Characterization of Randomized Shuffle and Sort Quantifiability in MapReduce Model

Kiran M.
Member, Technical Staff
Data Analytics Research Lab
Bangalore, India

Saikat Mukherjee
Senior Software Engineer
HP Research Lab
Bangalore, India

Ravi Prakash G.
Department of CS/IT
Alliance University
Bangalore, India

ABSTRACT
Quantifiability is a concept in MapReduce Analytics based on the following two conditions: (a) a mapper should be cautious, that is, should not exclude any reducer's shuffle and sort strategy from consideration; and (b) a mapper should respect the reducers' shuffle and sort preferences, that is, should deem a reducer's shuffle and sort strategy k_i infinitely more likely than k'_i if it premises the reducer to prefer k_i to k'_i. A shuffle and sort strategy is quantifiable if it can optimally be chosen under common shuffle and sort conjecture in the events (a) and (b). In this paper we present an algorithm that for every finite MapReduce operation computes the set of all quantifiable shuffle and sort strategies. The algorithm is based on the new idea of a key-value preference limitation, which is a pair (k_i, V_i) consisting of a shuffle and sort strategy k_i, and a subset of shuffle and sort strategies V_i, for mapper i. The interpretation is that mapper i prefers some shuffle and sort strategy in V_i to k_i. The algorithm proceeds by successively adding key-value preference limitations to the MapReduce.

Keywords
MapReduce analytics, quantifiability, key-value, preference limitation, shuffle and sort, Totally Ordered Data-Intensive Systems.

1. INTRODUCTION
In a MapReduce, it is natural to assume that a mapper reasons about its reducers before making a decision. Namely, in order to evaluate the possible consequences of a decision, the mapper must form some shuffle and sort conjecture about its reducers' choices which, in turn, must be based on some shuffle and sort conjecture about its reducers' conjecture about their reducers' choices, and so on. It is the goal of *MapReduce Analytics* [1] to formally describe such reasoning processes, and to investigate their behavioral implications.

Throughout this paper we take a mapper set perspective to analyze MapReduce-theoretic situations. That is, we always view the MapReduce from the perspective of mapper set, and put restrictions only on the conjecture of this particular mapper set – including conjecture about the reducers' conjecture – without imposing restrictions on the actual conjecture of the reducers. We premise this approach to be plausible; as we cannot look inside the reducers at the time we make a key-value choice. So, can only base key-value choice on conjecture about the reducers, and not on the actual conjecture and key-value choices of reducers. But then, if we want to analyze the reasonable key-value choices a mapper can make in a MapReduce, it is sufficient to concentrate only on the conjecture of this particular mapper set, as they encompass everything that can be used to make a decision. Although we premise the mapper set perspective to be very natural, it crucially differs from the usual approach to MapReduce in papers and articles, which typically proceed by imposing restrictions on the conjecture of all mapper set, and not only mapper set.

Quantifiability is a concept within MapReduce Analytics that is based upon the following two assumptions:

- a mapper should be *cautious*, that is, a mapper should not exclude any reducer's shuffle and sort strategy from consideration;
- a mapper should respect the reducers' shuffle and sort preferences, that is, if the mapper premises that an reducer prefers shuffle and sort strategy k_i to shuffle and sort strategy k'_i, then the mapper should deem k_i much more likely k'_i.

Any shuffle and sort strategy that can be chosen optimally under common shuffle and sort conjecture in these two events is called *quantifiable*.

In order to define quantifiability formally we can no longer model the mappers' conjecture by standard probability distributions. Suppose, for instance, that mapper 1 premises that mapper 2 prefers shuffle and sort strategy a to shuffle and sort strategy b. If mapper 1's shuffle and sort conjecture about 2's choice would be modeled by a single probability distribution then mapper 1 should assign probability 0 to b, since it must respect 2's shuffle and sort preferences. This, however, would contradict the assumption that it is cautious.

A possible way to define quantifiability is by means of *sequences of probability distributions*, or by using *totally ordered Data-intensive systems* [9], [10], [12], [14] [11]. Both frameworks can model a state of mind in which you deem some reducer's shuffle and sort strategy k_i infinitely more likely than some other shuffle and sort strategy k'_i, without completely discarding the latter choice.

The practical disadvantage of these richer frameworks is that, it makes the computation of quantifiable shuffle and sort strategies rather difficult. This is probably also the reason that quantifiability, despite its strong intuitive appeal, has not received as much attention as many other concepts in MapReduce Analytics. It would therefore be very useful to have an algorithm helping us to compute this quantifiable shuffle and sort strategies. A procedure, called *iteratively trembling*, that for any given $\alpha > 0$ yields the set of α-quantifiable shuffle and sort strategies. By letting α tend to zero, we finally would obtain the set of quantifiable shuffle and sort strategies. So, in a sense, this procedure only *indirectly* leads to the set of quantifiable shuffle and sort strategies, as we first have to apply the procedure for a sequence of small α's, and then let α go to zero.

There is another algorithm designed for quantifiability, called *iterated backward inference.* This procedure does not exactly yield the set of quantifiable shuffle and sort strategies, as its output may contain shuffle and sort strategies that are not quantifiable. The output, however, always *includes* the set of quantifiable shuffle and sort strategies.

In this paper we present an algorithm, called *iterated addition of key-value preference limitations* that *directly* delivers the set of all quantifiable shuffle and sort strategies in every finite MapReduce operation [17], [19]. The algorithm is based on the new notion of a *key-value preference limitation*. Formally, a key-value preference limitation for mapper i is a pair (k_i, V_i), where k_i is a shuffle and sort strategy and V_i a subset of shuffle and sort strategies for mapper i. The interpretation is that mapper i prefers some shuffle and sort strategy V_i in to k_i, without specifying which one (unless V_i contains only one shuffle and sort strategy, of course). A *totally ordered shuffle and sort conjecture* for mapper i about its reducers' shuffle and sort strategies is a finite sequence $\psi_i = (\psi_i^1, \ldots, \psi_i^P)$ of probability distributions on K_{-i}, the set of reducers' shuffle and sort strategy combinations, such that every shuffle and sort strategy combination k_{-i} in K_{-i} receives positive probability under some probability distribution ψ_i^p in this sequence. For every $p \in \{1, \ldots, P\}$, we call ψ_i^p the level p shuffle and sort conjecture.

The totally ordered shuffle and sort conjecture ψ_i deems some shuffle and sort strategy combination k_{-i} *infinitely more likely* than some other shuffle and sort strategy combination k'_{-i} if there is some level p such that k_{-i} receives positive probability under the level p shuffle and sort conjecture ψ_i^P, whereas k'_{-i} receives probability zero under the first p levels. We say that ψ_i *respects* a key-value preference limitation (k_j, V_j), for reducer j if it deems some shuffle and sort strategy in V_j infinitely more likely than k_j. This thus mimics the condition in quantifiability that i must respect j's shuffle and sort preferences. The totally ordered shuffle and sort conjecture ψ_i is said to *assume* a subset $E_{-i} \subseteq K_{-i}$ of shuffle and sort strategy combinations if it deems every element in E_{-i} infinitely more likely than every element outside E_{-i}.

