
Latency in Data Access: Depending on the replication strategy, there may be latency in accessing the most recent data. In scenarios where real-time access to the latest data is critical, this latency could be a significant drawback.
Impact on Performance: The process of replicating data can impact the performance of the system, especially during heavy read and write operations. The overhead of maintaining multiple copies can degrade performance if not managed carefully.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Study Activity 5
Remember/Understanding Question:
What is a view server used for when using a primary server with backup servers and how does it do this?
Answer: A view server is a server that lets clients figure out which of the servers (replicas) is the primary. This ensures that clients only talk to the primary server and are able to switch to the new primary server if the existing one goes down. It does this by keeping track of the various servers that exist, and regularly communicating with them to ensure they are still functioning correctly. Then, the client asks the view server for the primary server, and then it can make a request to this primary server.
Apply/Analyze Question:
In the lectures we discussed what happens if the primary server can't reach the view server (in this case it changes to no longer be the primary server). However, what if the view server goes down completely (none of the servers can reach it)? Give some ideas of how you could deal with this to ensure clients can still talk to your service.
Answer: Some ideas to deal with this:
- Have backup view servers. In this case, when the view server goes down, a new view server could take its place.

- Switch over to a distributed quorum-based system. If the servers can no longer contact the view server, they can automatically switch over to deciding amongst themselves who the primary is. In a sense, the servers could act as a distributed view server. In this case, the clients would need some way to directly find/contact one of the servers to figure out who the primary is.
- Have no view server at all. Instead, let the clients contact any of the servers and treat it as the primary. Although this is a temporary solution that can cause the replicas to become inconsistent, this may be a good solution if the view server is only down briefly (and then there is some way to resolve any inconsistenties).

Evaluate/Create Question:

In the lectures, there was only ever 1 view server for all clients. However, as we've seen previously, it's typically advantageous to have replicas all over the world to reduce latency for a global audience. How could you have multiple replicas of the view server all over the world, while still ensuring there is only 1 primary server (i.e. all view servers agree on the same primary).

Answer: This is a similar idea to figuring out how to keep a distributed DB consistent (week 7), and many of the same ideas can be applied:

-

-

Whenever any data inside the view server is updated (who the primary is, who is offline, etc), wait for the data to be consistent across all view servers. Thus, all clients will always have the exact same primary.

The above can be sped up by using quorum-based decision making. Only wait until some/all of the view servers have received the data.

Have one “primary” view server which holds ground truth. Clients can still query the closest view server to get the primary, and then in turn each view server regularly polls the primary view server for an updated view of the world. Thus, although there may be some delay in changing the primary, the performance will be greatly improved over the above two ideas.

%%

Group 34 -> Week 10, slides 20-25

Q1

Q: **What is the purpose of the "Shuffle and Sort" phase in the MapReduce framework?**

A: The "Shuffle and Sort" phase in MapReduce is responsible for organizing and rearranging the output generated by the Map phase. It ensures that the data from the map tasks is grouped and sorted based on the keys before the reduce phase. This phase is crucial for the efficient and correct processing of data in the subsequent reduce phase.

Q2
Question: **Provide an example of a scenario where MapReduce can be applied for efficient data processing including what happens to the data at each phase.**
Answer:
Input Data: Unstructured user activity data.
Input Splitting: Data is split into chunks for map tasks.
Map Phase: Maps process user activity entries, counting likes, comments, and shares.
Shuffle and Sort Phase: System groups data by user or content ID.
Reduce Phase: Reduces calculate total likes, comments, and shares per user or content.
Output: Structured data with user or content ID and engagement metrics

Q3
Q: **You are building a cloud monitoring and analytics platform that provides real-time insights**
into the performance of applications and infrastructure. Where would you use batch processing
and where would you use stream processing?
A: Ingesting data feeds for monitoring is most synergetic with the characteristics of stream processing as it requires low-latency processing of continuous data. On the other hand, analysis of historical data or generating reports are better suited for batch processing.

