IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

RAIDX: Event-Driven Architecture for Real-Time Kabaddi Live Score Streaming and Tournament Management

Jeet S. Mhatre, Ketan Zagade, Pradumn Gurav, Sujal Chandekar, Moushmee Kuri

Student, Student, Student, Student, Professor

School Of Computing,

MIT-Art, Design and Technology University, Pune, Maharashtra, India.

Abstract: Kabaddi matches, championships and tournaments, essentially at the local and grass-root level, there are limited as well as manual scoring and no live score management systems, due to which there can occur less accessibility for viewers to actually view the scores, less efficient co-ordination between the organizers and may occur delays for match updates. Thus, there is a lack of a digital infrastructure which also caters the ability to maintain a centralized player information, events ledger and scoring mechanism for kabaddi at the grass-root level. As the popularity of Kabaddi is growing, there is a requirement of a scalable digital platform which will be able to manage the live scoring and its broadcasting for the events of kabaddi including matches, tournaments and championships for frequent match updates.

This paper represents our platform RAIDX, which is a real-time Kabaddi live scoring and score broadcasting platform for all kabaddi events including the tournament and championships which is designed using the event-driven architecture. The proposed system designed utilizes the scorers having the organizer role to record the various events occurring in the match per raid, which is then updated to the backend server which is implemented in Golang using a persistent WebSocket connection. Those match events are processed by the match event engine at the backend that updates the match score data instantly and streams it to all the connected viewer connections. This robust infrastructure is implemented using the Fiber framework using the Redis as in-memory cache to accommodate the high frequency match updates. The persistent tournament, or championship data, including the player accounts, team information, fixtures and invites are stored using MongoDB.

For secure and scalable user management, the current design involves the hybrid mechanism including the JSON Web Token (JWT) with server-side session tracking, to allow the device-based session management (by controlling and revoking access). Also, there is an implementation of the Role-Based Access Control (RBAC) model that manages the different user roles including the organizers, team owners and players with viewers as the default role for all the mentioned roles. The system is also incorporated with the invitation framework which supports both direct as well as the token-based invite links for both team and match event invites.

The proposed system architecture determines how a scalable sports event management platform can be implemented combining the application of both the modern web technologies as well as real-time communication protocols. The RAIDX system improves tournament coordination, enhances the viewer engagement by quick score updates, and providing a structured digital ecosystem for managing Kabaddi competitions.

Index Terms - Kabaddi Digitization, Real-Time Sports Streaming, Event-Driven Architecture, WebSocket Communication, Redis Runtime State Management, NoSQL Databases, Low-Latency Broadcasting, Tournament Management Systems, Role-Based Access Control (RBAC), Scalable Web Architecture, Match Data Archiving, Player Performance Analytics.

I. Introduction

Kabaddi is one of the oldest indigenous sport famous especially among the Indian subcontinent which been gaining popularity in recent years due to the emerging leagues at the national and internation levels. Even with such growth, the technological or digital infrastructure which supports the Kabaddi competitions and events especially at the grass-root and local levels is very limited. Majority of the competitions still rely on the manual scoring, pen-paper based fixtures and via the verbal communication between the organizers, players and team owners. these traditional practices lead to delays in updating the scores, less transparent and certain times compromised statistics which results in the limited accessibility for the score-viewers. Additionally, the absence of a central digital system restricts the efficient Kabaddi tournament management.

Due to the advancements of web technologies and the distributed system architectures, it has enabled the development of real-time digital platforms having the ability process the high frequency events and broadcasting efficiently to large number of users. Sports including cricket, football and basketball have significantly improved viewer engagement and the tournament managing apps. But the similar digital applications for Kabaddi certainly remained under-developed especially for the local level or medium sized tournaments where the usually the access to the expensive broadcasting infrastructure is not feasible.

