Managing PMU Data Sets with Bitmap Indexes

Ben McCamish
School of Engineering and Computer Science
Washington State University
Vancouver, WA 98686. USA

David Chiu
Department of Mathematics and Computer Science
University of Puget Sound
Tacoma, WA 98416. USA

Miles Histand
Jordan Landford
Robert B. Bass
Department of Electrical and Computer Engineering
Portland State University
Portland, OR 97201. USA

Rich Meier
Eduardo Cotilla-Sanchez
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97331. USA

***Abstract*—Large databases and data warehouses are becoming prevalent for the storage and management of energy data. Accelerating the rates at which data can be retrieved is beneficial not only to allow for more efficient search of the data, but also to be integrated with other energy system tools. In this paper, a fast indexing and data retrieval method, known commonly as a *bitmap index*, is created to facilitate intelligent querying and indexing of data generated by phasor measurement units (PMU) at a rate of 60 Hz. We find that bitmaps are amenable to managing efficient access to large amounts of PMU data. Furthermore, the bitmap-management process will provide decreased access time for data retrieval as well as decreased memory usage. From our experiments, our system is able to achieve approximately 30 times speedup on queries resulting in tuples that are in the database. Conversely, reducing the time taken to answer queries that result in tuples not contained in the database to nearly instantaneous.**

***Keywords*—PMU, synchrophasor, data management, data warehousing, bitmap indexing**

I. Introduction

To support real-time situational awareness, power utilities are deploying phasor measurement units (PMU)[1] over the grid. At a high-level, PMUs are sensing devices that measure electrical waveforms at short fixed intervals [1]. A unique feature of PMUs is that they are equipped with global positioning systems (GPS), allowing multiple PMUs in space to be synchronized across time. Their mass deployment can offer utility operators a holistic and live sense of the grid's health and status.

Today's PMUs have become extremely sophisticated, generating up to 60 measurements per second. Each PMU's data stream is collected and coalesced by a device known as a phasor data concentrator (PDC) before finally being written to large, but slow, non-volatile storage, *e.g.*, hard disks. When data streams from many PMUs are combined, it can amount to massive volumes of data each year. However, common data processing tasks, such as *ad hoc* querying, retrieval for analysis, and visualization may require scanning or randomly accessing large amounts of PMU data over on disk, which would take a prohibitive amount of time. We must therefore leverage fast storage and retrieval mechanisms to accelerate data access times.

One data structure, known as a *bitmap index* [2], has become popular for managing large amounts of data in the context of scientific applications [3]–[5], network traffic monitoring [6], and data warehousing [7], [8]. A bitmap B is an $m \times n$ matrix where the n columns represent range-bins, and the rows correspond to the m records (*e.g.*, PMU measurements). A bit $b_{i,j} = 1$, if the ith record falls into the specified value/range of the jth bin, and $b_{i,j} = 0$, otherwise.

Records	**Bins**						
	X				Y		
	x_1	x_2	...	x_{50}	y_1	y_2	y_3
t_1	0	1	...	0	0	0	1
t_2	0	0	...	0	0	1	0
t_3	0	0	...	1	0	0	1
...	...	...	...	...	...	...	...

TABLE I. An Example Bitmap Index

Consider the bitmap in the Table I. Suppose this example data has two attributes, X and Y, and the values of X are known to be integers in the range $(0, 50]$ and that the values of Y can be any real number. Due to its small cardinality, we can generate a bin x_j for each possible value of X. Because the values of Y are continuous and unbounded, we must discretize its values, *i.e.*, decide on an appropriate cardinality of bins to represent Y, and select the range of values associated with each bin. In our example, we chose to

[1]Also known as *synchrophasors*, we refer to them as PMUs throughout this paper.

use only three bins, $y_1 = (-\infty, -5]$, $y_2 = (-5, 5)$, and $y_3 = [5, \infty)$.

Suppose we want to retrieve all records from disk where $X < 25$ and $Y = 0$. We can identify the candidate records by computing the following boolean expression,

$$v_R = (x_1 \vee ... \vee x_{24}) \wedge y_2$$

The bits with a value of 1 in v_R corresponds the set of candidate records on disk,

$$R = \{t \mid (t[X] < 25) \wedge (-5 < t[Y] < 5)\}$$

Intuitively, there could be false positives in R, which requires checking, but only the records $r_i \in R$ with a corresponding bit $v_R[i] = 1$ must be retrieved from disk and examined to ensure they meet the selection criteria. All records r_i with a corresponding bit $v_R[i] = 0$ are *pruned* immediately and do not require retrieval from disk. Because a well-designed bitmap is sparse and compressible, it can be stored in core-memory, which is orders of magnitude faster than disk.

