Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2022, 9(2):56-62

Research Article ISSN: 2394 - 658X

Backend-Driven Efficient Upload Algorithm: Balancing Data Freshness and Device Energy Consumption

Sambu Patach Arrojula

Email: sambunikhila@gmail.com

ABSTRACT
In mobile applications, maintaining a balance between data freshness and device energy consumption is a critical yet challenging task. Frequent uploads of event data ensure real-time insights but can significantly drain battery life and consume network data, especially over cellular connections. Conversely, infrequent uploads conserve energy but lead to delayed data availability, impairing the ability of app owners to make timely and informed decisions. This paper proposes a backend-driven efficient upload algorithm that dynamically balances these conflicting requirements. The algorithm leverages various trigger events at clients such as new data additions, connectivity changes, and elapsed time since the last upload and derives an initial equation to determine the optimal upload timing., then introduces additional constraints controlled by the backend, targeting data freshness and energy consumption KPIs to add separate weight/pressure on upload decisions so that overall KPIs move in target KPI direction. Through this hybrid and adaptive strategy, the proposed solution prioritizes Wi-Fi uploads when available and optimizes cellular data usage. Simulation results demonstrate that our approach effectively balances data freshness with energy efficiency, outperforming existing methods. This solution empowers mobile application developers to enhance user experience while maintaining operational efficiency.

Keywords: Mobile application, analytic events, data freshness, efficient upload, adaptive strategy, backend driven, near real time insights.

INTRODUCTION

Maintaining an optimal balance between data freshness and energy consumption is a critical challenge in mobile application development. Real-time data uploads ensure that backend systems have the most current information, enabling timely and informed decision-making. However, this practice can quickly deplete battery life and increase data costs, particularly when using cellular networks. Conversely, batching data for less frequent uploads conserves energy but results in outdated information at the backend, undermining the application's responsiveness and utility.

Mobile applications frequently generate event data that needs to be transmitted to backend servers for processing and analysis. Uploading every event as it occurs is inefficient, leading to excessive battery drain and high data usage. Hence, caching events locally on the device and uploading them in batches becomes necessary. This approach, however, introduces the problem of determining the optimal timing and conditions for these uploads to balance data freshness and energy efficiency.

Our proposed solution is a backend-driven efficient upload algorithm designed to dynamically balance these conflicting requirements. The algorithm is triggered by various events such as the addition of new data to the cache, changes in network connectivity, and the elapsed time since the last successful upload. Key parameters, including the threshold size of un-uploaded data, maximum waiting time for uploads, data consumption limits on cellular connections, critical battery levels, and minimal data sizes for Wi-Fi uploads, are considered to make upload decisions.

By integrating these parameters into a decision-making equation, the algorithm dynamically adjusts its behavior based on current conditions. Additionally, it incorporates constraints controlled by the backend to meet specific Key Performance Indicators (KPIs) related to data freshness and energy consumption. This backend control allows for fine-tuning the algorithm based on historical data and overall system requirements.

Our approach employs a hybrid strategy that prioritizes Wi-Fi for uploads when available and optimizes cellular data usage to minimize costs and energy expenditure. The proposed solution ensures that the backend receives timely data updates while conserving device energy, ultimately enhancing the user experience and operational efficiency of mobile applications.

APPROACH

Our main approach involves defining a generic yet adaptable formula for making upload decisions, aiming to balance energy consumption and data freshness effectively. This formula considers the key aspects of upload operations, including threshold event count, waiting time, data consumption limits, battery levels, and minimal data sizes. The primary challenge lies in deriving reliable and practical weightages for these constraints, as the optimal balance varies based on diverse usage patterns and conditions.

While historical data can be used to deduce these weightages, our approach in this paper introduces an innovative method: adding more constraints and allowing the backend to control these new constraints. This enables fine-tuning of the end results, such as Key Performance Indicators (KPIs), dynamically according to real-time needs.