Also, a challenge in designing a digital sports infrastructure is the requirement of low-latency and real-time communication between the scorer (organizer) and the viewers. During a Kabaddi match, to score the raids, tackles, bonus points and player eliminations occurs very frequently and within a very short period of time intervals. Thus, a scoring system should be capable to capture these events instantly and should process them efficiently while distributing those updates to the viewers with least possible delay. the traditional request-response architectures based on periodic polling are less suitable as they may introduce unnecessary network overhead and latency.

Another important challenge is managing the multiple user roles within the system. In a normal Kabaddi competition, there exists multiple stakeholders including the organizers, team owners, players, scorers and viewers. Now each category of these users will require the different level of access to the resources of our system.

In order to address these challenges, the research proposes **RAIDX**, which is a real-time Kabaddi live score streaming and tournament management platform which is designed using the modern web design patterns. This platform introduces an event-driven architecture that enables the kabaddi competitions and corresponding events to be processed and broadcast in real-time using the persistent WebSocket connections. This architecture allows the raid events which are generated by the scoring interface to be sent instantly to the backend server which is then processed by the match event engine and thus get broadcasted to all the connected viewer clients.

The backend of RAIDX is implemented using the **Golang with the Fiber web framework**, as it provides the very high-performance HTTP routing and can easily handle the concurrent requests. Real-time communication between the between the client and the server is achieved through the WebSocket connections, which allows the persistent bi-directional data exchange. thus, to handle the high-frequency match score updates, the system will use the **Redis as an in-memory state store**, which enables the quick read and write operations during the live matches. Persistent data of tournaments or competitions are stored using **MongoDB**, as it is a document-oriented NoSQL scalable database which provides flexible schema design and high query performance.

In addition to this real-time event processing, the platform also has **a secure authentication and session management framework** which is a hybrid of **JSON Web Token (JWT)** authentication along with the **server-side session tracking**. This approach allows the system to maintain the scalable stateless authentication along with centralized user sessions and device-based login management. Along with this, the resource-oriented **Role Based Access Control (RBAC)** mechanism that dynamically manages the access privileges based on the user role.

The RAIDX backend also employs a flexible **invite and participation framework** that enables the users to join the teams or the kabaddi competition events through both direct invites as-well-as token-based invite links. This design makes the participant onboarding easier at the same time have a check on RBAC.

The primary objective of this research is to design and implement a scalable system architecture which is capable of supporting the real-time low latency highly scalable sport event streaming system while handling the tournament management capabilities. The proposed solution targets to demonstrate how modern web technologies and event-driven architectures can be applied to transform traditional Kabaddi competitions like tournaments, championships and matches workflows into a digital ecosystem which will overall improve the transparency, accessibility and the viewer endorsement and engagement.

The main contributions of this work include:

1. The architecture design of an event-driven system enabling the real-time Kabaddi match and tournament scoring and broadcasting.
2. The implementation of a hybrid approach containing auth and session management using JWT with server-side session tracking.
3. Implementing the dynamic RBAC authorization for management of multiple user roles.
4. To integrate the Redis-based runtime state management for handling the high-frequency match state changes efficiently.
5. Design of a match, tournament and championship management system supporting the supports team management, invitation workflows, and real-time match streaming.

Through these contributions, the RAIDX platform provides a scalable and extensible technological framework for modernizing Kabaddi tournament management and thus enables a real-time engagement for viewers and all participants.

II. RELATED WORK / LITERATURE SURVEY

The development of live scoring digital platforms for sports management and event broadcasting has gained great attention with the advancement of web technologies and the distributed compute systems. The modern sports platforms usually aim to provide the real-time match updates, with centralized data management, and enhanced viewer engagements though the digital interfaces. However, multiple existing systems had primarily focused on the globally famous sports including cricket, football, and basketball, but the traditional sports like kabaddi or Kho-Kho still don't have any significant digital infrastructure, essentially at the grass-root or local level.

It is observed that usually all the digital sports management systems rely on the centralized databases and performs periodic data refresh in order to update the match statistics. These systems provide the functionalities to store the matches records and displaying the results, but doesn't support the real-time event streaming or low-latency communication between the match official and the viewers. Thus, the viewer experience often includes the delayed score updates, and the system usually struggles to handle the high-frequency match event changes efficiently.