The algorithm we present proceeds by inductively adding key-value preference limitations [6], [2], [4], [3], [8], [5], until no further key-value preference limitations can be produced. At round 1, we start with the empty set of key-value preference limitations for all mappers. In every subsequent round, we add a key-value preference limitation (k_i, V_i) for mapper i if every totally ordered shuffle and sort conjecture on K_{-i} that respects all current key-value preference limitations for i's reducers, assumes some subset $E_{-i} \subseteq K_{-i}$ on which k_i is weakly dominated by some randomized shuffle and sort strategy on V_i. We continue this process until no further key-value preference limitation can be added. Among the final set of key-value preference limitations for mapper i, we look for those shuffle and sort strategies k_i that are not part of any key-value preference limitation (k_i, V_i). We show that these shuffle and sort strategies are exactly the quantifiable shuffle and sort strategies for mapper i.

So, at every round the algorithm produces, for each mapper, a set of key-value preference limitations. As the set of key-value preference limitations can only grow at every round, and there are only finitely many possible key-value preference limitations, the algorithm must stop after finitely many rounds.

Not only can this algorithm be used to *compute* the quantifiable shuffle and sort strategies in a MapReduce, it also represents a natural *inductive reasoning procedure* for the mappers that eventually lead them to quantifiable shuffle and sort choices. The central object in this reasoning process is that of a key-value preference limitation. If we add a key-value preference limitation (k_i, V_i) for mapper i, then normally this means that i's reducers premises that i prefers some shuffle and sort strategy in V_i to k_i. Moreover, if i's reducers respect i's shuffle and sort preferences, as we assume in quantifiability, then i's reducers will also deem some shuffle and sort strategy in V_i infinitely more likely than k_i. Thus, by adding key-value preference limitations at every round, we further and further limit the possible totally ordered conjecture that mappers can plausibly hold about their reducers' choices. In a sense, what the algorithm shows is that, in order to reason your way toward quantifiable shuffle and sort strategies, it is sufficient to keep track of the mappers' key-value preference limitations. At every round, by considering the current key-value preference limitations, we can possibly derive new key-value preference limitations, thus further limitations the mappers' possible totally ordered conjecture, until this reasoning process cannot produce any new key-value preference limitations. This is where the reasoning procedure ends, and by looking at the final key-value preference limitations we can find the entire quantifiable shuffle and sort strategies in the MapReduce.

In the algorithm we present, the objects of output are different than in previous procedure. There, the procedure delivers at every round and for every mapper i, a set of full support probability distributions on mapper i's shuffle and sort strategy, where this set becomes smaller with every round. As there are infinitely many possible sets of full support probability distributions, previous procedure can produce infinitely many possible outputs in every round. This is a major difference with the algorithm we propose, where at every round there are only a finite number of possible outputs, namely the key-value preference limitations at that round.

Note also that the algorithm in this paper is fundamentally different from most other inductive concepts in MapReduce Analytics, which usually proceed by successively eliminating shuffle and sort strategies from the MapReduce. Think, for instance, of iterated elimination of strictly (weakly) dominated shuffle and sort strategies. So, why did we not base the algorithm on elimination of shuffle and sort strategies as well? The reason is that iterated elimination of strategies cannot work for quantifiability. In Section 2 we provide an algorithm for quantifiability must necessarily be of a different nature than the ones we are used to.

The outline of the paper is as follows. In Section 2 we show, why successive elimination of shuffle and sort strategies does not work for quantifiability. In Section 3 we give a formal necessary and sufficient condition of quantifiability, by making use of totally ordered Data-intensive systems [9], [10], [12], [14] [11]. In Section 4 we present the algorithm, illustrate it by means of our main proposition showing that the algorithm produces exactly the set of quantifiable shuffle and sort strategies. In Section 5 we discuss some important properties of the algorithm: We show how the algorithm can be viewed as a natural inductive reasoning procedure, and explain why the order in which we add key-value preference limitations does not matter for the eventual output. In section 6 we include conclusion and future scope.

2. WHY ELIMINATION OF SHUFFLE AND SORT STRATEGIES DOES NOT WORK

Most algorithms in the MapReduce Analytics literature [1] proceed by successively modifying shuffle and sort *strategies* from the operation cycle. Think, for instance, of iterated elimination of strictly (weakly) dominated shuffle and sort strategies. As announced, the algorithm we propose for quantifiability is of a different nature since it is based on successively adding *key-value preference limitations* rather than eliminating shuffle and sort *strategies*. A natural question is why we do not stick to the process of eliminating shuffle and sort strategies here. In this section we show why elimination of shuffle and sort strategies does not work for quantifiability.

Let us first be precise about the class of shuffle and sort strategy elimination procedures we consider. All the elimination procedures mentioned above have in common that at each round, only weakly dominated shuffle and sort strategies in the reducer cycle of MapReduce cycle [18], [20] (but not necessarily all) are eliminated. Now, say that a shuffle and sort strategy elimination procedure is *regular* if at every round, it eliminates a (possibly empty) subset of the set of weakly dominated shuffle and sort strategies in the reducer of MapReduce cycle [18], [20].

3. NECESSARY AND SUFFICIENT CONDITION OF QUANTIFIABILITY

3.1 Totally ordered Data-intensive systems

Totally ordered Data-intensive systems have been formally introduced as a possible way to represent a decision maker's shuffle and sort conjecture about the data-intensive state of the data-intensive world. The essential feature is that it allows the decision maker to deem one data-intensive state much more likely (in fact, infinitely more likely) than some other data-intensive state, without completely ignoring the latter data-intensive state when making a decision.

More formally, let N be some finite set of data-intensive states. By $\theta(N)$ we denote the set of all probability distributions on N. A *Totally ordered Data-intensive systems* (TODIS) on N is a finite sequence of probability distributions

$$\psi = (\psi^1, \psi^2, \ldots, \psi^P),$$

with $\psi^p \in \theta(N)$ for all $p \in \{1, \ldots, P\}$. We refer to ψ^1 as the decision maker's *level* 1 *shuffle and sort conjecture*, to ψ^2 as its *level* 2 *shuffle and sort conjecture*, and so on. The interpretation is that the decision maker attaches much more importance to its level 1 shuffle and sort conjecture than to its level 2 shuffle and sort conjecture, attaches much more importance to its level 2 shuffle and sort conjecture than to its level 3 shuffle and sort conjecture, and so on, without completely discarding any of these conjecture. For every data-intensive state $n \in N$, let $lp(n,\psi)$ be the first level p for which $\psi^p(n)>0$. If $\psi^p(n) = 0$ for every $p \in \{1, \ldots, P\}$, set $lp(n,\psi) = \infty$. We call $lp(n,\psi)$ the *rank* of data-intensive state n within the TODIS ψ. We say that the TODIS ψ deems data-intensive state n *infinitely more likely* than some other data-intensive state *Proposition* if n has a lower rank that *Proposition*.

