Optimization of Long-Running Media Aggregation Queries

Sigurður Þórarinsson
CRESS
Reykjavik University
Reykjavík, Iceland
sigurdurt24@ru.is

Björn Þór Jónsson
CRESS
Reykjavik University
Reykjavík, Iceland
bjorn@ru.is

Omar Shahbaz Khan
DSAR
IT University of Copenhagen
Copenhagen, Denmark
omsh@itu.dk

Abstract

Unlike traditional retrieval, dynamic exploration of multimedia collections may require complex aggregation queries whose performance is highly sensitive to dataset size, filters and grouping applied at any time. Such queries often yield unstable response times, thus undermining interactivity. Inspired by the online aggregation approach from the database community, we investigate whether progressively refined intermediate results, accompanied by quality estimates, can enable users to decide whether to accept partial results or continue processing. We evaluate this approach within the Multidimensional Media Model, where media collections are explored through metadata tagsets and hierarchies. Our study analyses operator placement, showing that moving deduplication and grouping from the database to the server reduces time-to-first-result by over 90% while maintaining steady quality improvement. We further examine join ordering and selectivity estimation for reliable progress prediction. Our results demonstrate that online aggregation principles can substantially improve responsiveness in multimedia exploration systems.

CCS Concepts

• **Information systems** → **Multimedia and multimodal retrieval**; **Users and interactive retrieval**; **Query optimization**.

Keywords

Multimedia exploration, Interactive query processing, Multidimensional media model

ACM Reference Format:
Sigurður Þórarinsson, Björn Þór Jónsson, and Omar Shahbaz Khan. 2026. Optimization of Long-Running Media Aggregation Queries. In *International Conference on Multimedia Retrieval (ICMR '26), June 16–19, 2026, Amsterdam, Netherlands.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3805622.3810667

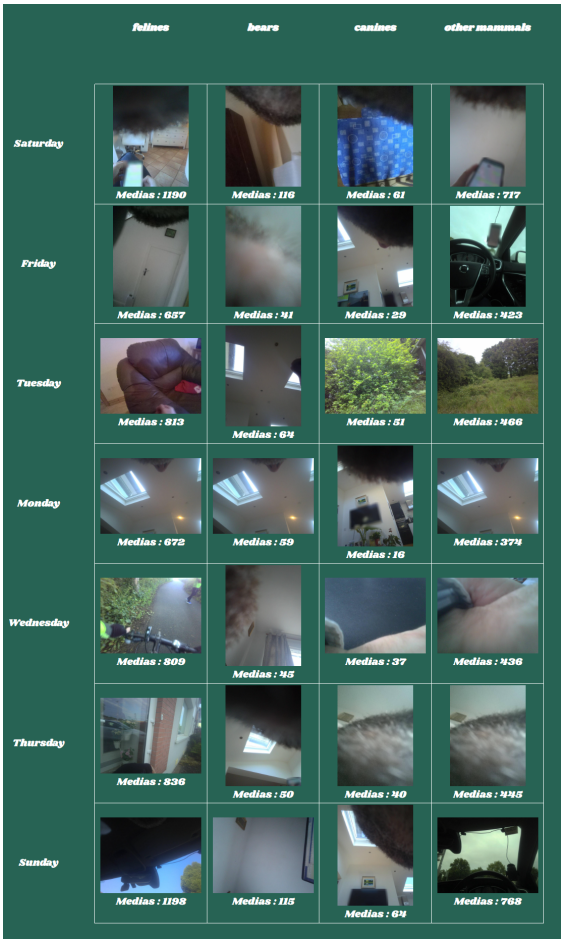


Figure 1: Example of a *browsing state* in the M^3 model. The x-axis shows the four main categories of mammals detected in the lifelog collection, while the y-axis shows the seven days of the week; additionally, a filter is applied to show only images from 2020. Each block in the browsing state is called a *cell* and represents all lifelog images from 2020 associated with one mammal category and one day of the week.

1 Introduction

A significant portion of the data we generate is media files, both in our day-to-day lives, through personal photo and video collections, and through business-related activities, such as medical imagery or surveillance. As digital media collections grow in size and diversity, interactive multimedia exploration, a practice that emphasises browsing and interacting with media content along multiple facets, has become an increasingly important tool. Users of these datasets often do not have a specific query in mind but rather a vague idea of what they are looking for, such as relationships between media subsets or patterns that would be difficult to identify using search alone. As collections grow, so does the importance of providing scalable multimedia exploration and analytics systems [24].

ICMR '26, Amsterdam, Netherlands

ACM ISBN 979-8-4007-2617-0/26/06
https://doi.org/10.1145/3805622.3810667

Any scalable interaction with a media collection involves running some query (or queries) to a data management system. For traditional multimedia search, where the goal is to provide the user with a small set of relevant items, optimisation of such queries is relatively easy: provide an (approximate) index that supports a notion of relevance (typically similarity of features or embeddings) and probe a small portion of that index to produce the result. This process is generally both scalable and stable, which means that the query performance is consistently perceived by users as interactive across all queries. In contrast, exploration queries often involve aggregation and dynamic filters, where the data management system must aggregate a large proportion of the entire collection to produce a summary of the collection, and that proportion varies significantly depending on the filters required at each time. Such systems therefore are both less scalable and less stable in their performance, resulting in some interactions taking much longer time than others and, importantly, taking more time than users perceive as interactive performance.

The database community has historically studied methods to optimise long-running aggregation queries for relational databases, including an approach called *online aggregation* [11]. Overall, the goal of online aggregation processing is to maintain a running aggregation, and present to the user both the current result of the aggregation query and a statistical estimate of its quality. With this approach, the user has a choice: if they are satisfied with the current answer, they may stop the query and move on to the next interaction in their exploration process, but if they do not yet trust the answer, they may allow the query to run longer and possibly to completion. With the online aggregation approach, the user gains control of the aggregation process, even when it is applied to an extremely large collection. Inspired by this approach, this paper studies its application to one particular and important type of aggregation queries for multimedia collections.

