Distributed Outlier Detection using Compressive Sensing

Ying Yan[1], Jiaxing Zhang[1], Bojun Huang[1], Xuzhan Sun[2*], Jiaqi Mu[3*],
Zheng Zhang[4†], Thomas Moscibroda[1]
[1]Microsoft Research, [2]Peking University, [3]UIUC, [4]NYU Shanghai
[1]{ying.yan, jiaxz, bojhuang, moscitho}@microsoft.com,
[2]sunxuzhan@pku.edu.cn, [3]jiaqimu2@illinois.edu, [4]zz@nyu.edu

ABSTRACT

Computing outliers and related statistical aggregation functions from large-scale big data sources is a critical operation in many cloud computing scenarios, e.g. service quality assurance, fraud detection, or novelty discovery. Such problems commonly have to be solved in a distributed environment where each node only has a local slice of the entirety of the data. To process a query on the global data, each node must transmit its local slice of data or an aggregated subset thereof to a global aggregator node, which can then compute the desired statistical aggregation function. In this context, reducing the total communication cost is often critical to the overall efficiency.

In this paper, we show both theoretically and empirically that these communication costs can be significantly reduced for common distributed computing problems if we take advantage of the fact that production-level big data usually exhibits a form of sparse structure. Specifically, we devise a new aggregation paradigm for outlier detection and related queries. The paradigm leverages compressive sensing for data sketching in combination with outlier detection techniques. We further propose an algorithm that works even for non-sparse data that concentrates around an unknown value. In both cases, we show that the communication cost is reduced to the logarithm of the global data size. We incorporate our approach into Hadoop and evaluate it on real web-scale production data (distributed click-data logs). Our approach reduces data shuffling IO by up to 99%, and end-to-end job duration by up to 40% on many actual production queries.

1. INTRODUCTION

Large-scale data-intensive computation has become an indispensable part of industrial cloud computing, and building appropriate system support and algorithms for efficiently analysing massive amounts of data has become a key focus, both in industry as well as in the research community. For example, data volume within Microsoft almost doubles every year and there are thousands of business critical data analytics jobs running on hundreds of thousands of machines everyday.

While each of these jobs is different, there are nevertheless a few common characteristics. First, instead of requiring the raw data records, many applications typically issue queries against *aggregated values* (e.g. sum...group by...). For example, our investigation in one of our own production clusters reveals that 90% of 2,000 regular analytics jobs in this cluster issue queries against such aggregated values. Secondly, data analytics jobs over big data are often processed in a shared-nothing distributed system that scales to thousands of machines. In other words, the entirety of the data is distributed across a large set of nodes each of which contains a slice of the data, and each node needs to transmit necessary information about its data to some aggregator node which then computes the desired function.

In this paper, we focus on a particularly important problem that falls within the above categorization: the *distributed outlier problem*. We explain the problem using a concrete scenario that arises in Microsoft Bing web search service quality analysis engine. Here, to measure the users' satisfaction of search query results, each search query is assigned either a positive (Success Click) or negative (Quick-Back Click) score, depending on the user's reaction to the shown search results. These scores are logged in search event logs at data centers across the globe, and stored in thousands of remote and geo-distributed machines. In order to determine the quality of a particular search query result, the task is now to aggregate these scores from all the various distributed search event logs. In a typical implementation of this aggregation (Figure 1), each node first locally (and partially) aggregates the scores (i.e., the values) grouping them by market and vertical segments (i.e., the keys). Then, all the nodes transmit their key-value pairs to a central node for final aggregation. Figure 1(a) shows a typical global aggregated result from the production logs. It can be seen that the result exhibits a "sparse-like" structure: while a majority of the values are concentrated around the *mode* b (1800 in the example) and there is a small fraction of *outliers* (marked with circles) whose values diverge greatly from b. Data from different individual data centers may not be sparse when considered independently. For obvious reasons, finding such outliers across different markets and vertical segments is of

*This work was done when the authors were doing their internships at Microsoft Research.
†This work was done when the author was working at Microsoft Research.

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.

http://dx.doi.org/10.1145/2723372.2747641.

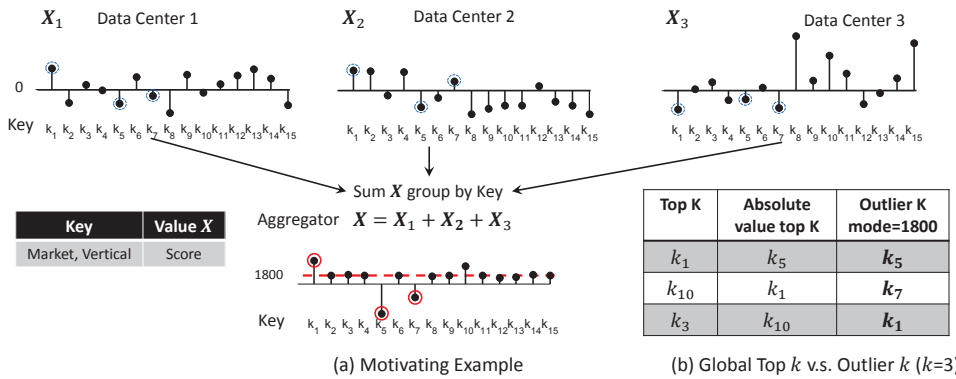


Figure 1: Distributed outlier detection in web search service quality analysis

critical importance to the analysis of our company's web search service quality.

There are three challenges that make it difficult to find an efficient solution to the above problem. 1) We have found that the local outliers and mode are often very different from the global ones. Without transmitting all the data, the true global outliers can only be approximated, and finding a high quality result is not guaranteed. As shown in Figure 1(a), the key k_5 in the remote data centers appear 'normal'. However, after aggregation, it is an obvious outlier compared to other keys. 2) As new data arrives—recall that terabyte of new click log data is generated every 10 mins—, the global outliers and mode will naturally change over time. Any solution that cannot support incremental updates is therefore fundamentally unsuited. 3) The number of distributed data centers considered in the aggregation may also change. For example, when a new data center joins the network, or the number of data centers considered in the analysis changes, the global outliers and mode may also change accordingly. We seek a solution with the ability of incremental addition and removal of data centers involved in the aggregation.

Formally, the above problem can be captured as a *k-outlier detection* problem, which seeks to find the k values furthest away from the mode b.[1] In the context of a large-scale production environment where data is located in many geographically distributed nodes, each node i holds a data slice that is an N-dimensional vector $\mathbf{x}_i \in \mathbb{R}^N$ (ordered according to a global key dictionary). In our concrete application, each entry in the vector $\mathbf{x}_i$ corresponds to the log entry of one specific search query result stored at that particular node i. The global data is the aggregation of these slice vectors by summing up the values of each entry, i.e., $\mathbf{x} = \sum_i \mathbf{x}_i$. In our application, the global data is therefore the vector where we sum up the log-entries for each query stored in all the different nodes. Outlier detection seeks the top-k outliers from this global data vector $\mathbf{x}$.

The commonly-used baseline implementation for the distributed outlier problem is to first aggregate the slices in "MapReduce-style". The problem with this approach is that the outliers and mode of the slices in different nodes are different from each other, and also different from the global outliers and mode after aggregation (see Figure 1). Therefore, in order to get a global result of reasonable accuracy, all data (or at least a large fraction of the data) would have to be transmitted to the global aggregator node. This entails heavy communication across machines, accompanied with expensive disk I/O and computation (sort, partitioning and so on). In many practical scenarios, the aggregation communication cost dominates system performance. In one study on SCOPE clusters in Microsoft [42], for example, data-shuffling I/O caused by aggregation incurs 58.6% of cross-rack traffic. It is therefore not surprising that new techniques to reduce aggregation communication have been researched (different data compression, partial aggregation techniques, optimizing the user-defined operators, etc [23]).

In this paper, we take a fundamentally different approach. The key insight is that practical aggregated big data frequently exhibits *sparse structure*. The above search quality application is a typical example: most of the values concentrate around zero or some constant bias, yet users are mostly interested in the analytics determined by the values that diverge from the majority of mass, such as querying for outliers, mode and top-k. Intuitively, such queries have the potential for reducing the required communication for aggregation by exploiting the underlying data's sparse structure. However, capitalizing on this potential is challenging, because different local data slices may not be equally sparse. In Figure 1, for example, data $\mathbf{x}_1, \mathbf{x}_2$ and $\mathbf{x}_3$ do not have modes, while –after aggregation–,the global $\mathbf{x}$ shows a clear mode. Thus, whether we can exploit sparsity for our problem depends on the existence of an efficient lossy compression algorithm that *keeps high accuracy on the high-divergence values while being insensitive to the distribution difference between local slices and the global data*. Fortunately, we find that *compressing sensing* is a suitable choice for such a compression algorithm, due to its linearity in measurement (compression) as well as its focus and "greediness" on the significant components in recovery (decompression).

Compressive Sensing (CS) is an efficient technique for sampling high-dimensional data that has an underlying sparse structure. Since its emergence, CS has found several key applications, for example in the domain of photography [16], image reconstruction and facial recognition [39], or network traffic monitoring [43, 7]. In this paper, we show that applying CS can also yield significant benefits (both in theory and practice) when applied to a classic distributed computing problem. While we focus here on the specific distributed outlier problem at hand, our techniques may also be extended to

[1]Note the difference between the k-outlier problem and the top-k problem: the keys with most exceptional values are not necessarily the top-k (or absolute top-k)keys (cf. Figure 1(b)).

solve similar aggregation queries (mean, top-k, percentile,...) in large-scale distributed systems.

In this paper, we make the following contributions:

- We devise a new communication-efficient aggregation paradigm for the distributed outlier problem. The solution leverages compressive sensing techniques and exploits the underlying sparse structure of the data. Technically, the algorithm compresses the data by a random projection at each local node. This compressed sketch is transmitted to the aggregator node, which then uses a compressive sensing-based recovery algorithm to determine the mode and the outliers simultaneously (see Figure 2).

- We propose Biased OMP (BOMP), a new recovery algorithm to recover both the outliers and the mode which can be any *unknown value* that the values of the aggregate data are concentrated around. BOMP thus supports a much wider set of scenarios compared to existing compressive sensing-based recovery algorithms [34, 37], which can only be applied to sparse data concentrating at zero (sparse at zero).

