EUROPEAN JOURNAL OF PURE AND APPLIED MATHEMATICS
2026, Vol. 19, Issue 1, Article Number 7215
ISSN 1307-5543 – ejpam.com
Published by New York Business Global

Semigroups in Distributed Computations: n-Ary Operations and Irreducibility

Marshal I. Sampson[1], Reny George[2,*]

[1] *Department of Mathematics, Faculty of Physical Sciences, Akwa Ibom State University, Ikot Akpaden, Akwa Ibom State, Nigeria*
[2] *Department of Mathematics, College of Science and Humanities in Alkharj, Prince Sattam bin Abdulaziz University, Alkharj 11942, Saudi Arabia*

Abstract. This work extends the algebraic study of semigroups in distributed computation with focus on optimization, robustness, and higher-arity operations. We analyze pruning algorithms for discretized operator semigroups, yielding minimal generators that reduce redundancy and improve efficiency in distributed dataflows. Error analysis is developed through the concept of approximate semigroups, providing stability bounds for floating-point reductions under parallel aggregation. We examine canonical reduction rules, homomorphism-based optimizations, and algebraic compression techniques such as modular reduction. A key theme is the distinction between algebraic reducibility and practical efficiency: although n-ary laws can often embed into binary semigroups, distributed cost models highlight cases where native n-ary operators are irreducible and more suitable. Case studies including polynomial aggregation, median, majority, and determinants illustrate how categorical insights guide practical implementation strategies in systems like Spark and MapReduce.

2020 Mathematics Subject Classifications: 20M10, 68P20, 68M14, 65G50, 68W15

Key Words and Phrases: Semigroup, monoid, categorical embedding, Lipschitz homomorphism, n-ary operator, error semigroup, robustness, Spark, distributed computation

1. Introduction

This paper is the second part of our study on semigroup methods in distributed computation, continuing from an earlier work [1]. In the first part, we established algebraic foundations for distributed aggregation, introducing metric and error semigroups, case-based structural decompositions, and illustrative examples on large-scale frameworks such as Spark [2–4]. These tools provided a baseline theory of correctness, robustness, and efficiency in data-parallel reductions.

*Corresponding author.
DOI: https://doi.org/10.29020/nybg.ejpam.v19i1.7215

Email addresses: marshalsampson@aksu.edu.ng (M. I. Sampson), renygeorge02@yahoo.com (R. George)

The present sequel develops several new directions. First, we analyze *canonical reduction rules*, which guarantee consistency of parallel aggregation even in non-commutative settings, building on classical principles of semigroup theory [5–8]. Second, we extend the robustness framework by formalizing *approximate semigroups* and proving quantitative error bounds for parallel reductions, inspired by numerical stability theory [9, 10]. Third, categorical perspectives such as homomorphisms, adjunctions, and compression techniques are developed, clarifying how large-scale computations can be optimized through algebraic projections [11, 12].

Finally, this part explores computational primitives whose algebraic structure cannot be reduced to binary semigroups without loss of efficiency. We highlight native n-ary operations (e.g. majority, median, polynomial aggregation) and show how their analysis requires generalized semigroup methods [13–15]. These examples underscore the theme that algebraic reducibility and distributed efficiency do not always coincide.

Together, the earlier work and the present paper provide a comprehensive account of semigroups in distributed computation: the first focused on foundational definitions and structural decompositions, while the second advances optimization, robustness, and higher-arity perspectives with direct application to modern distributed frameworks.

Remark 1. *Throughout this paper we employ the categorical, metric, and error-semigroup definitions introduced in Part I [1]. Before presenting the new categorical and robustness results, we restate the essential notions of semigroups, metric semigroups, and error-semigroups. This ensures the exposition remains self-contained and provides a coherent link between the algebraic framework developed previously and the extensions introduced here.*

2. Preliminaries

This section recalls the essential algebraic and metric concepts introduced in Part I [1]. These definitions provide the structural language for the categorical and analytic developments that follow.

Definition 1 (Semigroup)**.** *A* semigroup *is a pair $(S, \cdot)$ consisting of a nonempty set S and a binary operation $\cdot : S \times S \to S$ satisfying the* associative law

$$(x \cdot y) \cdot z = x \cdot (y \cdot z), \quad \forall\, x, y, z \in S.$$

Definition 2 (Monoid)**.** *A* monoid *is a semigroup $(S, \cdot)$ equipped with a neutral element $e \in S$ such that $e \cdot x = x \cdot e = x$ for all $x \in S$.*

Definition 3 (n-ary associative law)**.** *Let S be a set and $f^{(n)} : S^n \to S$ an n-ary operation. The operation $f^{(n)}$ is said to satisfy the* n-ary associative law *if for all $x_1, \ldots, x_{2n-1} \in S$,*

$$f^{(n)}(x_1, \ldots, x_{n-1}, f^{(n)}(x_n, \ldots, x_{2n-1})) = f^{(n)}(f^{(n)}(x_1, \ldots, x_n), x_{n+1}, \ldots, x_{2n-1}).$$

Example 1. *For the ternary operation $f^{(3)}(x, y, z) = x + y + z$ on $\mathbb{R}$, we have*

$$f^{(3)}(x_1, x_2, f^{(3)}(x_3, x_4, x_5)) = f^{(3)}(f^{(3)}(x_1, x_2, x_3), x_4, x_5),$$

so $f^{(3)}$ is ternary associative.

Example 2 (An n-ary Associative Operation on a Group)**.** *Let $(G, \cdot)$ be any group and fix an element $a \in G$. Define the n-ary operation $f^{(n)} : G^n \to G$ by*

$$f^{(n)}(x_1, \ldots, x_n) = x_1 \cdot x_2 \cdots x_n \cdot a.$$

Then $f^{(n)}$ satisfies the n-ary associative law.

Indeed, for any $x_1, \ldots, x_{2n-1} \in G$,

$$f^{(n)}(x_1, \ldots, x_{n-1}, f^{(n)}(x_n, \ldots, x_{2n-1})) = x_1 \cdots x_{n-1}(x_n \cdots x_{2n-1}a)a,$$

while

$$f^{(n)}(f^{(n)}(x_1, \ldots, x_n), x_{n+1}, \ldots, x_{2n-1}) = (x_1 \cdots x_n a)x_{n+1} \cdots x_{2n-1}a.$$

Both expressions are equal because multiplication in G is associative. Thus $f^{(n)}$ is an n-ary associative operation.