Along with the advancement of the modern web technologies, various research studies have identified the use of **real-time communication protocols** for live sports broadcasting. The WebSocket based communication have made it available to actually enable the persistent bidirectional duplex connections between the client and the server machines, enabling the event updates to get transmitted quickly without the actual need to repeatedly making the HTTP requests. This event-driven approach is becoming more popular in applications which usually requires the continuous data streams like the financial systems, online gaming platforms and the live sports analytics systems.

The next significant part of research includes the application of the **event-driven architectures** which processes the high-frequency data streams. Considering such systems, the events produced by the users or the devices are being sent to the processing pipelines or engines which then updates the application states and then distribute the updates on the client-side devices. This architecture significantly reduces the latency and the network overhead when we compare this with our traditional pooling-based systems. Such architectures are preferably suitable for the sports scoring systems where the quick updates of the state occur during the sessions.

In addition to the real-time communication technologies, the modern sport platforms now rely on the **distributed data management techniques** to enhance the metrics and performance at scale. The in-memory cache like Redis have been widely embraced to manage the high-frequency run-time data as it enables the very high speed read and write operations. Due to separations of transient runtime states from the persistent storage systems, such interfaces now can efficiently process the real-time updates while storing and maintaining the long-term records in the traditional databases.

The authentication and access-control mechanisms play a vital role in any multi-user sports management interfaces, these systems which supports the multiple roles ensures that only the authorized users can perform a particular action on the platform. Traditional access control systems usually rely on static role assignments, which does not provide the required flexibility in the dynamic environments where the actual user permission may depend upon the relationship with specific resources.

Authentication and access control mechanisms also play a crucial role in multi-user sports management platforms. Systems that support multiple user roles must ensure that only authorized users can perform specific actions within the platform. Traditional access control systems often rely on static role assignments, which may not provide sufficient flexibility in dynamic environments where user permissions depend on relationships with specific resources. Consequently, recent research has been conducted on the use of the Resource-Oriented **Role-Based Access Control (RBAC)** model, whereby the system checks the permission based on the role of the user and the ownership of the system entity such as the team, the tournament, or the match.

Multiple digital sports platforms now incorporate the **team management and participation frameworks** which enables the players and organizers to co-ordinate with the tournament events through the online interfaces. These systems usually provide the functionality for player registrations, team creations, tournament fixtures scheduling, and match overall tracking. Certainly, many pre-existing solution architectures usually lack these integrated real-time scoring facilities and the flexible participation structures to allow the players to get in the teams or tournaments through the invitation workflows.

Thus, there remains a significant gap in the availability of digital platforms which specifically crafted for the Kabaddi tournaments and competitions. In most local tournaments, manual scoring and fragmented data management practices takes place. Thus, the absence of a scalable and real-time scoring system will limit the ability of the organizers to actually broadcast the match updates to maintain the historical structured data for the players and the respective teams.

The RAIDX platform seeks to solve these limitations by merging modern real-time communication technologies with a scalable backend setup. This creates a complete Kabaddi tournament management system. It incorporates WebSocket-based event streaming, Redis-based runtime state management, hybrid authentication methods, and dynamic role-based access control. The system offers a single digital platform that can handle real-time scoring for Kabaddi matches and manage tournaments efficiently.

III. SYSTEM ARCHITECTURE

The RAIDX platform is actually designed utilizing the modular and a scalable architecture which enables the real-time scoring, tournament management and live match or tournament score event broadcasting. The architecture follows the approach of a layered design, where the user interactions, application logic, real-time communications, caching (state management) and persistent storage are separates into different layers or the system components. This enables the separation of concerns thereby improving the system maintainability, enhances scalability, and allows the platform to handle high-frequency match events efficiently.

The overall architecture of RAIDX consists of the following key layers including **frontend interface layer, API and service layer, authentication and authorization layer, real-time event processing layer, runtime state management layer, and the persistent data storage layer**. Each layer of these plays a significant role in handling the real-time operation of the system and ensuring the smooth interaction between different users including the players, owners, organizers and viewers.