As we can see, bitmaps can help reduce disk accesses when properly discretized, resulting in a space/accuracy tradeoff. More precise pruning may have been possible had we split the attribute Y into even more, finer-grained, bins. However, each additional bin effectively adds an entire dimension, increasing the bitmap index's size, challenging its ability to fit in core-memory. Furthermore, appropriate range assignment of the bins will also affect accuracy.

In this paper, we describe the design and implementation of a bitmap-supported PMU data management system. Specifically, we make the following contributions: (1) We obtained one year of real data collected by the Bonneville Power Administration's network of PMU sites. We characterized the prominent features of this data set to inform discretization decisions for bitmap indexing. (2) Based on the PMU data characterization, we generated a bitmap index over our data set with high selectivity. We integrated this index into our existing bitmap compression and processing framework [9].

The remainder of this paper is organized as follows. Section II describes the Word-Aligned Hybrid code, a popular technique for bitmap compression. An overview of our data management and retrieval system is described in Section III. In Section IV, we present our PMU data characterization results. Preliminary evaluation results are presented in Section V. We discuss future work in Section VI and conclude in Section VII.

II. Background: Word-Aligned Hybrid Code

Several specialized bitmap compression schemes have been introduced that allow for fast query execution directly over the compressed bitmap codes. The Word Aligned Hybrid code (WAH) [10] is one such efficient bitmap compression and querying scheme. In WAH sequences of bits in the bitmap are grouped into *words*, which are defined by the hardware (*e.g.*, $w = 32$ bits). Each size w word contains either a *fill* or *literal* encoding of $w - 1$ bits. A fill-word is used to represent a run of either 1's or 0's. The structure of a fill-word is (*flag-bit*, *fill-bit*, *length*). The flag-bit indicates whether the word is a fill or a literal (1 or 0, respectively). The fill-bit represents the value of the run. The remaining bits represent the number of size $w - 1$ runs of the fill-bit there are. In contrast, a literal-word is encoded with (*flag*, *exact encoding*). The flag-bit $(= 0)$ signifies that the word is a literal. This is then followed by the exact encoding, which is the literal translation of the $(w - 1)$ bits. In this paper, we assume that word size $w = 32$.

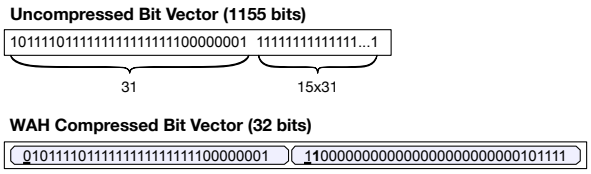

Fig. 1. WAH Compression Example

Fig. 1 illustrates an example of compressing a bit vector using a 32-bit word. The example shown in Fig. 1 shows the compression of a bit vector that is 1155 bits long. Compressing with WAH using 32-bit words gives a compression ratio of approximately 18 times resulting in a bit vector containing only 64 bits.

Logical operations between bit vectors occur by performing a bitwise operator between them. Performing these operations on compressed bit vectors happens at the level of a single word. One important aspect during bitwise operations is the *alignment* of the words, *i.e.*, that each bit is operated on against its respective bit in another bit vector. This is already done through the design of WAH, since each word represents an equal sets of bits, $w - 1 = 31$ in our case, for words of size 32 bits. These words can either contain a single set of 31 bits or many, depending on whether they are literals or fills respectively.

When performing operations between each bit vector, a variable *active word* is created for each bit vector. The active word keeps track of the word that is being operated on in that bit vector, starting with the first. When the bitwise operation is applied to all the encoded bits in the active word, it then moves onto the next word in that bit vector. Three scenarios can occur when performing bitwise operations between two active words:

Fill Word vs. Fill Word: In this scenario, if both active word variables are encoding the same number bits, then the bitwise operation can be performed without extracting any additional sets of bits. That is, for a word of size 32, only that many bits need to be operated

on, allowing for queries to take place without decompression. In the case where the active words do not encode the same number of bits, consider the following. Given two words, X and Y where X encodes more bits than Y, the equal number of bits are operated on, reducing the *length* of X to the difference and exhausting the bits encoded in Y. The active word that represented Y reads the next word while X remains with a reduced run length.