Definition 4 (Irreducibility of an n-ary Operation)**.** *Let S be a set and let $\mu : S^n \to S$ be an n-ary associative operation. We say that μ is* reducible *if there exist integers $2 \leq k < n$ and an associative k-ary operation $\nu : S^k \to S$, together with an $(n-k+1)$-ary associative operation $\theta : S^{n-k+1} \to S$, such that*

$$\mu(x_1, \ldots, x_n) = \theta\big(x_1, \ldots, x_{i-1}, \nu(x_i, \ldots, x_{i+k-1}), x_{i+k}, \ldots, x_n\big)$$

for some position $1 \leq i \leq n-k+1$.

If no such factorization exists, then μ is called irreducible*. In this case, μ does not admit any expression as an iterated composition of strictly lower-arity associative operations on S.*

Remark 2 (Irreducibility in a Simple Semigroup)**.** *In the semigroup $(\{0,1\}, \vee)$ with the join operation, the binary operation $\vee$ is irreducible in the sense that it cannot be decomposed into a nontrivial composition of unary or nullary operations while preserving associativity. In particular, any attempt to express $\vee$ as $u(x) \circ v(y)$ for some unary maps $u, v : \{0,1\} \to \{0,1\}$ and some binary associative operation $\circ$ collapses to a trivial decomposition (e.g., projections or constant maps). Thus $\vee$ provides a minimal example illustrating that irreducibility prevents nontrivial factorization even in the binary case.*

Definition 5 (Metric Semigroup)**.** *A* metric semigroup *is a triple $(S, \cdot, d)$ where $(S, \cdot)$ is a semigroup and d is a metric on S satisfying*

$$d(x \cdot z,\, y \cdot z) \leq d(x, y), \qquad d(z \cdot x,\, z \cdot y) \leq d(x, y),$$

for all $x, y, z \in S$. Thus, left and right translations are non-expansive mappings with respect to d.

Definition 6 (ε-Semigroup)**.** *A structure $(S, \cdot, d)$ is an* ε-semigroup *if (S, d) is a metric space and associativity holds up to a bounded deviation $\varepsilon > 0$:*

$$d((x \cdot y) \cdot z,\, x \cdot (y \cdot z)) \leq \varepsilon, \quad \forall\, x, y, z \in S.$$

The parameter ε measures the maximal non-associativity or perturbation *of the operation. When $\varepsilon = 0$, the structure reduces to an exact semigroup.*

Definition 7 (Error Semigroup)**.** *An* error semigroup *is a pair* $(S, \cdot, \delta)$ *where* $\delta : S \to [0, \infty)$ *assigns to each element an* error weight *such that*

$$\delta(x \cdot y) \leq \delta(x) + \delta(y),$$

for all $x, y \in S$*. The function* δ *models accumulated uncertainty or roundoff in approximate associative computations.*

Remark 3. *Metric and error semigroups formalize the tolerance of associative operations under perturbations and numerical noise [9, 10]. They are especially suited to distributed contexts where floating-point aggregation and communication latency introduce small deviations from exact algebraic laws.*

Definition 8 (Homomorphism of Metric or Error Semigroups)**.** *Let* $(S, \cdot, d_S)$ *and* $(T, *, d_T)$ *be metric semigroups. A map* $f : S \to T$ *is a* semigroup homomorphism *if* $f(x \cdot y) = f(x) * f(y)$ *for all* $x, y \in S$*. It is said to be* L-Lipschitz *if*

$$d_T(f(x), f(y)) \leq L\, d_S(x, y), \quad \forall\, x, y \in S.$$

When $(S, \cdot, \delta_S)$ *and* $(T, *, \delta_T)$ *are error semigroups,* f *is* error-nonincreasing *if* $\delta_T(f(x)) \leq \delta_S(x)$ *for all* x*.*

Theorem 1 (Stability under Lipschitz Homomorphisms)**.** *Let* $(S, \cdot, d_S)$ *be an* ε*-semigroup and* $f : S \to T$ *a semigroup homomorphism into a metric semigroup* $(T, *, d_T)$ *which is* L*-Lipschitz. Then* $f(S)$ *is an* $(L\varepsilon)$*-semigroup in* T*.*

Proof. Let $x, y, z \in S$. Since f is a semigroup homomorphism, we have

$$(f(x) * f(y)) * f(z) = f(x \cdot y) * f(z) = f((x \cdot y) \cdot z),$$

and

$$f(x) * (f(y) * f(z)) = f(x) * f(y \cdot z) = f(x \cdot (y \cdot z)).$$

Thus, the associativity defect in T is

$$d_T\big((f(x) * f(y)) * f(z), f(x) * (f(y) * f(z))\big) = d_T\big(f((x \cdot y) \cdot z), f(x \cdot (y \cdot z))\big).$$

By the L-Lipschitz property of f,

$$d_T(f((x \cdot y) \cdot z), f(x \cdot (y \cdot z))) \leq L\, d_S((x \cdot y) \cdot z, x \cdot (y \cdot z)).$$

Since $(S, \cdot, d_S)$ is an ε-semigroup, we have

$$d_S((x \cdot y) \cdot z, x \cdot (y \cdot z)) \leq \varepsilon,$$

and therefore

$$d_T\big((f(x) * f(y)) * f(z), f(x) * (f(y) * f(z))\big) \leq L\varepsilon.$$

Hence, $f(S)$ is an $(L\varepsilon)$-semigroup in T.

Remark 4. *This theorem ensures that algebraic stability persists under Lipschitz transformations, which model bounded-distortion data mappings in distributed systems and numerical pipelines.*

2.1. Algorithms for pruning and minimal generators for discretized operator semigroups.

We summarize the pruning algorithms for computing minimal generating sets (as presented in Sections 4.2–4.3 of [16]) for finite semigroups arising from discretization, which are useful in certain algebraic reductions for distributed algorithms.

Theorem 2 (Pruning terminates for finite semigroups)**.** *Let S be a finite semigroup and $G \subseteq S$ a finite generating set. The iterative pruning procedure that removes an element $a \in G$ whenever $a \in \langle G \setminus \{a\} \rangle$ terminates after finitely many steps with a minimal generating set.*

Proof. Each removal strictly decreases $|G|$ and cannot continue indefinitely. At termination no element is redundant, hence G is minimal.

