IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.57
Volume 4, Issue 1, August 2024

Interactive Document Editing and Distributed Synchronization Using Azure Cosmos and WebSockets

Sri Rama Chandra Charan Teja Tadi
Lead Software Developer, Austin, Texas

Abstract: *As interactive (collaborative) work transitions to real-time multi-user environments, the underlying infrastructure handling the interactions must also evolve in parallel. This architecture blends WebSocket-based streaming with Azure Cosmos DB's distributed consistency model to synchronize document states across clients at low latency. Edits are encoded as deltas, which are captured through the database's change feed and dispatched using stateless Azure Functions. Operational transforms resolve concurrent edit conflicts to maintain intent across overlapping user activity. Document histories are managed through snapshot intervals and temporal indexing, while security is enforced through stream-level isolation and scoped authentication. The result is a robust event-driven foundation for large-scale, interactive editing.*

Keywords: Interactive Editing, Collaborative Systems, Real-Time Synchronization, WebSocket Communication, Azure Cosmos DB, Operational Transforms, Change Feed, Serverless Architecture

I. FOUNDATIONS OF CLOUD-BASED COLLABORATIVE DOCUMENT EDITING

Collaborative Document editing has been characterized by great technological progress that promotes the user experience with real-time, multi-user collaboration. The earliest collaborative editing only involved static systems where the users could not visualize each other's modifications until they saved the document and sent it back and forth using email. However, with the emergence of the internet and cloud computing, even more advanced platforms that provide simultaneous editing facilities have evolved, wherein several participants can edit in real-time and see the updates as they go on [5]. This phenomenon requires an effective architecture capable of resolving the conflicts that will result from concurrent editing. It has the ability to get all participants to see the same copies of the document.

The inclusion of Azure Cosmos DB in this architecture is warranted because it has a distributed design and multi-model database support, making scalability and reliability possible in document state synchronization among various geographic locations. As emphasized by current implementations, Azure Cosmos DB offers a distinctive value proposition through its change feed feature, where applications can observe changes in real time. This functionality is crucial in collaborative editing as it enables updates made by one user to be shared among all other users in real time, hence lowering latency and improving the interactive session. Second, it allows for support for a conflict resolution protocol that conforms to operational transformation, allowing seamless reconciliation of documents being edited in parallel by different users.

Furthermore, the requirement of near-instant synchronization in shared environments calls for low-latency communication channels like WebSockets [16]. Normal HTTP requests, while being optimal for most applications, can impose massive latency in multi-user environments largely because of their request-response model. WebSockets, by virtue of their long-lived, low-latency channels, enable steady streams of data between servers and clients, which support real-time document editing [13]. This transition from polling-based mechanisms to persistent connections is a significant user-experience advance, making collaborative editing tools more responsive and efficient.

In general terms, the building blocks of cloud-based collaborative document editing are fundamentally based on the intersection of revolutionary database offerings, such as Azure Cosmos DB, and low-latency communication protocols, such as WebSockets. By integrating these technologies, developers can design more reliable collaborative editing systems that are interactive and efficient in terms of document integrity and real-time synchronization. As collaborative

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.57
Volume 4, Issue 1, August 2024

environments become more popular and sophisticated, an understanding of these underlying principles will be essential to advancing the functionality of online editing platforms.

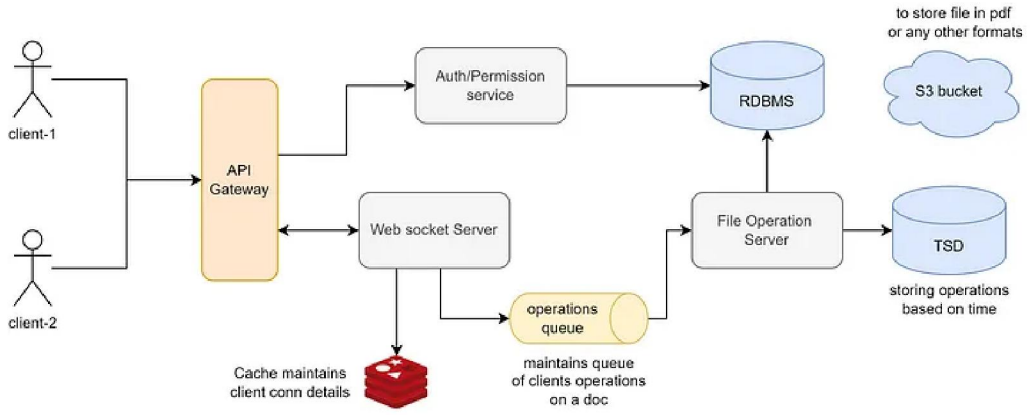

Figure 1: System Architecture for Real-Time Collaborative Document Editing
Source: Adapted from [21]

WebSocket Communication for Low-Latency Interaction

WebSocket protocol is particularly important to interactive document editing in the sense that it enables responsive and bidirectional server-client data communication. Contrary to request-response-based HTTP connections, which are inherently delay-prone and inefficient, WebSockets provide a persistent connection over which data can stream continuously and unobstructed. This bidirectional communication mechanism supports real-time updates and interaction of several users in real-time, which is fundamental to a shared editing experience [13]. The values of low latency and the capability of real-time data transfer provided by WebSockets enable applications to synchronize the changes made by various users optimally and faultlessly without bottlenecks such as those associated with conventional request-response protocols.