Literal Word vs. Fill Word: With a literal and a fill, a single set is again subtracted from the fill and compared with the literal set of bits encoded in the literal word. Then the active word moves to the next word in the respective bit vector. If there are no more bits encoded in the fill, then the active word moves to the next word in its respective bit vector.

Literal Word vs. Literal Word: The bitwise operations between two literals is performed normally on the encoded bits.

There is a direct correlation between how compressed a bit vector is with how fast queries can be performed. The more compressed a bit vector is, the faster the query will occur. This is because of how operations occur between fill words amortizes the cost of applying the logical operation between two bit vectors.

III. System Design

Given a user query that selects a subset of records from the PMU data archive, the naïve approach to respond to the query would be to perform a linear scan of the database, comparing each record for selection, and then returning the matching records. For a real-time application such as the power operators' situational awareness, this operation would be too expensive because disk I/O operations are slow. Our PMU data management system, depicted in Fig. 2, has multiple software components allowing the user to build a bitmap index over raw data, and to efficiently query records that match specifications.

The *Bitmap Creator* inputs the raw PMU data and generates a bitmap using the binning strategy specified in Section IV. When new files are added to the database, these records will simply be appended onto the index. Once the bitmap is created, the *Compressor* will compress the index using WAH. After compression, the system is ready to receive queries from the user. These queries will give selection conditions on which values of particular attributes the user is interested in. The *Query Engine* then translates the query into boolean operations over the specified bins in the compressed index. This will produce a *Result Bit Vector* v_R, which contains information on which records we need to retrieve from disk.

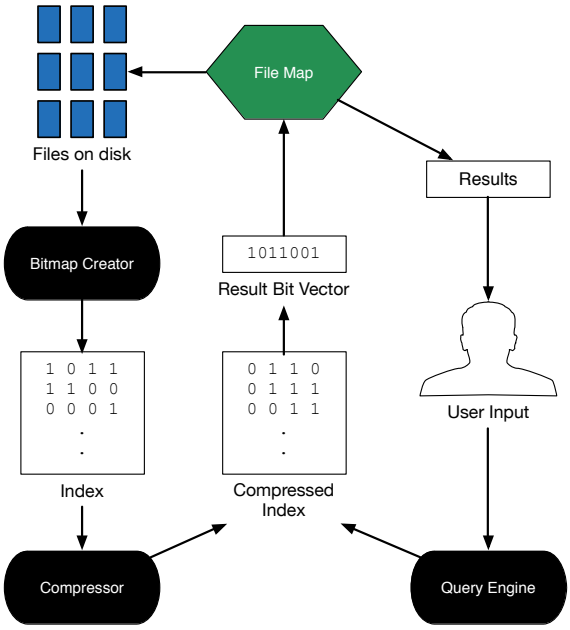

Fig. 2. PMU Data Management System Architecture

While v_R holds the selected record information (all bits with a value of 1), it is the actual data on disk that must be returned. An intermediate data structure, the *File Map*, was created to facilitate this role. The File Map is an intermediate data structure that holds metadata on the files and how many tuples [2] they each contain. There are two values per File Map entry: *totalRowCount* and *filePointer*. The *totalRowCount* contains the total number of tuples up to and including that particular file. The *filePointer* holds a pointer to the corresponding file on disk that contains the next set of tuples. To retrieve files with this method, the result bit vector is first scanned and a *count* is kept for the number of bits that have been read. For each hit, the count is hashed to its corresponding index in the File Map. This is an upper-bound hash, meaning that the count value is hashed to the closest *totalRowCount* value, without being greater than it. This will give the corresponding file that is desired.

Fig. 3 illustrates a small example of a bit vector and where the bits hash to the filemap. Bits one through three are hashed to the first row in the File Map structure. Bits 4 and 5 are hashed to the second row since these bits represent tuples 4 and 5, which are stored in *fileB*. With the upper bound hash, bits 4 and 5 hash to *totalRowCount* 6, since they are both greater than 3 but less than or equal to 6. Bits 60, 62, and 63 are not hashed since they are not hits. Only bits in the bit vector that have value one will be hashed. This leads to improved performance when there are long stretches of zeroes in the bit vector.