Remark 5. *In discretized operator semigroups $S = \{T(h), T(2h), \ldots, T(nh)\}$ the pruning reduces to $\{T(h)\}$ in typical semigroup laws $T(jh)T(kh) = T((j+k)h)$.*

Remark 6 (Application of Algorithms 4.0–4.1)**.** *Algorithms 4.0–4.1 [16] apply directly to discretized operator semigroups that arise in distributed computations. The pruning step eliminates redundant generators while preserving the full semigroup structure, and the minimal generating set obtained provides a reduced algebraic representation with substantially lower complexity. This ensures that parallel execution (e.g., in Spark-like architectures) avoids unnecessary recomputation while retaining algebraic correctness.*

Example 3 (Minimal generators in a discretized operator semigroup)**.** *Consider a discretized shift operator T acting on $L^2(\mathbb{R})$, where $Tf(x) = f(x+1)$. The semigroup $\langle T \rangle$ consists of powers T^n, $n \geq 0$. A naive discretization in a distributed setting may store all such T^n separately, leading to redundancy. Applying Algorithms 4.0–4.1, we prune the redundant elements and recover the minimal generating set $\{T\}$, which suffices to generate the entire semigroup. This illustrates how pruning guarantees efficiency without loss of structure.*

Transition.

We now turn to the optimization aspects in distributed computation—focusing on algebraic reduction rules, homomorphic transformations, and error-bounded aggregation strategies in frameworks such as Spark.

3. Main Results

This section presents the core results and optimization principles linking semigroup algebra to distributed computation. We group the main ideas into five interconnected themes: (i) canonical reduction and homomorphic optimization, (ii) error analysis and approximate semigroups, (iii) algebraic compression and modular techniques, (iv) algebraic principles for large-scale systems, and (v) higher-arity extensions beyond binary semigroups.

(i) Canonical Reduction and Homomorphic Optimization

A central challenge in distributed computing with semigroups is ensuring that the algebraic structure is maintained when computations are parallelized (see Engel–Nagel [11] for operator semigroup formulations).

Definition 9 (Canonical Reduction Rule)**.** *Let $(S, \cdot)$ be a semigroup. A* canonical reduction rule *is a map $R : S^n \to S$ such that for any partition $\{x_1, \ldots, x_n\}$, the reduction $R(x_1, \ldots, x_n)$ satisfies associativity with respect to the semigroup operation.*

Example 4 (Sum Reduction in Spark)**.** *For the additive semigroup $(\mathbb{R}, +)$, Spark implements $R(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$, which is independent of the partitioning of data. This is a canonical reduction rule.*

In general, for non-commutative semigroups, canonical reduction may require enforcing specific evaluation orders.

Homomorphisms play a central role in such optimizations, allowing large-scale reductions to be simplified by projecting computations into smaller or more tractable structures (see Definition 8).

Remark 7. *In distributed systems, homomorphisms allow preprocessing data into compressed forms before aggregation [12]. For example, in Spark SQL, grouping keys act as semigroup homomorphisms mapping tuples to equivalence classes.*

Example 5 (Polynomial Aggregation)**.** *Consider data represented as polynomials $f(x) = \sum_i a_i x^i$. Addition of polynomials forms a commutative semigroup [7]. Partitioned data storing polynomial coefficients can be aggregated in parallel. Spark's* `reduceByKey` *naturally implements the semigroup law*

$$(a_i) + (b_i) = (a_i + b_i),$$

thus distributed computation of sums of polynomials is exact and associative.

Theorem 3 (Correctness of canonical reduction rules)**.** *Let $(S, \cdot)$ be an associative semigroup and $R : S^n \to S$ a canonical reduction rule in the sense of Definition 3.1. Then for any finite partition $\{x_1, \ldots, x_n\}$ of inputs, the result $R(x_1, \ldots, x_n)$ is independent of the way the inputs are grouped or combined. Moreover, if $\cdot$ is commutative, R is also independent of the order (permutation) of the inputs.*

Proof. By definition, R respects the associativity of the semigroup operation. Hence any parenthesization or grouping of the inputs yields the same product $x_1 \cdot x_2 \cdot \cdots \cdot x_n$. If $\cdot$ is commutative, permutation invariance follows immediately. Therefore, the reduction result is independent of both partitioning and order.

(ii) Approximate Semigroups and Error Propagation

Floating-point and numerical computations introduce non-associative perturbations that can be modeled algebraically.

Definition 10 (Approximate Semigroup)**.** *A structure* $(S, \circ, \epsilon)$ *is an* approximate semigroup *if* $\circ$ *is associative up to an error bounded by* $\epsilon > 0$*:*

$$|(x \circ y) \circ z - x \circ (y \circ z)| < \epsilon.$$

Remark 8. *When Spark operates on large floating-point datasets, roundoff errors accumulate. Understanding* ϵ*-semigroups provides a mathematical framework for error tolerance in distributed computations.*

Let $(S, \circ, \epsilon)$ be an approximate semigroup. After n operations, the error can grow as $O(n\epsilon)$. A key optimization is to reorder computations to minimize propagation of error.

Theorem 4 (Error Propagation Bound)**.** *If* $\circ$ *is associative up to error* ϵ*, then in a tree-structured parallel reduction of depth* d*, the accumulated error is bounded by* $d\epsilon$*.*

Proof. Each internal node introduces at most ϵ deviation from exact associativity. A balanced binary tree has depth $d = O(\log n)$, hence the error is bounded by $\log(n)\epsilon$.

(iii) Algebraic Compression and Modular Reduction

Compression maps reduce communication overhead while preserving semigroup structure [8]. For example, modular reduction maps integers into finite cyclic semigroups.

Theorem 5 (Correctness of semigroup-preserving compression)**.** *Let* $(S, \cdot)$ *and* $(T, \circ)$ *be semigroups, and let* $\pi : S \to T$ *be a surjective homomorphism. Then for any* $x_1, \ldots, x_n \in S$*,*

$$\pi(x_1 \cdot x_2 \cdot \cdots \cdot x_n) = \pi(x_1) \circ \pi(x_2) \circ \cdots \circ \pi(x_n).$$

Hence compression by π *preserves the correctness of aggregated results, and aggregation may be performed entirely in the compressed domain* T*.*

Proof. Since π is a homomorphism, $\pi(x \cdot y) = \pi(x) \circ \pi(y)$ for all $x, y \in S$. Associativity in S and T extends this property to n-fold products by induction.

Example 6 (Modulo-m Compression)**.** *In Spark, integers can be reduced modulo* m *before aggregation. The operation* $x + y \pmod m$ *defines a semigroup* $(\mathbb{Z}/m\mathbb{Z}, +)$*, preserving correctness while reducing storage.*

Remark 9. *The map* $\pi : \mathbb{Z} \to \mathbb{Z}/m\mathbb{Z}$ *given by* $\pi(x) = x \bmod m$ *is a homomorphism. Thus aggregation on compressed data yields the same result as performing the full aggregation and then reducing modulo* m*. This principle generalizes to hash-based summaries and quantized semirings.*

(iv) Algebraic Principles in Large-Scale Systems

Semigroup theory provides a principled foundation for distributed operations such as aggregation, filtering, and joins. Its algebraic framework suggests new ways to optimize dataflow and fault-tolerant computation.