Generic Formula for Upload Decision Making

The proposed formula incorporates essential parameters affecting upload decisions:

- **Threshold Event Count:** To avoid excessive cache buildup, which can lead to reduced data freshness and potential data loss.
- **Waiting Time:** To prevent long delays in data uploads, ensuring timely updates to the backend.
- **Data Consumption Limits:** Especially relevant for cellular connections to manage costs and energy consumption.
- **Battery Levels:** To avoid uploads when the device is critically low on battery.
- **Minimal Data Size for Wi-Fi:** To ensure efficient use of available Wi-Fi connections.

However, determining precise weightages for these factors is challenging. The optimal weightages can vary significantly based on each application's context and user behavior patterns. Instead of relying solely on historical data for static weightage determination (a feasible approach), our approach introduces additional constraints managed by the backend, allowing for dynamic adjustments. This method is not intended to find optimal weightages for each device, as deriving an optimal balance for every device is impractical. Rather, our approach aims to incorporate a normalized global constraint into upload decisions along with local constraints from device context, ensuring that overall global metrics converge in a desired direction.

Backend-Driven Constraints

By integrating backend-controlled constraints, we address the challenge of dynamically balancing KPIs:

- **Data Freshness KPI:** Backend can dynamically set a target median wait time for events to be uploaded, ensuring data remains fresh without overwhelming the network.
- **Energy Consumption KPI:** Backend can specify an average time to be spent on upload operations, managing overall energy usage effectively.

Advantages of Backend-Driven Control

Dynamic KPI Adjustments: The backend can change these KPIs in real-time without requiring client-side updates. This flexibility allows the system to adapt quickly to changing conditions and requirements.

Individual Client Control: The backend can fine-tune constraints for individual clients if needed, optimizing performance based on specific user behavior or device capabilities.

Enhanced Responsiveness: Backend control enables a more responsive system, capable of adjusting upload behaviors based on real-time data and analytics.

Scalability: This approach scales well with large user bases, as backend-driven adjustments can be applied uniformly or selectively, maintaining optimal performance across the board.

In summary, our approach leverages a generic formula for upload decision-making, enhanced by backend-driven constraints that allow for dynamic and fine-tuned adjustments. This method ensures that mobile applications can efficiently manage data uploads, maintaining operational efficiency and enhancing user experience.

DETAILS

1. Caching Added Events

Events generated by the application are first cached locally. Instead of uploading each event as it occurs, events are grouped into smaller chunks to optimize upload efficiency. Each payload for the upload API is defined as a chunk of 100 events. This chunking reduces the frequency of uploads, balancing the need for fresh data with energy conservation.

2. Metadata Maintenance

To manage uploads effectively, certain metadata is maintained:

- **Median Timestamps of Each Payload:** Each payload contains events with a median timestamp representing the average time the events have been waiting in the cache.

• **Last Uploaded Time:** The timestamp of the last successful upload is stored to calculate elapsed time for subsequent upload decisions.

3. Trigger Events for Making Upload Decisions

The algorithm uses specific trigger events to decide when to upload:

• **New Event Added to Cache:** Whenever a new event is cached, it triggers an evaluation.

• **Connectivity Change:** Network availability changes or transitions between Wi-Fi and cellular networks prompt a decision.

• **Elapsed Time Threshold:** When the time since the last successful upload exceeds a pre-defined threshold, an evaluation is triggered.

4. Generic Formula Constraints and Weight Calculation

The generic formula incorporates five key constraints, each influencing the upload decision either positively or negatively. We need all these constraints' weights to be between 0 and 1 to properly allocate these weights in the final equation.

• **Threshold Event Count (Nmx):** Prevents excessive cache buildup.

$$Cweight = \max(0, 1 - \frac{Nmx}{Ne})$$

Where Ne is the current number of events in the cache. This weight increases if the cache size exceeds Nmx which could be provided by the application owner/developer as he knows more about app conditions and requirements. This weight will remain neutral (0) until the cache size reaches Nmx. If the cache exceeds this size, the weight will gradually increase towards 1, applying more pressure for the upload.

