Symmetric and Asymmetric Aggregate Function in Massively Parallel Computing(extended version)

Chao Zhang, Farouk Toumani, Emmanuel Gangler

Symmetric and Asymmetric Aggregate Function in Massively Parallel Computing

ZHANG Chao
Supervised by Prof. Farouk Toumani, Prof. Emmanuel GANGLER

LIMOS, Université Blaise Pascal, Aubière, France
{zhangch, ftoumani}@isima.fr

ABSTRACT

Applications of aggregation for information summary have great meanings in various fields. In big data era, processing aggregate function in parallel is drawing researchers' attention. The aim of our work is to propose a generic framework that enables to map an aggregation function into an efficient massively parallel algorithm that can be executed on modern large-scale data-processing systems. We describe our preliminary results regarding classes of symmetric and asymmetric aggregation functions that can be mapped, in a systematic way, into efficient MapReduce algorithms.

1. INTRODUCTION

The ability of summarizing information is drawing increasing attention for information analysis [11, 6]. Simultaneously, under the progress of data explosive growth processing aggregate function has to experience a transition to massively distributed and parallel framework, e.g. Hadoop, Spark, Flink etc. Therefor aggregation function requires a decomposition approach in order to execute in parallel due to its inherently property of taking several values as input and generating a single value based on a certain criteria. Decomposable aggregation function can be processed in a way that computing partial aggregation and then merge them at last to obtain final result.

Decomposition of aggregation function is a long standing research problem that has been addressed in various fields. In distributed computing frameworks like Hadoop, decomposability of aggregate function can push aggregation before shuffle phase [17, 3]. This is usually called initial reduce, with which the size of data transmission on network can be substantially reduced. For wireless sensor network, the need to reduce data transmission is more necessary because of limitation of power supply [15]. In online analytical processing (OLAP), decomposability of aggregate function enables aggregation across multi-dimensions, such that aggregation queries can be executed on pre-computation results instead of base data to accelerate query answering [8]. An important point of query optimization in relational database is to reduce table size for join [10], and decomposable aggregation brings interests [4].

When an aggregation function is decomposable, how to decompose it and when a decomposition is 'efficient' is a hard nut to crack. Previous works identify interesting properties for decomposable aggregation. A very relevant classification of aggregation functions, introduced in [11], is based on the size of sub-aggregation (i.e., partial aggregation). This classification distinguishes between distributive and algebraic aggregation having sub-aggregate with fixed sizes, and holistic functions where there is no constant bound on the storage size of subaggregate. Some algebraic properties, such as associativity and commutativity, are identified as sufficient conditions for decomposing aggregation [17, 3]. Compared to these researches, our work provides a generic framework to identify and process any symmetric aggregation in parallel and generates the corresponding generic algorithm. Moreover, all but few work in the literature consider symmetric functions. Asymmetric aggregation are inherently non-commutative functions and this makes their processing in parallel and distributed environment far from being easy. In [16], a symbolic parallel engine (SYMPLE) is proposed in order to automatically parallelize User Defined Aggregations (UDAs) that are not necessarily commutative. Although interesting, the proposed framework lacks of genericity in the sense that it is up to the user to encode a function as SYMPLE UDA. Moreover, symbolic execution may have path explosion problem.

My research focuses on designing generic framework that enables to map symmetric and asymmetric aggregation functions into an efficient massively parallel algorithm. To achieve this goal, we firstly identify a computation model and an associated cost model to design and evaluate parallel algorithms. We consider MapReduce-like (MR) framework and use the MRC [12] cost model to define *'efficient'* MR algorithms. We rest on the notion of well-formed aggregation [4] as a canonical form to write symmetric aggregation and provide a simple and systematic way to map well-formed aggregation function α into a MR algorithm, noted by $MR(\alpha)$. Moreover we provide reducible properties to identify when the generated $MR(\alpha)$ is efficient (when $MR(\alpha)$ is a MRC algorithm). Then we extend our framework to a class of asymmetric aggregation functions, position-based aggregation, and propose extractable property to have generic MRC algorithm. Our main results are theorem 1 and theorem 2, proofs for which are illustrated in an extended report[1].

Proceedings of the VLDB 2017 PhD Workshop, August 28, 2017. Munich, Germany.