One of the data models proposed for exploration of media collections is M^3 (Multidimensional Media Model [14], or *emm-cube*). M^3 focuses on exploration based on media metadata, where metadata includes operational metadata (e.g., media creation time or media dimensions) and semantically generated metadata (e.g., colour information or model-generated tags). In M^3, each media object can be described by multiple metadata tags. Each tag belongs to a single tagset and (multiple) tag hierarchies can be used to organise each such tagset. Media collections can be explored by projecting tagsets or hierarchy nodes to visualisation axes (x-y axes for 2D screen-based visualisation, or x-y-z axes in VR), and by defining additional filters on nodes, tagsets or tags. At any given time in an exploration, the set of applied projections and filters is termed a *browsing state*; Figure 1 shows an example of such a browsing state for a lifelog application.

Seen from an underlying relational database, the browsing state query is a dynamic and complex aggregation query, consisting of a set of joins, followed by deduplication and grouping. It aggregates the collection based on the projected tagsets and hierarchies, returning for each cell of the browsing state the number of media items belonging to the cell and providing one representative example media item. The performance of this aggregation query is highly sensitive to both the size of the collection and which data is projected and filtered. The goal of this paper, inspired by the online aggregation approach, is to study whether users can be provided with sufficient information at any point during query processing to make *an informed decision to accept the intermediate result or let the system keep improving it.*

This paper makes the following contributions:

- We analyse how the placement of deduplication and grouping operators within a client-server-database architecture affects query performance and result quality over time. Overall, the results show that moving these operators from the database to the M^3 server improves the time to see the first result by more than 90% in all cases, with result quality improving steadily over the lifetime of the queries.
- We study the impact of join ordering on performance and quality of the optimised approach. The results show that query performance is relatively insensitive to join ordering.
- A reliable estimate of query selectivity is needed to give users a reasonable estimate of the query progress and result quality. We study the applicability of standard database query optimisation techniques to this problem and show that some statistical information is needed on relationships between tagsets.

The remainder of the paper is organised as follows. Section 2 presents the necessary background information for the work. Section 3 presents the policies studied for operator placement, join ordering and query optimization. Section 4 discusses the experimental setup, including the lifelog collection and browsing states studied. Section 5 then presents and discusses the results, before we conclude in Section 6.

2 Background

Through the years, the multimedia community has developed a number of systems aimed at multimedia exploration, including PhotoMesa [3], Scenique [2], MediaMill [23], MediaTable [20], Navigu [1], PicArrange [15], Exquisitor [16] and MeGraS [21]. The earlier systems dealt with relatively small collections, while the more recent systems aim at exploring larger collections; none of these consider dynamic aggregations in the same manner as we do in this paper.

In the remainder of this section, we first outline the M^3 data model and then present the system architecture we use to implement the model. We then describe the SQL query that implements the browsing state; optimising this query is the focus of this paper. Finally, we describe the online aggregation approach and related optimisation efforts from the database community.

2.1 M^3 Model

The M^3 model, which is derived from historical work in business analytics, considers multimedia objects to reside in a multidimensional metadata space and allows exploring that space via visualisation operations that project it down to conventional 2D/3D space [7, 14]. The following concepts define this metadata space and the operations that enable its exploration.

Media: These can be images, video clips, audio files, or any other media, represented by file URIs containing the media items.

Tags and Taggings: Each object is described by a set of metadata *tags*. These metadata tags can be (i) created with the media object, (ii) extracted from the media content (e.g., using semantic classifiers), or (iii) provided by users. Each association between a media object and a tag is called a *tagging*.

Tagsets, Hierarchies, Dimensions and Metadata Space: Each tag resides in a particular *tagset*; these are used to group together semantically related tags. A tagset can then be organised using one or more *hierarchies*, where each hierarchy *node* represents a tag; a media item is considered to be tagged by a node if it is tagged by its tag, or by a tag represented by any node within one of its sub-trees. Both tagsets and hierarchies can be used as exploration *dimensions* and the cross-product of all exploration dimensions yields the multi-dimensional *metadata space*. Note that the metadata space is naturally sparse, but when projected down to 2D/3D (described next) it generally becomes much more dense.

Projections and Cells: By associating a tagset with one of the visible axes of the 2D/3D visualisation, the collection is projected onto this axis by grouping the objects of the collection based on their association with the tags in the tagset. Correspondingly, by associating a hierarchy node with one of the visible axes, the collection is projected onto this axis by grouping the objects of the collection based on their association with subnodes of the projected (parent) node. Each of these groups is called a *cell*. When two (or more) dimensions are projected onto axes, each cell now represents one tag or child-node for each axis, as shown in Figure 1. Note that media items that are not associated with any tags/nodes along one of the axes are not included in the visualisation, while media items that correspond to multiple tags/nodes along a dimension are included multiple times.

Implicit and Explicit Filters: Since media items that are not associated with one of the axes are not included in the visualisation, the projected dimensions thus define implicit filters on the media collection. The contents of cells can be further constrained by defining explicit filters that require association with a tag (or tags), a tagset, or a node for a media item to be included in their cells.

Exploration Actions: During exploration, the visualisation can be modified in a variety of ways, in addition to projecting a dimension or explicitly applying filters. These exploration actions include traversing hierarchies—*drilling down* into a subtree by projecting a child node to the axis previously visualising its parent node or *rolling up* from a child node to a parent node—and *pivoting*—replacing one projected dimension with another and adding a corresponding explicit filter for the replaced dimension.