%%
Q: (Remember) **What are some common approaches to solving reliability problems?**
A: The concept of transactions, which provide atomicity and isolation throughout a distributed system is an approach used in modern-day systems that solve reliability problems such as write conflicts and losing commits due to a system crash.
Q: (Create) **Suppose now instead of operating with data directly from a database, the company**
wants you to build an S3 object storage system that handles incoming data from a scraper (for
example, in JSON format), utilize the concept of transactions to build a reliable system of S3 and
the scraper.
A: For example, we could utilize the concept of updates and commits. We could build a status system on both ends. S3 should track the status of the scraper on its last commit, (e.g. where did
the scraper stop on the last commit, was the scraper still working, etc.). The scraper will also have an update and commit system where it updates its result locally, and after a certain amount
of updates, it commits an update by pushing all JSON objects to S3. This way even if the scraper
crashes, no corrupted data will be committed, and it knows where to resume from by checking its
last status from S3.

Q: (Evaluate) **Suppose the company wants you to build a new database system while providing**
atomicity. The company also wants the system to be easy to maintain and understand by possible
new interns, and have less frequent write operations since the company wants to spend on
something other than extra storage, would you choose the Write-ahead Logs approach or the
Shadow Copies approach and why?
A: Shadow Copies are more ideal under the given constraint. Shadow Copies are generally simpler to explain and understand for beginners or new interns. In Shadow Copies, data modifications are written into some temporary storage area, and only when the transaction is ready to be committed The Write-ahead Logs approach records any change to the database before it is applied, in a log that can be played back in case of system crashes. This requires extra
storage and an understanding of logging and rollback operations which can get complex.
The Shadow Copies approach usually requires less frequent write operations compared to the Write-ahead Logs approach, since in Shadow Copies, the writes are batched together and done once on commit, whereas in Write-ahead Logs, every single change is logged immediately.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Q1: **Remember: What are the 3 advantages of ROS?**
1. Modularity: reusable components for a wide range of tasks
2. Scalability: good framework for both small and large scale projects
3. Flexibility: supports a large array of configurations, resulting in wide use case

Q2: **Analyze: In what scenarios would a publisher/subscriber architecture be preferred over a push notification architecture?**
Both architectures are able to asynchronously push notifications triggered by an event to consumers. Push notifications require knowledge of the consumer, and in many cases, a message queue for each consumer is needed. This does not scale well, especially in cases where there are many consumers interested in the same data, since maintaining knowledge and a message queue for each consumer can become infeasible. A publisher/subscriber architecture decouples the publishers from the subscribers and allows for multiple consumers to subscribe to a topic at a time.
Q3: Evaluate: **You are building a distributed architecture with a few simple components, some of**
which may be hosted cross-platform. The components may be unreliable; they may not be able
to immediately respond to messages. Because of this, communication between components is
event-driven. You are building this with the goal of servicing a small number of users. Would a
publisher/subscriber architecture be a good choice here? Explain.
The publisher/subscriber architecture seems good at first glance, as one of its main strengths is

handling cross platform communication in non-real time communication settings. However, it is likely overkill here as there are only a few simple components and not many users, meaning we do not expect there to be large amounts of messages being sent/received. This means the scalability that the publisher/subscriber architecture provides cannot be fully leveraged. A simpler push notifications architecture is likely a better fit.

%%

Remember/Understanding

Q: **Why use a Pub Sub messaging Pattern? Give 2 reasons.**

A: The Pub Sub messaging pattern allows for (2 of): decoupling of components, flexibility, scalability, feal-time data handling,ease of integration of hardware and software, and facilitates complex interactions among nodes.

Apply/Analyze

Q: **Suppose a manufacturer is thinking of incorporating the Robotics Operating System (ROS) into their development of autonomous humanoid robots designed to perform high-endurance tasks. What's one advantage of encorporating such software?**

A: ROS allows developers to design and implement individual components separately due to it's modularity. Thus, this facilitates easier integration of various components such as sensors and algorithms, making it suitable for the various complex components used in autonomous robots.