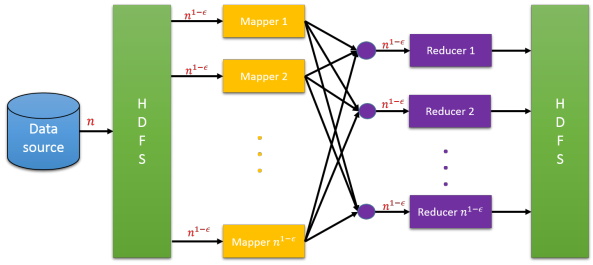

Figure 1: MapReduce flowchart with MRC constraints

2. MRC ALGORITHM

Several research works concentrate on complexity of parallel algorithm. MUD [7] algorithm was proposed to transform a symmetric streaming algorithm to a parallel algorithm with nice bounds in terms of communication and space complexity, but without any bound on the time complexity. This disqualifies MUD as a possible candidate cost model to be used in our context. MRC [12] is another popular framework that has been used to evaluate whether a MapeReduce algorithm is efficient. The constraints enforced by MRC w.r.t. to total input data size can be summarized as following: sublinear number of total computing nodes, sublinear space for any mapper or reducer, polynomial time for any mapper or reducer, and logarithm round number. We illustrate these constraints besides round number in a simplified MapReduce flowchart in figure 1 where $\epsilon > 0$.

Hence, the MRC model considers necessary parameters for parallel computing, communication time, computation space and computing time, and makes more realistic assumptions. A MapReduce algorithm satisfying these constraints is considered as an efficient parallel algorithm and will be called hereafter a MRC algorithm.

3. SYMMETRIC AGGREGATION WITH MRC

Definition(Symmetric Aggregation) [9]: *An aggregation α is symmetric if $\alpha(X) = \alpha(\sigma(X))$ for any $X = \{x_1, \ldots, x_n\} \subseteq I$ and any permutation σ, where $\sigma(X) = (x_{\sigma(1)}, ..., x_{\sigma(n)})$.*

Symmetric aggregation result does not depend on the order of input data. In this section, we define a generic framework to map symmetric aggregation into a MRC algorithm.

3.1 A Generic Form for Symmetric Aggregations

To define our generic aggregation framework, we rest on the notion of well-formed aggregation [4]. A symmetric aggregation α defined on a set of values $\{d_1, \ldots, d_n\}$ can be written in well-formed aggregation as following:

$$\alpha(d_1, \ldots, d_n) = T(F(d_1) \oplus \ldots \oplus F(d_n)),$$

where F is translating function(tuple at a time), $\oplus$ is a commutative and associative binary operation, and T is terminating function. *Average* can be easily transformed in well-formed aggregation as follows: $F(d) = (d, 1)$, $(d, k) \oplus (d', k') = (d+d', k+k')$ and $T((d, n)) = \frac{d}{n}$. It is worth noting that, any aggregate function can be rewritten in well-formed aggregation due to a flexible choice of $\oplus$, e.g $\oplus = \cup$.

Well-formed aggregation provides a generic plan for processing aggregate function in distributed architecture based on the associative and commutative property of $\oplus$: processing F and $\oplus$ at mapper, $\oplus$ and T at reducer. Table 1 depicts the corresponding generic MapReduce(MR) algorithm, noted by $MR(\alpha)$, where mapper input mapper is d and mapper output is $o$output, and $\sum_{\oplus}$ as concatenation of $\oplus$.

Table 1: $MR(\alpha)$: a generic MR aggregation algorithm

	operation
mapper	$\sum_{\oplus,j} F(d_j)$
reducer	$T(\sum_{\oplus,i} o_i)$

However, the obtained $MR(\alpha)$ are not necessarily an efficient MapReduce algorithm. We identify when $MR(\alpha)$ is a MRC algorithm using *reducibility* property of well-formed aggregation.

Definition 1. An aggregation function α is reducible if a well-formed aggregation $(F, \oplus, T)$ of α satisfies

$$\forall d_i, d_j \in I : |F(d_i) \oplus F(d_j)| = O(1).$$

With this reducible property, we provide a theorem identifying when a symmetric aggregation can be mapped into MRC algorithm.