[2] An entry (or record) in the database

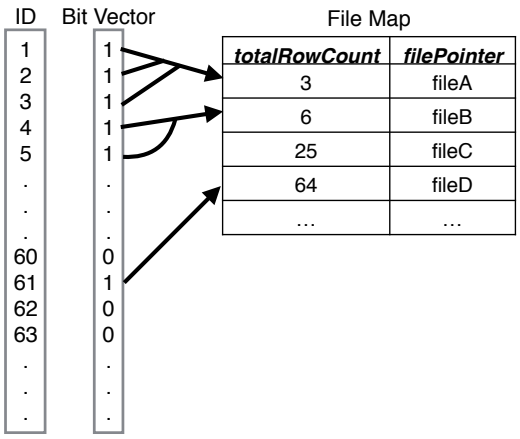

Fig. 3. File Map Structure

IV. Characterizing PMU Data for Bitmap Indexing

We obtained 950 GB of data from a number of PMUs within Bonneville Power Administration's (BPA) operating region from August 2012 to August 2013. At each PMU, a *phasor* measurement is sampled every $1/60$ sec. Each measurement is represented by a *date-time* and a *phasor*, which is a pair of values: the phase angle ϕ and the positive voltage magnitude V. The phasors from 20 PMUs are combined, resulting in 2×20 PMUs $= 40$ attributes. The phase angle ϕ is a time-varying real number that oscillates within the range of $[-180, 180]$. The voltage, on the other hand, is a non-negative real number. In order to define the bitmap ranges, we examined ϕ and V's distributions. We analyzed the distribution of ϕ and V over a sample size of 30 days ($155,520,000$ measurements).

To optimize for speed, the design of the bitmap must be informed by the queries that will be frequently executed. For frequently queried values in bitmap structures, a crippling factor in response time is the candidacy checks to identify true positives, which require disk access. Due to imperfect discretization, bins will often contain bits that indicate more than one value. It is therefore necessary to check whether that bit is an indication of the correct value. For example, if a bin has the range of five possible values then that means each bit in that bin is one of five different values. Performing this check, called a *candidacy check*, ensures that the *tuple* contains the desired value for the query. Choosing the correct binning strategy can therefore potentially improve our query times by reducing candidacy checks among values that were expected to be queried.

From discussions with power systems experts at BPA, queries typically comprise a specific range of dates, voltage V, phase angle ϕ, or any combination of these attributes. When generating the bitmap, the binning (discretization) strategy can minimize candidate record checks and provide fast query response times. Due to the low cardinality of the *date-time* attribute, it was simple to generate bins: 60 bins each for second and minute, 24 bins for hour, 31 bins for day, etc. with the exception of the year. In this case we used 11 bins for the year, starting at 2010. Since there were no range bins, no candidacy checks were necessary when performing queries on the dates. Because ϕ and V are real values, we discretize based on their distribution. In order to find the distributions of both ϕ and V, the cumulative distribution function (CDF) plots were constructed. These distributions determined what binning strategies were used.

Fig. 4 illustrates the phase angle ϕ distribution. From this graph we can see that ϕ follows a uniform distribution. Because ϕ is also bounded, we apply an *equal-width* binning strategy over ϕ, meaning the range of each bin is equivalent. We designed the *bitmap creator* in such a way that this range can be assigned by the user before creation of the bitmap. For our experiments we set this value to 10, leaving 36 bins for each PMU attribute. Fig. 5 represents the phase angle values that were assigned to each bin.

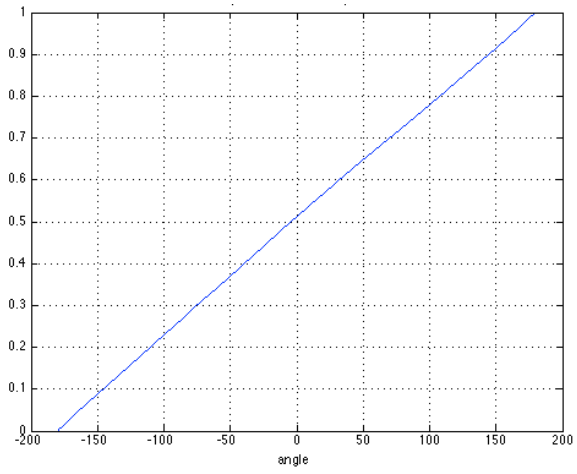

Fig. 4. Normal Phase Angle CDF

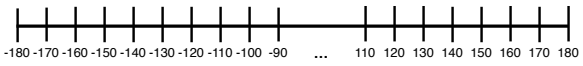

Fig. 5. Phase Angle Bins

Fig. 6 illustrates the distribution for normal operation of a PMU's voltage magnitude. The majority of the values occur between $[535, 545]$. For this attribute, we used a binning strategy which attempts to minimize candidacy checks for the values that are most likely to be queried. We assume the majority of queries from the