3.2 MapReduce Programming Model

Consider a finite static MapReduce $\delta = (K_i, x_i)_{i \in I}$ where I is the finite set of mappers, the finite set K_i denotes the set of strategies for mapper i, and $x_i : \prod_{j \in I} K_j \rightarrow \mathrm{F}$ denotes mapper i's utility function. We assume that mapper i does not only have a shuffle and sort conjecture about its reducers' shuffle and sort strategy choices, but also about the possible conjecture that its reducers could have about the other mappers' shuffle and sort strategy choices, and about the possible conjecture that the reducers could have about the possible conjecture that their reducers could have about the other mappers' shuffle and sort strategy choices, and so on. That is, mapper i hold a full *shuffle and sort conjecture hierarchy* about the reducers' choices and the reducers' conjecture. If we assume, moreover, that each of the conjecture in this hierarchy can be represented by a TODIS, this leads to the following MapReduce programming model [7], [13], [15], [16].

Necessary and sufficient condition 3.1 *(MapReduce programming model).* A finite MapReduce programming model for the MapReduce δ is a tuple $(T_i, \psi_i)_{i \in I}$ where, for all mappers i, T_i is a finite set of key-value types, and ψ_i is a function that assigns to every key-value type $t_i \in T_i$ some TODIS $\psi_i(t_i)$ on the set $K_{-i} \times T_{-i}$ of reducers' strategy–key-value type combinations.

Here, $K_{-i} := \prod_{j \neq i} K_j$ denotes the set of reducers' strategy combinations, and $T_{-i} := \prod_{j \neq i} T_j$ the set of reducers' key-value type combinations. The interpretation is that $\psi_i(t_i)$ represents the shuffle and sort conjecture that key-value type t_i has about its reducers' choices and conjecture. For instance, the marginal of $\psi_i(t_i)$ on P_j represents the shuffle and sort conjecture that t_i has about reducer j's choice. Since every reducer's type t_j holds a shuffle and sort conjecture about the other mappers' choices, we can derive from $\psi_i(t_i)$ as well the shuffle and sort conjecture that key-value type t_i has about the shuffle and sort conjecture that mapper j has about its reducers' choices, and so on. In fact, from $\psi_i(t_i)$ we can derive the full shuffle and sort conjecture hierarchy that mapper i has about its reducers' choices and conjecture.

The reader may wonder why we limit attention to MapReduce programming models with *finitely* many key-value types for every mapper. In principle, we could allow for infinitely many key-value types for every mapper, and define quantifiability for such infinite MapReduce programming models. But it can be shown that every quantifiable strategy in a finite MapReduce can be supported by a quantifiable key-value type within a MapReduce programming model with *finitely* many key-value types only. So, we do not "overlook" any quantifiable strategies by concentrating on finite key-value type spaces only. As working with finite sets of key-value types makes things easier, we have decided to solely concentrate on finite MapReduce programming models in this paper.

Note that within a MapReduce programming model, the totally ordered shuffle and sort conjecture $\psi_i(t_i) = (\psi_i^1, \ldots, \psi_i^P)$ of a key-value type t_i is, mathematically speaking, an TODIS on the set of data-intensive states $K_{-i} \times T_{-i}$. For every reducers' shuffle and sort strategy–key-value type combination $(k_{-i}, t_{-i}) \in K_{-i} \times T_{-i}$, we can thus define the *rank* $lp((k_{-i}, t_{-i}), \psi_i(t_i))$ of (k_{-i}, t_{-i}) within $\psi_i(t_i)$, being the lowest level p such that $\psi_i^p(k_{-i}, t_{-i})>0$. Remember that, by convention, $lp((k_{-i}, t_{-i}), \psi_i(t_i))=\infty$ whenever (k_{-i}, t_{-i}) does not receive positive probability anywhere in $\psi_i(t_i)$. We say that

key-value type t_i deems the shuffle and sort strategy–key-value type combination (k_{-i}, t_{-i}) *infinitely more likely* than some other combination (k'_{-i}, t'_{-i}) if the rank of (k_{-i}, t_{-i}) is lower than the rank of (k'_{-i}, t'_{-i}).

Similarly, we can define for every *event* $A \subseteq K_{-i} \times T_{-i}$ of reducers' shuffle and sort strategy–key-value type combinations the associated rank by

$$lp(A,\psi_i(t_i)) = min\{l((k_{-i}, t_{-i}), \psi_i(t_i)) | (k_{-i}, t_{-i}) \in A\}.$$

Hence, the rank of A is the lowest level p such that ψ_i^p assigns positive probability to some element in A. This necessary and sufficient condition then allows us to define the rank of an *individual* reducer's shuffle and sort strategy–key-value type pair (k_j, t_j), simply by taking the rank of the event

$$\{k_j\} \times \prod_{p \neq i,j} K_p \times \{t_j\} \times \prod_{p \neq i,j} T_p.$$

So, we first take the marginal of the TODIS $\psi_i(t_i)$ on $K_j \times T_j$, and then take the rank of (k_j, t_j) inside this marginal TODIS. In a similar fashion, we can also define the rank of an individual reducer's key-value type t_j, and of an individual reducer's shuffle and sort strategy k_j. As such, we can formally data-intensive state expressions like "$\psi_i(t_i)$ deems (k_j, t_j) infinitely more likely than(k'_j, t'_j) for reducer j" or "$\psi_i(t_i)$ deems k_j infinitely more likely than k'_j for reducer j", which means that the rank of the former is smaller than the rank of the latter.

We say that key-value type t_i *deems possible* some event $A \subseteq K_{-i} \times T_{-i}$ if there is some level p with $\psi_i^p(A) > 0$. That is, A is deemed possible if and only if $lp(A, \psi_i(t_i)) \neq \infty$. Since we have defined the rank also for individual shuffle and sort strategy–key-value type pairs (k_j, t_j) and for individual key-value types t_j, we can also formally define the event that key-value type t_i deems possible a shuffle and sort strategy–key-value type pair (k_j, t_j) for reducer j, and that t_i deems possible an reducer's key-value type t_j. It simply means that the associated rank is not ∞.

3.3 Cautious key-value Types

Intuitively, *caution* means that the mapper should not fully exclude any reducer's key-value choice from consideration. The formal necessary and sufficient condition is, however – in data-intensive states that a key-value type t_i should not exclude any strategy choice *for any reducer's key-value type t_j considers possible*. Hence, for every shuffle and sort conjecture hierarchy that t_i deems possible for its reducer j, and for every quantifiable strategy k_j that j can possibly choose, key-value type t_i should deem possible the event that its reducer holds this shuffle and sort conjecture hierarchy *and* chooses k_j.