THEOREM 1. *Let α be a symmetric well-formed aggregation and $MR(\alpha)$ be the generic algorithm for α, then $MR(\alpha)$ is a MRC algorithm if and only if α is reducible.*

PROOF. Firstly we prove reducible condition is equivalent to MRC space constraints. (Necessity) The worst case is that n copies of the same input, such that all outputs of mappers are transferred into one reducer, therefor machine number multiplies mapper output should be no bigger than reducer memory. Because MRC considers $\Theta(n^{1-\epsilon})$ machines and $O(n^{1-\epsilon})$ reducer memory, mapper output should be no bigger than $O(1)$. $MR(\alpha)$ has operations F and $\oplus$ at mapper, then α is reducible. (Sufficiency) If α is reducible, then it is trivial to see that mapper output of $MR(\alpha)$ is $O(1)$, such that $\Theta(n^{1-\epsilon})$ machines multiplying $O(1)$ mapper output $O(n^{1-\epsilon})$ is not bigger than $O(n^{1-\epsilon})$ reducer memory, then reducible$MR(\alpha)$ satisfying MRC space constraints. Time complexity of $MR(\alpha)$ at mapper phrase is $O(n)$ because translating function F is a tuple-at-a-time function and $\oplus$ takes every output of F. At reducer phrase, $\oplus$ combines results of constant size from sublinear number of mappers and T terminates aggregating, therefore reducer time complexity is $O(n^{1-\epsilon})$. $MR(\alpha)$ satisfies MRC time complexity constraint which is polynomial in n. □

3.2 Deriving MRC Algorithm from Algebraic Properties

In this section, we investigate several symmetric aggregation properties satisfying our theorem 1. If an aggregation α is in one of the following class, then α has a $MRC(\alpha)$ algorithm illustrated in table 1.

An aggregate function α is **associative** [9] if for set $X = X_1 \cup X_2$, $\alpha(X) = \alpha(\alpha(X_1), \alpha(X_2))$. Associative aggregation function can be transformed in well-formed aggregation $(F, \oplus, T)$ as following,

$$F = \alpha, \ , \oplus = \alpha, \ T = id \qquad (1)$$

where id denotes identity function. α is reducible because it is an aggregation. Therefore $MR(\alpha)$ of associative aggregation α is a MRC algorithm.□

An aggregation α is **distributive** [11] if there exists a combining function C such that $\alpha(X,Y) = C(\alpha(X), \alpha(Y))$. Distributive aggregation can be rewritten in well-formed aggregation $(F, \oplus, T)$ as following,

$$F = \alpha,\ \oplus = C,\ T = id. \tag{2}$$

Similarly, α is reducible and corresponding $MR(\alpha)$ is a MRC algorithm.□

Another kind of aggregate function having the same behavior as distributive aggregation is **commutative semi-group aggregate function** [5]. An aggregation α is in this class if there exists a commutative semi-group $(H, \otimes)$, such that $\alpha(X) = \bigotimes_{x_i \in X} \alpha(x_i)$. The corresponding well-formed aggregation $(F, \oplus, T)$ is illustrated as following,

$$F = \alpha,\ \oplus = \otimes,\ T = id. \tag{3}$$

It is clearly that α is reducible and $MR(\alpha)$ is a MRC algorithm.□

A more general property than distributive or commutative semi-group aggregation is preassociative aggregate function. An aggregate function α is **preassociative** [13] if it satisfies $\alpha(Y) = \alpha(Y') \implies \alpha(XYZ) = \alpha(XY'Z)$. According to a theorem in [13], many of preassociative aggregation functions(unarily quasi-range-idempotent and continuous and one to one function for the unary and binary part of the aggregation) can be constructed as following, $\alpha(\mathbf{X}) = \psi\left(\sum_{i=1}^{n} \varphi(x_i)\right), n \geq 1$, where ψ and φ are continuous and strictly monotonic function. The well-formed aggregation $(F, \oplus, T)$ for this kind of preassociative aggregation is illustrated as following

$$F = \varphi,\ \oplus = +,\ T = \psi. \tag{4}$$

The corresponding $MR(\alpha)$ is also a MRC algorithm.
□

All the above aggregate functions have a common characteristic that the output of translating function F is one element, in the following we introduce an aggregation with F having a pair output.