A. Frontend Interface Layer

The frontend layer provides the graphical interfaces through which the different stakeholders actually interact with the platform. These interfaces include the dashboards for all the three roles, score viewing and the scoring interfaces along with the profile pages per user. The scorer interface allows the authorized scorer (organizer)to record the events taking place during a raid including the raid points, tackles, bonus

etc. These all events are then transmitted to the backend in real-time. The viewer interface then accepts the incoming changes from the backend server and then updates those respectively.

The whole frontend views and statics are implemented using the standard web technologies including **HTML, CSS with Bootstrap, and JavaScript**, which enables the interfaces to work across multiple devices and browsers without actually requiring the specialized software. The normal communication between the frontend and backend occurs through **HTTP-based API requests for standard operations**, while for the live scoring and score streaming **WebSocket connection** are used.

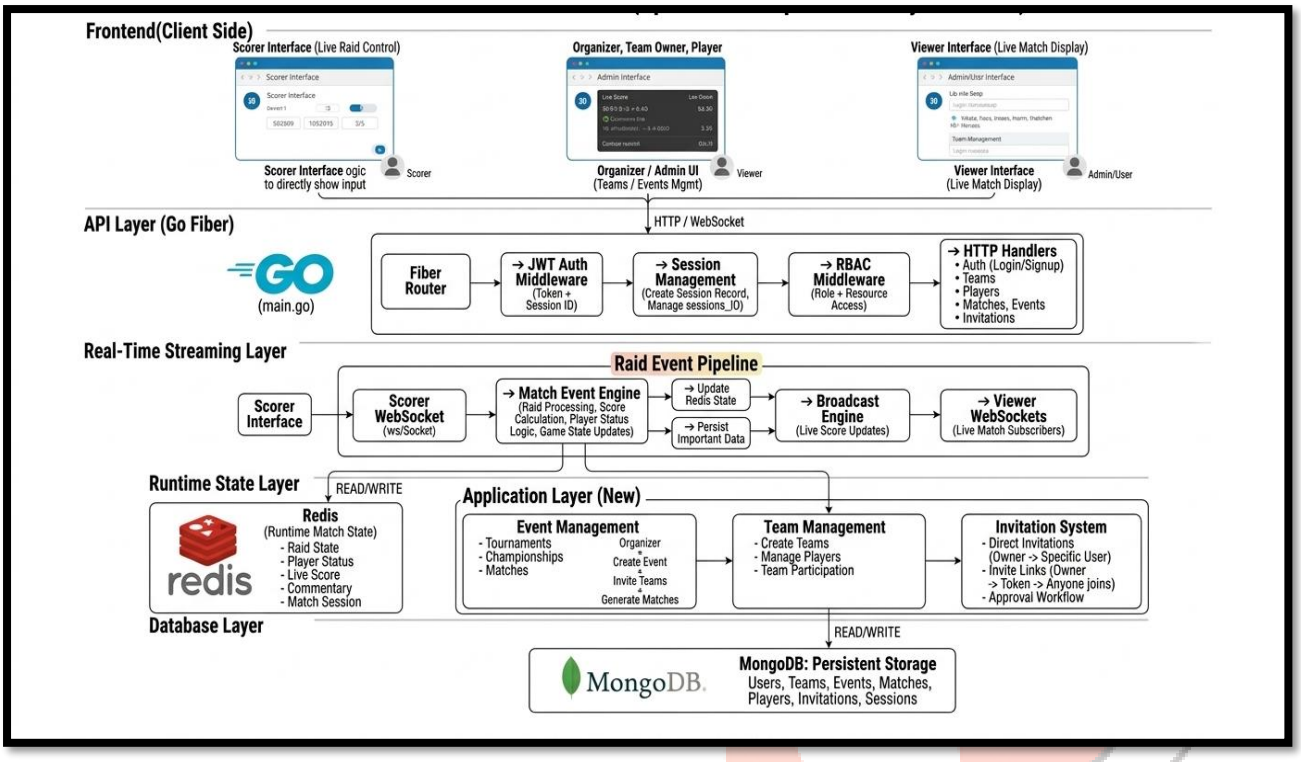

Figure 1 : System Architecture

B. API and Service Layer

The backend of the RAIDX platform is implemented using **Golang with the Fiber web framework**, as it provides the high-performance routing and efficiently handles the concurrent requests. This layer actually exposes the RESTful APIs and its routes which handles the operations including user authentication, team management, tournament configuration, invite processing and match management.

The API layer is actually the point of communication between the frontend interfaces and the backend server. The incoming requests are then processed through the structured request pipeline which includes the routing, middleware validation, and handler execution. Every API endpoint corresponds to a specific service operation inside the system.

Utilizing Golang and Fiber, the backend system gets the benefits like lightweight request handling and efficient asynchronous processing, allowing the platform to handle multiple concurrent users and match events without significant performance degradation.

C. Authentication and Session Management Layer

The secure user authentication is incorporated by using a **hybrid authentication architecture combining JSON Web Tokens (JWT) with server-side session tracking**. Thus, after successful login, a signed JWT is generated which has the user identity and possess a unique session identifier.

As the JWT provides the state-less authentication, it is usually used in distributed systems, as it enhances the overall security by storing the session records in the database. Due to this the features like the device-based tracking, token revocation, and multi-device login management are enabled. By combining token-based authentication with session tracking, the platform achieves both scalability and centralized control over active user sessions.