Necessary and sufficient condition 3.2 *(Cautious key-value type).* Consider a MapReduce programming model with sets of key-value types T_i for every mapper i. Key-value type $t_i \in T_i$ is cautious if, for every reducer j, every key-value type $t_j \in T_j$ it considers possible, and every shuffle and sort strategy choice $k_j \in K_j$, key-value type t_i deems possible the strategy–key-value type pair (k_j, t_j).

3.4 Respecting the reducers' shuffle and sort preferences

The key condition for quantifiability is that a key-value type should *respect its reducers'* key-value *shuffle and sort preferences.* In words it means that, whenever key-value type t_i premises that its reducer j prefers some shuffle and sort strategy k_j to some other shuffle and sort strategy k'_j, then it should deem k_j infinitely more likely than k'_j. We must first define what it means, within our MapReduce programming model, that a key-value type prefers some shuffle and sort strategy to another shuffle and sort strategy.

Consider a key-value type t_i with an TODIS $\psi_i(t_i) = (\psi_i^1, \ldots, \psi_i^P)$ on $K_{-i} \times T_{-i}$. Then, for every level $p \in \{1, \ldots, P\}$ and every shuffle and sort strategy k_i, we can define the *level p expected utility*

$$x_i(k_i, \psi_i^p) := \sum_{(k_{-i}, t_{-i}) \in K_{-i} \times T_{-i}} \psi_i^P(k_{-i}, t_{-i})\, x_i(k_i, k_{-i}).$$

This is the expected utility that would result by choosing k_i under the shuffle and sort conjecture ψ_i^P.

Necessary and sufficient condition 3.3 *(A key-value type's preference relation over* shuffle and sort *strategies).* Let $t_i \in T_i$ be a key-value type with TODIS $\psi_i(t_i) = (\psi_i^1, \ldots, \psi_i^P)$ on $K_{-i} \times T_{-i}$. Key-value type t_i prefers shuffle and sort strategy k_i to some other shuffle and sort strategy k'_i if there is some level $p \in \{1, \ldots, P\}$ such that $x_i(k_i, \psi_i^o) > x_i(k'_i, \psi_i^o)$ and $x_i(k_i, \psi_i^o) = x_i(k'_i, \psi_i^o)$ for all $o < p$.

For later purposes, we say that key-value type t_i *weakly prefers k_i to k'_i* if t_i does not prefer k'_i to k_i.

Necessary and sufficient condition 3.4 *(Respecting the reducers' shuffle and sort preferences).* Let $t_i \in T_i$ be a cautious key-value type. Key-value type t_i respects the reducer's shuffle and sort preferences if, for every reducer j, every key-value type $t_j \in T_j$ deemed possible by t_i, and every two shuffle and sort strategies k_j, k'_j such that t_j prefers k_j to k'_j, key-value type t_i deems the pair (k_j, t_j) infinitely more likely than the pair (k'_j, t_j).

3.5 Quantifiability

We say that a key-value type t_i is *quantifiable* if t_i is cautious and respects the reducers' shuffle and sort preferences, premises that all reducers are cautious and respect their reducers' shuffle and sort preferences, premises that all reducers premise that their reducers are cautious and respect their reducers' shuffle and sort preferences, and so on. In other words, t_i is cautious and respects the reducers' shuffle and sort preferences, and expresses common shuffle and sort conjecture in the event that mappers are cautious and respect the reducers' shuffle and sort preferences.

Necessary and sufficient condition 3.5 *(Common shuffle and sort conjecture in "caution and respect of the reducers' shuffle and sort preferences").* A key-value type t_i expresses common shuffle and sort conjecture in the event that mappers are cautious and respect the reducers' shuffle and sort preferences if t_i only deems possible reducers' key-value types that are cautious and respect their reducers' shuffle and sort preferences, only deems possible reducers' key-value types that only deem possible reducers' key-value types that are cautious and respect their reducers' shuffle and sort preferences, and so on.

By additionally assuming that t_i itself is cautious and respects the reducers' shuffle and sort preferences, we obtain the necessary and sufficient condition of a quantifiable key-value type.

Necessary and sufficient condition 3.6 *(quantifiable key-value type).* A key-value type t_i is quantifiable if it is cautious and respects the reducers' shuffle and sort preferences, and moreover expresses common shuffle and sort conjecture in the event that mappers are cautious and respect the reducers' shuffle and sort preferences.

Finally, we say that a shuffle and sort strategy k_i is quantifiable for mapper i if it is optimal for some quantifiable key-value type. Formally, a shuffle and sort strategy k_i is called *optimal* for key-value type t_i if t_i weakly prefers k_i to any other shuffle and sort strategy.

Necessary and sufficient condition 3.7 *(quantifiable* shuffle and sort *strategy).* A shuffle and sort strategy k_i for mapper i is quantifiable if there is some finite MapReduce programming model $(T_i, \psi_i)_{i \in I}$ and some quantifiable key-value type $t_i \in T_i$ such that k_i is optimal for t_i.

As we already mentioned before, the concept of a quantifiable shuffle and sort strategy would not change if we would allow for infinite MapReduce programming models here.

4. ALGORITHM

In this section we will present an algorithm that always delivers all quantifiable shuffle and sort strategies. Before doing so, we first provide some intuitive arguments that eventually will lead to the algorithm. Finally, we state our main result, namely that the algorithm yields precisely the set of quantifiable shuffle and sort strategies in every MapReduce.

4.1 Road to the Algorithm

In Section II we have seen that elimination of (subsets of) weakly dominated shuffle and sort strategies cannot work for quantifiability. So, what kind of procedure *could* work here? We start our informal investigation with the following well-known fact:

Step 1. Suppose that shuffle and sort strategy k_i is weakly dominated on K_{-i} by some randomized shuffle and sort strategy $\gamma_i \in \theta(V_i)$, where V_i is a subset of shuffle and sort strategies. Then, if mapper i is cautious, it will prefer some shuffle and sort strategy in V_i to k_i. We say that (k_i, V_i) is a shuffle and sort *key-value preference limitation* for mapper i.

Here, $\theta(V_i)$ denotes the set of probability distributions on V_i. The reason for this fact is simple: If k_i is weakly dominated by shuffle and sort γ_i, then under every cautious totally ordered shuffle and sort conjecture, k_i will be worse than γ_i, and hence there must be some $v_i \in V_i$ which is better than k_i under such a cautious totally ordered shuffle and sort conjecture. So, (k_i, V_i) will be a shuffle and sort key-value preference limitation for mapper i.

Suppose now that mapper i premise its reducers are cautious and that it respects its reducers' shuffle and sort preferences. If some reducer's shuffle and sort strategy k_j is weakly dominated on K_{-j} by some randomized shuffle and sort strategy $\gamma_i \in \theta(V_i)$, then we know by Step 1 that mapper j will prefer some shuffle and sort strategy in V_j to k_j in case it is cautious. As mapper i indeed premises it is cautious, and respects j's shuffle and sort preferences, mapper i must deem some shuffle and sort strategy in V_j infinitely more likely than k_j. We say that mapper i's totally ordered shuffle and sort conjecture respects the preference limitation (k_j, V_j). This leads to the following observation:

Step 2. Suppose mapper i premises its reducers are cautious, and respects its reducers' shuffle and sort preferences. Then, i's totally ordered shuffle and sort conjecture must respect every reducer's shuffle and sort key-value preference limitation (k_j, V_j) generated in Step 1.