An aggregate function α is barycentrically associative [14] if it satisfies $\alpha(XYZ) = \alpha(X\alpha(Y)^{|Y|}Z)$, where $|Y|$ denotes the number of elements contained in set Y and $\alpha(Y)^{|Y|}$ denotes $|Y|$ occurrences of $\alpha(Y)$. A well-known class of this kind of aggregate function is **quasi-arithmetic mean** having the following form $\alpha(\mathbf{X}) = f^{-1}\left(\frac{1}{n}\sum_{i=1}^{n} f(x_i)\right), n \geq 1$, where f is an unary function and f^{-1} is a quasi-inverse of f. With different choices of f, α can be different kinds of mean functions, e.g arithmetic mean, quadratic mean, harmonic mean etc. It is trivial to rewrite this kind of aggregation to well-formed aggregation $(F, \oplus, T)$ and the $MR(\alpha)$ is also a MRC algorithm,

$$F = (f, 1),\ \oplus = (+, +),\ T = f^{-1}(\frac{\sum_{i=1}^{n} f(x_i)}{n}). \tag{5}$$

4. ASYMMETRIC AGGREGATION WITH MRC

Many commonly used aggregation function is symmetric(commutative) such that order of data can be ignored, while *asymmetric aggregation function considers the order.* Two common asymmetric cases could be weighted aggregation and cumulative aggregation, where aggregated result will be changed if data order is changed. For example, WMA(weighted moving average) and EMA(exponential moving average)[2], which are used to highlight trends, have different weights such that order of data can not be changed.

4.1 A Generic Form for Asymmetric Aggregation

In contrast to symmetric aggregation, asymmetric function is impossible to rewrite into well-formed aggregation, because translating function F is a tuple at a time function and $\oplus$ is commutative and hence both functions are insensitive to the order. For this reason, we propose an extended form based on well-formed aggregation which is more suitable for asymmetric aggregation.

Definition 2. An asymmetric aggregation α defined on an ordered sequence $\bar{X}$ is an asymmetric well-formed aggregation if α can be rewritten as following,

$$\alpha(\bar{X}) = T(F^o(\bar{X}, x_1) \oplus ... \oplus F^o(\bar{X}, x_n)), \tag{6}$$

where F^o is order-influenced translating function, $\oplus$ is a commutative and associative binary operation, and T is terminating function.

For instance, $\alpha(\bar{X}) = \sum_{x_i \in \bar{X}} (1-z)^{i-1} x_i$[14] with a constant z can be rewritten as $F^o(\bar{X}, x_i) = (1-z)^{i-1}x_i$, $\oplus = +$, $T = id$, where i is the position of x_i in the sequence $\bar{X}$.

Asymmetric well-formed aggregation can rewrite any asymmetric aggregation α, and with the associative property of $\oplus$, α also has a generic MR algorithm $MR(\alpha)$: processing F^o and $\oplus$ at mapper, $\oplus$ and T at reducer. Similar to the behavior of symmetric well-formed aggregation, reducible property is needed to ensure MRC constraints. The reducible property for asymmetric well-formed aggregation is

$$\forall x_i, x_j \in \bar{X}:\ |F^o(\bar{X}, x_i) \oplus F^o(\bar{X}, x_j)| = O(1).$$

However, in order to have a correct generic MRC algorithm for asymmetric aggregation, reducible property is not enough, because asymmetric function considers data order such that operations for combining mapper outputs are more than $\oplus$, otherwise the aggregated result will not be correct. We illustrate this problem and identify properties to have correct MRC algorithm for a class of asymmetric well-formed aggregation in the following subsection.

4.2 Position-based Aggregation with MRC

We deal with a kind of asymmetric aggregation α called position-based aggregation, for which the F^o is $F^o(\bar{X}, x_i) = h(i) \odot f(x_i)$, where $h()$ is an unary function on i, and $\odot$ is a binary operation. The corresponding asymmetric well-formed framework is $\alpha(\bar{X}) = T(\sum_{\oplus, x_i \in \bar{X}} h(i) \odot f(x_i))$, where $\sum_{\oplus}$ is the concatenation of $\oplus$.