- We show theoretically that our algorithm can find outliers with the aggregation communication reduced to the logarithm of the global data size. Specifically, if the data of size N satisfies an s-sparsity condition, our algorithm can recover the outliers with $O(s^c \cdot \log N)$ communication cost, where c is a small constant.

- We implement our algorithm in Hadoop. We show that for the web search quality application described above, our technique significantly outperforms existing algorithms for outlier detection, reducing data-shuffling IO by up to 99%. In our concrete production environment, this IO reduction results in a reduction of end-to-end job duration by up to 40%.

The remaining of the paper is organized as follows. Section 2 introduces preliminaries of outlier detection and compressive-sensing techniques. Section 3 formulates the problem and presents our algorithm for compressive-sensing based distributed outlier detection. Section 4 theoretically analyses the performance of BOMP. Section 5 introduces how to implement our solution in Hadoop and discusses problems related to system implementation. Experimental results are reported in Section 6. Section 7 discusses related work and Section 8 concludes the paper.

2. PRELIMINARIES

2.1 Distributed Outlier Detection

We first give a formal definition of distributed outlier detection. Suppose a data vector $\mathbf{x} = [x_1, x_2, ..., x_N]^{\mathrm{T}} \in \mathbb{R}^N$ is distributed across L nodes such that each node l holds a partial slice of the data $\mathbf{x}_l = [x_{l1}, x_{l2}, ..., x_{lN}]^{\mathrm{T}} \in \mathbb{R}^N$ and $\sum \mathbf{x}_l = \mathbf{x}$. Now suppose an aggregator node wants to compute some statistics on $\mathbf{x}$. In order to do so each of the L nodes can send information about its local data slice to the global aggregator node.

One important data property that the aggregate data $\mathbf{x}$ may satisfy is *sparsity*:

Definition 1 (s-sparse). *A vector* $\mathbf{x} = [x_1, \ldots, x_n]^{\mathrm{T}} \in \mathbf{R}^N$ *is called* s-sparse *if it has s nonzero items.*

Typically, aggregated big data in the real world is not sparse in the strict sense of the above definition. However, it is "sparse-like" if we consider a more general data property on $\mathbf{x}$. We study the *Outlier Problem* under this more general property. In its most basic version, we assume the existence of a *dominating majority* of the values of $\mathbf{x}$. Formally, we define a *majority-dominated data* as follows.

Definition 2 (majority-dominated). *A vector* $\mathbf{x} = [x_1, \ldots, x_n]^{\mathrm{T}} \in \mathbf{R}^N$ *is* majority-dominated *if there exists* $b \in \mathbb{R}$ *such that the set* $O = \{i : x_i \neq b\}$ *has* $|O| > \frac{N}{2}$.

Given a majority-dominated data $\mathbf{x}$, we call the items in O the *outliers*. Note that the value of the majority b, if it exists, is unique by definition. The *k-outlier problem* is defined as finding a set $O_k \subseteq O$ with $\min(k, |O|)$ elements that are furthest away from b. Formally, for any $i \in O_k$ and $j \notin O_k$, we have $|x_i - b| \geq |x_j - b|$.

s-sparse data and majority-dominated data are simple concepts of sparse data. In practice, data often exhibits weaker notions of sparse structure. In Figure 1, for example, values concentrate around a mode b, but they are not necessarily equal to the exact b. Due to the imprecision of 'concentration', the outliers and the mode do not have unique definitions on such data.

As discussed before, in many real-world applications the number of keys N can be very large, so the goal is to find distributed algorithms in which the total data volume transmitted by all nodes is *sub-linear* (logarithmic, if possible) in N. Also, due to the possibly large latency of data transmissions, we focus on single-round algorithms: every remote node transmits information derived from its local data to the aggregator in parallel, the aggregator receives all this information, and then locally computes the global results.

2.2 Compressive Sensing and Recovery Algorithms

The basic ideas and mathematical foundations of compressive sensing were established in [8] [9] [15]. It is an efficient way to sample and compress data that has a sparse representation. Compressive sensing has two phases: i) sampling the data to a measurement and ii) recovering the data from the sampled measurement. The sampling is done by linearly projecting a data vector $\mathbf{x} = [x_1, x_2, ..., x_n]^{\mathrm{T}}$ with a measurement matrix $\mathbf{\Phi} = [\varphi_1, \varphi_2, ..., \varphi_n]$ into $\mathbf{y} = \mathbf{\Phi x}$. Due to the linearity in this sampling, compressive sensing seems uniquely suited for data that is distributed across different locations or data that is frequently updated, scenarios in which traditional compression techniques often face challenges and may not be applicable. In practice, $\mathbf{\Phi}$ is randomly generated, which was shown to be near-optimal [9] [15].

For the recovery algorithms, the ones most widely used are Basis Pursuit (BP) [13] and (Orthogonal) Matching Pursuit (OMP) [30, 34, 37], which try to decompose the signal to a sparse representation via an over-complete dictionary. BP recovers the data $\mathbf{x}$ by solving the optimization problem $\min_{\mathbf{x}} \|\mathbf{x}\|_1$ subject to $\mathbf{y} = \mathbf{\Phi x}$, which is transformed into a linear programming problem. OMP is a greedy algorithm. Starting from a residual $\mathbf{r} = \mathbf{y}$ and an empty matrix $\mathbf{\Phi}_*$, OMP performs a sequence of iterations. In iteration k, OMP selects the column vectors $\varphi \in \mathbf{\Phi}$ that has the largest inner

product with the current residual $\mathbf{r}$. OMP appends this column to $\mathbf{\Phi}_*$ and records its position in $\mathbf{\Phi}$ by I_k. $\mathbf{y}$ is projected to the subspace spanned by $\mathbf{\Phi}_*$ to get the representation $\mathbf{z} = [z_1, z_2, ..., z_k]^{\mathrm{T}}$. Then the residual is updated by $\mathbf{r} = \mathbf{y} - \mathbf{\Phi}_*\mathbf{z}$. When OMP is finished after some iterations, it gets the recovered $\mathbf{x}$ from $\mathbf{z}$ by setting $x_{I_k} = z_k$ and the other positions in $\mathbf{x}$ to zero.

Compared to BP, OMP has several advantages when used for the distributed k-outlier detection problem. First, OMP is simple for implementation and faster than BP. Second, in the outlier problem we are inherently interested are those 'significant' components instead of the entirety of the data. OMP naturally recovers the 'significant' components during the first several iterations, after which we can stop the execution of OMP without significant loss in accuracy. Exploiting this feature of OMP is critically important when applying compressive sensing to big data problems in general, and to the outlier problem in particular, because the computational burden of recovery would pose an unsurmountable obstacle otherwise.

3. ALGORITHM

In this section, we introduce our CS-based algorithm for the distributed k-outlier problem. We first show how we compress the distributed data, and then present the recovery algorithm that finds the outliers from the compressed data.

3.1 Distributed Compression

Each node contains up to N key-value pairs and we want to get the k outliers of the globally aggregated values grouped by the keys. Our solution consists of three phases. As shown in Figure 2, the first phase is data compression. We vectorize the key-value pairs into vectors and apply a random projection locally at each node to get the local compression (or measurement). Then, the compressed data is transmitted to the aggregator who then directly computes a compressed aggregate from the different compressed data. Finally, the outliers and mode are recovered using the BOMP algorithm. We now describe each of these steps in detail. Also, please refer to Figure 3 for an example of how to apply the procedure in the context of our web search quality application.

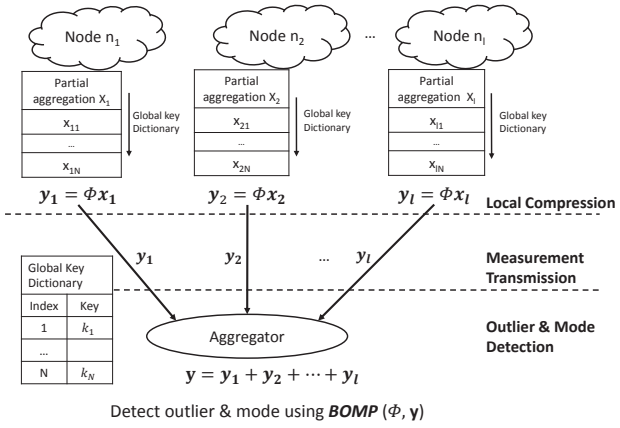


Figure 2: CS-Based Solution for Distributed Outlier and Mode Detection

Vectorization. Given a key space of size N, we can build a global *key dictionary*: The values on each node l are arranged by their key in a globally fixed order forming a vector $\mathbf{x}_l = [x_1, x_2, ..., x_N]^{\mathrm{T}}$. If a key does not exist on a node, we still keep an entry with value equals to 0. Thus, all keys have the same fixed position in all local vectors $\mathbf{x}_l$; i.e., looking up the key dictionary with the position of a value, we can find its key. Distributed outlier detection tries to seek the outliers on the aggregated data $\mathbf{x} = \sum_{l=1}^{L} \mathbf{x}_l$.

Local Compression. By a consensus, each node randomly generates the same $M \times N$ measurement matrix $\mathbf{\Phi}_0$ with column $\varphi_1, \varphi_2, ..., \varphi_N$. Locally, each node l then measures its data $\mathbf{x}_l$ by $\mathbf{y}_l = \mathbf{\Phi}_0\mathbf{x}_l$, resulting in a vector of length M. The fraction $\frac{M}{N}$ is the *compression ratio*, and we call $\mathbf{y}_l$ a *local measurement*.

Measurement Transmission. Each node now transmits its local measurement $\mathbf{y}_l$ to the aggregator. Assuming the system consists of L nodes, the communication cost is thus reduced from $O(N \cdot L)$ to $O(M \cdot L)$ compared to the trivial algorithm in which all data is transmitted to the aggregator.[2]

Global Measurement. Once the aggregator receives all local measurements $\mathbf{y}_l$, it aggregates them into a global measurement $\mathbf{y} = \sum_{l=1}^{L} \mathbf{y}_l$:

$$\mathbf{y} = \sum_{l=1}^{L} \mathbf{y}_l = \sum_{l=1}^{L} \mathbf{\Phi}_0\mathbf{x}_l = \mathbf{\Phi}_0 \sum_{l=1}^{L} \mathbf{x}_l = \mathbf{\Phi}_0\mathbf{x}. \qquad (1)$$

Notice that $\mathbf{y}$ exactly corresponds to the measurements on the aggregated data $\mathbf{x} = \sum_{l=1}^{L} \mathbf{x}_l$.