Say that a totally ordered shuffle and sort conjecture for mapper i *assumes* a set $E_{-i} \subseteq K_{-i}$ of reducers' shuffle and sort strategy combinations if it deems all shuffle and sort strategy combinations inside E_{-i} infinitely more likely than all shuffle and sort strategy combinations outside E_{-i}. Suppose now that i's totally ordered shuffle and sort conjecture is cautious, and assumes some set E_{-i} of reducers' shuffle and sort strategy combinations. Assume, moreover, that its shuffle and sort strategy k_i is weakly dominated on E_{-i} by a randomized shuffle and sort strategy $\gamma_i \in \theta(V_i)$. Then, i must prefer some shuffle and sort strategy in V_i to k_i. The argument is basically the same as for Step 1, if we would "reduce" the MapReduce to reducers' shuffle and sort strategy combinations in E_{-i}. We thus obtain the following step:

Step 3. Suppose that every totally ordered shuffle and sort conjecture for mapper i respecting all key-value preference limitations from Step 1, assumes some $E_{-i} \subseteq K_{-i}$ on which k_i is weakly dominated by some $\gamma_i \in \theta(V_i)$. Suppose, moreover, that mapper i is cautious, premises its reducers are cautious, and respects the reducers' shuffle and sort preferences. Then, i must prefer some shuffle and sort strategy in V_i to k_i. We say that (k_i, V_i) *is* a *new key-value preference limitation* for mapper i.

Of course, we can iterate this argument if we assume that mapper i is cautious, respects the reducers' shuffle and sort preferences, and expresses common shuffle and sort conjecture in the event that mappers are cautious and respect the reducers' shuffle and sort preferences. That is, if we assume that mapper i's key-value type is quantifiable. The inductive step would then look as follows:

Inductive step. Suppose that every totally ordered shuffle and sort conjecture for i that respects all key-value preference limitations generated so far, assumes some $E_{-i} \subseteq K_{-i}$ on which k_i is weakly dominated by some $\gamma_i \in \theta(V_i)$. Then, if i is of a quantifiable key-value type, it must prefer some shuffle and sort strategy in V_i to k_i. So, (k_i, V_i) would be a new key-value preference limitation for mapper i.

This would thus generate an inductive procedure in which at every step (possibly) some new key-value preference limitations would be added for the mappers. Since there are only finitely many possible key-value preference limitations for the mappers, this procedure must end after finitely many steps. Now, consider some mapper i, and its set of key-value preference limitations generated by the procedure above.

If mapper i is of some quantifiable key-value type, we know from our arguments above that it will never choose a shuffle and sort strategy k_i if it is part of some key-value preference limitation (k_i, V_i). In that case, namely, it would always prefer some shuffle and sort strategy in V_i to k_i, so k_i could not be optimal.

So, the procedure above rules out shuffle and sort strategies that is certainly not quantifiable. But what about the converse? So, what about shuffle and sort strategies that are not ruled out by the procedure above? The main proposition in this paper, Proposition 4.6, will show that the "surviving" shuffle and sort strategies are all quantifiable! Hence, the procedure above will always select *exactly* those shuffle and sort strategies that are quantifiable – not more and not less.

4.2 Description of the algorithm

Before we state the algorithm, we first formally necessary and sufficient condition the new concepts we described above, such as key-value preference limitations, what it means for a totally ordered shuffle and sort conjecture to respect a key-value preference limitation, and so on.

Necessary and sufficient condition 4.1 *(Key-value preference limitation).* A key-value preference limitation for mapper i is a pair (k_i, V_i) where k_i is a shuffle and sort strategy, and V_i a nonempty subset of shuffle and sort strategies.

The interpretation is that mapper i prefers at least one shuffle and sort strategy from V_i to k_i. Now, consider a totally ordered shuffle and sort conjecture ψ_i on K_{-i}, which is simply a TODIS on K_{-i}. From here on, we will always assume that such a totally ordered shuffle and sort conjecture ψ_i has full support on K_{-i}, that is, every shuffle and sort strategy combination in K_{-i} receives positive probability in some level of ψ_i.

Necessary and sufficient condition 4.2 *(Respecting a key-value preference limitation).*A totally ordered shuffle and sort conjecture ψ_i on K_{-i} respects a key-value preference limitation (k_j, V_j) *for* mapper j if ψ_i deems some shuffle and sort strategy in V_j infinitely more likely than k_j.

This, in a sense, mimics the requirement that mapper i must respect j's shuffle and sort preferences.

Necessary and sufficient condition 4.3 *(Assuming a set of reducers' shuffle and sort strategy combinations).* Consider a subset $E_{-i} \subseteq K_{-i}$ of reducers' shuffle and sort strategy combinations, and a totally ordered shuffle and sort conjecture ψ_i on K_{-i}. The totally ordered shuffle and sort conjecture ψ_i assumes the set E_{-i} if ψ_i deems all shuffle and sort strategy combinations inside E_{-i} infinitely more likely than all shuffle and sort strategy combinations outside E_{-i}.

Note that a totally ordered shuffle and sort conjecture $\psi_i = (\psi_i^1, \ldots, \psi_i^P)$ on K_{-i} assumes a subset $E_{-i} \subseteq K_{-i}$ if and only if, there is some level $p \in \{1, \ldots, P\}$ such that $\bigcup_{o \leq p} \text{supp}(\psi_i^o) = E_{-i}$. Here, $\text{supp}(\psi_i^o)$denotes the support of the probability distribution ψ_i^o. A *randomized shuffle and sort strategy* for mapper i is a probability distribution $\gamma_i \in \theta(K_i)$.on mapper i's shuffle and sort strategies. For a subset $V_i \subseteq K_i$, we denote by $\theta(V_i)$ the set of randomized shuffle and sort strategies that assign positive probability only to shuffle and sort strategies in V_i. For some reducers' shuffle and sort strategy combination $k_{-i} \in K_{-i}$, let

$$x_i(\gamma_i, k_{-i}) := \sum_{k_i \in K_i} \gamma_i(k_i)\, x_i(k_i, k_{-i})$$

denote i's expected utility from the randomized shuffle and sort strategy γ_i and the reducers' shuffle and sort strategy combination k_{-i}.

Necessary and sufficient condition 4.4 *(Weakly dominated shuffle and sort strategy).* Let $E_{-i} \subseteq K_{-i}$ be a subset of the reducers' shuffle and sort strategy combinations. Shuffle and Sort Strategy k_i is said to be weakly dominated by randomized shuffle and sort strategy γ_i on E_{-i} if $x_i(\gamma_i, k_{-i}) \geq x_i(k_i, k_{-i})$ for all $k_{-i} \in E_{-i}$, with strict in equality for at least some $k_{-i} \in E_{-i}$.