WebSockets are helpful in collaboration scenarios where user activity needs to be mirrored in real-time on all devices. When a user comments or edits, the edit or comment needs to be relayed to other users in real time so that they can view updates in real time. This feature is crucial in alleviating the cognitive load on users since it offers a smooth interaction experience that supports increased collaboration [5]. This responsiveness enhances user satisfaction and produces more dynamic interactions, where team members can collaborate on the document simultaneously without delays or document state inconsistencies.

Secondly, the application of Azure Functions with WebSockets enhances the architecture's performance in processing real-time data streams. Azure Functions can process incoming requests and send updates to users in a timely manner, enabling developers to build event-driven architectures that respond in real time to user activity [7]. This procedure enables the application to hold a steady stream of changes so that all clients will be updated on the latest version of the document. It is easier to have a change feed from Azure Cosmos DB since it enables the system to listen for updates genuinely and present them to clients using WebSockets.

Security is another important feature in real-time communication, especially in collaborative work with confidential information. Stream-level isolation and authentication features are enabled by the architecture in a way that only allows users to access or edit the documents [15]. This allows for a guard level to create collaborative edit spaces with support for users, making their interactions open but safe from access and possible data exposure.

The use of WebSocket communication on interactive document editing apps enormously improves the user experience through the provision of low-latency interactions and strong data synchronization. By facilitating real-time updates and secure connections with backend services such as Azure Cosmos DB, WebSockets enable collaboration tools to function effectively and efficiently, providing an innovative and dynamic editing experience [12]. The real-time flow of data and ability to resolve conflicts in real-time make WebSockets an essential feature of modern collaborative document editing architecture.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.57
Volume 4, Issue 1, August 2024

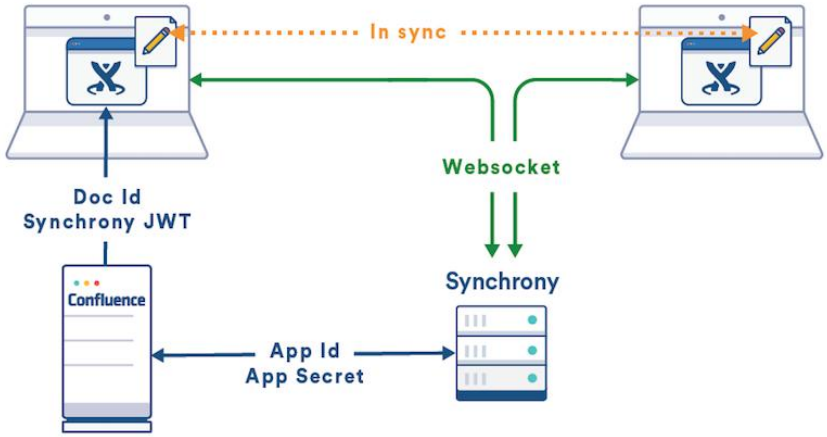

Figure 2: Collaborative Editing Architecture Using Synchrony and WebSocket Integration in Confluence
Source: Adapted from [22]

Leveraging Azure Cosmos DB for Distributed Data Synchronization

Azure Cosmos DB is a foundation for building distributed applications that need real-time synchronizations of document state and management across locations. Its support for globally distributed, partition-tolerant databases makes it particularly well-suited for interactive document editing applications. Perhaps the most impressive feature of Azure Cosmos DB is its global distribution, which allows applications to share data across regions. This replication is also essential in helping eliminate latency and provide users with instant access to the latest states of documents, no matter where they are located [9]. With the use of the database's own automatic and transparent data replication to various regions, developers can ensure that all concurrent edits are visible in real time, thus maximizing user interaction and productivity.

Apart from this, Azure Cosmos DB also has a robust consistency model that comes into play in collaborative edit scenarios when one can have several changes concurrently. The database provides developers with the ability to have different consistency models, i.e., strong consistency, bounded staleness, session consistency, or eventual consistency, depending on the application requirements. This capability helps ensure data correctness and integrity in shared environments by supporting different usages in the appropriate way [3]. For example, in cases where applications need to have strict data accuracy (e.g., account balances), strong consistency is used uniformly so that each user always has access to the most recent data available. At some point, for applications, consistency can be used such that updates allow for enhanced performance and robustness for network partition failures but sacrifice small lags for update visibility.

