Symmetric and Asymmetric Aggregate Function in Massively Parallel Computing

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

LIMOS, Université Clermont Auvergne, 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 enabling to map an arbitrary aggregation into a generic algorithm and identify when it can be efficiently executed on modern large-scale data-processing systems. We describe our preliminary results regarding classes of symmetric and asymmetric aggregation that can be mapped, in a systematic way, into efficient MapReduce-style algorithms.

1. INTRODUCTION

The ability to summarize 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 platforms, e.g. Hadoop MapReduce, Spark, Flink etc. Therefore aggregation function requires a decomposition approach in order to execute in parallel due to its inherent property of taking several values as input and generating a single value based on certain criteria. Decomposable aggregation function can be processed in a way that computing partial aggregation and then merging them at last to obtain final results.

Decomposition of aggregation function is a long-standing research problem due to its benefits in various fields. In distributed computing platforms, 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 a 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 aggregate 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 databases is to reduce table size for join [10], and decomposable aggregation brings interests [4].

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

When an arbitrary 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 decomposing 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-aggregation with fixed sizes, and holistic functions where there is no constant bound on the storage size of sub-aggregation. Some algebraic properties, such as associativity and commutativity, are identified as sufficient conditions for decomposing aggregation [17, 3]. Compared to these works, our work provides a generic framework to identify the decomposability of any symmetric aggregation and generate generic algorithms to process it in parallel. Moreover, all but few researches in the literature consider symmetric functions. Asymmetric aggregation is 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 guarantees for efficiency and accuracy in the sense that it is up to users 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 efficient massively parallel algorithms. To achieve this goal, we firstly identify a computation model, and an associated cost model to design and evaluate parallel algorithms. We consider MapReduce-style (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 an MR algorithm, noted by $MR(\alpha)$. Moreover, we provide reducible properties to identify when the generated $MR(\alpha)$ is efficient (when $MR(\alpha)$ is an MRC algorithm). Then we extend our framework to a class of asymmetric aggregation function, position-based aggregation, and propose extractable property to have generic MRC algorithms. Our main results are Theorem 1 and Theorem 2, of which proofs are provided in an extended report[2].

2. MRC ALGORITHM

Several research works concentrate on the complexity of parallel algorithms. MUD[7] algorithm was proposed to

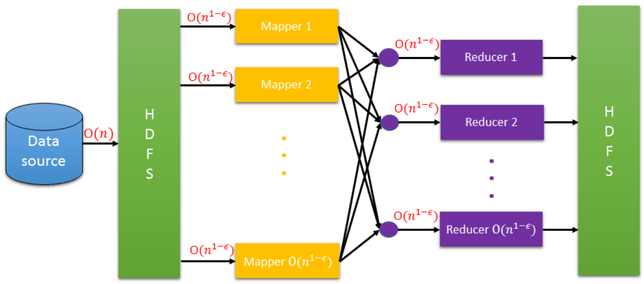

Figure 1: MapReduce flowchart with MRC constraints

transform symmetric streaming algorithms to parallel algorithms with nice bounds in terms of communication and space complexity, but without any bound on time complexity. This disqualifies MUD as a possible candidate cost model to be used in our context. MRC[12] is another popular model that has been used to evaluate whether a MapeReduce algorithm is efficient. The constraints enforced by MRC w.r.t. 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 an MRC algorithm.

3. SYMMETRIC AGGREGATION WITH MRC

Let I be a doamin, an *n-ary aggregation* α is a function[9]: $I^n \to I$. α is symmetric or commutative[9] if $\alpha(X) = \alpha(\sigma(X))$ for any $X \in I$ 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, therefore input is considered as a *multiset*. In this section, we define a generic framework to map symmetric aggregation into an MRC algorithm.

3.1 A Generic Form for Symmetric Aggregation

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

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

where F is translating function(tuple at a time), $\oplus$ is a commutative and associative binary operation, and T is terminating function. For instance, *average* can be easily transformed into well-formed aggregation: $F(d) = (d, 1)$, $(d, k) \oplus (d', k') = (d + d', k + k')$ and $T((d, n)) = \frac{d}{n}$. In fact, any symmetric aggregation can be rewritten into well-formed aggregation with 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(the case of one key and trivially extending to any number of keys), noted by $MR(\alpha)$, where mapper input is a submultiset X_i of X and mapper output is o_i, and $\sum_\oplus$ is the concatenation of $\oplus$.

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

	operation
mapper	$\sum_{\oplus, d_j \in X_i} F(d_j)$
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.

Definition 1. A symmetric aggregation function α defined on domain I is reducible if the 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 $MR(\alpha)$ of a symmetric aggregation is a 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 an MRC algorithm if and only if α is reducible.*

3.2 Deriving MRC Algorithm from Algebraic Properties

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

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

$$F = \alpha, \ \ , \oplus = \alpha, \ \ T = id \tag{1}$$

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

Another kind of aggregate function having the same behavior as symmetric and distributive aggregation is **commutative semigroup aggregate function** [5]. An aggregation α is in this class if there exists a commutative semigroup $(H, \otimes)$, 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 an MRC algorithm.□

A more general property than commutative semi-group aggregation is symmetric and preassociative aggregate function. An aggregation α is *preassociative* [13] if it satisfies $\alpha(Y) = \alpha(Y') \implies \alpha(XYZ) = \alpha(XY'Z)$. According to [13], some **symmetric and preassociative**(unarily quasi-range-idempotent and continuous) aggregation functions can be constructed as $\alpha(\mathbf{X}) = \psi\left(\sum_{i=1}^{n} \varphi(x_i)\right), n \geq 1$, where ψ and φ are continuous and strictly monotonic function. For instance, $\alpha(X) = \sum_{i=1}^{n} 2 \cdot x_i$, where $\psi = id$ and $\varphi(x_i) = 2 \cdot x_i$. 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 an MRC algorithm.□