user will pertain to some anomaly, that is values that are not apart of normal operations. Therefore, a bin with range $[535, 545]$ can be created to contain the regularly occurring values. Since the range of the bin is quite large, and it spans the values which occur most frequently, then the majority of tuples which fall into this category will require candidacy checks. However, our assumption is that queries will occur for values outside the normal range. This leads to a specific strategy for binning: There are ten bins on either side of the central bin representing the normal operational range. Each of these outer bins is capable of containing a value with a range of one. Fig. 7 represents the binning distribution for voltage magnitude. There is an additional bin for the value zero, since this is an indication of a data event at a PMU site. This strategy generates bins of small ranges for values of V that will be queried frequently and very large bins for those that aren't.

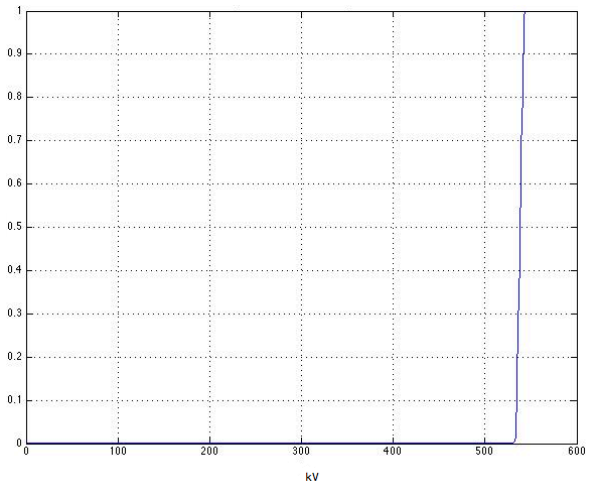

Fig. 6. Normal Voltage Magnitude CDF

Fig. 7. Voltage Magnitude Binning

In addition to the aforementioned attributes, we also introduced an attribute Δ, which represents the displacement between phase angles from the previous time-stamp, *i.e.*, $\Delta_t = |\phi_t - \phi_{t-1}|$. Δ is a coarse representation of rate of change and can be an indicator as to whether a power event occurred. Therefore, we bin Δ with smaller ranges, reducing the number of candidacy checks. Listed below are the bins that we used for each attribute. The total is 9,768 bins for each row in the bitmap index.

- Year: 11
- Month: 12
- Day: 31
- Hour: 24
- Minute: 60
- Second: 60
- Millisecond: 10
- ϕ (23 for each PMU): 40×36
- V (36 for each PMU): 40×23
- Δ (180 for each PMU): 40×180

V. Results

Queries were ran over the database to demonstrate the performance gains from analyzing and creating a bitmap index over the data. For these experiments, 4 million rows from the database were queried. File Map was used to retrieve the tuples from the database once a query has been serviced. The bitmap results are compared against the common linear scan that is performed when searching a database.

Table II shows results from six queries that were run. Query ID 1 is an example of a query where the user wishes to find when a specific PMU had a voltage magnitude of 533. An example of when this might happen is if the Correlation Visualization indicates there is an event occurring when that PMU has a voltage magnitude of 533. The exact same query to the bitmap engine provide a $68\times$ speed up on retrieval. Query IDs 2-4 demonstrate examples of requests for tuples at specific dates. These demonstrate that performing multiple queries with small adjustments does not require much additional time. Query IDs 5 and 6 shows queries for tuples that do not exist in this data set. Since the bitmap engine able to look at the bit vector results without every going to disk to see if the desired tuples are in the database, the speedup is many orders of magnitude greater. The bitmap query ID 5 takes slightly more time than ID 6 because ID 5 has to perform bitwise ANDs between each column, while ID 6 is simply checking a single column. There is very little time difference between the linear scan in ID 5 and 6.

The linear scan times are so similar because no matter the query given, it is necessary to scan the entire data set to ensure accuracy. Bitmap index query times can vary and primarily depend on how many columns need to be compared and how many tuples need to be pulled from disk. In fact the majority of the time spent for the bitmap index queries is simply retrieving the tuples from disk, making I/O the limiting factor.