Azure Cosmos DB change feed also supports real-time updating. The change feed is a log where all the update and insert operations that are committed to the database are tracked, enabling applications to observe changes and execute corresponding responses in real time [10]. The facility is beneficial for synchronizing states of documents in a collaborative editing environment because it gives an adequate vehicle for executing changes propagated by one user to all collaborating users involved in the session. For example, when a user performs an edit, the operation might be represented as a delta, the difference between old and new versions of the document, which is then sent directly to the other collaborators over pre-defined channels, such as WebSockets. With Azure Functions in conjunction with the change feed, it is possible for developers to implement serverless solutions that automatically update without having to poll or unnecessary long-lived HTTP requests [11].

Furthermore, Azure Cosmos DB's native JSON document support is ideal for satisfying collaborative editing application data structure requirements. JSON's extensibility offers effective data storage and management of complex data types, which are critical when dealing with document constituents like images, text, and format information. Having an efficient NoSQL database scaled for performance and scalability provides developers with a way to work on application functionality without considering the infrastructure constraints below [10]. Overall, Azure Cosmos DB

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.57
Volume 4, Issue 1, August 2024

offers a complete solution for the distributed synchronization of data in collaborative editing apps, maximizing performance and user experience with guaranteed consistency and availability of document states.

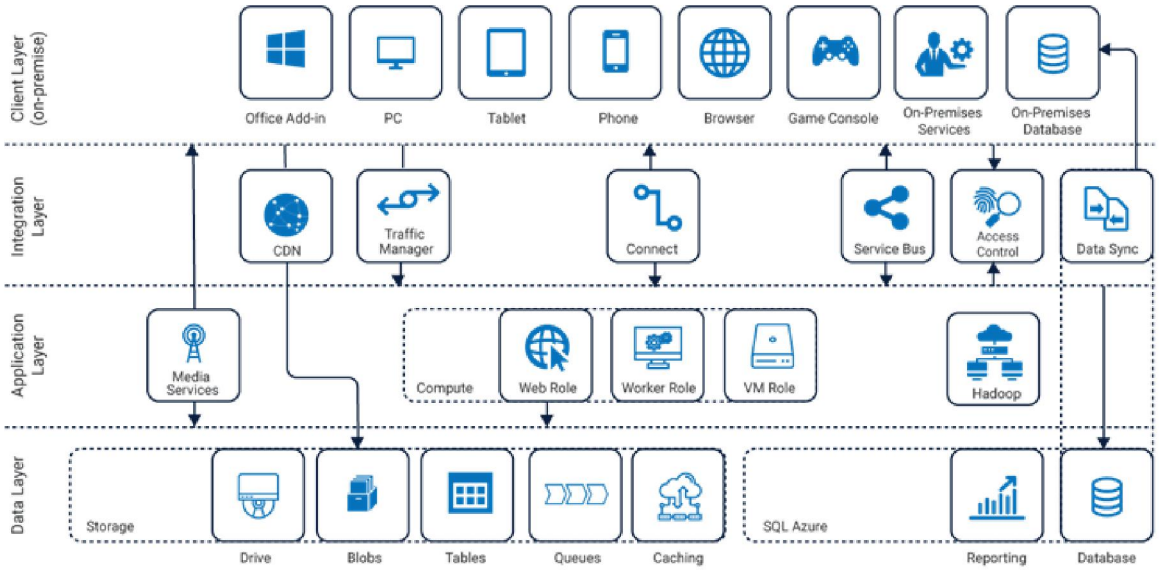

Figure 3: Azure Cosmos DB in a Multi-Layer Cloud Architecture
Source: Adapted from [23]

Concurrent Editing and Conflict Resolution Mechanisms

As collaborative editing develops, one of the most significant areas to develop is managing concurrent edits effectively. The simultaneous input by multiple users can result in conflicts, wherein two or more users make edits on the same section of a document. In response to such conflicts, various approaches have been employed, such as operational transforms (OT) and conflict-free replicated data types (CRDTs) [1]. These mechanisms not only support concurrent editing but also make sure that all editors have a standard view of the document on every device.

Operational transforms accomplish this by establishing a sequence of operations that can be composed so that each user's intent is preserved in case of overlapping edits. Whenever a user makes any change, it is redefined based on other outstanding changes so that all users can apply them without inconsistency [4]. For example, when two users enter the same location simultaneously, the operational transformation algorithm will assist in identifying the result of the document based on predetermined rules so that the edits are resolved without compromising the contribution of the two users. Such an adaptive method is most significant in ensuring usability and also enabling efficient collaboration between users with minimal technical knowledge.

In contrast to this, CRDTs aim to establish a mathematical underpinning that ensures changes from many users converge to one state irrespective of the order in which they are updated [2]. CRDTs are anticipated to handle conflicts naturally by applying a sound merger principle in which states can be combined without losing intent from any operation. For instance, within the user session where users have the ability to delete or alter the same area, CRDTs allow the system to handle those automatically, usually resulting in a merged and consistent document state that accommodates all users' inputs. Those mechanisms reduce conflict resolution overhead significantly, which is why they find applications in large-scale systems in which many users may simultaneously view the same data.

