Big Data Begets Big Database Theory*

Dan Suciu

University of Washington

1 Motivation

Industry analysts describe Big Data in terms of three V's: volume, velocity, variety. The data is too big to process with current tools; it arrives too fast for optimal storage and indexing; and it is too heterogeneous to fit into a rigid schema. There is a huge pressure on database researchers to study, explain, and solve the technical challenges in big data, but we find no inspiration in the three Vs. Volume is surely nothing new for us, streaming databases have been extensively studied over a decade, while data integration and semistructured has studied heterogeneity from all possible angles.

So what makes Big Data special and exciting to a database researcher, other for the great publicity that our field suddenly gets? This talk argues that the novelty should be thought along different dimensions, namely in communication, iteration, and failure.

Traditionally, database systems have assumed that the main complexity in query processing is the number of disk IOs, but today that assumption no longer holds. Most big data analysis simply use a large number of servers to ensure that the data fits in main memory: the new complexity metric is the amount of communication between the processing nodes, which is quite novel to database researchers.

Iteration is not that new to us, but SQL has adopted iteration only lately, and only as an afterthought, despite amazing research done on datalog in the 80s [1]. But Big Data analytics often require iteration, so it will play a center piece in Big Data management, with new challenges arising from the interaction between iteration and communication [2].

Finally, node failure was simply ignored by parallel databases as a very rare event, handled with restart. But failure is a common event in Big Data management, when the number of servers runs into the hundreds and one query may take hours [3].

The Myria project [4] at the University of Washington addresses all three dimensions of the Big Data challenge. Our premise is that each dimension requires a study of its fundamental principles, to inform the engineering solutions. In this talk I will discuss the communication cost in big data processing, which turns out to lead to a rich collection of beautiful theoretical questions; iteration and failure are left for future research.

* This work was partially supported by NSF IIS-1115188, IIS-0915054 and IIS-1247469.

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 1–5, 2013.

2 The Question

Think of a complex query on a big data. For example, think of a three-way join followed by an aggregate, and imagine the data is distributed on one thousand servers. How many communication rounds are needed to compute the query? Each communication round typically requires a complete reshuffling of the data across all 1000 nodes, so it is a very expensive operation, we want to minimize the number of rounds. For example, in MapReduce [5], a MR job is defined by two functions: `map` defines how the data is reshuffled, and `reduce` performs the actual computation on the repartitioned data. A complex query requires several MR jobs, and each job represents one global communication round. We can rephrase our question as: how many MR jobs are necessary to compute the given query? Regardless of whether we use MR or some other framework, fewer communication rounds mean less data exchanged, and fewer global synchronization barriers. The fundamental problem that we must study is: determine the minimum number of global communication rounds required to compute a given query.

3 The Model

MapReduce is not suitable at all for theoretical lower bounds, because it allows us to compute *any* query in *one* round: simply map all data items to the same intermediate key, and perform the entire computation sequentially, using one reducer. In other words, the MR model does not prevent us from writing a sequential algorithm, and it is up to the programmer to avoid that by choosing a sufficiently large number of reducers.

Instead, we consider the following simple model, called the Massively Parallel Communication (MPC) model, introduced in [6]. There are a fixed number of servers, p, and the input data of size n is initially uniformly distributed on the servers; thus, each server holds $O(n/p)$ data items. The computation proceeds in rounds, where each round consists a computation step and a global communication step. The communication is many-to-many, allowing a total reshuffling of the data, but with the restriction that each server receives only $O(n/p)$ amount of data. The servers have unrestricted computational power, and unlimited memory: but because of the restriction on the communication, after a constant number of rounds, each server sees only a fraction $O(n/p)$ of the input data. In this model we ask the question: given a query, how many rounds are needed to compute it? A naive solution that sends the entire data to one server is now ruled out, since the server can only receive $O(n/p)$ of the data.

