MUSICDB: A PLATFORM FOR LONGITUDINAL MUSIC ANALYTICS

Jeremy Hyrkas
University of Washington
hyrkas@cs.washington.edu

Bill Howe
University of Washington
billhowe@cs.washington.edu

ABSTRACT

With public data sources such as Million Song dataset, researchers can now study longitudinal questions about the patterns of popular music, but the scale and complexity of the data complicate analysis. We propose MusicDB, a new approach for longitudinal music analytics that adapts techniques from relational databases to the music setting. By representing song timeseries data relationally, we aim to dramatically decrease the programming effort required for complex analytics while significantly improving scalability. We show how our platform can improve performance by reducing the amount of data accessed for many common analytics tasks, and how such tasks can be implemented quickly in relational languages — variants of SQL. We further show that expressing music analytics tasks over relational representations allows the system to automatically parallelize and optimize the resulting programs to improve performance. We evaluate our system by expressing complex analytics tasks including calculating song density and beat-aligning features and showing significant performance improvements over previous results. Finally, we evaluate expressiveness by reproducing the results from a recent analysis of longitudinal music trends using the Million Song dataset.

1. INTRODUCTION

Over the past decade, a concerted investment in building systems to help extract knowledge from large, noisy, and heterogeneous datasets — big data — has had a transformative effect on nearly every field of science and industry. Progress has also been fueled by the availability of public datasets (e.g., the Netflix challenge [1], early releases of Twitter data [14], Google's syntatic n-grams data [7], etc.), which have focused and accelerated research in both domain science and systems. The field of Music Information Retrieval (MIR) appears to been less affected, as complications from copyright-encumbered properties have limited the introduction of big datasets to the community. Now, however, such datasets are finally making their way into the field.

In other fields we have observed that as the data size and scope of problems increased, issues of scale became the bottleneck: single-site solutions written in R or python give way to distributed shared-nothing systems that can readily handle large datasets. These systems relieve the user of worrying about issues such as memory management, concurrency and distributed computing by focusing on limited data models and APIs. General purpose systems such as Hadoop [21] and Spark [22] provide MapReduce-style dataflow computations [4], but can be hard to program and optimize because of their generality and relatively low-level interfaces. Increasingly, programmers are experimenting with the models and languages of relational databases [11,12,17] for non-relational analytics over timeseries, graphs, images, and more due to their higher-level programming abstractions, simpler data models, and automatic optimization.

In this paper, we propose a platform for *longitudinal music analytics* built on a relational big data system. Because music data (which may include multi-dimensional arrays and timeseries of extracted features) is ostensibly not relational upon collection, we describe a "relationalization" of the data to afford distributed, parallel processing. Then, we present four algorithms found in music analytics tasks in the MIR literature and show that they can be expressed in declarative relational languages similar to SQL, affording scalability, portability, and automatic optimization, and freeing the programmer from systematic concerns related to memory management, concurrency, and distributed processing. The four algorithms are song density, feature beat-alignment, pitch keyword distribution by year, and timbre keyword distribution by year. The first two algorithms are common music analysis tasks, and the latter two are highly computational algorithms presented in a high profile MIR study over the Million Song Data set [20]. We reproduce prior results with only a few lines of code and a significant performance improvement over prior experiments on similar large-scale systems.

We propose our model as an approach for platforms to raise the level of abstraction for longitudinal music analytics and reduce the barrier to entry for researchers in musicology and sociology.

1.1 Million Song Dataset

The key dataset used in our experiments is the The Million Song Dataset (MSD) [3]. The dataset includes metadata and extracted features from one million pop music songs from a period of decades, including information such as genre tags, chroma measurements, timbre and loudness

measurements, detected beats, artist and song metadata such as year, duration and location, and many other attributes. This dataset has been highly influential in the MIR community, leading to the Million Song Dataset Challenge [16], as well as being a key data set used to study music history [20] and MIR tasks such as cover song detection [2, 10]. The MSD is available as available as a large collection of HDF5 [5] files; at over 250GB even when heavily compressed, it is the largest public dataset in MIR and is the first true "Big Data" dataset in the field.

The MSD is available in a number of formats, including a relational database. However, the database representation include the metadata only, and cannot be used for the content-based longitudinal analytics tasks we aim to support with MusicDB.