With these approaches, combined with the features of Azure Cosmos DB, a very responsive and user-friendly editing experience is achieved. The UI design can leverage the change feed with OT or CRDTs so that users' edits are processed in real time and propagated to others via WebSockets. This event-driven strategy maintains the history of the changes to the document, so backtracking is easy, and there is an unambiguous audit trail of changes, building user confidence and encouraging an open, collaborative culture.

Additionally, snapshot intervals or temporal indexing approaches can be used as a native system feature for document version management so that users can roll back to previous versions if conflicts happen or in the event of unwanted

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.57
Volume 4, Issue 1, August 2024

changes. Such an organized process not only facilitates conflict resolution but also helps strengthen the integrity of the collaborative editing platform so that users can work more freely with the content being edited. Therefore, pairing a good conflict resolution approach with a robust backend like Azure Cosmos DB is a good solution for efficient concurrent editing in contemporary cloud applications.

Delta-Based Synchronization and Change Feed Integration

The use of delta-based synchronization in shared document editing software effectively optimizes data propagation and conflict resolution efficiency. In concurrent document editing, which occurs in the multi-user setup, logging changes as deltas, that is, changes between states, is critical. Through the conversion of edits to deltas, the software minimizes network data and eliminates the potential for conflicts [5]. This approach is utilized for the detection of user intent with low payload for transmission to the storage layer, where the strengths of Azure Cosmos DB can be utilized for additional optimization of integration.

To enable change detection, the system needs to utilize sophisticated algorithms that are capable of identifying differences in document versions dynamically. Such processes, normally based on diffing algorithms, can be executed at various levels based on the data type of the documents involved. For instance, in a popular JSON schema with Azure Cosmos DB, one can track changes at the property level, document object properties are tracked for updates, or at an abstract level where one checks entire objects for changes [6]. Using such hierarchical monitoring provides very high-grained synchronization control and lightens the computational burden since only the changed portion of the document is run.

The change feed feature of Azure Cosmos DB supports developers in streaming such delta updates in real time whenever a change to the document happens. Change feed is an append-only, persistent log of all changes from which applications can infer real-time updates [11]. With Azure Functions, developers can define serverless processing logic that will listen to this change feed and call functions to push the changes. For example, when a user modifies a document, the modifications are noticed, encoded as deltas, and pushed over the change feed; Azure Functions can then push these modifications over WebSocket connections to all connected clients. This method not only reduces latency but also provides the users with a guarantee of always being able to see the most recent state of the document as soon as changes are made.

While designing schemas, a structured document with tangible fields for version history can significantly optimize the synchronization process. Each document may have metadata fields to store the date last modified, the document modifier, and a change-log array with the deltas made during the document's lifetime. This organization helps store a complete audit trail and facilitates the application to revert back to earlier versions quickly without violating the integrity of collaborative editing sessions.

Further, employing real-time processing logic improves the smoothness of user interaction. Upon reflecting delta changes, the application does not need to reload entire documents per edit but, instead, can merge arriving changes with its current in-memory document version via client-side operational transforms or similar reconciliatory methods [17]. This helps cut server-side computation load and enhances user experience via the provision of instant updates without perceivable lag.

Through the encoding and streaming of document modifications as deltas, applications can take advantage of Azure's architecture to improve user interaction without undermining the integrity of the collaborative process.

Architecture for Real-Time Update Propagation

The real-time propagation of updates in interactive document editing scenarios requires an event-based model that naturally includes several Azure services integrated together, such as Azure Cosmos DB, Azure Functions, and WebSocket relays. The architecture is also modular and enables each component to handle a single task, all of which come together to enhance performance and responsiveness. Essentially, the architecture is founded on the assumption that any updates to a document are propagated to all clients in real time and synchronized across distributed users.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.57
Volume 4, Issue 1, August 2024

Modular Architecture Overview

The architecture's primary elements are:

Client Applications: These can be web clients or mobile applications where users specify editing tasks. Each client has an open WebSocket to enable real-time communication.

WebSocket Servers: These servers enable two-way client communication. These servers get update notifications from the clients and propagate them to all connected users in real time.

Azure Cosmos DB: It is used as a horizontally scalable database backend. This is where the document data is kept, and changes are captured via its change feed, which is imperative in delta-based synchronization.

Azure Functions: They are serverless functions that offer event-driven process logic. They are invoked by changes in the Azure Cosmos DB change feed and execute the logic thereafter to push such changes to clients in real time.

Change Feed Integration: Cosmos DB change feed provides real-time monitoring of changes. It is a method of causing the system to respond instantly to changes that have been made to the document.