Let $\bar{X}$ be an ordered sequence $\bar{X} = \bar{S_1} \circ ... \circ \bar{S_m}$, where $\bar{S_l}$ is a subsequence of $\bar{X}$, $l \in \{1, ..., m\}$ and $\circ$ is concatenation of sequence, and i be the holistic position number in $\bar{X}$ and j be the relative position in subsequence $\bar{S_l}$. Then $\sum_{\oplus} F^o$ of α on any subsequence S_l is $\sum_{\oplus, x_j \in \bar{S_l}} F^o(x_j) = \sum_{\oplus, x_j \in \bar{S_l}} h(j+k) \odot f(x_i)$, where $j+k$ $(j+k = i)$ is the holistic position of the jth element x_j in $\bar{S_l}$. In order to process α in parallel on these subsequences, the first requirement is to

have l, which means in Hadoop data set is split into ordered chunks and chunk numbers can be remembered. It is trivial to implement this[16]. Secondly, k is needed, the number of elements before $\bar{S}_l$. However, sequentially distributing subsequence count value then starting aggregation is costly due to too many times of data transferring on network. If k can be extracted out of $\sum_{\oplus, x_j \in \bar{S}_l} h(j+k) \odot f(x_i)$, α can be processed without distributing counts because operations relating to counts can be pushed to reducer. We identify conditions to extract k which we call *extractable* property.

LEMMA 1. *Given an ordered sequence $\bar{X}$, a position-based asymmetric well-formed aggregation α defined in $(F^o, \oplus, T)$ and $F^o(\bar{X}, (x_i)) = h(i) \odot f(x_i)$ for any $x_i \in \bar{X}$, where $h()$ and $f()$ are unary functions, is* ***extractable*** *if there exists a binary operation $\otimes$ making $h()$ satisfy $h(i+k) = h(i) \otimes h(k+c)$ where c is a constant, and $\oplus$, $\otimes$ and $\odot$ satisfy one of the following conditions,*

- *$\otimes$, $\odot$ and $\oplus$ are same,*
- *$\otimes$ and $\odot$ are same and they are distributive over $\oplus$,*
- *$\otimes$ is distributive over $\odot$ which is same as $\oplus$.*

The behavior of $h()$ is similar to group homomorphism however they are not exactly same, and our intention is to extract k instead of preserving the exact operation.

THEOREM 2. *Let α be a position-based well-formed aggregation and $MR(\alpha)$ be the generic algorithm for α, then $MR(\alpha)$ is a MRC algorithm if α is reducible and extractable.*

Extractable property of position-based aggregation α allows previous subsequence count value 'k' to be extracted out of mapper operation, then α can be correctly processed by $\sum_{\oplus} F^o$ at mapper phrase. To combine mapper output at reducer phrase, more than $\oplus$ and T are needed and specific additional operation depends on different extractable condition.

PROOF. Similarly to symmetric well-formed aggregation function, reducible properties ensures MRC space and time complexity. Moreover, extractable property of position-based aggregation α allows previous subsequence count value 'k' to be extracted out of mapper operation, then α can be correctly processed by $\sum_{\oplus} F^o$ at mapper phrase. To combine mapper output at reducer phrase, more than $\oplus$ and T are needed and specific additional operation depends on different extractable condition. □

We take EMA as an example and illustrate that it is a position-based aggregation with a MRC algorithm. Given an input sequence $\bar{X} = \{x_1, ..., x_n\}$ and assuming x_1 is the most recent data, then EMA has the formula, $EMA(\bar{X}) = \frac{\sum_{i=1}^{n}(1-a)^{i-1} \cdot x_i}{\sum_{i=1}^{n}(1-a)^{i-1}}$, where a is a constant between 0 and 1. The asymmetric well-formed aggregation of EMA is illustrated as following,

$$
\begin{aligned}
F^o &: F^o(\bar{X}, x_i) = \{h(i) \cdot x_i,\ h(i)\}, \\
\oplus &: \{h(i) \cdot x_i,\ h(i)\} \oplus \{h(i+1) \cdot x_{i+1},\ h(i+1)\} \\
&\quad = \{h(i) \cdot x_i + h(i+1) \cdot x_{i+1}, h(i) + h(i+1)\}, \\
T &: T(\sum_{i=1}^{n} h(i) \cdot x_i, \sum_{i=1}^{n} h(i)) = \frac{\sum_{i=1}^{n} h(i) \cdot x_i}{\sum_{i=1}^{n} h(i)},
\end{aligned}
$$

where $h(i) = (1-a)^{i-1}$. It is clearly that EMA is a position-based aggregation, and also EMA is reducible because $\oplus$ is a pair of addition. Moreover $h(i+k) = h(i) \cdot (1-a)^k$, and the three binary operations $\otimes = \cdot$, $\odot = \cdot$, $\oplus = +$ satisfy the second extractable condition. Therefor EMA has a MRC algorithm(the generic MRC algorithm for the second extractable condition) illustrated as following where we assume input sequence $\bar{X} = \bar{S_1} \circ ... \circ \bar{S_m}$ and one mapper processes one subsequence S_l, $l \in \{1, ..., m\}$, and $count(S_0) = 0$