Evaluate/Create:

Q: **Why does ROS's Publisher subscribe architecture not utilize a message broker?**

A: A message broker is most suitable when we need broad information broadcasting, i.e. a Node would need to send messages to plenty of other Nodes. It's also best used when we immediate consumer responses are not required. Both of these aren't the case for robots. Generally sensors are mainly interacting with the robot's brain, which in turn communicates to the controls. Also, we will want fast responses to quickly react to environment change (e.g. autonomous car surprised by pedestrian needs quick reaction)

%%

Tier 1/2:

Q: **What is 2 Phase Locking (2PL), and why does it generate conflict serializable graphs?**

A: 2PL means that an action cannot proceed until it has acquired a lock, and once the action releases its lock it cannot acquire another one. This generates conflict serializable graphs because inability to reacquire a lock after one has been released eliminates the possibility of circular dependencies. This lack of circular dependencies ensures that all graphs are conflict serializable.

Tier 3/4:

Q: **For the code below, write each of the edges of the conflict graph, and conclude whether the**
schedule is conflict serializable.
T1: read(x)
T3: write(z, x)
T2: read(x)
T1: read(z)
T3: write(x, 30)
T1: write(y, 10)
T1: read(z)
A:
T1.1 -> T3.2
T3.1 -> T1.2
T3.1 -> T1.4
Thus because the graph is cyclic, the schedule is not conflict serializable.
We can see that T1.1 performs a read of X before a future write to X from T3,
Then T3.1 writes to z, before T1 is able to read z at T1.2.
Thus a conflict.
Tier 5/6:
Q: **You are a member of an engineering team developing a website for Oceanic Airlines. You**
have recently incorporated a feature enabling clients to select their seats during the flight booking process. One team member found a bug: booking two tickets simultaneously could
cause the same seat to be assigned to two different tickets. Identify which ACID property is not
respected by the booking service and propose a solution to prevent this issue.
A: The booking service is not respecting the Isolation property. To ensure isolation, we can use two-phase locking. Before the seat selection can be processed, the client's instance must acquire the lock corresponding to the seat before being able to indicate it as taken. That way, when the second seat selection scheduled will receive the lock, the seat would be already taken and an error message could be displayed to the user.

%%
Types of threats and Chrome browser security
Q1. (Understand)
Identify the type of security attack where:

- Bob is reliable and everyone has a good opinion about Bob.
- Mallory tells Alice that "Bob is unreliable"
A1. Misrepresentation
Q2.(Apply)
Google Chrome focuses on 3 factors for security:
(1) reducing the severity of vulnerabilities

(2) reducing the window of vulnerability
(3) reducing the frequency of exposure
Describe the method they use on how they address each factor.
A2.
(1) severity - sandboxing architecture so that potential vulnerabilities are contained in a controlled environment.
(2) window of vulnerability - auto updates and encouraging users to update frequently
(3) frequency of exposure - warning about malicious sites
Q3. (Evaluate)
The chromium browser architecture uses a modular architecture with two main components: the
Rendering Engine and the Browser Kernel. Additionally it uses sandboxing to isolate the rendering engine.
The Tahoma browser architecture also uses a modular architecture using a virtual machine
monitor. Sites are defined by manifest files (file containing metadata for a group of related files),
and each site runs in a separate protection domain. Additionally, the user is able to approve
requests for each web site.

Describe the pros and cons of each approach.
A3.
Chromium Browser Architecture:
Pros:
- Modularity - separation of Rendering Engine and the Browser Kernel allows for clear role distinctions, making it easier to manage and maintain the codebase.
- Sandboxing - provides an additional layer of security. Even if a vulnerability is exploited, the attacker's access is limited, reducing the impact of potential breaches.
Cons:
- Resource Overhead: Running components in separate sandboxes may introduce some resource overhead, affecting performance to some extent.
Tahoma Browser Architecture:
Pros:
- Strong Isolation - running each site in a separate protection domain using a virtual machine monitor means compromising one site doesn't directly impact others.
- Manifest-files - clear definition of which resources belong to a particular site, aiding in security and resource management.
- User Approval - Allowing users to approve requests for each website enhances user control and awareness of potentially risky interactions.
Cons:
- User Approval - Allowing users to approve requests for each website also opens up a vulnerability as the user may incorrectly approve malicious sites.