In this design, any update keyed into the client application is first sent to the WebSocket server. The server may either process the update locally or pass it on to Azure Functions to be processed downstream. The Cosmos DB change feed tracks these updates, and each delta is caught and made available for downstream processing.

Client Applications

Client applications are the user interface by which users can interact with collaborative document editing systems. Client applications are web or mobile and are optimized to provide an easy-to-use user experience without cutting down on powerful editing features [20]. All clients can manage real-time updates to documents using WebSocket connections, and hence, all users can see changes instantly across all instances that are connected. The design of such apps is usually a responsive front-end, built using frameworks such as React or Angular, to achieve interactive user interfaces and handle interactions in an optimized way. In order to start a WebSocket connection, the client begins by initializing the connection with the server upon launch so that it can receive updates in real time.

In the code below, the server is connected through WebSocket, and event handlers are set up so that incoming messages, like the server broadcasting document changes, are handled. The `sendChange` function illustrates how one can serialize and send changes to the server to enable all of the changes to be notified instantaneously in support of collaborative synchronization.

```javascript
const socket = new WebSocket('wss://your-server-url/websocket');

// Event handlers
socket.onopen = function() {
console.log('WebSocket connection established.');
};

socket.onmessage = function(event) {
const delta = JSON.parse(event.data);
applyDeltaToDocument(delta);
};
socket.onclose = function() {
console.log('WebSocket connection closed.');
};
// Send changes to the server
function sendChange(delta) {
socket.send(JSON.stringify(delta));
}
```

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.57
Volume 4, Issue 1, August 2024

The client app not only keeps the WebSocket connection alive to handle real-time updates but also uses operational transforms or Conflict-free Replicated Data Types (CRDTs) to handle collaborative editing. The models enable different users to co-edit the same section of a document without conflict. The client system architecture is also developed to pull the initial states of the documents and have an updated view reflecting other users' updates, improving the collaborative user experience.

For mobile apps, the design can include the use of frameworks such as React Native or Flutter, with the same patterns of WebSocket communication but with added support for mobile-specific interactions and performance optimization [19]. With proper networking management, mobile clients can achieve data consistency and rapid responsiveness under different network conditions.

WebSocket Servers

WebSocket servers are the center of real-time interaction delivery in terms of bi-directional communication between the clients. Designed to handle several connections simultaneously, WebSocket servers are important for broadcasting updates to the users within an interactive document editing system. After a client becomes connected, the WebSocket server provides a permanent connection, hence enabling it to broadcast updates in real time without presuming the client to be requesting new information continuously.

The main duty of the WebSocket server is to accept updates from clients (i.e., document changes) and forward such updates to all the online users. The following code sets up a basic WebSocket server that facilitates real-time broadcasting of document changes (deltas) to all connected clients in a collaborative editing environment.

```javascript
const WebSocket = require('ws');
const server = new WebSocket.Server({ port: 8080 });

server.on('connection', (socket) => {
console.log('New client connected');

socket.on('message', (message) => {
const delta = JSON.parse(message);
server.clients.forEach(client => {
if (client.readyState === WebSocket.OPEN) {
client.send(JSON.stringify(delta));
}
});
});

socket.on('close', () => {
console.log('Client disconnected');
});
});
```

In this deployment over a server, upon establishing a connection, the socket is expecting incoming messages that are client edits. It then sends the received delta to all the connected clients, giving immediate visibility of changes in the document. This structure is important in ensuring that all the users are synchronized in their collaborative work.

In addition, scalability is a factor to consider, as the number of users that can be handled concurrently can fluctuate greatly. By hosting the WebSocket server on a cloud platform like Azure or AWS, developers can take advantage of load balancing to distribute traffic effectively, ensuring performance even as demand from users grows.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.57
Volume 4, Issue 1, August 2024

Azure Cosmos DB

Azure Cosmos DB serves as the backend database for collaborative editing infrastructure, featuring a very scalable document store that can handle real-time updating and delta synchronization. The database architecture supports horizontal scaling and global distribution, making it possible for end-users to view the latest document states from all over the globe with virtually no latency. This function is particularly important for applications that need real-time data synchronization among various users and locations [8].

The organization of the documents in custody with Azure Cosmos DB most commonly utilizes a JSON schema so that there can be flexibility in the retrieval and structuring of data. A document can contain a copy of the collaborative material as well as all the metadata required, such as timestamps and user IDs, in case of tracking changes. Performance can be improved through the partitioning of the documents so that Cosmos DB can successfully distribute data and loads over its internal resources. The following JSON structure illustrates how a collaborative document and its associated metadata can be organized for storage and efficient retrieval in Azure Cosmos DB.

```JSON
{
"id": "document-123",
"content": "This is the document text.",
"lastModified": "2023-10-01T12:00:00Z",
"changes": {
"operation": "insert",
"position": 10,
"text": "new text added"
},
"userId": "user-456"
}
```

With the change feed feature, Azure Cosmos DB supports real-time tracking of changes being applied, allowing smooth updates to be captured and shared. Change feed is a real-time stream of document changes, which serverless functions can use as triggers to make additional changes or inform clients.