We are now ready to present the algorithm. The idea is to start with the empty set of key-value preference limitations for all mappers, and at every round to add new key-value preference limitations, if possible. For that reason, the algorithm is called "iterated addition of key-value preference limitations".

Algorithm 4.5 *(Iterated addition of key-value preference limitations).* In round 1, begin for all mappers i with the empty set of key-value preference limitations.

At every further round $q \geq 2$, limit for every mapper i to those totally ordered shuffle and sort conjecture on K_{-i} that respect all reducers' key-value preference limitations generated so far. Add a new key-value preference limitation (k_i, V_i) for mapper i if every such totally ordered shuffle and sort conjecture assumes some set $E_{-i} \subseteq K_{-i}$ on which k_i is weakly dominated by some $\gamma_i \in \theta(V_i)$.

Since the number of key-value preference limitations is finite, this algorithm must end after a finite number of rounds. We say that shuffle and sort strategy k_i *survives* the algorithm of iterated addition of key-value preference limitations if k_i is not part of any key-value preference limitation (k_i, V_i) generated by the algorithm. Namely, if k_i were to be part of a key-value preference limitation (k_i, V_i) produced by the algorithm, then mapper i would prefer at least one strategy in V_i to k_i, and hence k_i could not be optimal.

4.3 Main Proposition

Our main proposition states that the algorithm of iterated addition of key-value preference limitations yields *exactly* the set of quantifiable shuffle and sort strategies for every mapper.

Proposition 4.6 *(Algorithm yields precisely the set of quantifiable shuffle and sort strategies). Consider a finite static MapReduce. Then, a shuffle and sort strategy k_i is quantifiable, if and only if, k_i survives the algorithm of iterated addition of key-value preference limitations.*

The easier direction is to show that every quantifiable shuffle and sort strategy survives iterated addition of key-value preference limitation. So, a quantifiable shuffle and sort strategy k_i can never be part of a key-value preference limitation (k_i, V_i) generated by the algorithm. The more difficult direction is to prove that every shuffle and sort strategy k_i that is not part of any such key-value preference limitation (k_i, V_i) is quantifiable. Hence, we must construct a MapReduce programming model in which each of this shuffle and sort strategies k_i is supported by some quantifiable key-value type. This construction is rather delicate.

From the proposition, we can easily derive the following observation: If in a given MapReduce *no* shuffle and sort strategy is weakly dominated, then *all* shuffle and sort strategies for the mappers are quantifiable. Namely, the algorithm we present will only generate key-value preference limitations at the first round if there is at least some shuffle and sort strategy that is weakly dominated within the full MapReduce. Otherwise, the algorithm will not generate any key-value preference limitation at all, and hence all shuffle and sort strategies would survive the algorithm.

4.4 A Finite Formulation of the Algorithm

The algorithm of iterated addition of key-value preference limitations as we have formulated it proceeds by adding key-value preference limitations and deleting totally ordered conjecture at every round. More precisely, we start with the empty set of key-value preference limitations and the full set of totally ordered conjecture. At the first round we see whether we can add some key-value preference limitations. If so, then this would reduce the set of totally ordered conjecture, which at the next round could add some further key-value preference limitations, and so on.

What is somewhat undesirable from a computational point of view is that there are infinitely many possible totally ordered conjecture in the MapReduce. This would suggest that at every round in the algorithm we must scan through infinitely many totally ordered conjecture. This, however, is not necessary. What matters for the algorithm is not so much the precise probabilities in the totally ordered shuffle and sort conjecture, but the induced "likelihood shuffle and sort ordering" on reducers' shuffle and sort strategy combinations. More precisely, let $\psi_i = (\psi_i^1, \ldots, \psi_i^P)$ be a totally ordered shuffle and sort conjecture on K_{-i}. Remember our convention that ψ_i has full support on K_{-i}, that is, every $k_{-i} \in K_{-i}$ receives positive probability in some level ψ_i^p. Let $O_i = (O_i^1, \ldots, O_i^Z)$ be the ordered sequence of disjoint subsets $O_i^z \subseteq K_{-i}$ such that (a) ψ_i deems every $k_{-i} \in O_i^z$ infinitely more likely than every $K_{-i} \in O_i^{z+1}$ for every $z \in \{1, \ldots, Z-1\}$, (b) for every m and every $k'_{-i}, k_{-i} \in O_i^z$, the TODIS ψ_i does not deem k_{-i} infinitely more likely than k'_{-i}, nor vice versa, and (c) the union of the sets in O_i is K_{-i}. We call O_i the *likelihood ordering* induced by ψ_i. Formally, we have the following necessary and sufficient condition.

Necessary and sufficient condition 4.7 *(Likelihood ordering).* A likelihood ordering for mapper i on the reducers' shuffle and sort strategy combinations is an ordered sequence $O_i = (O_i^1, \ldots, O_i^Z)$ where $O_i^1, \ldots, O_i^Z$ are pair-wise disjoint subsets of K_{-i} whose union is equal to K_{-i}.

So, the interpretation is that O_i deems all shuffle and sort strategy combinations in O_i^1 infinitely more likely than all shuffle and sort strategy combinations in O_i^2, deems all shuffle and sort strategy combinations in O_i^2 infinitely more likely than all shuffle and sort strategy combinations in O_i^3, and so on. It is clear that there are only finitely many likelihood orderings in the MapReduce, since there are only finitely many shuffle and sort strategies for every mapper.

We can now easily extend the necessary and sufficient condition of "respecting a key-value preference limitation" and "assuming a set of reducers' shuffle and sort strategy combinations" to likelihood shuffle and sort orderings. Say that a likelihood shuffle and sort ordering $O_i = (O_i^1, \ldots, O_i^Z)$ *respects a key-value preference limitation* (k_j, V_j) if O_i deems some shuffle and sort strategy in V_j infinitely more likely than k_j. Also, the likelihood ordering O_i is said to *assume the set* E_{-i} *of reducers'* shuffle and sort *strategy combinations* if O_i deems all shuffle and sort strategy combinations inside E_{-i}, infinitely more likely than all shuffle and sort strategy combinations outside E_{-i}. The algorithm of iterated addition of key-value preference limitations can thus alternatively be stated as follows:

Algorithm 4.8 *(Finite version).* In round 1, begin for all mappers i with the empty set of key-value preference limitations.

At every further round $q \geq 2$, limit for every mapper i to those likelihood shuffle and sort orderings on K_{-i} that respect all reducers' key-value preference limitations generated so far. Add a new key-value preference limitation (k_i, V_i) for mapper i if every such likelihood shuffle and sort ordering assumes some set $E_{-i} \subseteq K_{-i}$ on which k_i is weakly dominated by some $\gamma_i \in \theta(V_i)$.