• **Threshold Waiting Time (Tmx):** Ensures timely uploads.

$$Tweight = \max(0, 1 - \frac{Tmx}{Tw})$$

Where Tw is the time since the last upload. This weight increases if the elapsed time exceeds Tmx. which could be provided by the application developer/owner. This weight will remain neutral until the wait time reaches Tmx and if the wait time crosses threshold time this weight will gradually increase towards 1, to apply its own pressure for upload.

• **Data Consumption on Cellular (Dmx):** Manages data usage on cellular networks.

$$Dweight = -\max(0, 1 - \frac{Dmx}{Du})$$

An upload session involves consecutive uploads of cache chunks without another upload-trigger event. This means that once an event occurs and we decide to upload the cache, we upload it in chunks, making a decision for each chunk until the cache is fully uploaded or the decision is to stop. Du represents the total amount of data/events uploaded in the current session. The weight for the next upload decreases (as we can see above this is negative weight) as data consumption exceeds Dmx, a limit also set by the application developer.

• **Critical Battery Level (Bt):** Avoids uploads when the battery is low.

$$Bweight = -\max(0, 1 - \left(\frac{Bc}{Bt}\right)^{Bf})$$

This is also a negative weight. Here Bc is the current battery level, Bt is battery threshold set by application developer and Bf is a factor to exponentially increase the weight. We introduce an exponential factor to indicate a quicker weight contribution for this constraint. Additionally, we increase the local weight (weight in negative constraints) in the final equation because we aim to minimize energy consumption for uploads when the user's battery is critical. As the battery level decreases, the weight increases more rapidly, making the final upload decision more likely to be negative. And an extra weight has been added to Bweight parameter in the negative part of equation (as indicated below) to give significant impact on upload decisions unless the device is charging, in which case this constraint is bypassed.

• **Minimal Data Size for Wi-Fi (Nmin):** Ensures to skip insignificant uploads.

$$Sweight = -\max(0, 1 - \frac{Ne}{Nmin})$$

In general data is significant when it has good enough information that said we avoid uploads when cache is having too lesser events. This weight decreases if the cache size is below Nmin. a threshold set by application developers.
Upload Decision Formula:

$$UploadDecision = \frac{[Cweight + Tweight]}{2} - \frac{[Dweight + 3 \times Bweight + Sweight]}{3}$$

A positive or zero result indicates a decision to upload, while a negative result suggests not to upload.

5. Additional Constraints for KPIs

To dynamically manage the system's performance based on backend KPIs, two additional constraints are introduced:

• **Data Freshness KPI (Median Wait Time):**

$$MWweight = \max(0, 1 - \frac{MWt}{mw})$$

Where MWt is the target median wait time, returned by backend with every upload and mw is the current payload median wait time of its events. This weight increases if the payload has been waiting longer than the target.

• **Energy Consumption KPI (Average Upload Time):**

$$UTweight = \max(0, 1 - \frac{UTt}{ut})$$

• Where UTt is the target total upload time, returned by backend with every upload and ut is the current day total upload time application spent on upload operations. We send previous day's upload time to the backend in every upload request in return we get the target total upload time. This weight decreases if the current upload time exceeds the target.

Final upload decision formula:

$$UploadDecision = \frac{[Cweight + Tweight + MWweight]}{3} - \frac{[Dweight + 4 \times Bweight + Sweight + UTweight]}{4}$$

Using Metadata:

• **Median Timestamps Calculation:** For each payload, calculate the median timestamp of the events.

• **Average Time on Upload Operations:** Track the cumulative time spent on uploads per day and send this along with each payload to the backend.

Backend Interaction: With each upload, the client sends the median timestamp of the current payload and the previous day's average upload time to the backend. The backend processes this data and returns updated target KPIs for the median wait time and average upload time.