Azure Functions

Azure Functions form the foundation for event-driven computation in the collaborative document editing model. They are intended to run code based on certain triggers, like modifications in Azure Cosmos DB's change feed, allowing the system to respond to document changes in real time. Through this serverless architecture, developers can create scalable and fault-tolerant applications without concerning themselves with the infrastructure [18].

The Azure Function configuration is minimal and only needs to involve defining a trigger and the logic to process. In a collaborative editing setup, the function would probably listen for updates in the change feed, process each update, and then notify all connected clients through WebSockets. A simple example of straightforward Azure Function implementation to process changes can appear as follows:

Within this role, all the changes discovered in the change feed are being processed. WebSocket clients can send all the data pertaining to the updated document back, so the users remain synced with the new data.

This serverless function concept is elastic in nature; as the number of edits grows, Azure Functions scale out to accommodate numerous change feed events concurrently and manage batch updates with ease without any delay. The statelessness inherent within Azure Functions also helps with scalability and high availability, which makes them well-suited to real-time cooperative applications.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.57
Volume 4, Issue 1, August 2024

```javascript
const { CosmosClient } = require("@azure/cosmos");
const WebSocket = require('ws');

const cosmosClient = new CosmosClient(process.env.COSMOS_DB_CONNECTION_STRING);
const websocketClients = ; // This array would manage connected websocket
clients

module.exports = async function (context, documents) {
for (const document of documents) {
context.log(Processing document change: ${document.id});

websocketClients.forEach(client => {
if (client.readyState === WebSocket.OPEN) {
client.send(JSON.stringify(document));
}
});
}
};
```

Additionally, the integration of Azure Functions simplifies the workflow. For example, you may want to validate the content of documents to check for compliance or formatting prior to broadcasting the updates. Such processes can be integrated within the Azure Function without complicating the client and the WebSocket server too much.

Change Feed Integration

Merging the collaborative edit system with the Azure Cosmos DB change feed is crucial in synchronizing client applications with changes to documents. Insert and update operations are caught in real-time by the change feed, and thus, it is the best way to track changes that have to be sent to users. This functionality not only simplifies the implementation of a custom change notification system but also improves performance through resource-efficient usage.

When properly constructed, the change feed is an ordered sequence of updates that can be consumed by any part of the application structure. The change feed is provided using a simple API, where the developer can listen for changes to particular containers. The following code shows how to start a continuous listening session on the change feed. By handling incoming new entries as they arrive, the app keeps the document state current in real time. Moreover, this operation naturally integrates into Azure Function lifecycles, enabling serverless event processing that accommodates modern architectural patterns.

```javascript
const container =
cosmosClient.database("YourDatabase").container("YourContainer");
const changeFeedIterator = container.items.changeFeed();

while (changeFeedIterator.hasMoreResults()) {
const { resources: documents } = await changeFeedIterator.readNext();
documents.forEach((doc) => {
console.log(Detected change in document: ${doc.id});
// Logic to process and notify WebSocket clients
});
}
```

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.57
Volume 4, Issue 1, August 2024

By leveraging the change feed to invoke actions, the architecture becomes interactive. The change feed allows Azure Functions to react whenever there are changes in the document without polling, making it efficient and scalable.
In addition, this integration serves operational transforms and CRDTs well in that it ensures that clients all have a consistent view of the edits. With change propagation through the system, the conflicts caused by concurrent edits can be resolved in such a way that the integrity of the collaborative editing process is maintained.

Security Considerations in Real-Time Collaborative Systems

Security implications are of the highest priority in real-time collaborative systems, especially those employing tools such as Azure Cosmos DB and WebSockets. These systems normally handle sensitive data among numerous users who read and edit documents in real time. A good security system should thus include secure WebSocket connections, robust identity verification, and role-based access control to safeguard user data and enable reliable collaboration.

Secure WebSocket Sessions

Real-time systems need to safeguard WebSocket connections against data interception. TLS (Transport Layer Security) support for wss:// protocols for WebSocket communication is required. The secure connection prevents eavesdropping and man-in-the-middle attacks that are used to intercept sensitive information that is being communicated between the server and the users. Providing a secure channel significantly reduces unauthorized access to data, hence, overall integrity in the collaborative environment [14].

Identity Verification

Identity verification plays an important role in protecting collaborative systems. Proper mechanisms must exist to verify the user prior to providing access to collaborative functionalities. Token-based authentication (for instance, through the utilization of JSON Web Tokens (JWT)) is used to authenticate that every request sent to the server can be authenticated using a secure token given to the users upon their successful authentication. Such a token should come with expiration times and also be signed against tampering, hence further enhancing the protection against unauthorized use [14]. Moreover, the application of multi-factor authentication (MFA) also enhances the process of verifying identity by imposing further verification procedures beyond just simple usernames and passwords, hence offering layered security.