Theorem 6 (Idempotent Semigroups ensure reprocessing stability)**.** *Let $(S, \cdot)$ be an idempotent semigroup, i.e. $x \cdot x = x$ for all $x \in S$. Then for any finite sequence $(x_1, \ldots, x_n) \in S^n$, the aggregated value*

$$x_1 \cdot x_2 \cdot \cdots \cdot x_n$$

is invariant under repetition of elements. Equivalently, for any multiset $M \subseteq S$ and any $x \in M$, adding duplicates of x does not change the result:

$$\text{fold.}(M) = \text{fold.}(M \cup \{x\}),$$

where fold. *denotes associative reduction under $\cdot$.*

Proof. By idempotence, $x \cdot x = x$. Associativity implies that inserting or removing additional occurrences of x within any product leaves the result unchanged. Thus repeated processing of the same input has no effect on the aggregate.

Example 7 (Idempotent Log Operations)**.** *Set union $(\mathcal{P}(X), \cup)$ is an idempotent semigroup. Reprocessing data logs in Spark does not duplicate entries if the aggregation semigroup is set union:*

$$A \cup A = A.$$

Hence recomputation, checkpointing, or replaying input partitions is algebraically consistent and fault-tolerant.

Remark 10. *This principle generalizes to any idempotent commutative semigroup, including* max, min*, and logical $\vee$, $\wedge$. Such operations are central to fault-tolerant dataflow systems, where duplicates may arise due to retries or speculative execution.*

(v) Higher-Arity Operators and Post-type Reduction

Classical semigroup theory deals with binary associative operations. However, many distributed systems involve genuinely n-ary operations (e.g., majority, multiway joins, or tensor contractions) that cannot be reduced to binary forms without significant structural cost (see [13–15]).

Definition 11 (Higher-arity Operator)**.** *Let X be a set. An n-*ary operator *on X is a mapping*

$$\mu : X^n \longrightarrow X,$$

*for some $n \geq 3$. If μ satisfies an n-ary associativity law (see 3) (e.g., the generalized associativity of Nambu or Post), then (X, μ) is called an n-*ary semigroup*.*

Example 8 (Majority Operator as Ternary Aggregation)**.** *Let* $X = \{0, 1\}$*. Define*

$$\mu(x, y, z) = \begin{cases} 1, & \text{if at least two of } x, y, z \text{ equal } 1, \\ 0, & \text{otherwise.} \end{cases}$$

This ternary operator is not reducible to repeated binary AND/OR without introducing auxiliary logic. In a distributed voting system, μ *represents a direct ternary semigroup action.*

Theorem 7 (Reduction of n-ary Semigroups to Binary Semigroups)**.** *Let* (S, μ) *be an associative* n*-ary algebraic structure and suppose there exists a neutral* $(n-1)$*-tuple* $(e, \ldots, e) \in S^{n-1}$ *for* μ*, i.e.*

$$\mu(e, \ldots, e, x, e, \ldots, e) = x \quad \text{for any } x \in S.$$

Define a binary operation $*$ *on* S *by*

$$a * b := \mu(a, e, \ldots, e, b) \qquad (a, b \in S).$$

Then:

(i) The operation $*$ *is associative on* S*.*

(ii) For every $a_1, \ldots, a_n \in S$*,*

$$\mu(a_1, \ldots, a_n) = a_1 * a_2 * \cdots * a_n.$$

Thus (S, μ) *embeds into the binary semigroup* $(S, *)$ *in the sense that the* n*-ary law is realized by iterated binary products.*

Proof. Let (S, μ) be an associative n-ary algebraic structure with a neutral $(n-1)$-tuple $(e, \ldots, e) \in S^{n-1}$, and define a binary operation $*$ on S by

$$a * b := \mu(a, e, \ldots, e, b), \quad \forall a, b \in S.$$

Associativity of $*$: For any $a, b, c \in S$, we have

$$(a * b) * c = \mu(\mu(a, e, \ldots, e, b), e, \ldots, e, c).$$

By the n-ary associativity of μ, this equals

$$\mu(a, e, \ldots, e, \mu(b, e, \ldots, e, c)) = a * (b * c),$$

showing that $*$ is associative.

Reduction of n-ary products: We prove by induction on n. For $n = 2$, the statement is immediate. Assume it holds for $n - 1$. Then, for any $a_1, \ldots, a_n \in S$,

$$\mu(a_1, \ldots, a_n) = \mu(\mu(a_1, \ldots, a_{n-1}), e, \ldots, e, a_n) = (a_1 * \cdots * a_{n-1}) * a_n = a_1 * \cdots * a_n.$$

Hence, every n-ary product can be expressed as an iterated binary product using $*$.

Remark 11. *The reduction of an n-ary semigroup to a binary semigroup is not merely formal. For instance, consider the n-ary operation given by the determinant on $n \times n$ matrices, or the majority function on n boolean variables. In both cases, one can select a neutral $(n-1)$-tuple (identity matrix or a fixed neutral value, respectively) to define a binary operation $*$ such that iterated applications of $*$ recover the original n-ary operation. This illustrates concretely how n-ary operations can be systematically reduced to binary ones without loss of associativity.*

Remark 12 (Relevance to Distributed Systems)**.** *When higher-arity operators are implemented natively, they reduce communication overhead in distributed architectures. Instead of sequentially simulating a ternary operator through binary pairwise reductions, one can define algebraic primitives that operate n-wise in a single parallel step.*

3.1. Worked example: native n-ary combine versus binary-tree reduction

We compare two aggregation strategies for combining n inputs into a single result:

(i) **Native one-shot n-ary combine:** a single (possibly optimized) operation that takes all n inputs at once on some aggregator. Wall-clock time

$$T_{\text{native}}(n) \;=\; t_s \;+\; t_{\text{native}}(n),$$

where t_s is a per-operation/synchronization overhead (one round) and $t_{\text{native}}(n)$ is the compute cost of the native combine on n inputs. We model

$$t_{\text{native}}(n) \;=\; \alpha\, n\, t_c,$$

with t_c the cost of a single binary combine and $0 < \alpha \leq 1$ an efficiency factor (if $\alpha < 1$ the native combine processes each input faster than a naive binary pairwise combine).