The advantage of this formulation is that at every round, we only have to scan through finitely many objects, as there are only finitely many key-value preference limitations and likelihood shuffle and sort orderings in the MapReduce. Obviously, this algorithm generates precisely the same set of key-value preference limitations as the original procedure. As such, the quantifiable shuffle and sort strategies are precisely those shuffle and sort strategies that survive this alternative algorithm.

5. DISCUSSION

In this section we will discuss some important properties of the algorithm.

5.1 Algorithm as an inductive reasoning procedure

The algorithm is not merely a tool to compute the quantifiable shuffle and sort strategies in a MapReduce, but can also be interpreted as an inductive reasoning process that can be used by a mapper who reasons in the spirit of quantifiability. Consider namely a fixed mapper in the MapReduce, say mapper i. In round 2, the algorithm would add for every reducer j a key-value preference limitation (k_j, V_j) if k_j would be weakly dominated on K_{-j} by a mixture on V_j. In that case, mapper i would store the key-value preference limitation (k_j, V_j) in its mind, meaning that he premises that mapper j prefers some shuffle and sort strategy in V_j to k_j. If i respect j's shuffle and sort preferences, then it should consequently deem some shuffle and sort strategy in V_j infinitely more likely than k_j. That is, the key-value preference limitations that mapper i would store in its mind at round 2 would limit the possible totally ordered conjecture it could hold about its reducers' choices. Moreover, if mapper i premises that its reducers reason similarly, then mapper i can actually deduce the possible totally ordered conjecture that its reducers may hold at this round.

In the next round of its reasoning procedure, mapper i would then ask for every reducer j: Given its limited set of conjecture, would mapper j always assume some set $E_{-j} \subseteq K_{-j}$ on which some shuffle and sort strategy k_j would always be weakly dominated by a mixture on V_j? If yes, then mapper i will store (k_j, V_j) as a new key-value preference limitation in its mind. By doing so, mapper i would then further limit the possible totally ordered conjecture it could hold about its reducers. Mapper i could continue this inductive reasoning procedure until no new key-value preference limitation could be added, and hence its possible totally ordered conjecture could not be limited any further.

So we see that the algorithm may serve very well as an intuitive reasoning procedure for mappers that will eventually lead them to the quantifiable shuffle and sort strategies in the MapReduce. What is crucial in this reasoning procedure is

that a mapper only needs to keep track of key-value preference limitations, which substantially simplifies matters compared to the original necessary and sufficient condition of quantifiability. In that light, our main proposition thus says that in order to find the quantifiable shuffle and sort strategies in a MapReduce, it is sufficient for a mapper to think in terms of key-value preference limitations, and to reason in accordance with the algorithm.

In the MapReduce Analytics literature [1], there are other algorithms that can nicely be interpreted as intuitive reasoning procedures. Take, for instance, the concept of *common shuffle and sort conjecture in quantifiability* and the associated algorithm of *iterated elimination of strictly dominated shuffle and sort strategies.* Here, the algorithm can be seen as a reasoning procedure in which a mapper successively deletes reducers' shuffle and sort strategies, since they can no longer be optimal. At every round, this would then limit the mapper's possible conjecture as it must assign probability zero to these shuffle and sort strategies. These additional limitations on the mappers' conjecture could then induce further shuffle and sort strategies that can be deleted, and so on. So, in that procedure the mappers' possible (non-totally ordered) conjecture are limited further and further by deleting shuffle and sort strategies, whereas in our procedure the (totally ordered) conjecture are limited further and further by adding new key-value preference limitations.

A similar story can be told for the concept of *iterated assumption of quantifiability within a complete key-value type structure* and the associated algorithm of *iterated elimination of weakly dominated shuffle and sort strategies.* Here, the algorithm reflects a reasoning procedure in which a mapper with totally ordered conjecture iteratedly deletes weakly dominated shuffle and sort strategies from its mind. At every round of this procedure, the mapper will then deem all surviving shuffle and sort strategies infinitely more likely than all deleted shuffle and sort strategies, thus limiting the possible totally ordered conjecture it can hold. So also in this procedure, the mapper's possible conjecture is limited in every round by deleting shuffle and sort strategies.

5.2 Order Independence

For the algorithm, it can be shown that the order and speed in which we add preference restrictions does not matter for the eventual result. That is, it does not matter whether in every round we add *all* preference restrictions that can possibly be generated, or only *some* of these.

To see this, let us compare two procedures, Procedure 1 and Procedure 2, where in the first we always add *all* possible key-value preference limitations at every round, and in the second we only add *some* of the possible key-value preference limitations every time. Then, first of all, Procedure 1 will at every round generate at least as many key-value preference limitations as Procedure 2. Namely, at round 2 Procedure 1 generates as least as many key-value preference limitations, by necessary and sufficient condition. Therefore, at round 3 Procedure 1 limits to a smaller set of totally ordered conjecture than Procedure 2. But then, under Procedure 1 it will be "easier" to generate new key-value preference limitations at round 3 than under Procedure 2. Hence, at round 3 Procedure 1 will, again, generate at least as many key-value preference limitations as Procedure 2, and so on. So, eventually, Procedure 1 will generate at least as many key-value preference limitations as Procedure 2. The key argument here was that a larger set of key-value preference limitations will lead to a smaller set of possible totally ordered conjecture, and a smaller set of possible totally ordered conjecture will in turn lead to a larger set of induced key-value preference limitations. So, the algorithm is *monotone* in this sense.

On the other hand, it can also be shown that every key-value preference limitation generated by Procedure 1 will also eventually be generated by Procedure 2. Suppose, namely, that Procedure 1 would generate some key-value preference limitation that would not be generated at all by Procedure 2. Then, let p be the first round at which Procedure 1 would generate a key-value preference limitation, say (k_i, V_i), not generated by Procedure 2 at all. By construction of the algorithm, every totally ordered shuffle and sort conjecture for mapper i that respects all key-value preference limitations generated by Procedure 1 *before* round p, must assume some set E_{-i} on which k_i is weakly dominated by some $\gamma_i \in \theta(V_i)$. By our assumption, all these key-value preference limitations generated by Procedure 1 before round p are also eventually generated by Procedure 2, let us say before round $z \geq p$. But then, every totally ordered shuffle and sort conjecture for mapper i that respects all key-value preference limitations generated by Procedure 2 before round z, assumes a set E_{-i} on which k_i is weakly dominated by some $\gamma_i \in \theta(V_i)$.

Hence, Procedure 2 must add the key-value preference limitation (k_i, V_i) sooner or later, which is a contradiction since we assumed that Procedure 2 does not generate key-value preference limitation (k_i, V_i) at all. We thus conclude that every key-value preference limitation added by Procedure 1 is also finally added by Procedure 2. As such, Procedures 1 and 2 eventually generate exactly the same set of key-value preference limitations. So, indeed, the order and speed in which we add key-value preference limitations is irrelevant to the algorithm.