%%%
Understanding

recover(log):
commits = {}
for record r in log[len(log)-1] .. log[0]:
if r.type == commit:
commits.add(r.tid)
if r.type == update and r.tid not in commits:
cell_write(r.var, r.old_val) // undo
In the given code, what happens to the 'update' records in the log if their corresponding transaction ID is not found in the 'commits' list?

In this case, we notice that the code performs a rollback action for the 'update' record. This is done by using the old value of the variable back to the cell and effectively undos the change.

Analyze

A student has devised the following code for a read operation that attempts to satisfy atomicity in ACID:
read(log, var):
commits = {}
// scan backwards
for record r in log[len(log) - 1] .. log[0]:
// keep track of commits
if r.type == commit:
commits.add(r.tid)
// find var's last committed value
if r.type == update and
(r.tid in commits or r.tid == current_tid) and
r.var == var:
return r.new_value
Explain how it satisfies these principles?

In this situation, if an operation on a var fails before commit and we attempt to read the same var, since it has not commited, we cannot read this version of the var. We can only read a version of the var that has been commited (with commit history logged). This ensures atomicity

Evaluate

A student has devised the following code for a read operation that attempts to satisfy atomicity in ACID:
read(log, var):

commits = {}
// scan backwards
for record r in log[len(log) - 1] .. log[0]:
// keep track of commits
if r.type == commit:
commits.add(r.tid)
// find var's last committed value
if r.type == update and
(r.tid in commits or r.tid == current_tid) and
r.var == var:
return r.new_value
What might be a problem with this code, especially in terms of efficiency? How do we improve this ?
(Alternate: write the code for recover(log) that maintains the same functionality)
If every read needs to scan back in the logs, reads become quite inefficient. One way to improve is to abstract the log checking code into a separate `recover` function that is run after each crash. We can then perform a simple read assuming the state has been recovered and is correct during operation. Although recovery now takes a special operation, this might be an acceptable tradeoff if crashes are not frequent and recovery is allowed to take more time.

Create
recover(log):
commits = {}
for record r in log[len(log)-1] .. log[0]:
if r.type == commit:
commits.add(r.tid)
if r.type == update and r.tid not in commits:
cell_write(r.var, r.old_val) // undo

For the given code, can you write some brief pseudo code or algorithm that would help the provided code to handle commiting messages?
function handleCommitRecords(log):
for record in log:
if record.type == 'commit':
# Mark the transaction as committed
markTransactionAsCommitted(record.tid)
# Apply the changes made by the transaction
applyChanges(record.tid)
function markTransactionAsCommitted(tid):
# This function marks a transaction as committed
transactionTable[tid].status = 'committed'
function applyChanges(tid):

# This function applies all changes made by the committed transaction
for updateRecord in getUpdateRecords(tid):
applyUpdate(updateRecord)
function getUpdateRecords(tid):
Retrieve all update records for the given transaction ID
return filter(lambda record: record.tid == tid and record.type == 'update', log)
function applyUpdate(updateRecord):
# This typically involves writing the new value to the database
database[updateRecord.var] = updateRecord.new_val

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Questions
1. Remember/Understanding
Q: **What is misrepresentation in the context of security?**
A: A malicious actor calling a safe server unreliable or unsecure.
2. Apply/Analyze
Q: **How can a browser protect users from malicious online content? (hint: Chrome)**
A: Run the rendering engine in a low-privilege sandbox. This means even if the rendering engine is breached, there is little the malicious actor can do.
3. Create/Evaluate
Q: **Why use a trust management system?**
A: When you want to avoid the need to authenticate each user who attempts to access your application. It allows you to authorize based on the concept of trust which allows arbitrary services to query into your application. This is also helpful for decentralizing the authentication process and allowing an authorization algorithm to be run on distributed systems.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Question 1 (first two tiers): **How could/should one minimize the risks of token theft?**

Question 2 (middle two tiers): **Analyzing the role of each component within a JSON Web Token**
(JWT) structure, how does the specified signing algorithm in the header contribute to ensuring the
token's security, integrity, and authentication?