Browsing State: The state of exploration at any time is defined by the dimensions that are currently projected, which implicitly define associated filters, as well as any other filters that the user has applied. These dimensions and filters define the shape of the current visualisation, and can be easily translated into API calls, while the media items that pass through all its filters define its content; as the collection evolves over time, so does the visualisation content. We refer to the definition and content of the visualisation at each time as the *browsing state*.

```
// Media and Metadata Information
Media(id, URI, ...)
Tags(id, tagset_id, value, ...)
Taggings(object_id, tag_id)

// Dimension Information
Tagsets(id, name, ...)
Hierarchies(id, name, tagset_id, ...)
Nodes(id, hierarchy_id, parentnode_id, tag_id)

// Materialized Views for Query Performance
Tagset_Taggings(tagset_id, object_id, tag_id)
Nodes_Taggings(parentnode_id, object_id, node_id)
```

Figure 2: Overview of the relational database structure.

2.2 Database and Browsing State Queries

Figure 2 shows the tables and views in the relational database that are important for this work. The six tables in Figure 2 are straightforward representations of the corresponding model concepts. *Media* stores information about the multimedia items; *Tags* stores information about tags, including the identifier of the tagset it belongs to; and *Taggings* is a many-to-many relationship table that stores information about which tags are associated with which media items. The tags are then organized with the following tables: *Tagsets* enumerates the tagsets; *Hierarchies* enumerates hierarchies; and *Nodes* stores details of nodes in the hierarchies, including which tag the node represents.

The goal of a browsing state query is to produce data for a browsing state visualisation, such as shown in Figure 1, as efficiently and as possible. Before discussing the query, Figure 2 also shows two materialised views that facilitate efficient execution of browsing state queries [13]. For tagset projections, an (implicit) filter is first applied on the given *tagset_id*, then the *object_id* is used to determine which media objects pass the filter, and finally the grouping attribute is *tag_id*. For node projections, an (implicit) filter is first applied on the given *parentnode_id*, then the *object_id* is used to determine which media objects pass the filter, and finally the grouping attribute is *node_id*.

Turning to the browsing state query, itself, Figure 3 shows an example SQL query that delivers the browsing state in Figure 1. As outlined in the abstract representation on the right side of Figure 3, the browsing state query logically consists of three components; in this paper, we focus on practical optimisations of these components.

Joins: The role of the join operators is to retrieve information about all objects that contain tags that pass through all implicit and explicit filters that are defined in the browsing state. In the example query, the node projection uses the *Nodes_Taggings* view, the tagset projection uses the *Tagset_Taggings* view, while the two filters use the *Taggings* table directly.

Deduplication: The cell must provide an accurate count of the *unique* media items in each cell, achieved by retaining only one copy of each distinct media identifier for each browsing state cell.

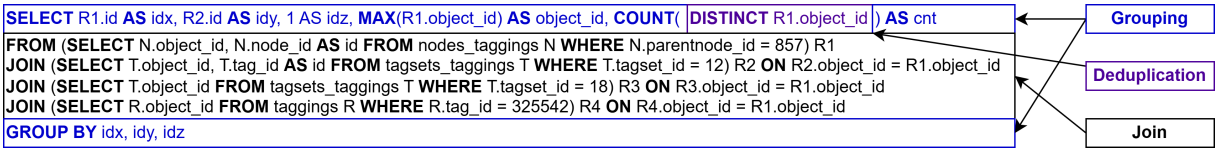


Figure 3: Abstraction of the baseline query for a browsing state.

Grouping: Finally, a cell is formed for each tag combination projected to the visualisation axes, by counting the retained copies of media identifiers and selecting one identifier to represent the cell; here, the highest identifier, thus most likely corresponding to the latest media item inserted.[1]

2.3 Processing of Long-Running Queries

Online aggregation—dynamically displaying incomplete but improving aggregation results—is a concept that was first introduced in the late 90s by Hellerstein et. al. [11]. The core idea of producing early join results quickly got adopted into the design of a new wave of non-blocking join algorithms, starting with sort-merge join [11], ripple join [9], progressive merge-join [6], hash-merge join [19], and many more. The concept has also been adopted towards distributed storage solutions [5] to show early results from distributed jobs and to improve network decisions for streaming services [10]. Databricks' evolving data frames is a new data model designed to support nested online aggregation operations [22], providing much deeper insight into early results of complex nested aggregations.

3 Methodology

In this work, we have extended the publicly available implementation of the M^3 model[2] to study performance aspects of the browsing state query. This implementation follows a traditional three-tier architecture:

Database: The database server is implemented using the relational PostgreSQL server, with the optimisations described above.[3]

Server: We have implemented a new M^3 server in Go using the gRPC communication model.[4] We chose gRPC for three reasons: (i) it has been shown to provide more efficient communications than REST APIs; (ii) it allows for streaming intermediate results, which is fundamental to the optimisations explored in this paper; and (iii) it is easy to create a REST API on top of the server, to facilitate migration of existing clients from the older REST-based server.

Client: For this paper, we have implemented a custom-made gRPC-based client to issue browsing state requests and evaluate their performance.

In the remainder of this section, we first describe the potential strategies for placement of and communication between the deduplication and grouping operators (Section 3.1), then describe the potential of join orderings (Section 3.2), and finally outline our initial approach to similarity estimation (Section 3.3).

3.1 Placement of Aggregation Operators

We are first interested in the impact that placement of aggregation operators has on query performance.[5] We identified seven meaningful variants that differ in (a) the placement of aggregation operators and (b) the communication between operators; these variants are shown in Figure 4. To identify the variants, we labeled each variant with three letters, each denoting one characteristic of the pipeline:

(1) The first letter is the placement of the deduplication operator, where the possible values are **D** for database, **S** for server and **C** for Client.
(2) The second letter is the placement of the grouping operator, with the same possible values of **D**, **S** and **C**.
(3) The third letter denotes the granularity of the data transfer from the server to the client, where **F** indicates that the full browsing state result is delivered at the end of processing (no incremental updates on client), **B** indicates that the server batches incremental updates to the browsing state, and finally **I** denotes a fully incremental delivery of the browsing state by transferring individual rows emerging from the join component and computing the browsing state at the client.