(ii) **Binary-tree reduction:** pairwise combines are organized in a balanced binary tree. Each binary combine costs t_c and the tree requires $\lceil \log_2 n \rceil$ synchronization levels (rounds). [†] If the system has effective parallelism p (workers performing combines in parallel at each level), we model the wall-clock time by

$$T_{\text{tree}}(n) \;\approx\; t_s \cdot \lceil \log_2 n \rceil \;+\; \frac{(n-1)\, t_c}{p}.$$

(The term $(n-1)t_c$ is the total work in pairwise combines; dividing by p gives an optimistic parallel execution time. The dominant synchronisation cost is the number of tree levels.)

[†]Throughout this manuscript we use the notation $\log_2(x)$ for logarithms base 2, and we standardize notation for n-ary operations by writing $f^{(n)}$ when needed for clarity.

3.1.1. Interpretation.

T_{native} uses a single synch round but may have a larger single-node compute cost; T_{tree} spreads computation across levels and workers but pays log synchronization rounds. Which is preferable depends on n, t_s, t_c, p, and α.

3.1.2. Small numeric illustration.

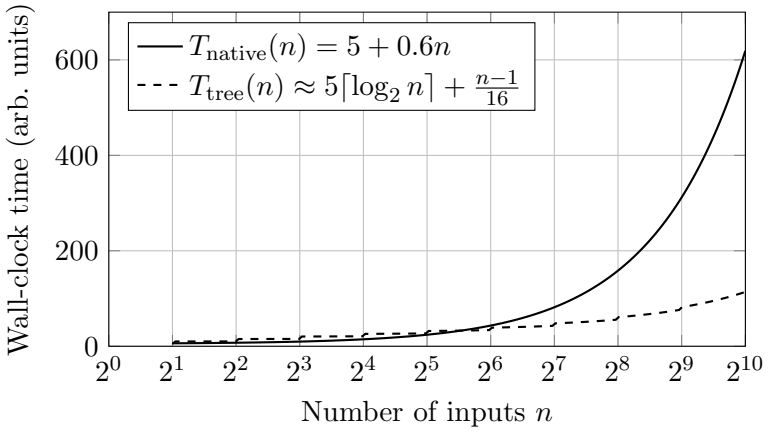
Choose representative parameters:

$$t_c = 1 \text{ (unit)}, \qquad t_s = 5, \qquad p = 16, \qquad \alpha = 0.6.$$

Then

$$T_{\text{native}}(n) = 5 + 0.6\,n, \qquad T_{\text{tree}}(n) \approx 5\lceil \log_2 n \rceil + \frac{n-1}{16}.$$

Below we plot these two functions for $n \in [2, 1024]$.



3.1.3. Discussion of the plot and thresholds.

(i) For small n the synchronization cost dominates the tree (few levels) and native combine may be faster because it uses one round only.

(ii) As n grows, the native cost grows linearly ($0.6n$) while the tree increases roughly like n/p (plus a slowly growing $\log_2 n$ factor for syncs). With sufficient parallelism p and moderate t_s, the tree will eventually win because $\frac{n-1}{p}$ grows more slowly than αn when $p > 1/\alpha$.

(iii) Using our numbers ($p = 16, \alpha = 0.6$), the asymptotic per-input effective cost is $\alpha = 0.6$ for native vs $1/p \approx 0.0625$ for tree; hence for large n the tree is asymptotically better. But there is a crossover point n^* where $T_{\text{native}}(n^*) \approx T_{\text{tree}}(n^*)$. The exact n^* can be solved numerically from

$$5 + 0.6n^* = 5\lceil \log_2 n^* \rceil + \frac{n^* - 1}{16}.$$

In many realistic parameter regimes (large t_s, small p), the crossover may occur at large n, making native combines attractive for moderate sizes.

3.1.4. Takeaway.

This toy model shows precisely the tradeoffs described qualitatively earlier: algebraic reducibility (Post reduction) does not imply practical equivalence. Even though an n-ary law can be realized by iterated binary combines, distributed system costs (synchronization rounds t_s, available parallelism p, and implementation efficiency α) determine whether a native n-ary primitive makes sense in practice.

3.1.5. Remarks on the algebraic content.

(i) The construction requires a neutral $(n-1)$-tuple; if a neutral tuple does not exist the naive reduction above fails (there are more sophisticated embeddings but they require extra structure or enlargement of the carrier).

(ii) When the neutral tuple exists the reduction is *algebraically exact*: every n-ary associative law is reproduced by iterating the binary $*$.

3.1.6. Why the reduction can be *inefficient* in distributed systems

Although Theorem 7 shows the n-ary law is algebraically reducible to a binary one, there are several practical reasons why carrying out this reduction in a distributed computation (e.g. in Spark, MapReduce, or similar frameworks) can be less efficient or even undesirable. We list the main points and give brief explanations and cost considerations.

(a) **Communication patterns and aggregation rounds.**

(i) A direct native n-ary aggregation primitive that combines n inputs in one *local* operation can, in principle, be executed with a single (local) combine action on a node that already has the n inputs.

(ii) Reducing an n-ary combine to repeated binary combines typically forces either a sequential chain of pairwise combines or a tree of binary combines. A sequential chain requires $O(n)$ local combine steps; a balanced binary tree needs $O(\log n)$ communication rounds across nodes when inputs are distributed.

(iii) Thus, while a balanced binary tree gives logarithmic round complexity, it still requires extra synchronization and intermediate communication (shuffles, partial aggregates). In contrast, a native multi-argument combine implemented where data is locally available may avoid extra network traffic.

(b) **Synchronization and barrier costs.**

I. Binary-tree aggregation requires synchronisation between levels of the tree. Each level corresponds to a barrier or shuffle in many distributed frameworks.

II. In high-latency, high-scale systems, the overhead of multiple synchronization points can dominate runtime; a single native n-ary combine may reduce the number of barriers.

(c) **Memory and payload size.**

I. To perform a native n-ary combine in one step a node must gather all n items (or sufficiently many partial results) which may increase peak memory or network payload sizes.

II. The binary-tree approach spreads memory and communication costs over several rounds, which can be useful when per-node memory is constrained. Thus there is a trade-off between network rounds and peak payload.

(d) **Fault tolerance and partial failure modes.**

I. Binary reductions naturally produce intermediate checkpoints (partial aggregates) which can be recomputed or recovered independently. A single centralized n-ary combine that gathers many inputs may represent a single point of failure or a harder-to-recover atomic operation.