Question 3 (last two tiers): **Compare and critique the implications of 2 options in token management in microservices on an overall system's security and scalability.**

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(Tiers 1-2, Remember) **What situations would require a server to initiate an interaction with a client?**

- Any of the following:
- Delivering a text message to a user
- Notifying a user that their Uber driver has arrived
- Notifying a user that an item on their wishlist has gone on sale
- etc.

(Tiers 3-4, Analyze) **Compare the advantages and disadvantages of email notifications and push notifications.**

- Email notifications can deliver a lot more content in the email body, while push notifications can deliver only a small amount of content.
- Email notifications arrive slower than push notifications.
- Push notifications require the user to install an app; email does not.
- Push notifications are more scalable.
- Emails are not interactive; push notifications can include interactivity in the app after the notification is clicked.

(Tiers 5-6, Evaluate) **Suppose you are a software engineer at a large company called Neta, and your company is planning to launch a new social media app called Strings. The company**
wants you to design a system that allows notifications (e.g., someone replied to your post,
someone messaged you) to be delivered to users without much delay (i.e., a delay of a few
minutes is acceptable). What method would you use to deliver notifications if you anticipate (a)
a low volume of notifications, or (b) a high volume of notifications? Your solution should be cost
effective given the expected traffic.

- (a) For a low volume of notifications, it may be easiest to use email notifications, or perhaps SMS notifications, to deliver information to users in a cost effective manner. Alternatively, an HTTP long-polling approach can be used to obviate the need for a dedicated notification service.
- (b) For a high volume of notifications, it would be more effective to use push notifications, such as by using the Apple Push Notification Service (APNs) or a custom notification service like RAMEN.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Question 1 (Remember)

How did RAMEN help Uber deliver a better user experience?

In the context of Uber, RAMEN’s goal is to eliminate polling (which overloads Uber servers) through push messaging. It does this by introducing a new microservice, Fireball, that decides on the right time to generate a message payload. Fireball can leverage existing polling endpoints, revoking the need to rewrite polling APIs. Messages sent minimize data usage. Fireball ensures messages are delivered even in bad network conditions by retrying the request within the message’s TTL.

Question 2 (Apply)

A company's mobile software architecture is suffering from backend degradation, and notices that the polling frequency for its push notification system varies, with polling calls potentially consuming the majority of the backend API gateway requests. How could they improve their push notification system to lighten the load on their APIs and deliver a better user experience?

RAMEN is likely the best bet to clean up this push notification system. RAMEN helps deliver a solution in several parts: the first is to introduce a microservice, separate from the API gateway and triggered by push triggers, to handle when and who to push notifications to. Then the API gateway constructs the message payload and forwards it to the push delivery system, which dispatches to users. The company can choose to implement the protocol, but one option for delivery would be the RAMEN protocol, which is based on a persistent TCP connection.

Question 3 (Evaluate)

Say Amazon, the online shopping platform, is considering implementing RAMEN over its current method, which implements a standard polling approach. Would you recommend the switch? Why or why not?

In general, it makes sense for Amazon to use a notification approach like RAMEN. Similar to Uber, Amazon has to deal with a ton of real-time data and many parameters (e.g. product catalog, user shopping cart, etc.). The demand for items and shopping time can change dramatically in a matter of seconds, which means any form of naive polling for messages will overload servers and lead to backend degradation. The state of product delivery changes frequently as well – from being in a fulfillment center to being shipped across an ocean, in a delivery truck, and finally sitting at your door. All this real-time data that can fluctuate intensely over short periods of time indicate that RAMEN is a necessary approach to push notifications for Amazon.

%%