Table of Contents

# Interval Merging Algorithm Explained

## 🎯 The Problem

When users rewind and re-watch parts of a video, we need to calculate **unique seconds watched** without double-counting overlapping time periods.

### Example Scenario

**User watches a 5-minute (300s) video:** 1. Watches from 0s to 30s (watches 30 seconds) 2. Rewinds to 20s 3. Watches from 20s to 50s (watches 30 seconds more)

**Question**: How many unique seconds did they watch?

**Wrong Answer**: 30 + 30 = 60 seconds ❌ - This counts seconds 20-30 twice!

**Correct Answer**: 50 unique seconds ✅ - They watched 0-50s continuously - The overlap (20-30s) should only be counted once

## 📊 Visual Example

### Raw Watch Segments

Timeline: 0 10 20 30 40 50 60  
 |-----|-----|-----|-----|-----|-----|  
Segment 1: [===========] (0-30s)  
Segment 2: [===========] (20-50s)  
  
Overlap: [===] (20-30s counted twice!)

### After Merging

Timeline: 0 10 20 30 40 50 60  
 |-----|-----|-----|-----|-----|-----|  
Merged: [=========================] (0-50s)  
  
Result: 50 unique seconds

## 🔧 The Algorithm (Step by Step)

The SQL query in [databricks\_video\_aggregation.py:506-572](databricks_video_aggregation.py#L506-L572) implements this in 4 steps:

### Step 1: Add Previous Segment Info

**Input**: Raw segments sorted by start time

| segmentStart | segmentEnd |  
|--------------|------------|  
| 0 | 30 |  
| 20 | 50 |

**Process**: Use LAG() window function to get previous segment’s end

LAG(segmentEnd) OVER (PARTITION BY userId, videoId, sessionId ORDER BY segmentStart)

**Output**:

| segmentStart | segmentEnd | prevEnd |  
|--------------|------------|---------|  
| 0 | 30 | NULL | ← First segment (no previous)  
| 20 | 50 | 30 | ← Previous ended at 30

### Step 2: Detect Overlaps

**Logic**: Does this segment overlap with the previous one? - **Overlap**: segmentStart <= prevEnd (segments touch or overlap) - **Gap**: segmentStart > prevEnd (there’s a gap between segments)

**Process**:

CASE  
 WHEN prevEnd IS NULL OR segmentStart > prevEnd THEN 1 -- New group (gap)  
 ELSE 0 -- Same group (overlap)  
END as newGroup

**Output**:

| segmentStart | segmentEnd | prevEnd | newGroup | Explanation |  
|--------------|------------|---------|----------|--------------------------------|  
| 0 | 30 | NULL | 1 | First segment = new group |  
| 20 | 50 | 30 | 0 | 20 <= 30 = overlap = same group|

**If there was a gap** (e.g., third segment starting at 60):

| segmentStart | segmentEnd | prevEnd | newGroup | Explanation |  
|--------------|------------|---------|----------|--------------------------------|  
| 0 | 30 | NULL | 1 | First segment |  
| 20 | 50 | 30 | 0 | Overlaps with previous |  
| 60 | 90 | 50 | 1 | 60 > 50 = gap = new group |

### Step 3: Assign Group IDs

**Logic**: Use running sum of newGroup to create group identifiers - All overlapping segments get the same groupId

**Process**:

SUM(newGroup) OVER (PARTITION BY userId, videoId, sessionId ORDER BY segmentStart)

**Output**:

| segmentStart | segmentEnd | newGroup | groupId | Calculation |  
|--------------|------------|----------|---------|-----------------|  
| 0 | 30 | 1 | 1 | 1 |  
| 20 | 50 | 0 | 1 | 1 + 0 = 1 |

**With gap example**:

| segmentStart | segmentEnd | newGroup | groupId | Calculation |  
|--------------|------------|----------|---------|-----------------|  
| 0 | 30 | 1 | 1 | 1 |  
| 20 | 50 | 0 | 1 | 1 + 0 = 1 |  
| 60 | 90 | 1 | 2 | 1 + 0 + 1 = 2 |

### Step 4: Merge by Group

**Logic**: For each group, take the earliest start and latest end - MIN(segmentStart) = when the merged interval starts - MAX(segmentEnd) = when the merged interval ends

**Process**:

SELECT  
 userId, videoId, sessionId,  
 MIN(segmentStart) as mergedStart,  
 MAX(segmentEnd) as mergedEnd  
FROM grouped  
GROUP BY userId, videoId, sessionId, groupId

**Output**:

| groupId | mergedStart | mergedEnd | Length | Explanation |  
|---------|-------------|-----------|--------|--------------------------|  
| 1 | 0 | 50 | 50s | Merged [0-30] + [20-50] |

**With gap example**:

| groupId | mergedStart | mergedEnd | Length | Explanation |  
|---------|-------------|-----------|--------|--------------------------|  
| 1 | 0 | 50 | 50s | Merged [0-30] + [20-50] |  
| 2 | 60 | 90 | 30s | Standalone segment |  
  
Total unique seconds: 50 + 30 = 80 seconds

## 🎓 Complete Example Walkthrough

### Scenario: Peter Watches Video 1

**Raw watch segments** (from events):

Segment 1: 0s → 30s (watched 30 seconds)  
Segment 2: 30s → 120s (watched 90 seconds)  
Segment 3: 110s → 120s (rewound, watched 10 seconds)