3.2 BOMP Algorithm

In principle, we can now apply any recovery algorithm used in the compressive sensing literature (e.g., OMP) to the global measurement $\mathbf{y}$ and use the recovered data to determine the outliers. However, when the values in $\mathbf{x}$ concentrate on an unknown non-zero bias b, all existing compressive sensing recovery algorithms are not applicable to this non-sparse data. Therefore, to deal with this more general scenario, we propose a novel recovery algorithm that uses the traditional OMP algorithm as a subroutine. We call the new algorithm "Biased OMP" (BOMP), because it can recover any set of values that concentrate around an unknown non-zero bias.

The BOMP algorithm is based on the following intuition. A data vector $\mathbf{x}$ whose values concentrate around b can be decomposed into vectors $\mathbf{x} = b + \bar{\mathbf{z}}$, where $\bar{\mathbf{z}}$ is near-sparse. By this decomposition, we reduce the recovery problem on a non-sparse $\mathbf{x}$ into an equivalent problem of recovering a near-sparse vector $\bar{\mathbf{z}}$ with a scalar b. Following this new representation of $\mathbf{x}$, measurement $\mathbf{y} = \mathbf{\Phi}_0\mathbf{x}$ can be written as

$$\mathbf{y} = \mathbf{\Phi}_0 \cdot (b + \bar{\mathbf{z}}) = \sum_{i}^{N} \varphi_i \cdot b + \mathbf{\Phi}_0\bar{\mathbf{z}}.$$

Carefully arranging the terms, we get another matrix-vector multiplication form

$$\mathbf{y} = [\frac{1}{\sqrt{N}} \sum_{i=1}^{N} \varphi_i, \mathbf{\Phi}_0] \cdot \begin{bmatrix} \sqrt{N}b \\ \bar{\mathbf{z}} \end{bmatrix} = \Phi \cdot \mathbf{x}. \qquad (2)$$

The form in (2) can be considered as a measurement on an extended vector $\mathbf{z} = [\sqrt{N}b, \bar{\mathbf{z}}]^{\mathrm{T}}$ with an extended measurement

[2]In practice, additional compression techniques can be applied on the data measurement for further data reduction.

matrix $\Phi = [\frac{1}{\sqrt{N}} \sum_{i=1}^{N} \varphi_i, \mathbf{\Phi}_0]$. In this way, we essentially give the vector $\mathbf{y}$—which is originally a measurement on $\mathbf{x}$—a new explanation: it is also a measurement on $\mathbf{z}$. Attractively, $\mathbf{z}$ is sparse and has a one-to-one mapping with $\mathbf{x}$. Thus, we can use any existing compressive sensing recovery algorithm, particularly OMP, to recover $\mathbf{x}$, and finally assemble the vector $\mathbf{x}$ from z. Overall, BOMP reduces the recovery problem on $\mathbf{x}$ to a recovery problem on $\mathbf{z}$ that is near-sparse by separating the bias b from $\mathbf{x}$ into an extra component to extend the vector that is to be recovered. This new recovery problem is solvable by the standard OMP algorithm. Notice that this new recovery problem has a different measurement matrix (the left term in (2)) and data vector dimension $(N+1)$ from the original one's.

Algorithm 1: BOMP

Input: Measurement matrix $\mathbf{\Phi}_0 = [\varphi_1, \varphi_2, ..., \varphi_N]$, measurement $\mathbf{y} = \mathbf{\Phi}_0\mathbf{x}$, iteration number R
Output: Outlier set O, mode b, recovered $\hat{\mathbf{x}}$

1: Extend the measurement matrix $\mathbf{\Phi} \leftarrow [\varphi_0, \mathbf{\Phi}_0]$, where $\varphi_0 = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} \varphi_i$
2: $\mathbf{z} = [z_0, z_1, ..., z_N]^{\mathrm{T}} \leftarrow$ OMP recovery on $\mathbf{y} = \mathbf{\Phi z}$ with R iterations.
3: $\hat{\mathbf{x}} \leftarrow [z_1 + \frac{z_0}{\sqrt{N}}, z_2 + \frac{z_0}{\sqrt{N}}, ..., z_N + \frac{z_0}{\sqrt{N}}]^T$, $b \leftarrow \frac{z_0}{\sqrt{N}}$, $O \leftarrow \{i | x_i \neq b\}$
4: **return** O, b, $\hat{\mathbf{x}}$

Our BOMP recovery algorithm for the compressive sensing-based outlier solution is illustrated in Algorithm 1 whose detailed workings are illustrated in Figure 3. Besides recovering the non-sparse data vector $\mathbf{x}$, BOMP also estimates its mode b and returns the outlier set O. Given the measurement $\mathbf{y} = \mathbf{\Phi}_0\mathbf{x}$ on data vector $\mathbf{x} = [x_1, x_2, ..., x_N]^T$, where $\mathbf{\Phi}_0 = [\varphi_1, \varphi_2, ..., \varphi_N]$ is a $M \times N$ measurement, BOMP first extends $\mathbf{\Phi}_0$ to $\mathbf{\Phi} = [\varphi_0, \varphi_1, ..., \varphi_N]$ by adding an extra column

$$\varphi_0 = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} \varphi_i. \tag{3}$$

Then, BOMP applies a standard OMP algorithm with R iterations to recover $\mathbf{z} = [z_0, z_1, ..., z_N]^T$ from $\mathbf{y} = \mathbf{\Phi z}$. Finally, we recover $\mathbf{x}$ through

$$\hat{\mathbf{x}} = [z_1 + \frac{z_0}{\sqrt{N}}, z_2 + \frac{z_0}{\sqrt{N}}, ..., z_N + \frac{z_0}{\sqrt{N}}]^T. \tag{4}$$

BOMP takes $b = z_0/\sqrt{N}$ as the mode of $\mathbf{x}$. Since the size of $\mathbf{z}$ is upper bounded by R, it follows (4) that the recovered $\mathbf{x}$ has at most $R-1$ components apart from b; we take these to be the outlier set O. Finally, the algorithm selects the k elements from O which are furthest away from b. These are the detected k outliers.

4. THEORETICAL ANALYSIS

We theoretically analyze BOMP to determine its applicability and efficiency to the problem at hand. The key fundamental difference between BOMP and the traditional OMP is that in OMP, all random entries in the measurement matrix are sampled independently. This matrix has been proved to have the restricted isometry property (RIP) [5], which is a guarantee for the recovery accuracy. However, in BOMP, the first column in the extended measurement matrix is the sum of all the other columns. This weak dependence becomes negligible as the number of columns grows, but it introduces substantial challenges to the theoretical analysis of BOMP.

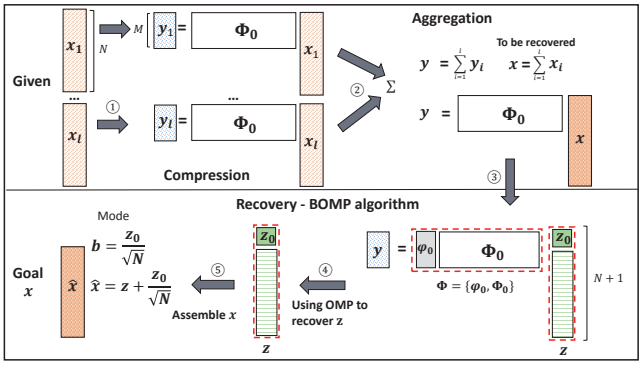


Figure 3: Example of CS-based approach

In this analysis, we apply BOMP to a *biased s-sparse vector*, which has s outlier components differing from an unknown bias, with all the other components equal to the bias. We prove that BOMP shows good recovery performance in Theorem 1. It requires $M = As^a \log(N/\delta)$ measurements to recover an N-dimensional s-sparse vector with probability $1-\delta$.

THEOREM 1. *Suppose that $\mathbf{x} \in \mathbb{R}^N$ is a biased s-sparse vector. Choose $M = As^a \log(N/\delta)$ where A and a are absolute constants. Draw a random $M \times N$ measurement matrix $\mathbf{\Phi}_0$ whose entries are i.i.d $\mathcal{N}(0, 1/M)$. BOMP can successfully recover $\mathbf{x}$ from measurement $\mathbf{y} = \mathbf{\Phi}_0\mathbf{x}$ with probability exceeding $1-\delta$. The theorem holds under the Near-Isometric Transformation and Near-Independent Inner Product conjectures described below.*

In the following, we give a detailed proof sketch of Theorem 1. Intuitively, OMP is a sequence of column selections (see Algorithm 2). In each iteration, the column of the measurement matrix $\mathbf{\Phi}$ with the largest inner product with the current residual $\mathbf{r}$ is selected. Then, the measurement vector $\mathbf{y}$ is projected to the subspace spanned by the selected column vector set $\mathbf{\Phi}_s$, and a new residual is calculated by $\mathbf{y} - \text{proj}(\mathbf{y}, \mathbf{\Phi}_s)$, where $\text{proj}(\mathbf{y}, \mathbf{\Phi}_s)$ is the projected result.