6. CONCLUSION AND FUTURE SCOPE

In this section we conclude, stating that the algorithm of iterated addition of key-value preference limitations selects exactly the set of quantifiable shuffle and sort strategies in the MapReduce. For our conclusion, we recall the necessary and sufficient condition of a *likelihood shuffle and sort ordering induced by a TODIS.* Consider a TODIS $\psi_i = (\psi_i^1, \ldots, \psi_i^P)$on K_{-i}. Remember our convention that ψ_i has full support on K_{-i}, that is, every $k_{-i} \in K_{-i}$ receives positive probability in some level ψ_i^p. Let $O_i = (O_i^1, \ldots, O_i^Z)$be the ordered sequence of disjoint subsets $O_i^z \subseteq K_{-i}$ such that (a) ψ_i deems every $k_{-i} \in O_i^z$ infinitely more likely than every $k'_{-i} \in O_i^{z+1}$, for every $z \in \{1, \ldots, Z-1\}$,(b) for every z and every $k_{-i}, k'_{-i} \in O_i^z$, the TODIS ψ_i does not deem k_{-i} infinitely more likely than k'_{-i}, nor vice versa, and(c) the union of the sets in O_i is K_{-i}. We call O_i the *likelihood shuffle and sort ordering* induced by ψ_i. Our conclusion characterizes, for a given shuffle and sort strategy k_i and set $V_i \subseteq K_i$, those likelihood shuffle and sort orderings on K_{-i} that admit an TODIS under which k_i is weakly preferred to all shuffle and sort strategies in V_i. Despite the progress on our interpretation is made the following three open problems are available for further research.

Open Problem 6.1 *Let ψ_i be a TODIS on K_{-i}, let k_i be a shuffle and sort strategy and $V_i \subseteq K_i$ a subset of shuffle and sort strategies.*(a) *If under the TODIS ψ_i, shuffle and sort strategy k_i is weakly preferred to all shuffle and sort strategies in V_i. Does ψ_i assume any $E_{-i} \subseteq K_{-i}$ on which k_i is weakly dominated by a mixture on V_i?* (b) *If ψ_i does not assume any $E_{-i} \subseteq K_{-i}$ on which k_i is weakly dominated by a mixture on V_i.*

Does some TODIS ϕ_i, inducing the same likelihood shuffle and sort ordering as ψ_i, under which k_i is weakly preferred to all shuffle and sort strategies in V_i?

Open Problem 6.2 *Let t_i be a quantifiable key-value type. Does t_i's totally ordered shuffle and sort conjecture on K_{-i} respects every key-value preference limitation in F^{∞}_{-i}?*

Open Problem 6.3 *(Property of key-value preference limitations not generated by the algorithm). For every mapper i, let F_i^{not} be the set of key-value preference limitations not generated by the algorithm. Does for every $(k_i, V_i) \in F_i^{not}$ is there an TODIS ψ_i on K_{-i} such that* (1) *under ψ_i, shuffle and sort strategy k_i is weakly preferred to all shuffle and sort strategies in V_i, and* (2) *for every reducer's shuffle and sort strategy k_j, the pair $((k_j, V_j^{-}(k_j, \psi_i))$ is in F_j^{not}?*

7. REFERENCES

[1] Rajagopal Ananthanarayanan, Karan Gupta, Prashant Pandey, Himabindu Pucha, Prasenjit Sarkar, Mansi Shah, and Renu Tewari. Cloud analytics: Do we *really* need to reinvent the storage stack? In *Proceedings of the 2009 Workshop on Hot Topics in Cloud Computing (HotCloud 09)*, San Diego, California, 2009.

[2] Mark Dredze, Alex Kulesza, and Koby Crammer. Multi-domain learning by confidence-weighted parameter combination. *Machine Learning*, 79:123–149, 2010.

[3] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert Gruber. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th Symposium on Operating System Design and Implementation (OSDI 2006)*, pages 205–218, Seattle, Washington, 2006.

[4] Arthur Asuncion, Padhraic Smyth, and Max Welling. Asynchronous distributed learning of topic models. In *Advances in Neural Information Processing Systems 21 (NIPS 2008)*, pages 81–88, Vancouver, British Columbia, Canada, 2008.

[5] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with YCSB. In *Proceedings of the First ACM Symposium on Cloud Computing (ACM SOCC 2010)*, Indianapolis, Indiana, 2010.

[6] Leon Bottou. Stochastic learning. In Olivier Bousquet and Ulrike von Luxburg, editors, *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146-168. Springer Verlag, Berlin, 2004.

[7] Thorsten Brants and Peng Xu. *Distributed Language models*. Morgan & Claypool Publishers, 2010.

[8] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swami Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP 2007)*, pages 205–220, Stevenson, Washington, 2007.

[9] Tony Hey, Stewart Tansley, and Kristin Tolle. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, Redmond, Washington, 2009.

[10] Tony Hey, Stewart Tansley, and Kristin Tolle. Jim Gray on eScience: A transformed scientific method. In Tony Hey, Stewart Tansley, and Kristin Tolle, editors, *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, Redmond, Washington, 2009.

[11] Eric Brill, Jimmy Lin, Michele Banko, Susan Dumais, and Andrew Ng. Data-intensive question answering. In *Proceedings of the Tenth Text REtrieval Conference (TREC 2001)*, pages 393–400, Gaithersburg, Maryland, 2001.

[12] Christopher Southan and Graham Cameron. Beyond the tsunami: Developing the infrastructure to deal with life sciences data. In Tony Hey, Stewart Tansley, and Kristin Tolle, editors, *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, Redmond, Washington, 2009.

[13] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: Towards a realistic model of parallel computation. *ACM SIGPLAN Notices*, 28(7):1-12, 1993.

[14] Wittawat Tantisiriroj, Swapnil Patil, and Garth Gibson. Data-intensive file systems for Internet services: Arose by any other name….Technical Report CMU-PDL-08-114,Parallel Data Laboratory, Carnegie Mellon University, 2008.

[15] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for MapReduce. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, Austin, Texas, 2010.

[16] Thomas Sandholm and Kevin Lai. MapReduce optimization using regulated dynamic prioritization. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '09)*, pages 299-310, Seattle, Washington, 2009.

[17] M. Mustafa Rafique, Benjamin Rose, Ali R. Butt, and Dimitrios S. Nikolopoulos. Supporting MapReduce on large-scale asymmetric multi-core clusters. *ACM Operating Systems Review*, 43(2):25–34, 2009.

[18] Jeffrey Dean and Sanjay Ghemawat. MapReduce: A flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010.

[19] Jonathan Cohen. Graph twiddling in a MapReduce world. *Computing in Science and Engineering*, 11(4):29–41, 2009.

[20] Michael Stonebraker, Daniel Abadi, David J. DeWitt, Sam Madden, Erik Paulson, Andrew Pavlo, and Alexander Rasin. MapReduce and parallel DBMSs: Friends or foes? *Communications of the ACM*, 53(1): 64–71, 2010.