2. RELATED WORK

Serra et al. provide a longitudinal analysis of music trends in popular music by utilizing the MSD [20] . The authors study the chroma, timbre, and loudness components of the MSD and find that the frequency of chroma keywords (which can be thought of as single notes, chords, etc) fit a power law distribution which is mostly invariant across time. However, the authors also find that transitions from one keyword to the next have become more uniform over time, suggesting less complexity in newer music. They also describe shifting trends in timbre and loudness, including numerical evidence that recorded music is increasingly louder on average. In Section 4 we describe these tasks in detail and present new algorithms for them using MusicDB. In Section 5 we evaluate our approach experimentally.

Raffek and Ellis analyzed MIDI files and matched them to corresponding songs in the MSD [18] . Their algorithm is fast but is not distributed; they estimate that running their approach on roughly 140k MIDI files against the MSD would take multiple weeks even when parallelized on their multi-threaded processor with 12 threads.

Bertin-Mahieux and Ellis described a method for finding cover songs in the MSD [2]. The method begins with an aggressive filtering step, which requires computing jumpcodes for the entire MSD and storing them in a SQLite database. Once the number of potential matches for a new song is filtered, a more accurate matching process is run to find cover songs. Using three cores, they computed the jumpcodes for the entire MSD in roughly three days, although matching cover songs once the jumpcodes are computed takes roughly a second per new song. The authors also mention that the jumpcodes had to be stored in many different SQLite tables, as they were unable to index roughly 1.5M codes in a single table. MusicDB provides a platform that can process the data directly, in parallel, without specialized engineering.

Humphrey, Nieto, and Bello also provide a method to detect cover songs [10]. Their method starts by transforming beat-aligned chroma of a song into a high-dimensional, sparse representation by projecting its 2D Fourier Magnitude Coefficients. They then use PCA to reduce dimensionality and use the results to find cover songs using a distance function. The authors claim that using ten threads on a machine with "plenty of RAM", various methods can take between 3-8 hours to complete this computation on the MSD.

Hauger et al. describe the million musical tweets dataset (MMTD) [9] collected from tweets with information about a user's location and what they were listening to at a certain time. This dataset, as well as others, can be used to augment the MSD for new MIR tasks. As it exists, the MSD is available as a directory hierarchy with hundreds of gigabytes of HDF5 files stored on AWS. Incorporating new data in analysis tasks over this dataset requires additional effort in the analysis pipeline, leaving either the authors of the data or the users of the data to write new code to handle the new data source and manually join it with the MSD. MusicDB provides a scalable substrate for such integration tasks.

3. DATA MODEL

A key step in efficiently analyzing the MSD is to represent its information in an appropriate data model. The representation of the MSD available on the website (on million HDF5 files) support efficient lookup by ID, but any more complex processing requires custom programs to be written, and parallelization, concurrency, distribution, and memory management are all the direct responsibility of the programmer. Moreover, tasks that require only a portion of metadata from each song must still access and load all song data from disk.

Instead, we can organize the music data as sets of records. In practice, this "relationalization" of timeseries and multidimensional data can significantly increase the size of the dataset. In our work, the end size of our relationalized data is roughly 500GB, about twice as large as the original dataset. However, all applications we have observed do not require the entire MSD, and the subset of relationalized data necessary for computation is much smaller than the entire MSD in HDF5 format. Further, representing the data as a set of records affords automatic partitioning and parallel processing, as we will see.

The steps to relationalization are as follows:

- Metadata that appears only once per song is inserted into one table (songs), with song ID as the key. This includes fields such as song duration, artist name, song name, etc.

- Nested fields are represented in separate tables, retaining a foreign key to the songs table. To represent the order within the nested field, an additiona column is added. For example, each song segment is represented as a record *(song_id, segment_number, value*, where *segment_number* explicitly encodes the implicit order in the original array. This additional field is one source of the space overhead we find in practice.

- We use additional tables to support specific components of the MSD, including a separate table for each of the following: beat-aligned chroma features, beat-aligned timbre features, and beat-aligned chroma features that have been transposed such that most songs are in C major or C minor.

table	key	arity	non-key fields
songs	song_id	33	duration, key, tempo, etc
segments	song_id, seg_num	31	timbre, loudness, and pitch measurements
mbtags	song_id, tag	3	tag count
bars	song_id, bar_num	4	bar start and confidence
beats	song_id, beat_num	4	beat start and confidence
sections	song_id, section_id	4	section start and confidence
terms	song_id, term	4	term frequency and weight
tatums	song_id, tatum_num	4	tatum start and confidence
similar artists	song_id, artist_id, similar_artist_id	3	N/A

Table 1. Core tables after relationalization of the MSD. Additional tables may be created, such as beat aligned features.