Algorithm 2: Column Selection in OMP [37]

Input: Matrix $\mathbf{\Phi} = [\varphi_0, \varphi_1, ..., \varphi_N]$, measurement vector $\mathbf{y}$
Output: selected column set $\mathbf{\Phi}_s$

1: Initialize $\mathbf{\Phi}_s$ as an empty set
2: $\mathbf{r} \leftarrow \mathbf{y}$
3: **while** $\|\mathbf{r}\|_2 > 0$ **do**
4: $\quad \varphi \leftarrow \arg\max_{\varphi \in \mathbf{\Phi}} |\langle \varphi, \mathbf{r} \rangle|$
5: $\quad \mathbf{\Phi}_s \leftarrow \mathbf{\Phi}_s \cup \{\varphi\}$
6: $\quad \mathbf{r} \leftarrow \mathbf{y} - \text{proj}(\mathbf{y}, \mathbf{\Phi}_s)$
7: **end while**
8: **return** $\mathbf{\Phi}_s$

Without loss of generality, assume the s outliers occur in the first s dimensions of $\mathbf{x}$. Accordingly, partition the extended measurement matrix $\mathbf{\Phi}$ into $[\mathbf{\Phi}_*, \mathbf{\Psi}]$, where $\mathbf{\Phi}_*$ includes the column vectors corresponding to the bias and the s outliers. It is easy to prove that, if the first $s+1$ iterations in Algorithm 2 always select the columns in $\mathbf{\Phi}_*$, BOMP will stop after $s+1$ steps and $\mathbf{x}$ is exactly recovered. In each iteration of Algorithm 2, statement 4 selects a new column by comparing the inner product. Thus, to ensure BOMP's successful execution, it is sufficient that for any $\mathbf{r}$ in the subspace span($\mathbf{\Phi}_*$) that is spanned by columns in $\mathbf{\Phi}_*$, there is

$$\rho(\mathbf{r}) = \frac{\max_{\psi\in\mathbf{\Psi}} |\langle\psi, \mathbf{r}\rangle|}{\max_{\varphi\in\mathbf{\Phi}_*} |\langle\varphi, \mathbf{r}\rangle|} < 1$$

in all the $s+1$ steps. In a single step

$$\begin{aligned}\mathbb{P}\{\rho(\mathbf{r}) < 1\} \geq & \mathbb{P}\left\{ \frac{\max_{\psi\in\mathbf{\Psi}} |\langle\psi, \mathbf{r}\rangle|}{\|\mathbf{\Phi}_*^{\mathrm{T}} \cdot \mathbf{r}\|_2/(\sqrt{s+1})} < 1 \right\} \\ \geq & \mathbb{P}\left\{ \frac{|\langle\psi, \mathbf{r}\rangle|}{\|\mathbf{\Phi}_*^{\mathrm{T}} \cdot \mathbf{r}\|_2} < \frac{1}{\sqrt{s+1}} \right\}^{N-s}. \end{aligned} \quad (5)$$

Due to the independence property, in (5) ψ can be any column in $\mathbf{\Psi}$. Thus, our goal is to evaluate the value of

$$\rho'(\mathbf{r}) = \frac{|\langle\psi, \mathbf{r}\rangle|}{\|\mathbf{\Phi}_*^{\mathrm{T}} \cdot \mathbf{r}\|_2}.$$

To achieve a high success probability, we would like $\rho'(\mathbf{r})$ to be as small as possible. That is $\|\mathbf{\Phi}_*^{\mathrm{T}} \cdot \mathbf{r}\|_2$ is much larger than $|\langle\mathbf{r}, \psi\rangle|$.

4.1 Near-Isometric Transformation

In $\|\mathbf{\Phi}_*^{\mathrm{T}} \cdot \mathbf{r}\|_2$, the vector $\mathbf{r}$ is linearly transformed by $\mathbf{\Phi}_*^T$. If all the entries in $\mathbf{\Phi}_*$ are independent, a simple proof in [5, 37] shows that the norm of $\mathbf{r}$ is preserved with high probability,

$$\mathbb{P}\{\|\mathbf{\Phi}_*^{\mathrm{T}}\mathbf{r}\|_2 \geq 0.5\|\mathbf{r}\|_2\} \geq 1 - e^{-cM}. \quad (6)$$

Now, the first column φ_0 in $\mathbf{\Phi}_*$ has a very weak dependence with the other s columns. Depicting this dependence, it can be seen that the covariance matrix between φ_0 and any other column only has a non-zero diagonal $1/\sqrt{N}$. When N is much larger than s, this dependence becomes smaller and smaller, and ultimately negligible. So, we believe that the near-isometric transformation property in (6) holds in spite of the weak-dependence. We formulate these intuitions in the following conjecture.

Conjecture 1 (Near-Isometric Transformation). *Given a $M \times (s+1)$ randomly generated matrix $\mathbf{\Phi}_*$ whose entries are all $\mathcal{N}(0, 1/M)$. The first column vector is weakly dependent with any other column with covariance matrix $\zeta\mathbf{I}$, where $|\zeta|$ is sufficiently small. The other s column vectors are independent with each other. For any $\mathbf{r} \in \mathrm{span}(\mathbf{\Phi}_*)$, there is $\mathbb{P}\{\|\mathbf{\Phi}_*^{\mathrm{T}}\mathbf{r}\|_2 \geq 0.5\|\mathbf{r}\|_2\} \geq 1 - e^{-cM}$ where c is an absolute constant.*

We conducted extensive numerical experiments to verify the validity of Conjecture 1. We find that when $s = 2$, $M > s$ and ζ gets its largest value $1/\sqrt{s}$: c is around 0.4. When M and s are larger than 10 which is the case in real applications, we observe $\|\mathbf{\Phi}_*^{\mathrm{T}}\mathbf{r}\|_2 \geq 0.5\|\mathbf{r}\|_2$ always holds by a large margin.

4.2 Inner Product between Random Vectors

We now discuss the second major component of the proof which relates the inner product between random vectors. Let $\mathbf{u} = \frac{0.5\mathbf{r}}{\|\mathbf{\Phi}_*^{\mathrm{T}}\mathbf{r}\|_2}$. Then

$$\rho'(\mathbf{r}) = 2|\langle\psi, \mathbf{u}\rangle|.$$

This is an inner product between two random vectors ψ and $\mathbf{u}$. All entries in φ are independent $\mathcal{N}(0, 1/M)$; by Conjecture 1, $\|\mathbf{u}\|_2 \leq 1$ with probability $P_1 = 1 - e^{-cM}$. If ψ and $\mathbf{u}$ are independent, then it follows from standard techniques (see [37]) that

$$\mathbb{P}\{|\langle\psi, \mathbf{u}\rangle| \leq \epsilon\} \geq 1 - e^{-\epsilon^2 M/2}.$$

In our case, φ is weakly dependent with $\mathbf{u}$. When $\mathbf{u} = \frac{\varphi_0}{\|\varphi_0\|_2}$, the dependence is the strongest. In that case, the covariance matrix only has a non-zero diagonal $1/\sqrt{N}$. When N is sufficiently large, this dependence is so small that its influence on the inner product becomes negligible. We formalize this insight as follows.

Conjecture 2 (Near-Independent Inner Product). *Suppose $\mathbf{x}$ and $\mathbf{y}$ are two M-dimensional vectors. Both of them have i.i.d $\mathcal{N}(0, 1/M)$. $\mathbf{x}$ and $\mathbf{y}$ are statistically dependent with each other with covariance $\mathbb{E}(\mathbf{x}\mathbf{y}^{\mathrm{T}}) = \zeta\mathbf{I}$. Normalize $\mathbf{y}$ as $\mathbf{y}' = \mathbf{y}/\|\mathbf{y}\|_2$. If $|\zeta|$ is sufficiently small, then $\mathbb{P}\{|\langle\mathbf{x}, \mathbf{y}'\rangle| \leq \epsilon\} \geq 1 - e^{-\epsilon^{2a}M/2}$ where a is an absolute constant.*

Again, we conducted extensive experimental evaluations to verify Conjecture 2. When setting $a = 1.1$, we never observed any counter-examples; in fact, the condition was satisfied by a wide margin in all cases.

4.3 Success Probability

Using the two previous conjectures, we now have the ingredients to conclude the proof. Specifically, it holds that

$$\begin{aligned}\mathbb{P}\left\{\rho'(\mathbf{r}) < \frac{1}{\sqrt{s+1}}\right\} &= \mathbb{P}\left\{|\langle\psi, \mathbf{u}\rangle| < \frac{1}{2\sqrt{s+1}}\right\} \\ &\geq \left(1 - e^{-(\frac{1}{2\sqrt{s+1}})^{2a}M/2}\right) \cdot P_1 \\ &\geq \left(1 - e^{-\frac{bM}{s^a}}\right)\left(1 - e^{-cM}\right),\end{aligned}$$

where a and c are the constant values defined in Conjectures 1 and 2 respectively; b is a constant derived from a. Plugging this probability into (5), we get

$$\begin{aligned}\mathbb{P}\{\rho(\mathbf{r}) < 1\} &\geq \left(1 - e^{-\frac{bM}{s^a}}\right)^{N-s} \cdot \left(1 - e^{-cM}\right)^{N-s} \\ &\geq \left(1 - (N-s)e^{-\frac{bM}{s^a}}\right) \cdot \left(1 - (N-s)e^{-cM}\right) \\ &\geq 1 - Ne^{-\frac{BM}{s^a}},\end{aligned}$$

where B is a new constant. Applying the union bound on all the $s+1$ steps, we can upper bound the final success probability

$$\begin{aligned}\mathbb{P}\{E_{\mathrm{succ}}\} &\geq 1 - (s+1)(1 - \mathbb{P}\{\rho(\mathbf{r}) < 1\}) \\ &\geq 1 - (s+1)Ne^{-\frac{BM}{s^a}}\end{aligned}$$

Let $\mathbb{P}\{E_{\mathrm{succ}}\} \geq 1 - \delta$. The final conclusion is that taking measurement size $M = As^a \log(N/\delta)$, the recovery failure

probability of BOMP recovery failure probability is δ, where A and a are constants.

5. HADOOP IMPLEMENTATION

We implement our solution on Hadoop, version 2.4.0. In this implementation, the mappers are responsible for collecting and aggregating the data for each key in the global key list with length N, and then compress the partial results into a vector with length M using the CS-Mapper method as introduced in Algorithm 3. The reducer receives the M-length vectors and then recovers the $k-$outlier and mode using the CS-Reducer method. The recovered results are output to HDFS. As illustrated in Algorithm 4, the recovery process calls the BOMP algorithm. In the implementation, we optimized the matrix computation in the recovery using QR factorization [6] with Gram-Schmidt process which is implemented with Intel Math Kernel Library(MKL) in C and is pre-built in shared objects. We use Java Native Interface(JNI) to call the functions from Hadoop code. Our solution can greatly save the communication cost, which consists of the mapper output IO and shuffling in the reducer. Although the recovery process may bring additional overhead, the end-to-end latency is still greatly improved. As the input file size becomes bigger, the saving of end to end time is more significant: This is because each mapper needs to process a larger number of file chunks, which in turn implies that our method yields biggest savings in terms of communication costs, thus further reduces the waiting time of reducers. Thus, for larger input files, reducers need to wait for a longer time period to get all the outputs from the mappers. Our compressive sensing-based solution speeds up the mapping time which correspondingly reduces the reducer's waiting time. As such, the total latency is greatly reduced.