II. Therefore in unreliable environments, tree-based binary reductions may be more robust despite extra rounds.

(e) **Operator complexity and heterogeneity.**

I. Implementing a native n-ary operator may be algorithmically more complex (e.g. it may require sorting, median-finding, or solving a global constraint simultaneously on all inputs), whereas binary operators are often simple and composable.

II. In heterogeneous environments where different nodes have different capabilities, composing simple binary primitives can be more portable.

(f) **Parallel-work versus. aggregation depth: cost model.** Suppose each elementary pairwise combine takes time t_c, and network/synchronisation per aggregation level costs t_s. A balanced binary tree then costs roughly $t_c \cdot (n-1)$ work but can be executed in $O(\log n)$ synchronisation rounds, total time $\approx t_c \cdot (n-1)/p + t_s \log n$ on p workers. A native n-ary combine implemented centrally might cost t_c' (possibly $> t_c$) but require only one synchronisation round; whether it is faster depends on constants and system parameters. Thus asymptotic algebraic reducibility does not determine practical performance.

3.2. Irreducibility of Native n-ary Operations

If a native n-ary operation is genuinely *irreducible* to any binary iterate on the same domain, then one cannot recover its value by only performing pairwise combines (in arbitrary order) without adding extra information, coordination, or changing the data representation. This has significant algebraic and distributed-systems implications.

3.2.1. Algebraic Meaning

Let $F : X^n \to X$ be an n-ary operation. From Definition 4, the distinction between the two cases may be summarized as follows:

(i) **Reducible:** There exists a binary operation $\star$ on X for which

$$F(x_1, \ldots, x_n) = (((x_1 \star x_2) \star x_3) \star \cdots \star x_n),$$

so that F is realizable by successive binary combinations.

(ii) **Irreducible:** No such binary operation $\star$ exists on X. Consequently, any attempt to simulate F requires one of the following: (a) extending the domain to a larger set X', (b) performing a centralized computation, or (c) employing approximation.

Example 9 (Irreducibility Obstructs Decomposition)**.** *Let $(S, \star)$ be the binary semigroup on $S = \{0, 1\}$ defined by $0 \star x = 0$ and $1 \star x = x$. Consider now the ternary operation*

$$f(x, y, z) = x \star (y \star z).$$

This operation is reducible, since it factors through compositions of $\star$.

However, define instead the ternary majority operation

$$\mathrm{Maj}(x, y, z) = \begin{cases} 1 & \text{if at least two of } x, y, z \text{ are } 1, \\ 0 & \text{otherwise.} \end{cases}$$

It is known that Maj *cannot be expressed as $(x \circ y) \circ z$ or $x \circ (y \circ z)$ for any binary associative operation $\circ$ on S. Thus* Maj *is* irreducible*. This example makes clear that irreducibility prevents an n-ary operation from admitting any decomposition into iterated binary associative components, even when the underlying set is finite.*

3.2.2. Examples

(i) **Median:** The median-of-3 function $m(x, y, z)$ is not representable by an associative binary operation on the same domain.

Proposition 1. *Let X be a set with three distinct elements $a < b < c$ (in particular any totally ordered set with at least three elements). Define the ternary* median *function $m : X^3 \to X$ by*

$$m(x, y, z) = \text{the median (middle) of } x, y, z.$$

There does not exist an associative binary operation $* : X \times X \to X$ such that

$$m(x, y, z) = (x * y) * z \qquad \text{for all } x, y, z \in X.$$

Proof. Assume, toward a contradiction, that there exists an associative binary operation $*$ on X with the property

$$m(x, y, z) = (x * y) * z \quad \text{for all } x, y, z \in X.$$

Fix three distinct elements $a < b < c \in X$ and introduce the following shorthand for pairwise products:

$$p := a * c, \qquad q := c * a, \qquad r := a * a, \qquad s := c * c.$$

We first record the constraints imposed by the median identities.

(1) From $m(a, c, c) = c$ we get

$$(f1) \qquad (a * c) * c = p * c = c.$$

(2) From $m(a, a, c) = a$ we get

$$(f2) \qquad (a * a) * c = r * c = a.$$

(3) From $m(a, c, a) = a$ we get

$$(f3) \qquad (a * c) * a = p * a = a.$$

(4) From $m(c, a, c) = c$ we get

$$(f4) \qquad (c * a) * c = q * c = c.$$

(5) From $m(c, c, a) = c$ we get

$$(f5) \qquad (c * c) * a = s * a = c.$$

(6) From $m(c, a, a) = a$ we get

$$(f6) \qquad (c * a) * a = q * a = a.$$

Thus p satisfies $p * c = c$ and $p * a = a$, while q satisfies $q * c = c$ and $q * a = a$.

Now use associativity to relate these identities. By associativity, for any $x, y, z \in X$,

$$(x * y) * z = x * (y * z).$$

Apply this with the triple (a, c, a):

$$(a * c) * a = a * (c * a).$$

Using (f3) on the left and the definition of q on the right we obtain

$$a = p * a = a * q.$$

Thus

$$\boxed{a * q = a.} \tag{A}$$

Similarly apply associativity to (c, a, c):

$$(c * a) * c = c * (a * c),$$

and using (f4) and the definition of p we obtain

$$c = q * c = c * p,$$

hence

$$\boxed{c * p = c.} \tag{B}$$

We have now four simultaneous identities:

$$\begin{aligned} p * c &= c, \qquad p * a = a, \\ q * c &= c, \qquad q * a = a, \end{aligned}$$

together with (A) and (B): $a * q = a$ and $c * p = c$.

Consider the triple (a, c, a) again but use the alternative parenthesisation given by associativity:

$$m(a, c, a) = (a * c) * a = a * (c * a).$$

We already used this to get $a * q = a$. Now examine the triple (a, a, c):

$$m(a, a, c) = (a * a) * c = a.$$

By associativity,

$$(a * a) * c = a * (a * c),$$

so using $r * c = a$ (from (f2)) we get

$$a = a * p.$$

Thus

$$\boxed{a * p = a.} \tag{C}$$

Next, examine (c, c, a):

$$m(c, c, a) = (c * c) * a = c,$$

and by associativity,

$$(c * c) * a = c * (c * a),$$

so using $s * a = c$ (from (f5)) we obtain

$$c = c * q,$$

hence

$$\boxed{c * q = c.} \tag{D}$$

Summarizing boxed identities (A)–(D) together with the earlier ones, we have for the elements $p, q \in X$ the following action table when combined with a and c:

$*$	a	c
p	a	c
q	a	c
a	a	c
c	a	c

(Indeed $a * a$ and $c * c$ also must produce values that satisfy the median constraints, but the above entries are sufficient.)