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 multiset Y and $\alpha(Y)^{|Y|}$ denotes $|Y|$ occurrences of $\alpha(Y)$. A well-known class of symmetric and barycentrically associative aggregation is **quasi-arithmetic mean** : $\alpha(\mathbf{X}) = f^{-1}\left(\frac{1}{n}\sum_{i=1}^{n} f(x_i)\right), n \geq 1$, 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 into well-formed aggregation $(F, \oplus, T)$ and the $MR(\alpha)$ is also an MRC algorithm,

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

4. ASYMMETRIC AGGREGATION

Many commonly used aggregation function is symmetric(commutative) such that the order of input data can be ignored, while *asymmetric aggregation 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, e.g. WMA(weighted moving average) and EMA(exponential moving average)[1], which are used to highlight trends.

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 of them 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

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

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 $\oplus$. We illustrate this problem and identify properties to have correct MRC algorithm for a class of asymmetric well-formed aggregation in the following.

4.2 Position-based Aggregation with MRC

We deal with a kind of asymmetric aggregation α called position-based aggregation, for which F^o is $F^o(\bar{X}, x_i) = h(i) \odot f(x_i)$, where $h()$ and $f()$ are unary functions, 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 the concatenation of subsequence, and i be the holistic position of x_i in $\bar{X}$ and j be the relative position of x_j in subsequence $\bar{S}_l$. Then $\sum_{\oplus} F^o(\bar{X}, x_i)$ of α on any subsequence S_l is

$$\sum_{\oplus, x_i \in \bar{S}_l} F^o(\bar{X}, x_i) = \sum_{\oplus, x_j \in \bar{S}_l} h(j + k) \odot f(x_i),$$

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 distributed and parallel computing data set is split into ordered chunks and chunk indexes can be stored. It can be trivially implemented in Hadoop[16]. Secondly, k is needed, the number of elements before $\bar{S}_l$. Sequential distributing subsequence count values then starting aggregation is costly due to too many times of data transferring on network. If k can be extracted out of $\sum_{\oplus, x_j \in \bar{S}_l} h(j+k) \odot f(x_i)$, then α can be processed without distributing counts because operations relating to count 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 $h(i + k) = h(i) \otimes h(k + c)$ with a constant c, 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 exact operations.

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

Extractable property of position-based aggregation α allows previous subsequences count value 'k' to be extracted out of mapper operation, then α can be correctly processed by $\sum_{\oplus} F^o$ or $(\sum_{\oplus} f(x_i), \sum_{\oplus} h(i))$ at mapper phase. To combine mapper outputs, more than $\oplus$ and T are needed and specific combining operation depends on the three different extractable conditions (provided in our extended report[2]).

For instance, given an input sequence $\bar{X} = (x_1, ..., x_n)$, then $EMA(\bar{X}) = \frac{\sum_{i=1}^{n}(1-a)^{i-1} \cdot x_i}{\sum_{i=1}^{n}(1-a)^{i-1}}$, where a is a constant between 0 and 1. We give below the asymmetric well-formed aggregation of EMA, where $h(i) = (1-a)^{i-1}$,

$$F^o : F^o(\bar{X}, x_i) = \Big(h(i) \cdot x_i,\ h(i)\Big),$$

$$\oplus : \Big(h(i) \cdot x_i,\ h(i)\Big) \oplus \Big(h(i+1) \cdot x_{i+1},\ h(i+1)\Big) = \Big(h(i) \cdot x_i + h(i+1) \cdot x_{i+1},\ h(i) + h(i+1)\Big),$$

$$T : T(\sum_{i=1}^{n} h(i) \cdot x_i, \sum_{i=1}^{n} h(i)) = \frac{\sum_{i=1}^{n} h(i) \cdot x_i}{\sum_{i=1}^{n} h(i)}.$$

It is clearly that EMA is a position-based aggregation, and EMA is reducible because $\oplus$ is a pair of addition. Moreover $h()$ satisfies $h(i+k) = h(i) \cdot h(k+1)$, and the corresponding three binary operations $\otimes = \cdot$, $\odot = \cdot$, $\oplus = +$ satisfy the second extractable condition. Therefore 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 mapper input is $S_l,\ l \in \{1, ..., m\}$, and $count(S_0) = 0$,

- mapper:$\Big(OM_l^{'} = \sum_{x_j \in S_l} h(j) \cdot x_j, OM_l^{''} = \sum_{x_j \in S_l} h(j), OM_l^{'''} = count(S_l)\Big)$,
- 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 algorithms and we identified properties that enable to efficiently execute symmetric and asymmetric aggregations using MapReduce-style platforms. For symmetric aggregation, we proposed the reducible property within well-formed aggregation framework to satisfy space and time complexity of MRC. Several algebraic properties of symmetric aggregation leading to a generic MRC algorithm 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 reducible property of asymmetric well-formed aggregation to have MRC algorithms.

Our future work will be devoted to the implementation and experimentation. We will study the extension of our framework to mainstream parallel computing platforms (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 a complex composition of aggregation functions).

6. REFERENCES

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

[2] Symmetric and asymmetric aggregate function in massively parallel computing(extened version). https://hal-clermont-univ.archives-ouvertes.fr/hal-01533675.

[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.