Algorithm 3: CS-Mapper

Input: Raw data, M, KeyList and *seed*
Output: Compressed data vector y with length M
$N \leftarrow$ KeyList.length
Generate random $M \times N$ measure matrix $\mathbf{\Phi}_0$ with *seed*
Load raw data and do partial aggregation for each key in the global KeyList, vector $x \leftarrow$ aggregation values ordered by KeyList
$y \leftarrow \mathbf{\Phi}_0 x$
return y;

Algorithm 4: CS-Reducer

Input: Compressed vector y, KeyList, K and *seed*
Output: Recovered $\mathbf{s}$, outlier set O, mode b
$N \leftarrow$ KeyList.length, $M \leftarrow y$.length
Generate random $M \times N$ measure matrix $\mathbf{\Phi}_0$ with *seed*
return BOMP($\mathbf{\Phi}_0$, y, $f(k)$)

One practical problem we encountered arises due to floating point precision when using the Gram-Schmidt process for QR factorization. The floating point error becomes non-negligible compared to the values in the remaining uncompressed vector as the recovery iteration increases, which may significantly influence the recovery result. To reduce this effect, our solution is to terminate the recovery process once the residual stops decreasing which seems to be very effective and accurate in practice.

An important optimization to speed up the recovery process is to select a proper iteration count R. The choice of R is affected by the value of k and data sparsity. In practice, as in Algorithm 4, we denote R as $f(k)$ since through parameter tuning, we find that $R \in [2k, 5k]$ is good enough for both recovery accuracy and efficiency.

Finally, note that the recovery algorithm can be accelerated with 30X $\sim$ 40X speed-up [2, 20] using GPU which is our on-going work.

6. PERFORMANCE EVALUATION

We begin by evaluating the performance of the CS-based distributed k-outlier detection algorithm BOMP. To have a thorough performance evaluation, we first conduct experiments on the result quality and effectiveness of our solution in Section 6.1. Then, we implement a compressive sensing based Hadoop system to evaluate the efficiency of our technique in Section 6.2. We use both synthetic and real-world production workloads in the experiments.

6.1 Effectiveness of BOMP

The effectiveness metrics we focus on are *communication cost* and *recovery quality*. Given the true k-outliers O_T which is a set of key and value pairs $<O_{T\cdot Key}, O_{T\cdot Value}>$, $|O_T| = k$ and a k-outliers estimation O_E, $|O_E| = k$, the error of the estimation is measured by the errors on keys and values:

1. **Error on Key (E_K):** $E_K = 1 - \frac{|O_{T\cdot Key} \cap O_{E\cdot Key}|}{K}$, $E_K \in [0, 1]$. It is the error on precision of the key set.

2. **Error on Value (E_V):** Both O_T and O_E are ordered according to the value. $E_V = \frac{\|O_{T\cdot Value} - O_{E\cdot Value}\|_2}{\|O_{T\cdot Value}\|_2}$, $E_V \in [0, 1]$. It is the relative L_2 error on the ordered value set.

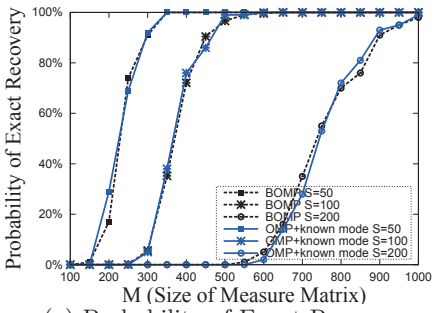
A good approximate k-outlier solution should have small E_K *and* E_V.

Our solution is independent of how the keys are distributed over the different nodes. Given a fixed number of nodes, the communication cost is only determined by the size of the measurement matrix M. The evaluations in this section focus on the recovery process using the BOMP algorithm with different measurement matrix size M.

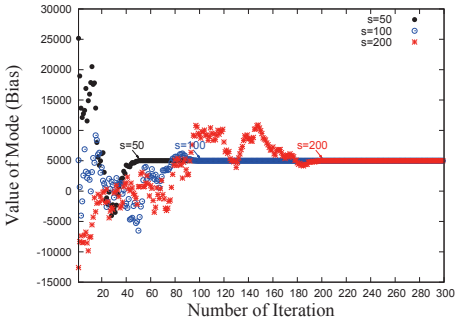
To evaluate the sensitivity of our solution to different data distributions and parameters, we use a synthetic data set in Section 6.1.1. In Section 6.1.2, we then show the effectiveness of our algorithm in terms of communication cost on a real search quality analysis production workload, and compare with alternative approaches.

6.1.1 Effectiveness of BOMP on Synthetic Data

We use two synthetic data sets as the aggregated scores in this group of experiments. The first one is a *majority-dominated* data set containing N observations with a mode b. There are $N - s$ observations having the value of b. The remaining s observations are not equal to b. b is set to be 5000. We change the sparsity of the data through varying the parameter s. The second one is a *Power-Law* distribution with skewness parameter α. Since it is distributed as a continuous heavy-tailed distribution, there is no pair of



(a) Probability of Exact Recovery



(b) Value of Mode(Bias) in Each Iteration

Figure 4: BOMP on majority dominated data

observations with the same value. The mode can be considered as the peak of its density function. We show that our algorithm can still show good performance over *Power-Law* distributions. The length of the data N to set to 1K in the experiments.

We being by examining the probability of exact recovery on *majority-dominated* data. In an exact recovery, $E_K = E_V = 0$ and the number of outliers k equals s. For each measurement matrix of size M, we test 1000 times each time with a randomly generated measurement matrix, from which the recovery percentage can be estimated. The number of recovery iterations is $\min\{M, s+1\}$. We also test the standard OMP algorithm under the assumption that the mode b is known in advance. We can see in Figure 4(a), BOMP has a similar performance as OMP even though BOMP does not need knowledge of b. However, if the mode is not known in advance (which is the more general case), OMP needs to transmit additional $2s+1$ data to get the mode. When $s = 200$, BOMP can save up to 40% of the communication cost.

Then, for each s, we choose M to be the size of 100% exact recovery and log the value of bias b (which is the mode) in each iteration in the recovery. The first 300 iterations of BOMP are plotted in Figure 4(b). We can see that when the iteration number equals the sparsity $s+1$, b starts to stabilize which is of great interest, because it confirms in practice the theoretical findings in Theorem 1.

In the second experiment, we examine the recovery quality over *Power-Law* distributed data with $\alpha = 0.9$ and 0.95, respectively and $N = 10K$. The errors E_K and E_V with $K = 5$, 10 and 20 are illustrated in Figures 5 and 6. For each M, we run 100 times compression and recovery with randomly generated measurement matrices. As such, we get the MAX, MIN and AVG E_K and E_V errors in the 100 runs. We can see that even for *Power-Law* distributed data, we can still correctly detect k-outliers with smaller M.

6.1.2 *Effectiveness of CS-Based Solution on Real Production Data*

In this group of experiments, we evaluate the performance of our solution with real production workload used for analysing Bing web search service quality as introduced in Section 1. We use three production workloads with the following template:

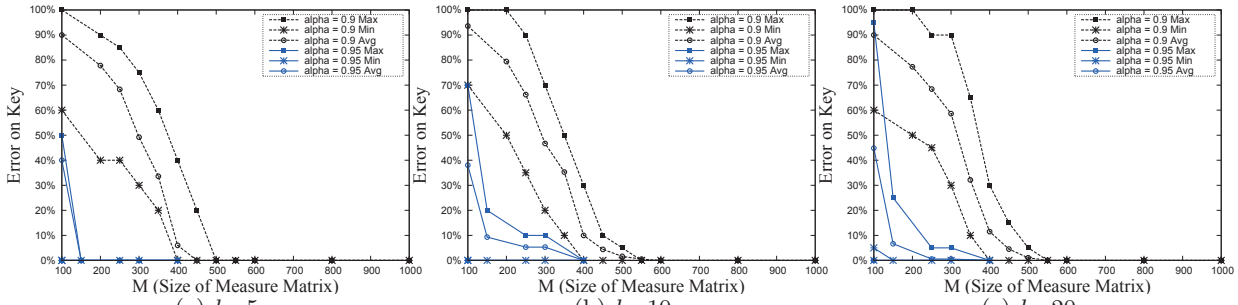
```
SELECT   Outlier K SUM(Score),G1...Gm
FROM     Log Streams PARAMS(StartDate,EndDate)
WHERE    Predicates
GROUP BY G1...Gm;
```

There are more than 600 such aggregation jobs running in production environment everyday, which consume large amounts of communication and computation resources. We choose three representative queries with different types of scores in the experiments: advertise click score, core-search click score, and answer click score. The GROUP-BY attributes including *QueryDate, Market, Vertical, RequestURL, DataCentre* and so on. The one-week log stream is 65TB total in size, and is merged from 8 geographically distributed data centers. The log contains the information from 49 Markets and 62 Verticals. After being filtered by the predicates, the total number of keys N that the three queries are interested in are 10.4K, 9K and 10K, respectively.

The performance metric we evaluate is communication cost computed from $N_t \cdot S_t$, where N_t is the number of tuples transmitted and S_t is the number of bytes we use to represent a tuple. In our solution, the tuple on each node is represented as a 1-dimension data vector with length N. Given L nodes, M measurements of S_M bytes each, the total communication cost is $L \cdot M \cdot S_M$. In our experiment, S_M is 64 bits.