D. Authorization and Access Control Layer

RAIDX has implementation of **Role-Based Access Control (RBAC) model** which regulates the access to system resources and its ownership relationships. The platform supports multiple roles organizers, team owners, players, scorers, and viewers. Despite of only relying only on the static role assignments, the same system now determines the permissions dynamically based on the relationship between the users and the resources. This resource-oriented authorization model provides fine-grained access control while maintaining system flexibility.

E. Real-Time Event Processing Layer

Also, another essential concept of RAIDX architecture is the **real-time event processing system** as it is responsible for managing the match score. During an ongoing match, the scorer interface sends the raid events to the backend through the WebSocket connection. Such events are then processed by a **Match Event Engine**, determining the raid outcomes and then sends those updates to all the connected viewer connections.

This event processing layer ensures that every action related to scoring is validated, processed and then applied to the match in the correct sequence. Hence, the updated match info is passed to all the viewers in connection.

This event-driven architecture allows the system to update the match events in real-time to the viewers without actually relying on the periodic polling mechanisms.

F. Runtime State Management Layer

To handle the high-frequency match updates, RAIDX uses the **Redis as an in-memory runtime state store.** Redis helps in maintaining the current state of the on-going matches, including live scores, raid outcomes, player status updates, and match session information. So, by storing the real-time data in Redis, the system will achieve the extremely high speed read and write operations as there are essential for the real-time scoring events. This approach reduces the load on the primary database too during the live matches.

G. Persistent Data Storage Layer

For the persistent data-storage the RAIDX platform uses the by **MongoDB**, which is a document-oriented NoSQL database. As the MongoDB stores the persistent data such as user accounts, teams, tournaments, match history, invitations, and player statistics it fulfils all the requirements.

MongoDB allows to efficiently undergoes and manage the complex querying and the analytics doe to its flexible schema design which actually helps to maintain the scalability when the system grows.

Through this layered architecture, RAIDX provides a scalable and efficient framework which is actually capable of supporting the real-time Kabaddi match scoring and tournament management that too across the multiple users and their multiple devices. With the combined use of Redis for the runtime data processing and the MongoDB for the persistent storage, RAIDX platform achieves a balance architecture that supports both real-time performance and long-term data management.