4. ALGORITHMS

We describe four algorithms for scalable analysis of the relationalized MSD dataset. The first two algorithms are common MIR tasks, and the latter two come from an influential study using the MSD [20].

4.1 Song Density

In 2011, Lamere described a Hadoop-based approach for calculating song density from the MSD [15]. A song's density is defined as the number of detected segments divided by the duration of the song in seconds. To calculate this metric for every song in the MSD, Lamere provides a MapReduce [4] algorithm that scans each song, extracts the segments, and computes the density. The map function was written in Java specifically for this purpose.

This task can be expressed directly with no custom general purpose code in SQL. In MusicDB, song density for a single song can be expressed as a simple count query over the segments table, followed by a join with the songs table and division by song duration. This method generalizes to the following query (in an imperative dialect of SQL used by the Myria system [8]) that computes the density for all songs.

Query 1. Lines 1-2 scan the relevant relations. Lines 4-7 count the number of segments per song and lines 8-14 calculate the density by dividing the number of segments by the duration in seconds. Line 15 stores the result.

```
1  segments = SCAN(SegmentsTable);
2  songs = SCAN(SongsTable);
3  -- implicit GROUP BY song_id
4  seg_count = SELECT
5      song_id,
6      COUNT(segment_number) AS c
7      FROM segments;
8  density = SELECT
9      songs.song_id,
10     (seg_count.c /
11         songs.duration) AS density
12     FROM songs, seg_count
13     WHERE songs.song_id =
14         seg_count.song_id;
15 store(song_density);
```

4.2 Beat-aligning features

While chroma and timbre data in the MSD are provided on a per-segment basis, it is often more useful to align these features to beats, which are easier to interpret musically. Beat-aligning these features is an extremely common and useful processing step that is used in cover song detection [2], longitudinal music studies [20], and many other MIR tasks, and is therefore a useful task to consider for MusicDB.

In 2011, Serra identifies dynamic time warping as one of the best methods for beat-alignment [19]. Dynamic time warping involves creating an SxB matrix, where S is the number of segments of a song and B is the number of beats. If a segment s overlaps with a beat b, the b, s entry of the matrix is set to the fraction of the segment contained in the beat (i.e. 1 if the segment falls entirely in the beat, .5 if the beat contains exactly half of the segment, etc). All other entries are set to 0, and then the rows are normalized such that each row of values sums to 1. Segment-based features such as chroma or timbre can then be beat-aligned by transposing the beat matrix and performing matrix multiplication on the features (a BxS matrix multiplied by a SxF matrix will result in a BxF matrix, where F is the number of features). Some additional regularization of rows is performed for chroma features.

In a relational system, the time warp operation can be computed using just two operations: a join and an aggregation. We first join the segments and beats table on the start and end time of each segment and bar, such that overlapping segments and beats are joined:

Query 2. Portion of a query that joins overlapping segments and beats of a song so that beat aligned features can be computed.

```
1  JOIN segments, beats WHERE
2      -- segment overlaps start of beat
3      (seg_start <= beat_start
4          AND seg_end <= beat_start)
```

```
    OR
    -- segment overlaps end of beat
    (seg_start < beat_end
        AND seg_end >= beat_end)
    OR
    -- segment fully inside beat
    (seg_start > beat_start
        AND seg_end < beat_end)
    OR
    -- beat fully inside segment
    (seg_start < beat_start
        AND seg_end > beat_end)
;
```

We can then perform two aggregations on the result, and re-join the aggregate queries to perform an operation identical to multiplying the features by a time warp matrix. Refer to Figure 1, which visualizes this query. In each aggregation, we will calculate the fraction of a segment that falls within a beat, which is defined as the length of the segment that falls within the beat divided by the length of the segment (or 1.0 if the segment spans the beat).

On the right side of the diagram, we compute the first aggregation. We use the fraction of the segment that falls within a beat to scale each feature of that segment (i.e. chroma or timbre features), and then take the sum of the scaled features per beat.

On the left side of the diagram, we perform a second aggregate which simply computes the sum of the segment fractions for each beat. We call this sum the divisor.

The two aggregates are then re-joined on beat id and the weighted sum from the first aggregate is normalized by dividing each value by the sum from the second aggregate. This divisor serves the same function as making sure rows of the time warp matrix sum to 1. The result of the joined aggregates is a table with the schema (song_id, beat_id, feature columns), which is a relational representation of the BxF feature matrix described above.