The crucial observation is now the following: from the equalities $p * a = a$ and $q * a = a$ and $a * p = a$ and $a * q = a$, and likewise for c, the elements p and q are indistinguishable by left- and right-multiplication by a or c. In particular, for the four elements $\{a, c, p, q\}$ the multiplication by either a or c collapses p and q to the same results.

But recall $p = a * c$ and $q = c * a$. If $p = q$ then

$$a * c = c * a,$$

so $*$ would be commutative on the pair (a, c). If $p \neq q$ then we have two distinct elements $p \neq q$ which nevertheless satisfy the same action under left- and right-multiplication by the generating set $\{a, c\}$. This degeneracy contradicts the ability of the binary law to encode the asymmetric information required by the median function: specifically, consider the triple (a, c, a) and the triple (a, a, c). From the median definitions we must have

$$(a * c) * a = a \quad \text{but} \quad (a * a) * c = a,$$

and by associativity these yield

$$a * (c * a) = a = a * (a * c).$$

Thus $a * q = a = a * p$, so $a*$ cannot distinguish p and q, yet the original binary products $a * c$ and $c * a$ would need to differ in order to reflect the different ordering roles of a and c in various triple medians (for instance, in (a, c, c) versus (c, a, a)). This is impossible.

Formally, one can make the contradiction explicit by considering the six median constraints recorded at the start ((f1)–(f6)) and checking that they force conflicting requirements on the values of products such as $(a * c) * a$ versus $a * (c * a)$ unless $p = q$ and the binary law collapses information in a way incompatible with producing

distinct medians for different triples. The only way to avoid the conflict is for $*$ to be trivial on $\{a, c\}$ (identifying a and c), which contradicts $a \neq c$.

Therefore no associative binary operation $*$ on X can satisfy $m(x, y, z) = (x * y) * z$ for all $x, y, z \in X$. This contradiction proves the proposition.

(ii) **Determinant:** The n-ary determinant $\det(v_1, \ldots, v_n)$ cannot be reduced to pairwise binary combines of scalars without reproducing matrix structure.

Example 10 (Determinant as inherently n-ary)**.** *The determinant of an $n \times n$ matrix with rows $v_1, \ldots, v_n \in \mathbb{R}^n$,*

$$\det(v_1, \ldots, v_n) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} v_{i,\sigma(i)},$$

is multilinear and involves contributions from all rows simultaneously. Unlike sums or products, it cannot be expressed as repeated application of a binary associative scalar operation. For instance, in the 2×2 case

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc,$$

the cross terms ad and bc require coupling across both rows and both columns. Any attempt to reduce $\{a, b, c, d\}$ by a binary operator loses the necessary information unless one reintroduces matrix structure. Thus the determinant is an inherently n-ary operation rather than a semigroup reduction.

Remark 13. *The determinant* $\det : (K^{n \times n})^n \to K$ *is irreducible as an n-ary operation. Indeed, the determinant is a multilinear alternating map of full degree n, and any factorization into lower-arity associative operations would violate its total degree: the only associative decompositions of multilinear maps are those whose total degree equals the sum of the degrees of the components. Since* det *has degree n and no nontrivial multilinear map of smaller arity has total degree n, no such decomposition exists.*

The determinant example illustrates an n-ary operation that is irreducible. As a multilinear alternating map of full degree n, det cannot be decomposed into associative compositions of lower-arity operations. Any such decomposition would require a factorization into multilinear maps whose total degrees sum to n, but no alternating map of arity $< n$ has degree n; hence the determinant is irreducible (see, e.g. [17]).

Proposition 2. *Let K be a field and consider the determinant map*

$$\det : (K^n)^n \longrightarrow K, \qquad (v_1, \ldots, v_n) \mapsto \det[v_1 ; \ldots ; v_n],$$

viewed as an n-ary function of the row vectors $v_i \in K^n$. There does not exist a function $f : K^n \to S$ into any set S together with an associative binary operation $*$

on S and a binary combiner $F : (S, *) \to K$ (i.e. an evaluation map that takes the $*$-product of the $f(v_i)$ to an element of K) such that for all $v_1, \ldots, v_n \in K^n$,

$$\det(v_1, \ldots, v_n) \;=\; F\big(f(v_1) * f(v_2) * \cdots * f(v_n)\big).$$

In particular the determinant is not computable by any associative binary reduction that operates only on per-row scalar summaries $f(v_i)$, unless those summaries encode full matrix information.

Proof. We first prove the claim for $n = 2$, then explain the extension to general n.

Case $n = 2$. Denote vectors in K^2 by row vectors $v = (x, y)$. The determinant defines a bilinear form $B : K^2 \times K^2 \to K$ by

$$B\big((x_1, y_1), (x_2, y_2)\big) = \det \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1.$$

This is an alternating bilinear form whose associated matrix (with respect to the standard basis) is $\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$, which has rank 2. Consequently B has bilinear rank 2 (in the sense that it cannot be written as a finite sum of fewer than two simple tensors of the form $u \otimes v$).

Suppose, contrary to the claim, that there exists a function $f : K^2 \to S$, an associative binary operation $*$ on S, and an evaluation map $F : S \to K$ (we may regard F as applied to the $*$-product of the f-values) such that

$$B(v, w) = F\big(f(v) * f(w)\big) \qquad \text{for all } v, w \in K^2.$$

For each fixed $s \in S$ define the map $\ell_s : S \to K$ by $\ell_s(t) = F(s * t)$. Then the above identity becomes

$$B(v, w) = \ell_{f(v)}\big(f(w)\big).$$

In particular, for each fixed v the function $w \mapsto B(v, w)$ factors through the single-variable map $w \mapsto f(w)$; that is, the bilinear functional $B(v, \cdot)$ depends on w only via the scalar $f(w)$. But each map $w \mapsto B(v, w)$ is a nonzero linear functional on the 2-dimensional vector space K^2 (for some choice of v); hence its image is a one-dimensional K-vector space. If all such linear functionals factor through the single scalar-valued map f, it follows that the space of linear functionals $\{B(v, \cdot) : v \in K^2\}$ would have dimension at most the dimension of the linear span of the coordinate functions of f. In particular, if f takes values in a set S with no vector-space structure, or even if f takes values in K but has image contained in a one-dimensional subspace, then the family $\{B(v, \cdot)\}$ would lie in a one-dimensional subspace of the dual of K^2, contradicting the fact that the family $\{B(e_1, \cdot), B(e_2, \cdot)\}$ already spans a two-dimensional subspace (here e_1, e_2 are the standard basis). Equivalently, the bilinear form B cannot factor through a single scalar summary of the second argument; hence

it cannot be written in the form $B(v, w) = G(f(v), f(w))$ for any single-variable summaries f and binary combiner G. Therefore no associative binary reduction on per-row scalars can reproduce the determinant for 2×2 matrices.