A very useful relaxation of this model is one that allows each server to receive $O(n/p \times p^{\varepsilon})$ data items, where $\varepsilon \in [0, 1]$. Thus, during one communication round the entire data is replicated by a factor p^{ε}; we call ε the *space exponent*. The case $\varepsilon = 0$ corresponds to the base model, while the case $\varepsilon = 1$ is uninteresting, because it allows us to send the entire data to every server, like in the MapReduce example.

4 The Cool Example

Consider first computing a simple join: $q(x, y, z) = R(x, y), S(y, z)$. This can be done easily in one communication round. In the first step, every server inspects its fragment of R, and sends every tuple of the form $R(a_1, b_1)$ to the destination server with number $h(b_1)$, where h is a hash function returning a value between 1 and p; similarly, it sends every tuple $S(b_2, c_2)$ to server $h(b_2)$. After this round of communication, the servers compute the join locally and report the answers.

A much more interesting example is $q(x, y, z) = R(x, y),\ S(y, z),\ T(z, y)$. When all three symbols R, S, T denote the same relation, then the query computes all triangles in the data[1], a popular task in Big Data analysis [7]. Obviously, this query can be computed in two communication rounds, doing two separate joins. But, quite surprisingly, it can be computed in a single communication round! The underlying idea has been around for 20 years [8,9,7], but, to our knowledge, has not yet been deployed in practice. We explain it next.

To simplify the discussion, assume $p = 1000$ servers. Then each server can be uniquely identified as a triple (i, j, k), where $1 \leq i, j, k \leq 10$. Thus, the servers are arranged in a cube, of size $10 \times 10 \times 10$. Fix three independent hash functions h_1, h_2, h_3, each returning values between 1 and 10. During the single communication round, each server sends the following:

- $R(a_1, b_1)$ to the servers $(h_1(a_1), h_2(b_1), 1), \ldots, (h_1(a_1), h_2(b_1), 10)$
- $S(b_2, c_2)$ to the servers $(1, h_2(b_2), h_3(c_2)), \ldots, (10, h_2(b_2), h_3(c_2))$
- $T(c_3, a_3)$ to the servers $(h_1(a_3), 1, h_3(c_3)), \ldots, (h_1(a_3), 10, h_3(c_3))$

In other words, when inspecting $R(a_1, b_1)$ a server can compute the i and j coordinates of the destination (i, j, k), but doesn't know the k coordinate, and it simply replicates the tuple to all 10 servers. After this communication step, every server computes locally the triangles that it sees, and reports them. The algorithm is correct, because every potential triangle (a, b, c) is seen by some server, namely by $(h_1(a), h_2(b), h_3(c))$. Moreover, the data is replicated only 10 times. The reader may check that, if the number of servers is some arbitrary number p, then the amount of replication is $p^{1/3}$, meaning that the query can be computed in one communication round using a space exponent $\varepsilon = 1/3$.

It turns out that $1/3$ is the *smallest* space exponent for which we can compute q in one round! Moreover, a similar result holds for every conjunctive query without self-joins, as we explain next.

5 Communication Complexity in Big Data

Think of a conjunctive query q as a hypergraph. Every variable is a node, and every atom is an hyperedge, connecting the variables that occur in that atom. For our friend $R(x, y), S(y, z), T(z, y)$ the hypergraph is a graph with three nodes

[1] The query $\mathsf{q(x,y,z)} = \mathsf{R(x,y),R(y,z),R(z,x)}$ reports each triangle three times; to avoid double counting, one can modify the query to $\mathsf{q(x,y,z)} = \mathsf{R(x,y),R(y,z),R(z,x),x<y<z}$.

x, y, z and three edges, denoted R, S, T, forming a triangle. We will refer interchangeably to a query as a hypergraph.

A *vertex cover* of a query q is a subset of nodes such that every edge contains at least one node in the cover. A *fractional vertex cover* associates to each variable a number ≥ 0 such that for each edge the sum of the numbers associated to its variables is ≥ 1. The value of the fractional vertex cover is the sum of the numbers of all variables. The smallest value of any fractional vertex cover is called the *fractional vertex cover* of q and is denoted $\tau^*(q)$.