IV. PERFORMANCE ANALYSIS

The actual performance of a real-time sports system is connected largely with respect to its connected users who faces the minimal delay. As Kabaddi matches includes the high frequency sequence of actions including the raids, tackles, bonus points and player eliminations, so the system must be capable of absorbing such high-speed real time incoming updates without actually affecting the system stability or the user experience. The RAIDX platform is designed keeping into consideration by combining an event-driven architecture, WebSocket-based communication, and in-memory runtime state management using Redis.

To determine the efficiency of the system, various aspects of the platform performance are considered, including **event processing latency, real-time data propagation, concurrent user handling, and database interaction efficiency**. Such factors overall determine the platform's capability to actually support the live match scoring and large number of viewers at the same time.

A. Real-Time Event Latency

An important performance measure for a real-time live scoring system is that the time required to process and then transmit the match events. In the RAIDX system, a scoring event starts when the organizer (scorer) records any raid actions through the scorer interface. This event is then broadcasted through a WebSocket connection to the backend server where the match event engine actually processes the requests and then updates the match state.

As the system use the Redis for real-time in memory cache, the rendering of the match events will occur entirely in the memory, which results in the obtaining the extremely low latency. Once the match state is updated on Redis, the broadcast engine will immediately transmit the updated score to all the viewers who are in connection. As this architecture will reduce the delay between the scoring action and its visibility on the actual viewers devices significantly.

B. WebSocket Communication Efficiency

The platform uses the persistent connection using the WebSockets which enables the continuous bidirectional communication between the clients and the backend server. Unlike the typical methods like the HTTP polling, which requires the repeated client requests to get the updates, the WebSocket connection allows the server so that the it pushes the updates directly to all the connected clients simultaneously.

Such communication model actually lowers the network overhead eventually improving the responsiveness. Whenever a match event is processed by the server, the broadcast engine will distribute the revised and updated match state to all the viewer clients who are in connection. Thus, all the viewers receive the synchronized score updates without needing to refresh or poll the interface.

C. Runtime State Performance Using Redis

The application of Redis as an in-memory cache implies an essential role in upscaling the system's overall performance. The Redis cache allows to store the active match data including the scores, raid events, player statuses in the memory itself instead of writing every update to the primary database.

This results in the reduction of time which is actually required to retrieve or update the match state information during actual scoring events. As Redis allows the system to absorb high speed read and write operations, the system now can easily handle the frequently scoring events without creating the bottlenecks in the actual backend infrastructure.

Also, by storing and maintaining the active matches in Redis, the platform now avoids the frequent database write operations during every update. This design actually improves the scalability and ensures multiple matches can be conducted simultaneously with high concurrency.

D. Concurrent User Handling

Another significant aspect of the system performance is the capability of the platform to support the multiple concurrent users. When a tournament occurs, a large number of viewers may connect the platform to view the match score live. The RAIDX architecture can now handle the same scenario through the combination of Golang's concurrency model and WebSocket broadcasting mechanisms.

The Golang's lightweight go-routines allows the backend server to actually manage the multiple simultaneous connections effectively. Thus, every viewer connection subscribes to any match updates through a closed WebSocket connection channel, as this allows the broadcast engine to transmits the match updates to all the connected clients without needing the individual request handling for each user.

This design ensures that the platform can support a large number concurrent viewer connections while managing the consistent real-time performance.

E. Database Interaction Efficiency

As the Redis easily handles the match score data during the live matches, MongoDB being responsible for storing the match persistent records like tournament information, team structures, user accounts, and finalized match results. In order to maintain the database efficiency, the system avoids writing

Rather, one a single finalized match data and its associated analytics are stored in the MongoDB once the match gets concluded. This approach will reduce the database load and will ensure that the persistent storage operations does not interferes with the real-time event processing.

F. System Scalability

The modular architecture of RAIDX contributes significantly to the scalability of the system. As different system components including the event processing, data storage, authentication, and broadcasting work independently, then system can be scaled to support many additional matches or users without actually requiring the major architectural changes.

The combinations of Golang's efficient concurrency handling, Redis runtime caching, WebSocket-based communication, and MongoDB persistent storage, the RAIDX platform can achieve a scalable, at the same time enables a low-latency infrastructure which is suitable for real-time sports event streaming.