In Figure 4, the arrows between database, server and client further indicate data transfer from one layer to the next, with solid arrows indicating transfer of the complete result, dashed arrows indicating transfer of intermediate results, and dotted arrows indicating transfer of individual result rows.

DDF: The **DDF** variant corresponds to the baseline browsing state query of Figure 3, where all processing takes place in the database and the final result is sent first to the server and then to the client. Since the deduplication and grouping operators are *blocking* on the database server, it is not possible to transfer intermediate results in this configuration.

DSF / DSB: As the grouping operator is moved from the database to the server, two variants are possible: **DSF** transfers the final complete result to the client, while **DSB** transfers intermediate browsing state results from the server to the client. Since the deduplication operator is still performed in a blocking manner in the database, however, grouping can only start when the deduplication is effectively finished,

[1] Since the example query returns a 2D visualisation, the z-axis is not needed; for simplicity of query generation and browsing state construction, missing axes always have the grouping value 1.
[2] https://github.com/Ok2610/PhotoCube-Server
[3] Additionally, *enable_hashjoin = off* query planner restriction is used for all browsing state queries. The reason is that the construction of hash tables takes up a significant portion of the query processing time when hash joins are utilized, which blocks the database from emitting results until the hash table computations are finished. This setting avoids this problem and therefore improves performance.
[4] https://grpc.io/
[5] Since the join component requires a great deal of communication, results of historical research in the database field demonstrate strongly that the join component should always be performed in the database.

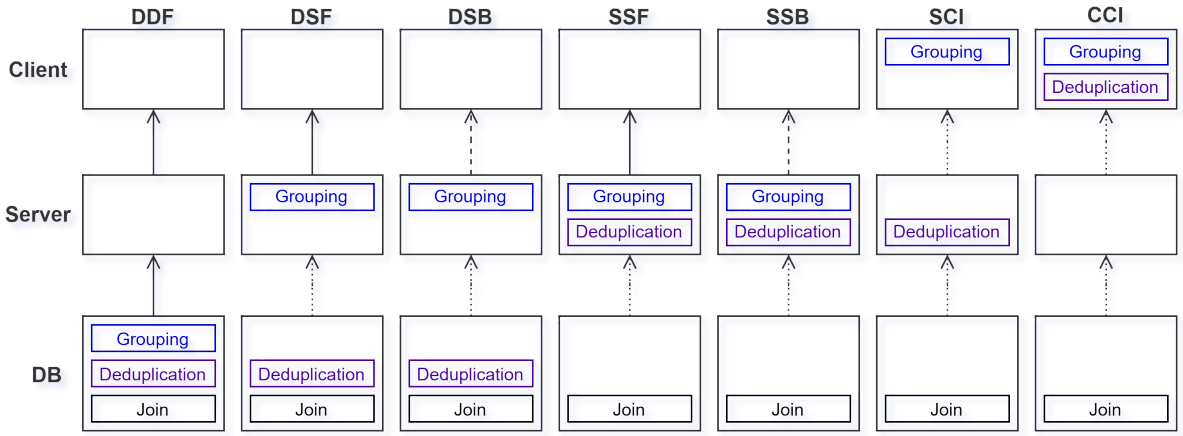


Figure 4: The seven variants of aggregation operator placement studied in this paper.

so deduplication and grouping cannot run in parallel; we therefore expect these variants to perform poorly.

SSF / SSB: As deduplication is moved to the server, we can implement it in a non-blocking manner, which means that deduplication and grouping can start to collect and transfer intermediate browsing state results, as soon as the first results are obtained from the join component. The **SSF** variant, which only transfers the complete result, fails to take advantage of this opportunity and should therefore always be slightly slower than **DDF**. The **SSB** variant, on the other hand, should deliver the first intermediate results quickly, even though the final result will be delivered at a similar time as with the **SSF** variant.

SCI / CCI: The final two variants both send individual rows from the server to the client, and then group them on the client. Since the **SCI** variant performs deduplication on the server, however, it may send fewer result rows to the client. Both these variants are expected to perform poorly, especially in a network constrained environments, and especially when the underlying database is large in comparison with the browsing state result. By including this variant, however, we can track the evolution of the quality of intermediate results in minute detail.

To summarise, we expect that **SSB** is the best variant for long-running queries, as it can start to return intermediate results rapidly, while only having a small performance overhead to deliver the complete result compared to the original **DDF** variant. In the experimental evaluation of Section 5.1, we study the query processing performance and the evolution of result quality in detail, confirming our intuition on query processing performance and showing that result quality improves steadily over time.

3.2 Join Ordering

In relational database systems, it is well known that the order in which joins are performed can significantly impact query processing performance, but also that modern query optimisers are very good at selecting query processing plans that avoid the worst plans, resulting in close to optimal query processing performance [17].

To explore how the order of projections and filters impacts the performance of the join component of the browsing state query, we run each browsing state repeatedly with all possible orders of filters and projections (if there are n projections and filters, this gives $n!$ possible join orders), measuring both query processing performance and result quality. Furthermore, we ran each join order with two settings: (a) forcing the database server to respect the given join order[6] and (b) allowing the query optimiser to rearrange the join order as it prefers.

As before, we study both query performance and result quality, as the order in which data is read is known to impact result quality [9, 11]. If the impact of join order on performance and result quality is insignificant, then we can simply trust the database optimizer to decide the join order. If the impact is significant, however, we could investigate what drives this performance difference and integrate that understanding into the implementation of our server. In Section 5.2 we study the impact of join ordering on query processing performance and intermediate results quality.