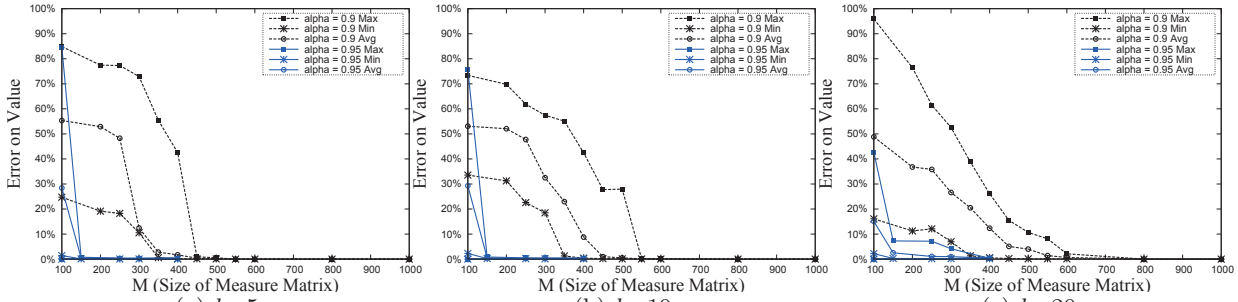
To the best of our knowledge, there is no existing distributed technique explicitly targeted for distributed outlier detection. In practice, all data is usually transmitted to the aggregator node. We consider this basic approach as one of our baselines. Using this vectorized approach, the communication cost is $L \cdot N \cdot S_v$, S_v is 64 bits. Alternately, if the number of non-zero keys n_i is small on each node, we can reduce the communication cost by shipping key-value pairs. As such, the communication cost is $\sum_{i=1}^{L} n_i \cdot S_t$, where S_t is the size of keyid-value pair which is 96 bits. Our experience working with the production data has shown that when transmitting all data, the communication cost of the vectorized approach is much smaller than shipping keyid-value pairs. We call this baseline ALL.

As a second baseline, we also implement an approximate approach called $k+\delta$ based on [10]'s three rounds framework. In the first round, each node samples g keys (g keys' ids are the same across different nodes) and sends them to the aggregator. Then, the aggregator computes an average value b from the aggregation values of g groups. In the second round, the aggregator transmits the value b to each remote node. In the last round, each remote node transmits back the $k+\delta-g$ outliers considering b as the mode. Finally, the aggregator outputs the k-outliers and mode b based on the aggregation results of the data received. As such, the communication cost by shipping keyid-value pairs is $L \cdot (K+\delta) \cdot S_t$, where S_t is the size of keyid-value pair which is 96 bits. Clearly, the communication cost of this approach



(a) k=5 (b) k=10 (c) k=20

Figure 5: Error on key for different k



(a) k=5 (b) k=10 (c) k=20

Figure 6: Error on value with different k

depends on how the values of different keys are distributed over different nodes. When the values are distributed with big standard deviations, the mode and outliers on each node are vastly different from the global ones and the estimation of outliers may have large errors. However, if the values are uniformly distributed across the nodes, the estimation based on the local data can be a good approximation of the global distribution. We therefore compare against this simple $k + \delta$-approach in order to determine how–on real data–this natural approach compares against our CS-based algorithm. We call this baseline K+δ.

In the first group of experiments, we evaluate the accuracy of outlier detection with different communication cost. We use three production workloads on finding k-outliers over sum of core-search, ads and answer click score. The keys are defined from grouping by *QueryDate, Market, Vertical, RequestURL* and *DataCentre* attributes. We partition the data exactly as it is partitioned on the different Microsoft data centers.

We compare the CS-Based algorithm with ALL and K+δ. For ALL, we adopt the vectorized approach because the total communication cost using keyid-value pairs is more than 3 times larger. In K+δ, the number of keys g to be sampled in the first round for the mode estimation should be big enough, since it is important to the estimation of mode, and thus the key to computing the outliers. From our investigation, the value of the sampled mode is relative stable when g is bigger than $(K + \delta)/2$. To this end, given a communication cost budget, we always choose g to be 50% of the communication cost in the evaluation in order to get a fair comparison among the three solutions. The communication cost of our CS-Based approach and the K+δ approach is normalized with the transmitting ALL solution.

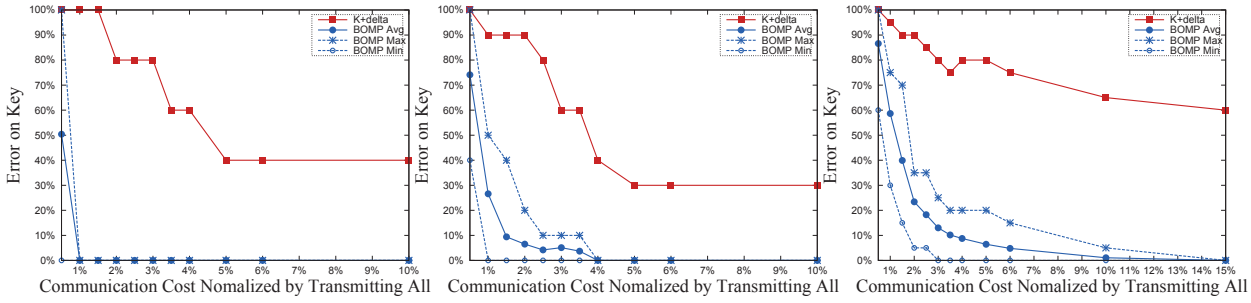
The evaluation results on core-search score data are shown in Figures 7 and 8. Given a target communication cost, we run the CS-based approach 100 times with different random measurement matrices and report MAX, MIN and AVG errors on key and value. We see that when the communicate cost is only 1% of ALL, the CS-Based approach yields accurate key results with $K = 5$. The error on value under the same setting is only 4%. When K grows bigger, we need more measurements to get the result with the same accuracy compared to smaller K. However, even when $K = 20$, *we only need 15% of the communication cost to get accurate key set.* In contrast, the K+δ approach returns big errors on both key and value even with much larger communication cost.

The trend in the results of ads and answer click score data are similar to the result above. In fact, the CS-based algorithm significantly outperforms the other approaches consistently over all benchmarks and data workloads.

The mode b in each iteration of the three production data sets are shown in Figure 9. The values on y-axis are anonymized. We can see that b becomes stable after 300, 650 and 610 iteration (M=500, 800 and 800, respectively), from which we can also derive the sparsity of production data.

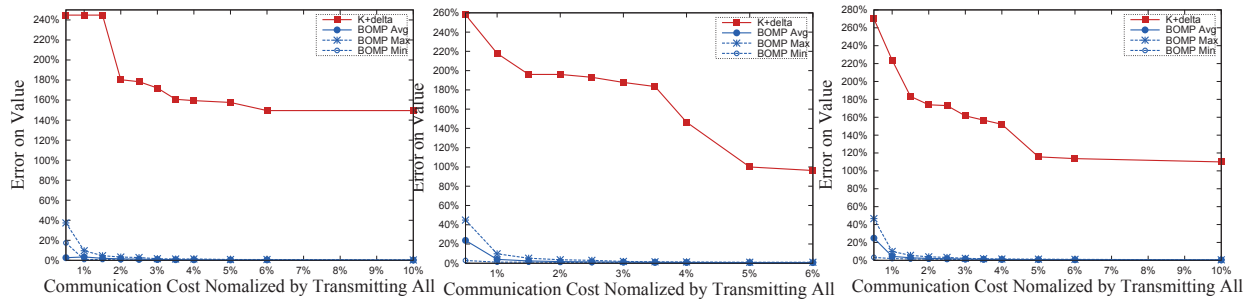
6.2 Efficiency of BOMP on Hadoop

In this group of experiments, we want to evaluate the performance of BOMP on MapReduce implementation. Since we can not find an existing Hadoop implementation on outlier detection, we compare our solution with traditional Top-k computing on Hadoop. Our solution can be extended directly to solve distributed Top-k problem when the data's mode is 0. All experiments are conducted on Hadoop 2.4.0 with both synthetic (*Power-Law* distribution with $\alpha = 1.5$, $N = 100Kto1M$) and product data set as introduced in Section 6.1.2. We change the data's mode to 0 by subtracting the model from all the data. All experiments are running on a 10-node cluster with Intel Core Xeon (CPU E5-2665@2.40GHz),



(a) k=5 (b) k=10 (c) k=20

Figure 7: Error on key over production data



(a) k=5 (b) k=10 (c) k=20

Figure 8: Error on value over production data

100GB of RAM and Cent OS 6.3 64bit. Network bandwidth is 1Gbps.

Firstly, we set $N = 100K$. Compared to traditional Top-k computation on Hadoop, our solution can greatly save the mappers' output IO cost and shuffling cost in reducer. However, it should also bring additional recovery cost. The amount of savings as well as overhead is mainly determined by the size of measurement matrix M. The input file size of synthetic data is 600M, as shown in Figure 10(a), the end to end time of our solution is smaller than the traditional implementation when $M < 1100$. The breakdown latencies of both mapper and reducer are shown in Figures 11(a) and 11(d). When we increase the input file size to 600G, the savings become more significant which is demonstrated in Figure 10(b). Interestingly, the savings on reducer as shown in Figure 11(e) is more significant. As discussed in Section 6.2, the reducer's waiting time can be further reduces if we can greatly save mapper time. BOMP can achieve that. For production data, the input file size is 12G. The comparison result is illustrated in Figure 10(c). When M grows bigger than 1600, the overhead of the recovery process in the reducer is much bigger. As such, end to end time can not be saved. Optimistically, both E_V and E_K are below 5% when $M = 400$ for $\alpha = 1.5$ and $M = 300$ for product data, under which settings our solution is more efficient than traditional Hadoop implementation.

To show scalability of our approach, and to examine the effects of key size N on the efficiency, we increase N from $100K$ to $5M$ with fixed input file size 10G in the following group of experiments. Both Traditional top-k approaches and BOMP with $k = 5$ are tested in each evaluation. As N increases, the IO and shuffling cost of the traditional approach increases, which slows down the query processing. The response time of our solution–BOMP–also increases with the increase of N. This is because of the overhead introduced from matrix operations when N is large. However, compared to the IO and shuffling cost, this overhead shows less impact on the overall efficiency compared to traditional approach. Thus, BOMP manages to effectively trade-off an increase of computation cost for a reduction of communication cost. As shown in Figure 12, with different N, our solution shows better efficiency over the traditional top-k approach.