For example, consider our query $q(x, y, z) = R(x, y), S(y, z), T(z, y)$. Any vertex cover must include at least two variables, e.g. $\{x, y\}$, hence its value is 2. The fractional vertex cover $1/2, 1/2, 1/2$ (the numbers correspond to the variables x, y, z) has value $3/2$ and the reader may check that this is the smallest value of any fractional vertex cover; thus, $\tau^*(q) = 3/2$. The smallest space exponent needed to compute a query in one round is given by:

Theorem 1. *[6] If $\varepsilon < 1 - 1/\tau^*(q)$ then the query q cannot be computed in a single round on the MPC model with space exponent ε.*

To see another example, fix some $k \geq 2$ and consider a chain query $L_k = R_1(x_0, x_1), R_2(x_1, x_2), \ldots, R_k(x_{k-1}, x_k)$. Its optimal fractional vertex cover is $0, 1, 0, 1, \ldots$ where the numbers correspond to the variables $x_0, x_1, x_2, \ldots$ Thus, $\tau^*(L_k) = \lceil k/2 \rceil$, and the theorem says that L_k cannot be computed with a space exponent $\varepsilon < 1 - 1/\lceil k/2 \rceil$.

What about multiple rounds? For a fixed $\varepsilon \geq 0$, let $k_\varepsilon = 2\lfloor 1/(1-\varepsilon) \rfloor$; this is the longest chain L_{k_ε} computable in one round given the space exponent ε. Let $\mathrm{diam}(q)$ denote the diameter of the hypergraph q. Then:

Theorem 2. *[6] For any $\varepsilon \geq 0$, the number of rounds needed to compute q within a space exponent ε is at least $\lceil \log_{k_\varepsilon}(diam(q)) \rceil$.*

6 The Lessons

An assumption that has never been challenged in databases is that we always compute one join at a time[2]: a relational plan expresses the query as a sequence of simple operations, where each operation is either unary (group-by, selection, etc), or is binary (join). We never use plans with other than unary or binary operators, and never compute a three-way join directly. In Big Data this assumption needs to be revisited. By designing algorithms for multi-way joins, we can reduce the total communication cost for the query. The theoretical results discussed here show that we must also examine query plans with complex operators, which compute an entire subquery in one step. The triangle query is one example: computing it in one step requires that every table be replicated $p^{1/3}$ times (e.g. 10 times, when $p = 1000$), while computing it in two steps requires reshuffling the intermediate result $R(x, y), S(y, z)$, which, on a graph like twitter's `Follows` relation, is significantly larger than ten times the input table.

[2] We are aware of one exceptions: the Leap Frog join used by LogicBlox [10].

References

1. Ceri, S., Gottlob, G., Tanca, L.: What you always wanted to know about datalog (and never dared to ask). IEEE Trans. Knowl. Data Eng. 1(1), 146–166 (1989)
2. Bu, Y., Howe, B., Balazinska, M., Ernst, M.D.: The haloop approach to large-scale iterative data analysis. VLDB J. 21(2), 169–190 (2012)
3. Upadhyaya, P., Kwon, Y., Balazinska, M.: A latency and fault-tolerance optimizer for online parallel query plans. In: SIGMOD Conference, pp. 241–252 (2011)
4. Myria, `http://db.cs.washington.edu/myria/`
5. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: OSDI, pp. 137–150 (2004)
6. Beame, P., Koutris, P., Suciu, D.: Communication steps for parallel query processing. In: PODS (2013)
7. Suri, S., Vassilvitskii, S.: Counting triangles and the curse of the last reducer. In: WWW, pp. 607–614 (2011)
8. Ganguly, S., Silberschatz, A., Tsur, S.: Parallel bottom-up processing of datalog queries. J. Log. Program. 14(1&2), 101–126 (1992)
9. Afrati, F.N., Ullman, J.D.: Optimizing joins in a map-reduce environment. In: EDBT, pp. 99–110 (2010)
10. Veldhuizen, T.L.: Leapfrog triejoin: a worst-case optimal join algorithm. CoRR abs/1210.0481 (2012)