Overall, the performance analysis demonstrates that the proposed architecture actually supports the real-time Kabaddi match scoring effectively while maintaining system responsiveness and the stability even under conditions high and concurrent user rate.

V. RESULTS AND DISCUSSION

To determine the efficiency of the proposed RAIDX architecture, various experimental tests were conducted focused on the **event processing latency, WebSocket broadcast performance, and concurrent user handling**. The aim of these tests was to evaluate the efficiency the platform with regard to the processing of the match events and its updates being broadcasted to all the connected viewers while a live kabaddi match was live.

To actually maintain the realistic conditions, the number of concurrent users was kept small intentionally, which will represent a typical grassroots Kabaddi tournament environment.

A. Event Processing Latency

The first test actually measured the time required for the system to process a raid event and then transmit the updated score to the connected viewers. The calculated latency was the time interval between the scorer entering the event in the interface and then the updated score appearing on the viewer interface.

Ten raid events were recorded during a simulated match and the corresponding latency values were measured.

Table 1: Event Processing Latency

Raid Event	Processing Time (ms)
Raid 1	48
Raid 2	52
Raid 3	46
Raid 4	50
Raid 5	49
Raid 6	53
Raid 7	47
Raid 8	51
Raid 9	45
Raid 10	50

These results actually shows that the average event processing latency is approximately **49 milliseconds**, which is considerably low for real-time score broadcasting. This demonstrates the efficiency of the event-driven processing pipeline and the Redis-based runtime state management system.

B. WebSocket Broadcast Performance

The second test was to determine the efficiency of WebSocket broadcasting when multiple viewers get connected to the platform. During this test, a simulated match was broadcasted to a small group of viewers who received live score updates.

Five viewers were connected simultaneously, and the time required for the score update to reach each viewer was measured.

Table 2: WebSocket Broadcast Delay

Viewer ID	Broadcast Delay (ms)
Viewer 1	55
Viewer 2	58
Viewer 3	56
Viewer 4	60
Viewer 5	57

The results indicates that the score updates were delivered to all viewers within approximately **55–60 milliseconds**, which demonstrates the effectiveness of the WebSocket-based broadcasting mechanism.

C. Concurrent Viewer Handling

Another important aspect of the system evaluation involved testing how the backend server handled multiple viewer connections during a match. In this test scenario, five viewers simultaneously connected to the platform while a referee recorded match events.

Table 3: Concurrent Viewer Handling

Number of Viewers	Average Response Time (ms)
1	47
2	47
3	48
4	47
5	48

The results indicate that the response time actually remained the same as the number of viewers increased. This shows that the Golang-based backend and the WebSocket communication model are capable of handling multiple connections efficiently.

D. Redis Runtime State Performance

To evaluate the effectiveness of Redis as the runtime state management layer, the time required to update and retrieve match state information was measured.

Table 4: Redis Runtime Operations

Operation	Average Time (ms)
Match State Update	2.3
Match State Retrieval	1.9

The results demonstrate that Redis operations occur within a few milliseconds, which is significantly faster than typical database queries. This confirms that Redis is well suited for handling high-frequency runtime updates during live matches.

E. Discussion

The experiments results had demonstrated that the RAIDX platform has successfully supported the real-time Kabaddi match scoring and broadcasting. The combination of all the components, including the **WebSocket communication, Redis runtime caching, and Golang concurrency mechanisms** helps in enabling the system to process the match events and the to distribute updates with very minimal latency.

Even through multiple connected viewers, the system had actually maintained a stable performance as well as delivered those score updates consistently across all the client viewer interfaces. The results actually confirms that the proposed architecture is highly suitable for supporting the real-time sports event streaming in small to medium-scale Kabaddi tournaments very efficiently.

Furthermore, the application of Redis for the runtime state management cache store has significantly reduced the load on the primary database which ensures that match updates has being processed efficiently. This hybrid storage architecture actually improves both the metrics, performance as well as the scalability.