Role-Based Access Control

Role-based access Control (RBAC) is significant in managing permissions in collaborative environments. It allows discrimination between user roles and some rights according to the user's role in the editing model. For example, roles can be "viewer," "editor," and "admin," with each having different privileges that define document changes and access levels. RBAC not only simplifies permissions management but also guarantees a secure system in which users are only permitted to do things within their scope. It prevents unapproved users from modifying documents or viewing information that is outside of their approved levels [14]. It is the systematized management of access that makes collaboration more effective in an environment of security and confidence.
Lastly, security for real-time collaborative systems should be multi-dimensional. To that effect, with secure WebSocket sessions, it is possible to impose stringent checks on identities and role-based access controls; an organization can indeed bolster the protection of sensitive data during collaboration processes. Through such measures, users can collaborate without fear of their document collaboration being vulnerable to real security threats.

Document Versioning, Recovery, and Historical State Management

Document versioning, recovery, and history state management are also important for collaborative editing systems to operate optimally. These are not only a requirement for the user interface but also for data correctness and integrity in the long term. Azure Cosmos DB offers strong support that enables versioning, snapshotting, and recovery functionality in collaborative settings.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.57
Volume 4, Issue 1, August 2024

Document Versioning

To maintain good document versioning, every modification to a document must be stored as a separate version within a well-structured versioning system. This might be done using a specific container in Cosmos DB that stores all document versions [6]. Any version can store metadata, i.e., document content, implementing user, a timestamp, and a version number. By maintaining such fine-grained metadata, users can recover past versions of the document with ease for inspection or recovery. Functionality can be offered by the system to query document versions so that users can track changes in detail and roll back to previous versions when necessary.

Recovery and Undo/Redo Functionality

Undo and redo functionality are basic features in document editing software. In order to perform such operations in the best possible manner, a stack-based mechanism can be used to store changes during editing. If a history stack of document states is kept, the application can alter the current content through state push or pop operations into and out of the stack. Whenever a user needs to undo an operation, the last state can be retrieved and set, which brings the document back to its original state. In the same way, the redo operation would push the current most recent undo state onto the stack, enabling users to move directly from one set of actions to another.

Working with Cosmos DB's TTL and Snapshots

Azure Cosmos Time-To-Live can be used in an innovative way to enable versioning as well as space savings by removing the old versions of documents periodically in a specified range. This saves storage expenses and ensures top-level performance, particularly where documents are constantly being edited. Simultaneously, snapshots can be created at important checkpoints or in response to particular user activities like saving or large modifications. These snapshots would act as decent recovery points so that users can roll back the work to a good known point when the need arises. By making use of Cosmos DB's feature for handling such historical records, the application is able to offer an optimal user experience without sacrificing performance.

In conclusion, versioning documents, recovery settings, and effective handling of past states are highly important in any collaborative editing scenario. With the capabilities of Azure Cosmos DB, including custom versioning collections, TTL, and snapshots, organizations can provide users with the ability to manage changes to documents in a secure manner, with the assurance that their collaborative work is safe and effective.

II. CONCLUSION

In essence, enabling real-time interactive editing requires the optimal integration of communication protocols, data models for storage, and consistency mechanisms. By combining WebSocket-based streaming and Azure Cosmos DB's distributed architecture and supporting it with reconciliation mechanisms like operational transforms, systems can obtain low-latency synchronizations while maintaining document integrity. Coupled with robust versioning, recovery schemes, and secure client coordination, such architectures form the basis for scalable, reliable, and user-friendly collaborative experiences.

REFERENCES

[1] C. Wu, L. Li, P. Changwei, Y. Wu, N. Xiong, and C. Lee, "Design and analysis of an effective graphics collaborative editing system," *Eurasip Journal on Image and Video Processing*, vol. 2019, no. 1, 2019. [Online]. Available: https://doi.org/10.1186/s13640-019-0427-6

[2] B. Haeupler, A. Shahrasbi, and E. Vitercik, "Synchronization strings: channel simulations and interactive coding for insertions and deletions," *arXiv preprint*, arXiv:1707.04233, 2017. [Online]. Available: https://doi.org/10.48550/arxiv.1707.04233

[3] B. Amhmed, "Distributed system design based on image processing technology and resource state synchronization method," *Distributed Processing System*, vol. 2, no. 4, pp. 28–35, 2021. [Online]. Available: https://doi.org/10.38007/DPS.2021.020404

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.57
Volume 4, Issue 1, August 2024

[4] A. Resmi and F. Taïani, "Filament: a cohort construction service for decentralized collaborative editing platforms," in *Lecture Notes in Computer Science*, vol. 10223, pp. 146–160, 2017. [Online]. Available: https://doi.org/10.1007/978-3-319-59665-5_11