This algorithm, while correct, is not necessarily optimal. Specifically, performing two aggregates over the same joined relation and then re-joining is an expensive operation. Relational engines that support *window functions* offer an alternative approach. A window function makes a single pass over a dataset, applying an aggregation over each window as defined by a grouping value or a fixed size. The engine on which MusicDB is based (a variant of the Myria system [8]) provides a generalization of window functions, but we do not employ that mechanism here to ensure reuse across platforms.

4.3 Pitch Keyword Distribution by Year

In 2012, Serra et al studied the progression of *chroma keywords* over time [20]. The authors form these keywords by transposing every song to an equivalent main tonality by correlating to tonal profiles provided by [13]. After that, the beat-aligned chroma values are discretized to binary values (1 if the value is greater than 0.5, 0 otherwise) and then concatenated. Intuitively, this discretization represents whether or not a certain pitch is present or not. These

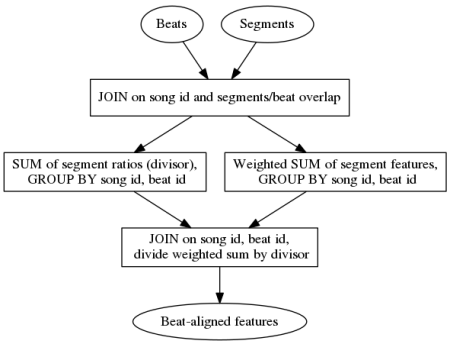

Figure 1. The relational algebra expression for beat-aligning features from segments. Segments and beats from a song are joined on song ID and overlap conditions. Two aggregations are performed and rejoined to form a table with features now aligned to beats instead of segments. The process of generated two results and joining them can be costly and can be aided by using window functions provided in many database systems.

keywords can then be summed over years and used to fit a distribution. The authors show that these chroma keywords fit a power law which has variables that are near invariant over time.

The discretization and sum of keywords can be implemented using our data model by the following SQL-like program:

Query 3. Lines 1-2 scan the appropriate tables. Lines 4-19 (with some lines emitted) create an integer keyword based on the value of each pitch column. Lines 21-26 count keywords by year and line 28 stores the result.

```
songs = SCAN(SongsTable);
pitch = SCAN(PitchTransposed);

keywords = SELECT
    p.song_id AS song_id,
    p.beat_number AS beat_number,
    CASE WHEN p.basis0 >= 0.5
        THEN int(pow(2, 11))
        ELSE 0
    END
    +
    CASE WHEN p.basis1 >= 0.5
        THEN int(pow(2, 10))
        ELSE 0
    END
    +
    ...
    AS keyword
    FROM pitch p;

-- implicit GROUP BY year, keyword
yearPitchKeywords = SELECT
    s.year, k.keyword,
    count(k.keyword)
    FROM songs s, keywords k
```

```
26      WHERE s.song_id = k.song_id;
27
28  store(yearPitchKeywords);
```

4.4 Timbre Keyword Distribution by Year

Similar to the pitch keywords in Section 4.3, the timbre values from the MSD can be discretized and studied over time. Serra et al sample timbre values from by year such that no year is more represented from than any other [20]. From this sample set, they estimate the tertiles for each timbre value. The tertile values are then used to discretise the timbre values similarly to the pitch keywords. For each timbre column, the value is converted to a 0 if it is less than the first tertile, 1 if it is less than the second tertile, and 2 otherwise; concatenating these values makes one timbre keyword.

The authors fit power laws to the timbre keywords as before. However, they find that the power law distributions significantly differ over time. They conclude that while the chroma distribution appears to be time invariant, timbre information (which encodes many complex factors such as instrument use, tone, and production style) has changed over time; additionally, the authors find that while there are local shifts in timbre values, the distribution is slowly converging.

The query for finding timbre keywords is similar to the query in Section 4.3, but slightly more complex. It requires access to a quantiles function that takes a quantile constant and a column, and returns an integer representing which quantile a row's column value falls in (for example, quantile(3, col) returns 0 if col is in the first tertile of the values contained in col, 1 if it in the second, and 2 if it is in the third).