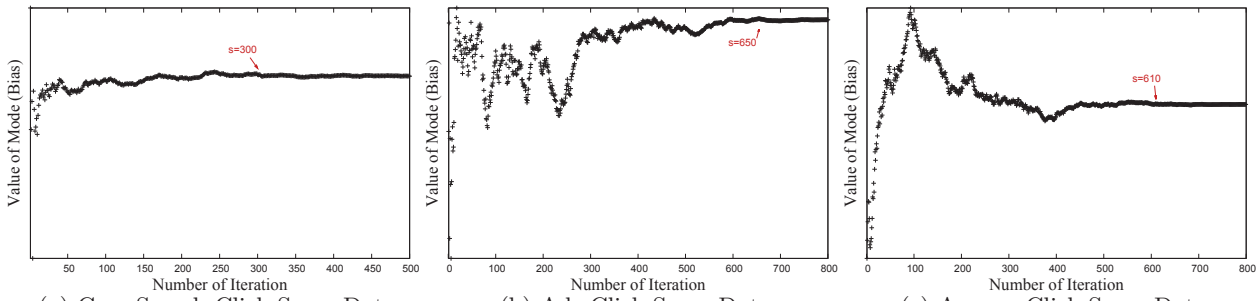
7. RELATED WORK

7.1 Theory of Distributed Outlier Detection

In the most basic version of distributed data statistics, the partial data $\mathbf{x}_l \in \{0,1\}^N$ is a binary vector. A well-known problem in this setting is the *set disjointness problem* [25] [36] [28] [1]. Suppose each of two nodes has a set of n elements S_1 and S_2, the problem asks the two nodes to exchange as few bits of information as possible to determine whether the two sets are disjoint (i.e. $S_1 \bigcap S_2 = \phi$). Razborov proved in [36] that $\Omega(n)$ bits are needed for any algorithm to solve set disjointness with constant success probability. Alon et al. [1] extended this linear lower bound to the multi-node scenario, showing that for L nodes with sets $|S_1| = \cdots = |S_L| = n$ and $n \geq L^4$, $\Omega(n/L^3)$ bits are necessary to know whether the sets $S_1 \ldots S_L$ are pairwise disjoint. These lower bounds also apply to problems in more general settings with non-binary data, and they have been used as building blocks to prove various other lower bounds.

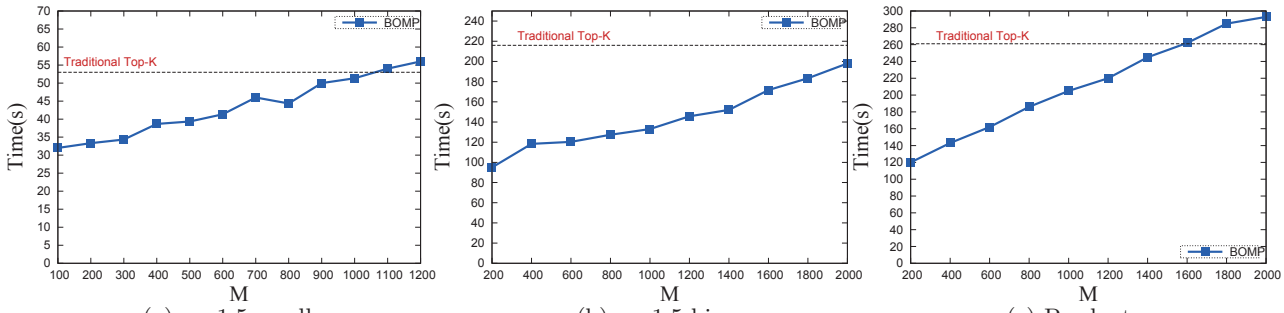
Numerous studies consider the communication complexity to learn statistical features of distributed *positive* data. In these works, the partial data $\mathbf{x}_l \in \mathbb{N}^N$ is a vector of natural numbers that typically corresponds to some counters over the set $\{1, \ldots, N\}$. Various features have been considered:

- Kuhn et al. [27] consider the aggregation data $\mathbf{x}$ as a frequency histogram of a sorted list $a_1 \leq a_2 \leq \cdots \leq a_m$



(a) Core Search Click Score Data (b) Ads Click Score Data (c) Answer Click Score Data

Figure 9: Mode in the recovery iterations



(a) α=1.5 small (b) α=1.5 big (c) Product

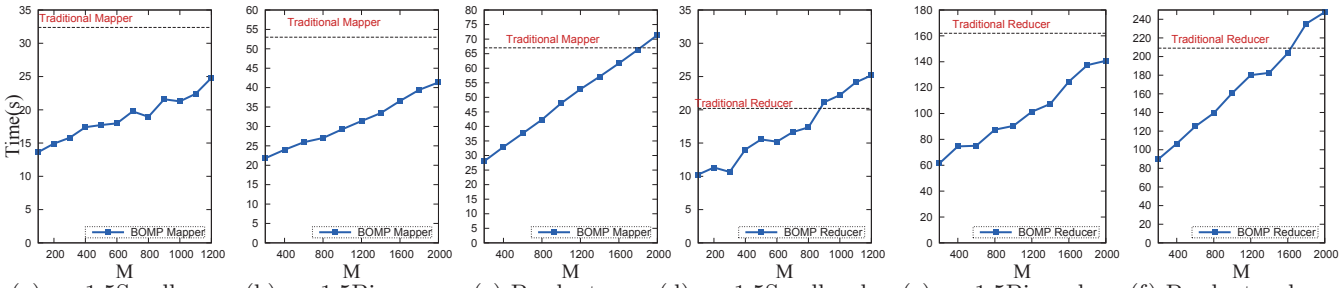
Figure 10: End-to-end time on Hadoop

(where $m = \sum_{i=1}^{N} x_i$ and $a_j \in \{1 \dots N\}$), and study the problem of determining the *median* element in this sorted list $\{a_j\}$, and more generally of determining the k^{th} smallest element a_k for any $1 \leq k \leq m$.

- Alon et al. [1] study the problem of determining the k-order *frequency moment* of the aggregation data $F_k = \sum_{i=1}^{N} (x_i)^k$. In this seminal paper, the authors consider the space complexity (i.e. how many bits of space are needed) to solve the problem in a streaming model (where the input data can only be processed in one pass), the results also apply to the communication complexity scenario in distributed computing. Specifically, their work adapts the lower bound technique in [28] and shows that for $k \geq 6$, at least $\Omega(N^{1-5/k})$ bits need to be exchanged between the nodes for any randomized algorithm in order to approximate F_k by $\hat{F}_k$ such that $\mathbb{P}[|\hat{F}_k - F_k| > 0.1F_k] < 1/2$. On the other hand, an $O(N^{1-1/k} \log N)$ algorithm was proposed in [1], which works in both the streaming model and the distributed model.
- A special case of the frequency moment problem is to determine the *sparsity* of the aggregation data $\mathbf{x}$, which asks how many components in $\mathbf{x}$ are non-zero (i.e. F_0). Again, the set disjointness lower bound implies a lower bound of $\Omega(N)$ for any algorithm to exactly solve the sparsity problem [35]. Efficient algorithms to compute F_0 in expectation were proposed in [21] [17].
- Finding the largest value in $\mathbf{x}$ is another problem closely related to the frequency moment problem in that $\lim_{k\to\infty} F_k = \max x_i$. Kuhn et al. point out that the lower bound of the set disjoint problem, again, applies to this problem, leading to a randomized lower bound of $\Omega(N/(x_{max})^5)$, where $x_{max} = \max x_i$ [26]. On the other hand, this work also proposes a randomized algorithm that solves the problem with high probability using $O(\frac{F_2}{x_{max}^2} \cdot \log N)$ bits. Note that $\frac{F_2}{x_{max}^2} \leq F_0$ is guaranteed to be less than the number of non-zero items in $\mathbf{x}$, and may break the linear lower bound when F_0 is sublinear to N. The key idea of the algorithm is to randomly partion all N values into two groups and sum up the values of keys assigned to the same group, so that the key with the largest value is more likely to stay in the group with larger sum. Repeating this routine (in parallel) multiple times can amplify this probabilistic fact to make the largest value stand out quickly.

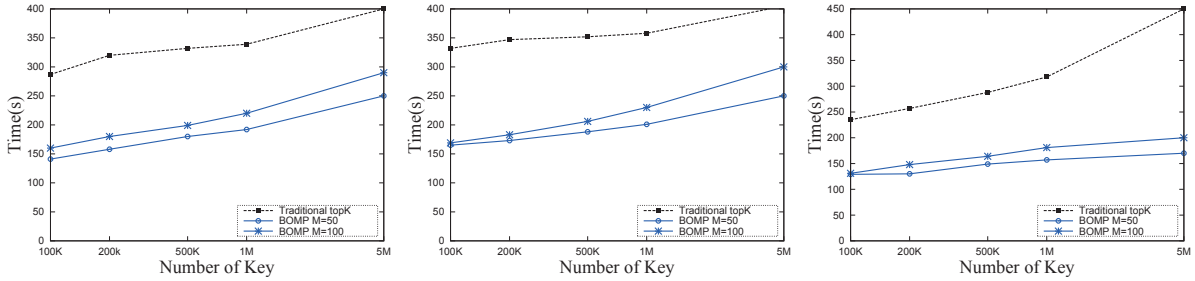
The *top-k problem* is a generalization of the maximum problem in which the k largest values in $\mathbf{x}$ have to be identified. This problem has been extensively studied in the database community assuming a limited number of nodes [18] [31] [3] [32] [4] [19]. In particular, a seminal work by Fagin et al. in [19] proposed the famous *Threshold Algorithm (TA)*, which was also independently discovered in [32] and [4]. The TA algorithm runs in multiple rounds. The idea is to calculate the SUM of the values ordered at the same rank in each node as a threshold. The algorithm stops when there are k keys whose SUM values are bigger than the threshold, because the threshold is an upper bound on the total value for the keys it has not seen.