3.3 Selectivity Estimation

When query results are presented incrementally, it is vital to provide users with a picture of the query evaluation progress that is as accurate as possible. Such a picture has two components: (a) an understanding of the evolution of result quality over time; and (b) an estimation of the expected running time of the query. We expect that the quality evaluation results for the different operator placement variants will give us the understanding needed of result quality evolution over time; in this section we consider whether traditional methods from the database community can offer a robust estimation of result size of the join component, which in turn can be used to estimate how much of query processing effort remain.

For queries like the join component of the browsing state query, estimation of join result size traditionally rests on two assumptions.

[6] Join order is enforced by parameter setting *join_collapse_limit = 1*.

First, that the results of filter constraints are uniform in size regardless of the value of the filtering constant, and second, that the filters are independent. If both assumptions hold, the expected cardinality (result size) of a multi-way equijoin can be approximated as the product of the input relation cardinalities divided by the join-key domain size raised to the power of the number of joins [4, 12]. More precisely, for a query joining k filtered relations on a common key, the estimated result size $\widehat{T}$ is:

$$\widehat{T} = \frac{\prod_{i=1}^{k} N_i}{U^{k-1}}$$

where N_i denotes the number of rows produced by the i-th filter, and U denotes the size of the join-key domain.

Considering the first assumption, it is known that real-life data generally is not uniform, but is more likely to follow power-laws or other distributions. For this reason, database systems often keep track of histograms of common values to avoid severely underestimating results sizes. In our case, we pre-compute statistics for each possible filter using a materialized view that records the number of taggings associated with each metadata value. This gives, for all filters, an accurate count of N_i. In a real system, this overhead would be too high, but the traditional approach of keeping track of the most common values would be sufficient to avoid serious underestimates.

Because this approach relies on uniformity and independence assumptions, we expect it to perform poorly when filters are strongly correlated or anti-correlated. In multimedia collections, correlations are common: semantic concepts such as a polar bear and a beach ball are unlikely to co-occur within the same object, while others, like a desk and a coffee mug, may appear together frequently. In such cases, the independence assumption is likely to lead to substantial under- or overestimation of the query's selectivity. However, we expect that when correlations between filters are weak or absent, the classical estimator will provide a reasonably accurate approximation of the query selectivity. Turning to the second assumption, this formula is expected to underestimate results size when attributes are correlated and overestimate when they are anti-correlated. In Section 5.3 we investigate the validity of this traditional approach.

4 Experimental Setup

Since there exists no benchmarks for media exploration as of yet, we have designed a workload to exercise different performance aspects of browsing state queries. In this section we first describe the media collection and its metadata dimensions, then the browsing states we use in our performance evaluation, and finally the hardware and metrics used.

4.1 Media Collection: Lifelog Search Challenge

We have used the latest collection from the Lifelog Search Challenge [8]. This collection is of large enough scale, consisting of 725,226 lifelog images taken across 18 months, and importantly it is distributed with a diverse set of associated metadata. In total, we have created 27 tagsets and 12 hierarchies. The most relevant tagsets for this paper are (i) based on timestamps: minute (within the hour, 60 values), day of week (string or number, 7 values each), year (4 values), month (12 values), year-month-day (530 values), and time zone (36 values); (ii) image content: ImageNet tags with 1,000 distinct possible tags and 10 taggings per image; and (iii) other metadata: sleep level (7 values) and distance (1,052 values). Simple hierarchies are defined on the following tagsets: year-month-day (three levels: root, year, month), day of week (two levels: root, weekdays/weekend), and sleep level (two levels: root, asleep/awake/unknown). For the ImageNet tagset, ChatGPT was prompted to create 6 hierarchies that are roughly balanced but meaningfully separate the tags, resulting in the following hierarchies: animals, plants and fungi, objects and tools, scenes and environments, food and beverages, and person-related. These tagsets and hierarchies allow us to create browsing state queries that have interpretable behavior and stress different aspects of the query execution.

4.2 Browsing States

We have defined and experimented with 13 browsing states in total. While analysing our results, we observe the importance of the following two main parameters, in this order: how many rows are read and returned from the join component of the query and how many cells are created. Both parameters impact the quantity of data transferred between system layers, but differently depending on the placement of deduplication and grouping operators. In our presentation, we therefore mainly focus on the four browsing states in Table 1, which represent high and low values for these two parameters. We expect that the RH and HW browsing states will be the most time consuming to process, but that the high number of states for GH and especially HW can also impact performance when intermedia results are delivered across a network to the client.

4.3 Hardware and Metrics

The entire client-server-database architecture was run on three separate Docker Containers, all hosted on the same computer: a high-end Lenovo ThinkPad mobile workstation with an Intel Core Ultra 7 165H (16-core/22-thread), 64 GB LPDDR5X-7467 memory, and an NVIDIA RTX 2000 Ada GPU. Note that this setup does artificically reduce the impact of network traffic, especially for the SCI and CCI variants.

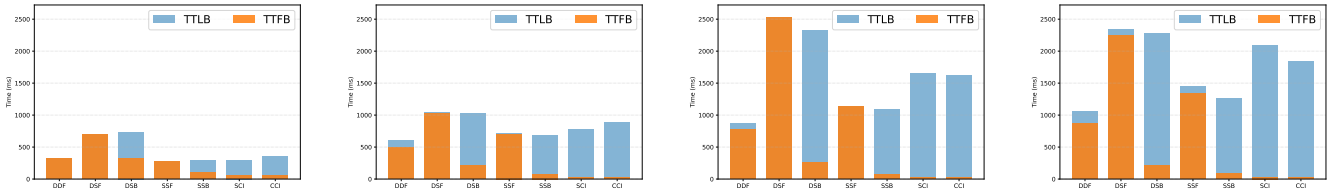
For each browsing state variant and each parameter setting, we first ran a query to warm the cache and then report averages of 30 runs of the query. We study a variety of query performance metrics, but here we focus on the time-to-first-byte (TTFB) and time-to-last-byte (TTLB) from the perspective of the client.