Overall, the experimental evaluation has indicated that the RAIDX platform provides a reliable and a highly-responsive infrastructure which can digitize the Kabaddi tournaments as well as delivers the real-time match updates to the viewers.

VI. CONCLUSION

This research presented **RAIDX**, a real-time Kabaddi live score streaming and tournament management platform which is designed to address the limitations of traditional tournament management practices. Many grassroots and the local Kabaddi tournaments still rely on manual pen-paper based scoring and fragmented coordination between referees, organizers, and spectators. These approaches will usually result in the delaying of match updates, the lack of structured data management, and a limited accessibility for audiences who are not able to attend those matches physically. The proposed system demonstrates how the modern web technologies and event-driven architectures can be applied so that Kabaddi tournaments can be digitized and provide a reliable platform for real-time match monitoring.

The RAIDX platform introduces a scalable architecture that will integrate multiple system components including **WebSocket-based real-time communication, Redis runtime state management, MongoDB persistent storage, and Golang-based backend services**. Through the use of persistent WebSocket connections, the platform actually enables the organizers (scorers) to transmit the raid events instantly to the backend server, which allows the system to process the scoring actions and then to broadcast the updated match information to all the connected viewer clients with minimal latency possible. This real-time communication model will significantly improve viewer engagement by ensuring that the score updates are well-synchronized across all the viewing devices.

Another important aspect of the system is the implementation of a hybrid authentication and the session management framework, which combines JSON Web Tokens with server-side session tracking. This design actually allows the platform to maintain scalable stateless authentication while enabling a centralized control over the active user sessions. In addition, the system also incorporates a resource-oriented Role-Based Access Control (RBAC) mechanism that dynamically manages the permissions for different user roles including the organizers, team owners, players, and viewers.

The experimental evaluation of the system had demonstrated that the platform is actually capable of processing raid events and the broadcasting updates with significantly low latency. The implementation of the Redis cache for runtime state management store, allows the high-frequency match updates to be handled efficiently without adding load on the persistent database. Even with multiple connected viewers, the system had maintained consistent performance and delivered real-time score updates reliably.

Overall, the RAIDX platform establishes a digital infrastructure that manages Kabaddi tournaments while enabling the real-time broadcasting of match events. By combining scalable backend architecture, event-driven communication, and modular system design, the platform actually provides a foundation for improving tournament organization and the viewer engagement in Kabaddi competitions.

The future work may focus on scaling the capabilities of the platform by adding features including advanced match analytics, automated player performance evaluation, video-based match streaming, and the machine learning models for the predictive sports analysis and outcomes. These enhancements could further strengthen the role of digital technologies in transforming Kabaddi tournaments into fully data-driven sporting events.

VII. ACKNOWLEDGMENT

We, all authors, would like to sincerely like to thank **MIT ADT University, Pune**, for providing the support, academic environment and the resources that helped us in developing and testing the RAIDX platform. We are also grateful to **Prof. Moushmee Kuri Ma'am** for her guidance support and constant encouragement throughout this project journey. The constructive feedback and support provided during the course of this project played an important role in shaping the research and implementation presented in this paper.

REFERENCES

[1] I. Fette and A. Melnikov, "The WebSocket Protocol," *Internet Engineering Task Force (IETF) RFC 6455*, Dec. 2011.

[2] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, Feb. 1996.

[3] S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80–83, Nov. 2010.

[4] S. Sanfilippo and P. Noordhuis, "Redis: An Open Source In-Memory Data Structure Store," Redis Documentation, 2023.

[5] MongoDB Inc., "MongoDB Architecture Guide," MongoDB Documentation, 2024.

[6] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA, USA: Addison-Wesley, 2004.

[7] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley, 2003.

[8] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[9] Fielding, R. T., "Architectural Styles and the Design of Network-Based Software Architectures," Doctoral Dissertation, University of California, Irvine, 2000.

[10] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A Distributed Messaging System for Log Processing," *Proceedings of the NetDB Conference*, 2011.