Query 4. Lines 1-2 scan the appropriate tables. Lines 4-15 (with some lines emitted) create an integer keyword based on the tertile each timbre column falls in (the function returns 0 through 2). Lines 17-22 count keywords by year and line 24 stores the result.

```
1   songs = SCAN(SongsTable);
2   timbre = SCAN(TimbreBeatAligned);
3
4   keywords = SELECT
5       t.song_id as song_id,
6       t.beat_number AS beat_number,
7       int(pow(10, 11)) *
8           QUANTILE(3, t.basis1)
9       +
10      int(pow(10, 10)) *
11          QUANTILE(3, t.basis2)
12      +
13      ...
14      AS keyword
15      FROM timbre t;
16
17  -- implicit GROUP BY year, keyword
18  yearTimbreKeywords = SELECT
19      s.year, k.keyword,
20      count(k.keyword)
21      FROM songs s, keywords k
22      WHERE s.song_id = k.song_id;
23
24  store(yearTimbreKeywords);
```

5. EXPERIMENTAL EVALUATION

We evaluate the feasibility of our relationalized approach by measuring the wall-clock performance of our implementation on a 72-worker cluster and comparing performance qualitatively with reports from the literature. We find that the entire MSD dataset can be analyzed in seconds or minutes, where previous results on large-scale platforms report tens of minutes and required custom code, while smaller-scale implementations reported taking hours or days.

5.1 Song Density

We ran the song density query described in Section 4.1 on a MusicDB cluster with 72 worker threads. The computation takes roughly half a minute, a far cry from the 20 minutes described in [15]. These two results are not directly comparable; the example in [15] was run on virtual machines in EC2, so the hardware, software, number of nodes, and most other factors are not comparable. However, the Hadoop implementation required custom code, and the underlying platform on which we implemented these algorithms (a variant of the Myria system [8]) has been previously shown to significantly outperform Hadoop on general tasks.

By using a relational model to represent the MSD and using a distributed analytics database, we can quickly analyze the MSD using a simple query and allowing the system and optimizer handle the complexities of computation.

5.2 Pitch Keyword Distribution by Year

We ran the query described in Section 4.3 on our 72-node production cluster of MusicDB. Computing the pitch keywords took about three minutes, while counting keywords by year took an additional minute. The resulting dataset contains the frequency for each keyword per year, and has the schema (Keyword, Year, Count) with (Keyword, Year) as the primary key. It is small enough ($<$ 1GB) to download locally and perform more complicated statistical tasks, such as fitting power law distributions over counts per year as in [20].

Figures 2, 3, and 4 show the power law distributions for pitch keywords in the years 1965, 1975, and 1985. We confirm the author's results [20] that the power law coefficient is invariant over time. We used the R package poweRlaw [6] to perform this post-processing analysis.

The database system we used does not have complex functions such as power law fitting, so the last step of the computation must be performed out of the system. However, this step could be run in parallel as the data for each year is independent. Alternatively, in a more general distributed system, the keyword counts for each year could be

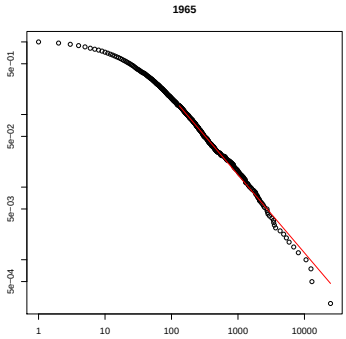

Figure 2. Power law distribution for pitch keywords from songs released in 1965.

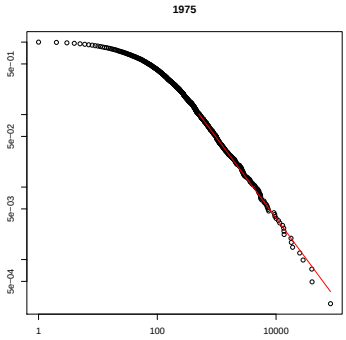

Figure 3. Power law distribution for pitch keywords from songs released in 1975.

partitioned and evaluated in a distributed manner. We did not demonstrate this capability in this case to avoid a contrivance; the resulting data's size was not large enough to justify the approach.

6. DISCUSSION AND FUTURE WORK

Distributed analytics systems have made it easier and faster to perform complex analysis on large datasets. In Section 2 we briefly mentioned several recent studies using the MSD. The authors of these studies ran experiments that ran on single-node systems, often taking hours or days to complete. However, most of these tasks are embarrassingly parallel and could be run not only in parallel on a single machine, but on thousands of nodes in a distributed system. Big data systems exist to empower users to easily analyze data in such a distributed environment. As more large dataset become available in the MIR community, it is no longer feasible or necessary to run single or mutli-core algorithms locally for weeks at a time.