- mapper: ($OM_l^{'} = \sum_{x_j \in S_l} h(j) \cdot x_j$, $OM_l^{''} = \sum_{x_j \in S_l} h(j)$, $OM_l^{'''} = count(S_l)$),
- reducer: $\frac{\sum_{l=1}^{m} OM_l{}' \cdot (1-a)^{\sum_{j=0}^{l-1} OM_j^{'''}}}{\sum_{l=1}^{m} OM_l^{''} \cdot (1-a)^{\sum_{j=0}^{l-1} OM_j{}'''}}$.

5. CONCLUSION AND FUTURE WORK

In this work, we studied how to map aggregation functions in a systematic way into generic MRC algorithm and we identified sufficient properties that enable to efficiently execute symmetric and asymmetric aggregations using MapReduce style framework. For symmetric aggregation, we proposed reducible property of well-formed aggregation to satisfy space and time complexity constraint of MRC. Several symmetric aggregation properties leading to an efficient $MR(\alpha)$ have been identified. Moreover, we extended the notion of well-formed aggregation to asymmetric aggregation and showed how it can be exploited to deal with position-based asymmetric aggregation. Through identifying the problem for parallelizing it, we proposed extractable property and merged it with the similar reducible property to have MRC algorithm for asymmetric aggregation.

Our future work will be devoted to the implementation and experimentation. We will study the extension of our framework to mainstream parallel computing paradigm (e.g., Apache Spark). Moreover, we also plan to extend our framework to cover additional classes of asymmetric aggregations. Finally, we plan to investigate how to generalize our approach to nested aggregation functions (i.e., functions defined as complex composition of aggregation functions).

6. REFERENCES

[1] Aggregation in parallel computing. `https:~`.

[2] Moving average. `https://en.wikipedia.org/wiki/Moving_average`.

[3] C.Liu, J.Zhang, H.Zhou, Z. S.McDirmid, and T.Moscibroda. Automating distributed partial aggregation. In *SOCC'14*, pages 1–12, 2014.

[4] S. Cohen. User-defined aggregate functions: bridging theory and practice. In *SIGMOD'06*, pages 49–60, 2006.

[5] S. COHEN, W.NUTT, and Y.SAGIV. Rewriting queries with arbitrary aggregation functions using views. *ACM TODS*, 31(2):672–715, June 2006.

[6] A. Cuzzocrea. Aggregation and multidimensional analysis of big data for large-scale scientific applications: models, issues, analytics, and beyond. In *SSDBM'15*, 2015.

[7] J. Feldman, S.muthukrishnan, A. Sidiropoulos, C. Stein, and Z. Svitkina. On distributing symmetric streaming computations. *ACM TALG*, 6(4), August 2010.

[8] M. Franklin. An overview of data warehousing and olap technology. *ACM SIGMOD Record*, 26(1):65–74, March 1997.

[9] M. Grabisch, J.-L. Marichal, R. Mesiar, and E. Pap. Aggregation function: Means. *Information Sciences*, 181(1):1–22, January 2011.

[10] H.Garcia-Molina, J.D.Ullman, and J.Widom. *Database System Implementation*. Prentice-Hall, New Jersey, 2000.

[11] J.Gray, A.Bosworth, A.Layman, and H.Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, Janaury 1997.

[12] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for mapreduce. In *SODA'10*, pages 938–948, 2010.

[13] M.Jean-Luc and T.Bruno. Preassociative aggregation functions. *Fuzzy Sets and Systems*, 268:15–26, June 2015.

[14] M.Jean-Luc and T.Bruno. Strongly barycentrically associative and preassociative functions. *Fuzzy Sets and Systems*, 437(1):181–193, May 2016.

[15] S.Madden, M.J.Franklin, J.M.Hellerstein, and W.Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *OSDI'02*, pages 131–146, 2002.

[16] V.Raychev, M.Musuvathi, and T.Mytkowicz. Parallelizing user-defined aggregaions using symbolic execution. In *SOSP'15*, pages 153–167, 2015.

[17] Y.Yu, M.Isard, and P. Gunda. Distributed aggregation for data-parallel computing: Interfaces and implementations. In *SOSP'09*, pages 247–260, 2009.