To evaluate quality of intermediate results, we consider the sizes of cells in a browsing state to form a distribution, and evaluate how quickly the cell size distribution of the streamed intermediate result approaches the cell size distribution of the final result, by using the *Jensen–Shannon divergence (JSD)*. Given two discrete probability distributions P and Q over the same support, JSD is a symmetric, bounded divergence defined as the average of the Kullback–Leibler divergences from each distribution to their mean mixture $M = \frac{1}{2}(P + Q)$. Using $\log_2$, the divergence is measured in bits and satisfies $\mathrm{JSD}(P\|Q) = 0$ iff $P = Q$ [18].

$$\mathrm{JSD}(P\|Q) = \frac{1}{2}\sum_i p_i \log_2\left(\frac{2p_i}{p_i + q_i}\right) + \frac{1}{2}\sum_i q_i \log_2\left(\frac{2q_i}{p_i + q_i}\right)$$

Table 1: Overview of the representative browsing states.

State	Description	Rows	Cells	Definition	Axis	Type	Tagset / Hier.	Nodes / Tags	Images
LW	LightWeight	10,592	28 / 28	Projections	x	Node	Mammals	4 sub-nodes	61,574
					y	Tagset	Day of week	7 tags	725,226
				Filters		Tagset	Year	4 tags	725,226
						Tag	2020	1 tag	203,335
GH	Grouping-Heavy	77,072	4,097 / 32,612	Projections	x	Tagset	Distance	1,052 tags	535,829
					y	Tagset	Day of month	31 tags	725,226
				Filter		Node	Clothing	6 sub-nodes	198,987
RH	Reading-Heavy	724,487	268 / 378	Projections	x	Node	Year Month	18 sub-nodes	724,487
					y	Tagset	Day of week	7 tags	725,226
					z	Node	Sleep Level	3 sub-nodes	725,226
HW	HeavyWeight	725,226	31,553 / 32,940	Projections	x	Tagset	Year Month	549 tags	725,226
					y	Tagset	Minute	60 tags	725,226
				Filter		Node	Day of week	2 sub-nodes	725,226



(a) LightWeight **(b) Grouping-Heavy** **(c) Reading-Heavy** **(d) HeavyWeight**

Figure 5: Experiment 1—Query processing performance of the seven aggregation placement variants.

5 Results

In this section, we report on three experiments to study: the impact of aggregation operator placement (Section 5.1); the impact of join ordering (Section 5.2); and the accuracy of query selectivity estimation (Section 5.3).

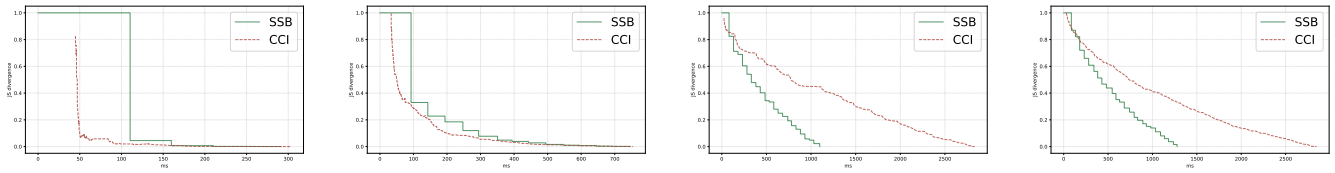
5.1 Experiment 1: Operator Placement

In this experiment, we study the impact of aggregation operator placement. We evaluate the seven variants of Figure 4 for the four representative browsing states from Table 1, studying both query processing performance and result quality evolution.

For the batching variants (**DSB** and **SSB**), whenever a row is scanned, the corresponding cell is marked dirty. The system periodically takes snapshots of dirty cells, converts them into response messages, and sends them over the gRPC stream. Our experiment uses a 50 ms batching interval, meaning that the first batch is sent 50 ms after the database query is initiated, and a new batch is sent every 50 ms after that; the TTFB can thus never be less than 50 ms. The fully incremental variants (**SCI** and **CCI**) convert each scanned row to a response message and send it over the stream as soon as it made available from the database. The remaining three variants (**DDF**, **DSF** and **SSF**) only send one response message per cell in the final browsing state once the aggregation process has completed.

Figure 5 shows the query processing performance, represented by TTFB and TTLB, for the four browsing states. The LW and GH browsing states require reading much less data and are thus computed rapidly, while the RH and HW browsing state require about 1 second of processing time with the **DDF** variant. For all four browsing states, the results are as predicted: the **SSB** variant shows the best tradeoff between TTFB and TTLB, with the first results appearing already after (the minimum) 50 ms (with the exception of the LW browsing state), and the final full results displayed only 10-20% slower than with the **DDF** variant.

Turning to result quality, Figure 6 shows the evolution of the Jensen–Shannon divergence (JSD) over time for the four browsing states. In each figure, the x-axis shows the elapsed time, while the y axis shows the JSD at each time. Each figure shows two variants: **SSB** as the best batching variant, and **CCI** as the variant that allows the most fine-grained evaluation of quality. The results indicate that convergence is very fast for the LW and GH browsing states that read less data, but convergence is also rapid (and sub-linear) for the RH and HW browsing states. The figures show that the shape of the convergence curve of **SSB** is identical to that of **CCI**, only significantly faster. Overall, the results indicate a very steady improvement in the result quality shown to users, indicating that **SSB** also performs best in terms of result quality evolution.