By incorporating these backend-controlled constraints, the system can dynamically adjust to meet real-time needs, optimizing both data freshness and energy consumption. This backend-driven approach ensures that the system remains flexible and adaptive, capable of responding to varying conditions without requiring client-side changes.

```
# Pseudocode for Upload Decision Making Function

# Constants and parameters provided by the app developer
Nmx = 1000  # Threshold event count
Tmx = 480  # Threshold waiting time in minutes
Dmx = 10000  # Max events for cellular upload in a session
Bt = 15  # Critical battery level percentage
Nmin = 20  # Minimal number of events for Wi-Fi upload
Bf = 2.5  # Battery factor

# Backend-provided dynamic KPIs
MWt = backend_provided_target_median_wait_time
UTt = backend_provided_target_upload_time

# Metadata maintained by the client
Ne = current_number_of_events_in_cache
Tw = time_since_last_successful_upload
Du = number_of_events_uploaded_in_current_session
Bc = current_battery_level_percentage
mw = current_payload_median_wait_time
ut = total_time_spent_on_upload_operations_today

# Function to calculate constraint weights
def calculate_weights():
   Cweight = max(0, 1 - (Nmx / Ne))
   Tweight = max(0, 1 - (Tmx / Tw))
   MWweight = max(0, 1 - (MWt / mw))
   Dweight = max(0, 1 - (Dmx / Du))
   Bweight = max(0, 1 - (Bc / Bt) ** Bf)
   Sweight = max(0, 1 - (Ne / Nmin))
   UTweight = max(0, 1 - (UTt / ut))

   return Cweight, Tweight, MWweight, Dweight, Bweight, Sweight, UTweight

# Function to make upload decision
def should_upload():
   Cweight, Tweight, MWweight, Dweight, Bweight, Sweight, UTweight =
calculate_weights()

   # Calculate positive and negative contributions
   positive_contribution = (Cweight + Tweight + MWweight) / 3
   negative_contribution = (Dweight + 4 * Bweight + Sweight + UTweight) / 4

   # Final upload decision
   upload_decision = positive_contribution - negative_contribution

   # Return True if decision is to upload, False otherwise
   return upload_decision >= 0

# Trigger the decision-making process
when(upload_trigger_event):
   for each chunk in cache
if should_upload():
#Perform the upload operation
   upload_events()
   # Update metadata and target kpis returned by b/e after successful upload
   update_target_kpis()
   update_last_upload_time()
   update_uploaded_event_count()
else:
   # Skip the upload operation
   break;
  end for:
end when:
```

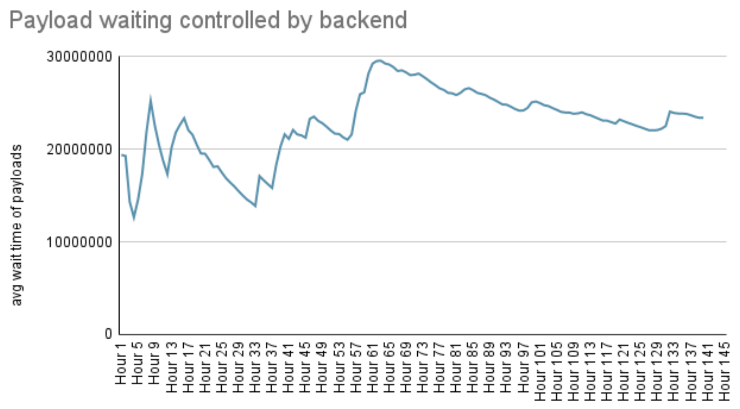
OBSERVATIONS

To validate the above concept, we developed a test application with the following setup:

• **Event Instrumentation and Cache System:** We instrumented a select number of events and integrated a caching subsystem that implements the smart upload algorithm described earlier.