VI. Future Work

The methods presented in this paper produced results that prove promising when retrieving this data. Presented below is the direction we plan to take the

ID	Selection Criteria	Linear Scan (sec)	Bitmap (sec)	Tuples Retrieved
1	Find all tuples where PMU1 has a magnitude Voltage Magnitude of 533.	25.859666	0.379387	160
2	Find all tuples that happened on exactly June 24, 2013 at 21:05 hours.	25.350993	0.854952	7204
3	Find all tuples that happened on exactly June 24, 2013 at 21:06 hours.	28.001001	0.922941	7204
4	Find all tuples that happened on exactly June 24, 2013 at 21:07 hours.	26.133607	0.785588	7204
5	Find all tuples that happened on exactly June 24, 2013 at 21:06 hours with PMU having a Voltage Magnitude of 533.	28.019449	0.001772	0
6	Find all tuples in 2012.	26.720291	0.0000601	0

TABLE II. QUERY PERFORMANCE

methods we currently have, making them scalable and more efficient.

Given large data sets, it is necessary to add additional methods of indexing for faster navigation and for queries to be returned in reasonable amounts of time. One such method that could be applied is sampling. This adds tiers of bitmaps, *i.e.*, bitmap indices for progressively more precise bitmaps, each one at a lower resolution of the data. For small amounts of data this is simply wasted space and too much overhead. When bit data such as this is introduced the sampling overhead begins to diminish as access times to the data doesn't scale up with the amount of data as quickly.

VII. CONCLUSION

In conclusion, we have shown that our bitmapping database minimizes data driven bottlenecks that are typically associated with data sets of this size. Specifically, compression of the data minimizes the space overhead required for indexing, allowing it to be operated on within memory. Query response times are also minimized due to the utilization of indexing coupled with the FileMap structure. This results in the ability to perform frequent queries leading to better analysis of the data.

This system is ideal to be coupled with other PMU analysis tools in order to better understand the data. One such tool that this system has already been coupled with is a correlation algorithm that is used to detect events occurring within the power system [11]. The combination of these two systems allow for fast retrieval of data in order identify anomalies among other events.

ACKNOWLEDGMENT

This research was funded in part by an Oregon BEST Center grant to E. Cotilla-Sanchez, D. Chiu, and R. Bass. We thank the Bonneville Power Administration for providing the PMU data and support.

REFERENCES

[1] A. Phadke, "Synchronized phasor measurements in power systems," *Computer Applications in Power, IEEE*, vol. 6, pp. 10–15, April 1993.

[2] P. E. O'Neil, "Model 204 architecture and performance," in *Proceedings of the 2nd International Workshop on High Performance Transaction Systems*, (London, UK), pp. 40–59, Springer-Verlag, 1989.

[3] R. R. Sinha and M. Winslett, "Multi-resolution bitmap indexes for scientific data," *ACM Transactions on Database Systems*, vol. 32, p. 2007.

[4] A. Romosan, A. Shoshani, K. Wu, V. M. Markowitz, and K. Mavrommatis, "Accelerating gene context analysis using bitmaps," in *SSDBM*, p. 26, 2013.

[5] Y. S. *et al.*, "Taming massive distributed datasets: data sampling using bitmap indices," in *HPDC*, pp. 13–24, 2013.

[6] F. Fusco, M. P. Stoecklin, and M. Vlachos, "Net-fli: On-the-fly compression, archiving and indexing of streaming network traffic," *PVLDB*, vol. 3, no. 2, pp. 1382–1393, 2010.

[7] K. Stockinger and K. Wu, "Bitmap indices for data warehouses," in *In Data Warehouses and OLAP. 2007. IRM*, Press, 2006.

[8] "Apache Hive Project, http://hive.apache.org."

[9] G. Guzun, G. Canahuate, D. Chiu, and J. Sawin, "A tunable aligned framework for bitmap indices," in *30th International Conference on Data Engineering (ICDE'14)*, 2014.

[10] K. Wu, E. J. Otoo, and A. Shoshani, "An efficient compression scheme for bitmap indices," 2004.

[11] R. Meier, E. Cotilla-Sanchez, B. McCamish, D. Chiu, B. Bass, J. Landford, and M. Histand, "Power system data management and analysis using synchrophasor data," in *IEEE Conference on Technologies for Sustainability*, (Portland, OR.), 2014.