(a) LightWeight (3) (b) Grouping-Heavy (11) (c) Reading-Heavy (2) (d) HeavyWeight (13)

Figure 6: Experiment 1—Result quality evolution of the SSB and CCI aggregation variants over time.

5.2 Experiment 2: Impact of Join Ordering

In this experiment, we study the impact of join ordering on query processing performance and result quality evolution. We run all possible join orders, both with the join order enforced and with the query optimiser allowed to change the processing plan. For comparison, we also report the baseline performance of the "default" join order, given by the browsing state definitions in Table 1.

Figure 7a shows the TTLB for all four browsing states for five join orderings, from left to right: best forced, best optimised, baseline, worst optimised, and worst forced. For the LW and GH browsing states, performance is as expected, with TTLB growing from left to right. For the reading-heavy RH and HW workloads, however, forcing the optimiser to respect the join order leads to worse performance, by depriving the database of some optimisations. The difference between the best and worst optimised join orderings is less than 20%, however, which indicates that browsing state query performance is relatively insensitive to join orderings. Note that, in all cases, TTFB is not impacted by the join ordering.

Turning to result quality, Figures 7b and 7c show the corresponding evolution of result quality for the reading-heavy RH and HW browsing states. As the figures show, the convergence curves largely retain their shape independent of the join order applied, which indicates that result quality is also relatively insensitive to join orderings. Overall, these results indicate that it is acceptable to apply the default join order in all cases.

5.3 Experiment 3: Selectivity Estimation

Finally, we examine how well the traditional database approach is able to predict the result size accurately. For this experiment, we examine all 13 browsing states and compare the estimated result size against the actual result size. Figure 7d shows the actual result size on the x-axis and the estimated result size on the y-axis. As the figure shows, the estimate is largely quite accurate; the worst underestimate is by 85% but this is for a very small browsing state, while the largest overestimate is by a factor of 2, and for 9 of the 13 browsing states, the estimate is correct.

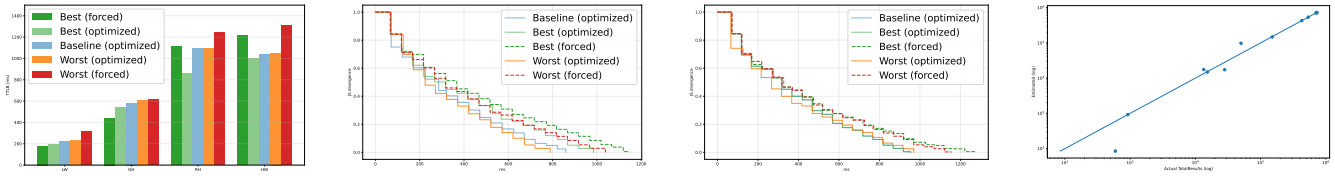
As expected, the underestimation is due to correlation, while the overestimation is due to anti-correlation. With perfect anti-correlation, we can expect a result set of size 0 so the formula above would overestimate result size. With perfect correlation, on the other hand, the smaller constraint would determine the result size and the formula above would underestimate the results size. Since it is only underestimation due to correlation that is a problem, for future work we propose to (a) always expect filters on two different semantic concepts to be correlated, and (b) run statistical processes on combinations of other tagsets to detect correlation, and take both kinds of correlation into account when estimating result sizes.

6 Conclusion

Inspired by the online aggregation approach from the database community, we investigate whether progressively refined intermediate results, accompanied by quality estimates, can enable users to decide whether to accept partial results or continue processing. We evaluate this approach within the M^3 model, where media collections are explored through metadata tagsets and hierarchies. The main contribution of our study is an analysis of aggregation operator placement, which shows that moving deduplication and grouping from the database to the server reduces time-to-first-result by over 90% while maintaining steady quality improvement. We further examine join ordering and selectivity estimation for reliable progress prediction. Our results demonstrate that online aggregation principles can substantially improve responsiveness in multimedia exploration systems.

Acknowledgments

This work was partially supported by the Icelandic Research Fund grant 239772-051.



(a) Query Processing Performance (b) Reading-Heavy (c) HeavyWeight (d) Selectivity Estimation

Figure 7: (a) Experiment 2—Query processing performance; (b-c) Experiment 2–Result quality evolution over time; and (d) Experiment 3—Evaluation of selectivity estimation.

References

[1] Kai Uwe Barthel, Nico Hezel, Konstantin Schall, and Klaus Jung. 2023. Navigu.Net: NAvigation in Visual Image Graphs Gets User-Friendly. In *Proceedings of the 2023 ACM International Conference on Multimedia Retrieval (ICMR '23)*. ACM, Thessaloniki, Greece, 654–658. doi:10.1145/3591106.3592248

[2] Ilaria Bartolini and Paolo Ciaccia. 2008. Scenique: a multimodal image retrieval interface. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI)*. ACM, Napoli, Italy, 476–477. doi:10.1145/1385569.1385664

[3] Benjamin B. Bederson. 2001. PhotoMesa: a zoomable image browser using quantum treemaps and bubblemaps. In *Proceedings of the Annual ACM Symposium on User Interface Software and Technology (UIST)*. ACM, Orlando, FL, USA, 71–80. doi:10.1145/502348.502359

[4] Surajit Chaudhuri. 1998. An Overview of Query Optimization in Relational Systems. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*. ACM, Seattle, WA, USA, 34–43. doi:10.1145/275487.275492

[5] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M Hellerstein, John Gerth, Justin Talbot, Khaled Elmeleegy, and Russell Sears. 2010. Online Aggregation and Continuous Query support in MapReduce. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. ACM, New York, NY, USA, 1115–1118. doi:10.1145/1807167.1807295