• **Test Execution:** The test application was installed on a device and operated through a script designed to simulate continuous user engagement. The script was paced to ensure the application generated at least one event per minute, simulating a scenario of heavy user interaction.

• **Backend Controlled KPI: For** the first 50 hours, the backend did not provide any target KPIs. However, at the 50th hour, it returned a target wait time of 20,000,000 milliseconds (approximately 5.5 hours).
• **Average Wait Time (AWT) Analysis:** We collected and analyzed the average wait times (AWT) for each payload. The observations revealed that before the 50th hour, the AWTs fluctuated significantly, reflecting the random nature of user interactions and client conditions. However, after the 50th hour, the fluctuations diminished considerably, and the AWTs gradually decreased—not immediately, but progressively, as shown in the accompanying chart.
• **Impact of Backend driven KPI:** The target KPI provided by the backend does not impose a strict boundary on the client but exerts substantial influence, guiding the client towards the target KPI over time.



NEXT STEPS

• Explore Dynamic Assignment of Target KPIs per Client:
• Tailoring KPIs dynamically for each client can ensure that specific needs and usage patterns are addressed more effectively. This involves analyzing individual client behavior and adjusting KPIs in real-time to optimize performance and user satisfaction.
• Implementing a machine learning model to predict and adjust target KPIs based on historical data and current usage patterns can enhance the system's adaptability and responsiveness.
• Consider Clients Having Their Own KPI Factors:
• Allowing clients to have their own KPI factors and adjust them as they evaluate performance can lead to more personalized and effective management of resources. Clients can fine-tune these factors based on their specific goals and operational requirements.
• Providing a user-friendly interface for clients to monitor and adjust their KPIs will empower them to take an active role in optimizing their data upload strategies, leading to better alignment with their unique needs and improved overall satisfaction.

CONCLUSIONS

This paper presents a novel backend-driven approach to balancing data freshness and energy consumption in mobile applications through an efficient upload algorithm. Unlike traditional strategies that rely solely on historical data to derive exact weightages for various parameters, our approach introduces dynamic backend-controlled constraints. This allows for real-time adjustments based on current conditions and specific requirements, making the system more adaptable and responsive.
By leveraging backend-driven control, this strategy offers several advantages over conventional methods. It allows for dynamic KPI adjustments without necessitating client-side updates, providing flexibility to adapt to changing conditions and requirements. Additionally, the backend can fine-tune constraints for individual clients, optimizing performance based on specific user behavior or device capabilities. This centralized control also enhances scalability, ensuring that the system performs optimally across a large user base.
In initial trials, this backend-driven approach has shown promising results. Although the backend KPI targets did not move as expected—specifically, enough KPIs were not impacted or did not moved quickly enough—the strategy did help to some extent by steering the KPIs in the required direction without altering the clients. Given that this backend-driven approach has additional advantages, it was able to meet expectations by balancing both data freshness and energy efficiency. The system's ability to dynamically balance these conflicting requirements has demonstrated significant improvements in operational efficiency and user experience. The results suggest that this approach can effectively address the challenges in mobile data upload management, providing a robust solution that adapts to real-time needs while maintaining optimal performance.

Overall, this backend-driven efficient upload algorithm represents a significant advancement in mobile application data management, offering a flexible, adaptive, and scalable solution to the critical challenge of balancing data freshness and energy consumption.

REFERENCES

[1]. https://www.singular.net/glossary/app-analytics/
[2]. https://mobisoftinfotech.com/resources/blog/mobile-app-analytics/
[3]. https://link.springer.com/article/10.1007/s11036-020-01650-z
[4]. https://www.mintmobile.com/blog/wifi-vs-cellular-data/
[5]. https://proandroiddev.com/dynamic-screens-using-server-driven-ui-in-android-262f1e7875c1
[6]. https://uxmag.com/articles/using-back-end-design-to-create-customizable-front-end-mobile-experiences