We have shown that representing the data in the MSD as tables can reduce the amount of data necessary for computations (for example, only reading the segment and song tables in Section 3, and only the segment and beat tables in Section 4.2). This works especially well in a relational system, where joining tables is a common task with many optimizations. However, reading and distributed less data is helpful outside of relational systems as well. Even though we showed that many common MIR tasks can be expressed relationally, some tasks are still very difficult to implement in an imperative language. If a distributed system such as Hadoop or Spark is more preferable for a given task, relationalizing the data can still be used to reduce the data necessary for computation in these systems. Finally, since these systems utilize higher level coding models that abstract away parallelization and distributed computation, they may empower musicologists who are less familiar with these concepts to ask quickly questions over larger data sets.

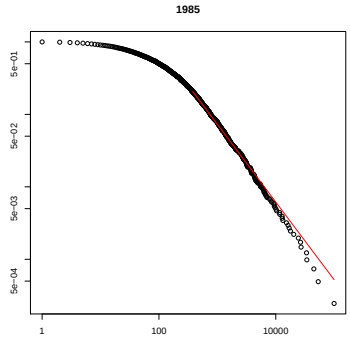

Figure 4. Power law distribution for pitch keywords from songs released in 1985.

7. REFERENCES

[1] James Bennett and Stan Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35, 2007.

[2] Thierry Bertin-Mahieux and Daniel PW Ellis. Large-scale cover song recognition using hashed chroma landmarks. In *Applications of Signal Processing to Audio and Acoustics (WASPAA), 2011 IEEE Workshop on*, pages 117–120. IEEE, 2011.

[3] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[4] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[5] Mike Folk, Albert Cheng, and Kim Yates. Hdf5: A file format and i/o library for high performance computing applications. In *Proceedings of Supercomputing*, volume 99, pages 5–33, 1999.

[6] Colin S. Gillespie. Fitting heavy tailed distributions: The poweRlaw package. *Journal of Statistical Software*, 64(2):1–16, 2015.

[7] Yoav Goldberg and Jon Orwant. A dataset of syntactic-ngrams over time from a very large corpus of english books. 2013.

[8] Daniel Halperin, Victor Teixeira de Almeida, Lee Lee Choo, Shumo Chu, Paraschos Koutris, Dominik Moritz, Jennifer Ortiz, Vaspol Ruamviboonsuk, Jingjing Wang, Andrew Whitaker, et al. Demonstration of the myria big data management service. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 881–884. ACM, 2014.

[9] David Hauger, Markus Schedl, Andrej Košir, and Marko Tkalčič. The million musical tweets dataset: What can we learn from microblogs. In *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR 2013)*, 2013.

[10] Eric J Humphrey, Oriol Nieto, and Juan Pablo Bello. Data driven and discriminative projections for large-scale cover song identification. In *ISMIR*, pages 149–154, 2013.

[11] Marcel Kornacker and Justin Erickson. Cloudera impala: Real time queries in apache hadoop, for real. *ht tp://blog. cloudera. com/blog/2012/10/cloudera-impala-real-time-queries-in-apache-hadoop-for-real*, 2012.

[12] Tim Kraska, Ameet Talwalkar, John C Duchi, Rean Griffith, Michael J Franklin, and Michael I Jordan. Mlbase: A distributed machine-learning system. In *CIDR*, 2013.

[13] Carol L Krumhansl. *Cognitive foundations of musical pitch*, volume 17. Oxford University Press New York, 1990.

[14] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, pages 591–600. ACM, 2010.

[15] Paul Lamere. How to process a million songs in 20 minutes. http://musicmachinery.com/2011/09/04/how-to-process-a-million-songs-in-20-minutes/, 2011.

[16] Brian McFee, Thierry Bertin-Mahieux, Daniel PW Ellis, and Gert RG Lanckriet. The million song dataset challenge. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 909–916. ACM, 2012.

[17] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: interactive analysis of web-scale datasets. *Proceedings of the VLDB Endowment*, 3(1-2):330–339, 2010.

[18] Colin Raffel and Daniel PW Ellis. Large-scale content-based matching of midi and audio files. In *16th International Society for Music Information Retrieval Conference (ISMIR 2015)*.

[19] Joan Serra. Identification of versions of the same musical composition by processing audio descriptions. *Department of Information and Communication Technologies*, 2011.

[20] Joan Serrà, Álvaro Corral, Marián Boguñá, Martín Haro, and Josep Ll Arcos. Measuring the evolution of contemporary western popular music. *Scientific reports*, 2, 2012.

[21] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

[22] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, volume 10, page 10, 2010.