### Step-by-Step Processing

**After Step 1** (add prevEnd):

| segmentStart | segmentEnd | prevEnd |  
|--------------|------------|---------|  
| 0 | 30 | NULL |  
| 30 | 120 | 30 |  
| 110 | 120 | 120 |

**After Step 2** (detect overlaps):

| segmentStart | segmentEnd | prevEnd | newGroup | Why? |  
|--------------|------------|---------|----------|---------------------|  
| 0 | 30 | NULL | 1 | First segment |  
| 30 | 120 | 30 | 0 | 30 <= 30 (touches) |  
| 110 | 120 | 120 | 0 | 110 <= 120 (overlap)|

**After Step 3** (assign groups):

| segmentStart | segmentEnd | newGroup | groupId |  
|--------------|------------|----------|---------|  
| 0 | 30 | 1 | 1 |  
| 30 | 120 | 0 | 1 |  
| 110 | 120 | 0 | 1 |

**After Step 4** (merge):

| groupId | mergedStart | mergedEnd | Unique Seconds |  
|---------|-------------|-----------|----------------|  
| 1 | 0 | 120 | 120 |

**Result**: Peter watched **120 unique seconds** (0-120s range) - Even though he rewound and re-watched 110-120s - Total watch time: 30 + 90 + 10 = 130s - Unique seconds: 120s (no double-counting)

## 🔍 Edge Cases Handled

### Case 1: Multiple Overlaps in Sequence

Segments: [0-30], [20-50], [40-80]

Timeline: 0 10 20 30 40 50 60 70 80  
 |-----|-----|-----|-----|-----|-----|-----|-----|  
Segment 1: [===========]  
Segment 2: [===========]  
Segment 3: [===========]  
  
Result: [=========================================]  
Unique seconds: 80 (not 30 + 30 + 40 = 100)

### Case 2: Non-Overlapping Segments (Gap)

Segments: [0-30], [50-80]

Timeline: 0 10 20 30 40 50 60 70 80  
 |-----|-----|-----|-----|-----|-----|-----|-----|  
Segment 1: [===========] (gap)  
Segment 2: [===========]  
  
Result: [===========] [===========]  
Unique seconds: 30 + 30 = 60

### Case 3: Fully Contained Segment

Segments: [0-100], [20-50]

Timeline: 0 20 50 100  
 |-----|---------|---------|  
Segment 1: [==========================]  
Segment 2: [=========]  
  
Result: [==========================]  
Unique seconds: 100 (segment 2 is fully inside segment 1)

### Case 4: Exact Duplicate

Segments: [0-30], [0-30]

Result: [===========]  
Unique seconds: 30 (duplicate completely ignored)

## 💡 Why This Algorithm Works

### Key Insight

By sorting segments by start time and checking if each segment overlaps with the previous one, we can identify **groups of overlapping segments**.

### The Magic of Running Sum

newGroup values: [1, 0, 0, 1, 0]  
Running sum: [1, 1, 1, 2, 2]  
 └────┘ └───┘  
 Group 1 Group 2

The running sum creates a unique ID for each group of overlapping segments!

### Efficiency

* **Time Complexity**: O(n log n) for sorting + O(n) for merging = O(n log n)
* **Space Complexity**: O(n) for intermediate results
* **Scalability**: Works efficiently in Spark with millions of segments

## 🧪 Testing This Algorithm

See [test\_data\_generator\_complete.py](../04_TESTING/test_data_generator_complete.py):

**TC-005: Skip Backward (Rewind)** - Tests overlapping segments from rewind behavior - Expected: unique seconds correctly calculated

**TC-009: Complex Navigation** - Tests multiple overlaps and gaps - Expected: all intervals merged correctly

**TC-021: Zero Duration Segment** - Tests edge case of segment with same start/end - Expected: handled gracefully (contributes 0 seconds)

## 📚 Additional Resources

### Related Code

* Main implementation: [databricks\_video\_aggregation.py:480-582](databricks_video_aggregation.py#L480-L582)
* Method documentation: [databricks\_video\_aggregation.py:477-492](databricks_video_aggregation.py#L477-L492)

### SQL Window Functions

* LAG(): Gets value from previous row
* SUM() OVER(): Running/cumulative sum
* Learn more: [Spark SQL Window Functions](https://spark.apache.org/docs/latest/sql-ref-syntax-qry-select-window.html)

### Alternative Approaches

* **Explode method**: Expand each segment to individual seconds, then deduplicate
  + Pros: Easier to understand
  + Cons: Very memory-intensive for long videos (300s video = 300 rows per segment!)
* **Interval tree**: More complex data structure
  + Pros: Faster for dynamic updates
  + Cons: Overkill for batch processing, harder to implement in SQL

## 🎯 Summary

**Problem**: Calculate unique seconds watched when users rewind

**Solution**: Merge overlapping time intervals

**Algorithm**: 1. Sort segments by start time 2. Detect overlaps with previous segment 3. Assign group IDs to overlapping segments 4. Merge each group by taking MIN(start) and MAX(end)

**Result**: Accurate unique seconds count without double-counting

**Complexity**: O(n log n) time, O(n) space

**Key Benefit**: Scales efficiently to millions of user-video combinations in Spark

*This algorithm is a classic computer science problem (interval merging) applied to video analytics. Understanding it helps ensure accurate engagement metrics!*