Although working well in many database scenarios, the threshold algorithm suffers from limited scalability with respect to the number of nodes as it fundamentally runs in multiple rounds. Inspired by Fagin's work, Pei Cao and Zhe Wang [10] proposed the TPUT algorithm, which consists of three rounds: i) estimate the lower bound of the kth value, ii) prune keys using the lower bound and iii) exact top-k refinement. Charikar et al. [11] proposed an approximation algorithm that finds k keys with values at least $(1-\epsilon)x_k$ with high probability under the streaming model, using $O((k + F_2/x_k^2) \cdot \log N)$ bits of memory. This algorithm can also be adapted to approximate the top-k results between distributed nodes with $O((k + F_2/x_k^2) \cdot \log N)$ bits.



(a) α=1.5Small map (b) α=1.5Big map (c) Product map (d) α=1.5Small reduce (e) α=1.5Big reduce (f) Product reduce

Figure 11: Breakdown time on Hadoop



(a) End-to-end Efficiency (b) Map Efficiency (c) Reduce Efficiency

Figure 12: Efficiency with different key size N

Different from all work above, our work extends the partial data $\mathbf{x}_l \in \mathbb{R}^N$ to the real field. Particularly x_{ij} can be negative numbers. The top-k problem over the natural number field can be seen as a special case of the k-outlier problem over the real field as considered in this paper. However, note that the fact that the partial data can be negative in the k-outlier problem invalidates key assumptions made in [19, 10], namely that the *partial sum is a lower bound on the aggregated sum*. This means that TA and TPUT proposed for the top-k problem cannot be easily adapted to the k-outlier problem. Besides, the top-k values are not necessarily the k-outlier values when negative values are involved.

The top-1 algorithm proposed in [26] may be adapted to solve the k-outlier problem with $O(k^2 \log N)$ messages with high probability. However, this would require multiple iterations and rounds (specifically, k rounds) of communication. In contrast, in this paper we focus on a *non-adaptive* algorithm that finds all k outliers in a single round.

7.2 Compressive Sensing and Data Sketches

Compressive Sensing (CS) is a breakthrough in the signal processing community. Its basic ideas and mathematical foundations have been established in [8, 9, 15, 37]. CS is an efficient approach to sample and reconstruct *sparse* data, and has found many applications (e.g., photography [16], image reconstruction and recognition [39], and network traffic monitoring [43]). In our work, we use CS to compress and recover sparse data vectors.

One technique for reducing communication cost is data compression. Lossless compression usually does not help a lot when each slice contains a large number of different non-zero real-values. Alternatively, some existing approaches conduct lossy compression (also known as sketches) [12, 24, 22] on the slice in each node. However, these scattered slices could have different distributions or structures from each other. The traditional sketching techniques applied locally may encounter the risk of losing substantial information that is necessary for accurately answering the query on the global data.

7.3 Communication-Efficient System Design

Communication-efficient distributed computation of statistics is also attractive to distributed system design, e.g. [41]. Many efforts have been made to optimize query processing in MapReduce-type distributed systems. [33, 38] propose techniques designed for reducing the Mapper and Reducer's communication cost through job sharing. The authors of [29, 14, 40] apply sampling to reduce the MapReduce jobs processing cost. Although these works are not tailored for outlier computation, they could be adjusted and applied together with our solution to further speed up MapReduce jobs.

8. CONCLUSIONS

Production-level big data usually exhibits a sparse structure. It is attractive to leverage this sparsity to reduce the communication cost in various distributed aggregation algorithms, such as the detection of outliers, mode, top-k, and so on. However, the distribution skew among data slices allocated in a shared-nothing distributed system impedes traditional techniques that are based on first compressing (sketching) local slices separately and then issuing the query to these aggregated sketches. In this paper, we make a case that compressive sensing-based compression and recovery techniques may be ideally suited in such a distributed context, especially in MapReduce-type systems. Specifically, we show in this paper empirically and theoretically that such an approach can be beneficial for the distributed outlier detection problem. More generally, we believe that there will be many more applications of compressive sensing in distributed computing.

9. REFERENCES

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 20–29, New York, NY, USA, 1996. ACM.

[2] M. Andrecut. Fast gpu implementation of sparse signal recovery from random projections. *Engineering Letters*, 17(3):151–158, 2009.

[3] B. Babcock and C. Olston. Distributed top-k monitoring. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 28–39. ACM, 2003.

[4] W.-T. Balke and W. Kießling. Optimizing multi-feature queries for image databases. *VLDB,(Sep 2000)*, pages 10–14, 2000.

[5] R. Baraniuk, M. Davenport, R. DeVore, and M. Wakin. A simple proof of the restricted isometry property for random matrices. *Constructive Approximation*, 28(3):253–263, 2008.

[6] T. Blumensath and M. E. Davies. On the difference between orthogonal matching pursuit and orthogonal least squares, 2007.

[7] T. Bu, J. Cao, A. Chen, and P. P. Lee. A fast and compact method for unveiling significant patterns in high speed networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1893–1901. IEEE, 2007.

[8] E. J. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *Information Theory, IEEE Transactions on*, 52(2):489–509, 2006.

[9] E. J. Candes and T. Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *Information Theory, IEEE Transactions on*, 52(12):5406–5425, 2006.

[10] P. Cao and Z. Wang. Efficient top-k query calculation in distributed networks. In S. Chaudhuri and S. Kutten, editors, *PODC*, pages 206–215. ACM, 2004.

[11] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Automata, Languages and Programming*, pages 693–703. Springer, 2002.

[12] S. Chaudhuri. An overview of query optimization in relational systems. In *PODS*, pages 34–43, 1998.

[13] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM journal on scientific computing*, 20(1):33–61, 1998.

[14] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, J. Gerth, J. Talbot, K. Elmeleegy, and R. Sears. Online aggregation and continuous query support in mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1115–1118. ACM, 2010.

[15] D. L. Donoho. Compressed sensing. *Information Theory, IEEE Transactions on*, 52(4):1289–1306, 2006.

[16] M. F. Duarte, M. A. Davenport, D. Takhar, J. N. Laska, T. Sun, K. F. Kelly, and R. G. Baraniuk. Single-pixel imaging via compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):83–91, 2008.

[17] M. Durand and P. Flajolet. Loglog counting of large cardinalities. In *Algorithms-ESA 2003*, pages 605–617. Springer, 2003.

[18] R. Fagin. Combining fuzzy information from multiple systems. In *Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 216–226. ACM, 1996.

[19] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In P. Buneman, editor, *PODS*. ACM, 2001.

[20] Y. Fang, L. Chen, J. Wu, and B. Huang. Gpu implementation of orthogonal matching pursuit for compressive sensing. In *Proceedings of the 2011 IEEE 17th International Conference on Parallel and Distributed Systems*, ICPADS '11, pages 1044–1047, Washington, DC, USA, 2011. IEEE Computer Society.

[21] P. Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[22] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, SIGMOD '98, pages 331–342, New York, NY, USA, 1998. ACM.

[23] Z. Guo, X. Fan, R. Chen, J. Zhang, H. Zhou, S. McDirmid, C. Liu, W. Lin, J. Zhou, and L. Zhou. Spotting code optimizations in data-parallel pipelines through periscope. In *OSDI*, pages 121–133, 2012.

[24] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*, 40(4):11, 2008.

[25] B. Kalyanasundaram and G. Schintger. The probabilistic communication complexity of set intersection. *SIAM J. Discret. Math.*, 5(4):545–557, Nov. 1992.

[26] F. Kuhn, T. Locher, and S. Schmid. Distributed computation of the mode. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 15–24. ACM, 2008.

[27] F. Kuhn, T. Locher, and R. Wattenhofer. Tight bounds for distributed selection. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 145–153. ACM, 2007.

[28] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, New York, NY, USA, 1997.

[29] N. Laptev, K. Zeng, and C. Zaniolo. Early accurate results for advanced analytics on mapreduce. *Proceedings of the VLDB Endowment*, 5(10):1028–1039, 2012.

[30] S. G. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *Signal Processing, IEEE Transactions on*, 41(12):3397–3415, 1993.

[31] A. Marian, N. Bruno, and L. Gravano. Evaluating top-k queries over web-accessible databases. *ACM Transactions on Database Systems (TODS)*, 29(2):319–362, 2004.

[32] S. Nepal and M. Ramakrishna. Query processing issues in image (multimedia) databases. In *Data Engineering,*

1999. Proceedings., 15th International Conference on, pages 22–29. IEEE, 1999.

[33] T. Nykiel, M. Potamias, C. Mishra, G. Kollios, and N. Koudas. Mrshare: sharing across multiple queries in mapreduce. *Proceedings of the VLDB Endowment*, 3(1-2):494–505, 2010.

[34] Y. C. Pati, R. Rezaiifar, and P. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on*, pages 40–44. IEEE, 1993.

[35] B. Patt-Shamir. A note on efficient aggregate queries in sensor networks. *Theoretical Computer Science*, 370(1):254–264, 2007.

[36] A. A. Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390, 1992.

[37] J. A. Tropp and A. C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *Information Theory, IEEE Transactions on*, 53(12):4655–4666, 2007.

[38] G. Wang and C.-Y. Chan. Multi-query optimization in mapreduce framework. *Proceedings of the VLDB Endowment*, 7(3), 2013.

[39] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma. Robust face recognition via sparse representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(2):210–227, 2009.

[40] Y. Yan, L. J. Chen, and Z. Zhang. Error-bounded sampling for analytics on big sparse data. *PVLDB*, 7(13):1508–1519, 2014.

[41] J. Zhang, Y. Yan, L. J. Chen, M. Wang, T. Moscibroda, and Z. Zhang. Impression store: Compressive sensing-based storage for big data analytics. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*, Philadelphia, PA, June 2014. USENIX Association.

[42] J. Zhang, H. Zhou, R. Chen, X. Fan, Z. Guo, H. Lin, J. Y. Li, W. Lin, J. Zhou, and L. Zhou. Optimizing data shuffling in data-parallel computation by understanding user-defined functions. In *NSDI*, volume 12, pages 22–22, 2012.

[43] Y. Zhang, M. Roughan, W. Willinger, and L. Qiu. Spatio-temporal compressive sensing and internet traffic matrices. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 267–278. ACM, 2009.