[6] Jens-Peter Dittrich, Bernhard Seeger, David Scot Taylor, and Peter Widmayer. 2002. Progressive Merge Join: A Generic and Non-blocking Sort-based Join Algorithm. In *Proceedings of the International Conference on Very Large Databases (VLDB)*. Elsevier, Morgan Kaufmann, Hong Kong, 299–310. doi:10.1016/B978-155860869-6/50034-2

[7] Aaron Duane and Björn Þór Jónsson. 2022. ViRMA: Virtual Reality Multimedia Analytics. In *Proceedings of the International Conference on Multimedia Retrieval (ICMR)*. ACM, Newark, NJ, USA, 211–214. doi:10.1145/3512527.3531352

[8] Cathal Gurrin, Liting Zhou, Graham Healy, Werner Bailer, Duc-Tien Dang-Nguyen, Steve Hodges, Björn Þór Jónsson, Jakub Lokoc, Luca Rossetto, Minh-Triet Tran, and Klaus Schöffmann. 2024. Introduction to the Seventh Annual Lifelog Search Challenge, LSC'24. In *Proceedings of the International Conference on Multimedia Retrieval (ICMR)*. ACM, Phuket, Thailand, 1334–1335. doi:10.1145/3652583.3658891

[9] Peter J. Haas and Joseph M. Hellerstein. 1999. Ripple Joins for Online Aggregation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. ACM, Philadelphia, PA, USA, 287–298. doi:10.1145/304182.304208

[10] Benjamin Heintz, Abhishek Chandra, and Ramesh K. Sitaraman. 2015. Optimizing Grouped Aggregation in Geo-Distributed Streaming Analytics. In *Proceedings of the International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*. ACM, Portland, OR, USA, 133–144. doi:10.1145/2749246.2749276

[11] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. 1997. Online Aggregation. In *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD)*. ACM, Tucson, AZ, USA, 171–182. doi:10.1145/253260.253291

[12] Yannis E. Ioannidis and Stavros Christodoulakis. 1991. On the Propagation of Errors in the Size of Join Results. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. ACM, Denver, CO, USA, 268–277. doi:10.1145/115790.115835

[13] Björn Þór Jónsson, Aaron Duane, and Nikolaj Mertz. 2022. Relational Database Performance for Multimedia: A Case Study. In *Proceedings of the International Conference on Content-based Multimedia Indexing (CBMI)*. ACM, Graz, Austria, 186–190. doi:10.1145/3549555.3549558

[14] Björn Þór Jónsson, Grímur Tómasson, Hlynur Sigurþórsson, Áslaug Eiríksdóttir, Laurent Amsaleg, and Marta Kristín Lárusdóttir. 2015. A Multi-Dimensional Data Model for Personal Photo Browsing. In *Proceedings of the International Conference on MultiMedia Modeling (MMM)*. Springer, Sydney, NSW, Australia, 345–356. doi:10.1007/978-3-319-14442-9_41

[15] Klaus Jung, Kai Uwe Barthel, Nico Hezel, and Konstantin Schall. 2022. PicArrange - Visually Sort, Search, and Explore Private Images on a Mac Computer. In *Proceedings of the International Conference on MultiMedia Modeling (MMM)*. Springer, Phu Quoc, Vietnam, 452–457. doi:10.1007/978-3-030-98355-0_38

[16] Omar Shahbaz Khan, Ujjwal Sharma, Gonçalo Marcelino, Stevan Rudinac, and Björn Þór Jónsson. 2026. Exquisitor at the Video Browser Showdown 2026: Temporal Queries Revisited. In *Proceedings of the International Conference on MultiMedia Modeling (MMM)*. Springer, Prague, Czech Republic, 245–251. doi:10.1007/978-981-95-6963-2_27

[17] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Thomas Neumann, and Alfons Kemper. 2015. How Good Are Query Optimizers, Really? *Proceedings of the VLDB Endowment* 9, 3 (2015), 204–215. doi:10.14778/2850583.2850594

[18] Jianhua Lin. 1991. Divergence Measures Based on the Shannon Entropy. *IEEE Transactions on Information Theory* 37, 1 (1991), 145–151. doi:10.1109/18.61115

[19] Mohamed F Mokbel, Ming Lu, and Walid G. Aref. 2004. Hash-Merge Join: A Non-blocking Join Algorithm for Producing Fast and Early Join Results. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE, Boston, MA, USA, 251–262. doi:10.1109/ICDE.2004.1320002

[20] Ork Rooij, Jarke van Wijk, and Marcel Worring. 2010. MediaTable: Interactive Categorization of Multimedia Collections. *IEEE Computer Graphics and Applications* 30, 5 (2010), 42–51. doi:10.1109/MCG.2010.66

[21] Luca Rossetto and Florian Ruosch. 2025. MeGraS: An Open-Source Store for Multimodal Knowledge Graphs. In *Proceedings of the ACM International Conference on Multimedia*. ACM, Dublin, Ireland, 13644–13647. doi:10.1145/3746027.3756872

[22] Nikhil Sheoran, Supawit Chockchowwat, Arav Chheda, Suwen Wang, Riya Verma, and Yongjoo Park. 2023. A Step Toward Deep Online Aggregation. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–28. doi:10.1145/3589269

[23] Marcel Worring, Cees Snoek, Ork de Rooij, Giang Nguyen, Richard van Balen, and Dennis Koelma. 2006. Mediamill: Advanced Browsing in News Video Archives. In *Proceedings of the International Conference on Image and Video Retrieval (CIVR)*. Springer, Tempe, AZ, USA, 533–536. doi:10.1007/11788034_62

[24] Jan Zahálka and Marcel Worring. 2014. Towards Interactive, Intelligent, and Integrated Multimedia analytics. In *Proceedings of IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, Paris, France, 3–12. doi:10.1109/VAST.2014.7042476