[5] M. Babi and W. Zhao, "Towards trustworthy collaborative editing," *Computers*, vol. 6, no. 2, p. 13, 2017. [Online]. Available: https://doi.org/10.3390/computers6020013

[6] A. Mishra, *Work with Azure Cosmos DB*, pp. 161–179, 2022. [Online]. Available: https://doi.org/10.1007/978-1-4842-8251-9_8

[7] R. Valavandan, B. Gothandapani, A. Gnanavel, N. Ramamurthy, M. Balakrishnan, S. Gnanavel, and S. Ramamurthy, "Leveraging Azure platform data services for efficient data analytics in the oil & gas industry: A case study of Nature Labs project at leading oil & gas company," *International Journal of Research Publication and Reviews*, vol. 4, no. 6, pp. 1730–1741, 2023. [Online]. Available: https://doi.org/10.55248/gengpi.4.623.45595

[8] W. Lu, X. Liu, and T. Chen, "Adaptive algorithms for synchronization, consensus of multi-agents and anti-synchronization of direct complex networks," *arXiv preprint*, arXiv:2007.14635, 2020. [Online]. Available: https://doi.org/10.48550/arxiv.2007.14635

[9] M. Sharma, *Azure Cosmos DB Overview*, pp. 11–59, 2018. [Online]. Available: https://doi.org/10.1007/978-1-4842-3682-6_2

[10] J. Paz, *Querying an Azure Cosmos DB Database*, pp. 159–201, 2018. [Online]. Available: https://doi.org/10.1007/978-1-4842-3351-1_5

[11] A. Satapathi and A. Mishra, *Build an IoT Solution with Azure IoT Hub, Azure Functions, and Azure Cosmos DB*, pp. 193–218, 2022. [Online]. Available: https://doi.org/10.1007/978-1-4842-9004-0_8

[12] J. Paz, *Learning Azure Cosmos DB Concepts*, pp. 25–59, 2018. [Online]. Available: https://doi.org/10.1007/978-1-4842-3351-1_2

[13] D. Chen, "Application of web socket in dynamic face recognition system," *Destech Transactions on Computer Science and Engineering (ICEITI)*, 2018. [Online]. Available: https://doi.org/10.12783/dtcse/iceiti2017/18820

[14] O. Awuor, "Review of the security challenges in web-based systems," *World Journal of Advanced Engineering Technology and Sciences*, vol. 8, no. 2, pp. 204–216, 2023. [Online]. Available: https://doi.org/10.30574/wjaets.2023.8.2.0099

[15] B. Joshi, *Azure SQL Database, Azure Cosmos DB, and MongoDB*, pp. 381–442, 2019. [Online]. Available: https://doi.org/10.1007/978-1-4842-5509-4_9

[16] Z. Huang, L. Chen, L. Zhang, S. Fan, and D. Fan, "Research on software synchronization method of real-time Ethernet distributed motion control system," *Assembly Automation*, vol. 39, no. 5, pp. 904–916, 2019. [Online]. Available: https://doi.org/10.1108/aa-12-2018-0265

[17] A. Taivalsaari, T. Mikkonen, C. Pautasso, and K. Systä, "Client-side cornucopia: Comparing the built-in application architecture models in the web browser," in *Web Engineering*, pp. 1–24, 2019. [Online]. Available: https://doi.org/10.1007/978-3-030-35330-8_1

[18] A. Satapathi and A. Mishra, *Serverless API Using Azure Functions and Azure Cosmos DB*, pp. 203–232, 2021. [Online]. Available: https://doi.org/10.1007/978-1-4842-7122-3_9

[19] B. Dornauer and M. Felderer, "Energy-saving strategies for mobile web apps and their measurement: Results from a decade of research," in *IEEE MobileSoft*, 2023. [Online]. Available: https://doi.org/10.1109/mobilsoft59058.2023.00017

[20] S. Konstantinov, F. Assad, W. Azam, D. Vera, B. Ahmad, and R. Harrison, "Developing web-based digital twin of assembly lines for industrial cyber-physical systems," in *IEEE ICPS*, pp. 219–224, 2021. [Online]. Available: https://doi.org/10.1109/icps49255.2021.9468227

[21] K. Jha, "Designing Systems: Implementing Collaborative Editing," Medium, 2023. [Online]. Available: https://medium.com/@krishnajha853/designing-systems-implementing-collaborative-editing-c7b556c5bb3

[22] Atlassian, "Collaborative editing for Confluence Data Center," Atlassian Developer Documentation, 2017. [Online]. Available: https://developer.atlassian.com/server/confluence/collaborative-editing-for-confluence-server/

[23] P. Gaurav, "Azure Cosmos DB – A Guide to Microsoft's Database Service," Cuelogic Blog, 2018. [Online]. Available: https://www.cuelogic.com/blog/azure-cosmos-db-a-guide-to-microsofts-database-service