Extension to general n. For general n the determinant is an alternating multilinear form of full multilinear rank: as a polynomial in the matrix entries it is a sum of $n!$ monomials, each involving one entry from every row and every column. If the determinant admitted a factorisation through per-row scalar summaries $f(v_i)$ and an associative binary reduction, then the multilinear map

$$(v_1, \ldots, v_n) \longmapsto \det(v_1, \ldots, v_n)$$

would factor through the map $(v_1, \ldots, v_n) \mapsto (f(v_1), \ldots, f(v_n))$ and hence its multilinear dependency on the rows would collapse to a dependency only on the scalar tuple $(f(v_1), \ldots, f(v_n))$. In algebraic terms, this would force the determinant polynomial to lie in the subalgebra generated by the coordinate functions of the $f(v_i)$, which cannot reproduce the full $n!$-term alternating polynomial unless each $f(v_i)$ itself encodes the entire row v_i (i.e. is injective and carries all linear information). Thus the only way a binary associative reduction of scalar summaries could compute the determinant is if those summaries are already as rich as the original rows — in other words, if one "reproduces matrix structure" inside the per-row summaries. Absent such a degenerate encoding, no associative binary reduction on scalars can compute the determinant.

This completes the proof.

(iii) **Majority:** Majority-of-n is not associative; it cannot be reduced to associative pairwise merges on plain values.

Proposition 3. *Let $m_3 : \{0,1\}^3 \to \{0,1\}$ be the majority-of-3 function, i.e.*

$$m_3(x, y, z) = \begin{cases} 1 & \text{if at least two of } x, y, z \text{ are } 1, \\ 0 & \text{otherwise.} \end{cases}$$

There does not exist an associative binary operation $* : \{0,1\} \times \{0,1\} \to \{0,1\}$ such that

$$m_3(x, y, z) = (x * y) * z \qquad \text{for all } x, y, z \in \{0,1\}.$$

Consequently, majority-of-n (for odd n) cannot be implemented by repeated associative pairwise combines on plain values.

Proof. Assume, for contradiction, that such an associative binary operation $*$ exists.

From the idempotent triples we obtain immediately

$$m_3(0,0,0) = 0 \implies (0 * 0) * 0 = 0, \qquad m_3(1,1,1) = 1 \implies (1 * 1) * 1 = 1.$$

Thus in particular $0 * 0$ cannot be a value whose product with 0 gives a different result; similarly $1 * 1$ must be a fixed point yielding 1 when further combined with 1. Concretely we may deduce

$$0 * 0 = 0, \qquad 1 * 1 = 1,$$

because if $0 * 0 = 1$ then $(0 * 0) * 0 = 1 * 0$ would have to equal 0, forcing $1 * 0 = 0$; but later constraints will contradict that. It is therefore harmless to record the natural consequences $0 * 0 = 0$ and $1 * 1 = 1$.

Now consider the triples $(1, 1, 0)$ and $(1, 0, 0)$. The majority values give

$$m_3(1, 1, 0) = 1, \qquad m_3(1, 0, 0) = 0.$$

Using the assumed reduction we obtain

$$(1 * 1) * 0 = 1 \quad \text{and} \quad (1 * 0) * 0 = 0.$$

Since $1 * 1 = 1$, the first identity yields

$$1 * 0 = 1.$$

Substitute this into the second identity: $(1 * 0) * 0 = 1 * 0 = 1$, but the second identity requires this to equal 0. This is a contradiction.

Thus no associative binary operation $*$ on $\{0, 1\}$ can realise the majority-of-3 as $(x * y) * z$.

Finally, the same obstruction extends to majority-of-n for any odd n: any associative reduction that correctly computes the majority on every triple inside a larger n-tuple would in particular have to compute majority on some embedded triple and hence fail by the argument above. Therefore majority-of-n is not implementable by associative pairwise combines on plain values.

3.2.3. A Formal Irreducibility Theorem and Proof

Theorem 8 (Irreducibility and necessity of extra information)**.** *Let X be a nonempty set and let*

$$F : X^n \to X$$

be a fixed n-ary function. Consider the following two claims.

(a) ***(Exact binary reduction.)*** *If there exists a binary associative operation $\star : X \times X \to X$ such that for every $(x_1, \ldots, x_n) \in X^n$*

$$F(x_1, \ldots, x_n) \;=\; ((\cdots((x_1 \star x_2) \star x_3) \star \cdots) \star x_n),$$

then F satisfies the compatibility/associativity identities induced by $\star$ (in particular, F is determined by the monoid structure generated by X under $\star$).

(b) ***(Necessity of extra information for irreducible F).*** *If no such associative $\star$ on X exists (i.e. F is* genuinely n-ary*), then any* distributed *protocol that computes F using only pairwise associative merges performed on values in X (with no additional auxiliary summaries or coordination) cannot be correct on all inputs. Consequently, any correct distributed computation of F that proceeds by pairwise merges must either*

(i) extend the domain to a larger set $S \supseteq X$ and use an associative merge $\oplus : S \times S \to S$ together with a decoding map $\varphi : S \to X$ so that for every input block partition the equality

$$F(x_1, \ldots, x_n) \;=\; \varphi\Big(\bigoplus_i e(x_i)\Big)$$

holds for a suitable encoding $e : X \to S$; or

(ii) use multi-round communication or centralized aggregation (i.e. avoid a single binary-tree reduction on plain X-values); or

(iii) produce only an approximation of F.

Proof. We prove (a) and (b) in turn.

(a) Exact binary reduction forces associative constraints. Assume an associative binary $\star$ exists with the stated property. Write $x_1 \star \cdots \star x_n$ for any fully left-associated product (associativity makes bracketing irrelevant). Fix any partition of the n inputs into two blocks: the first k inputs and the remaining $n-k$ inputs. For arbitrary $u = (x_1, \ldots, x_k)$ and $v = (x_{k+1}, \ldots, x_n)$ define

$$U := x_1 \star \cdots \star x_k, \qquad V := x_{k+1} \star \cdots \star x_n.$$

By definition of F via $\star$,

$$F(x_1, \ldots, x_n) = U \star V.$$

But U and V themselves are values in X obtained by iterated $\star$. Thus F is compatible with the binary combine $\star$ in the sense that combining the block-results by $\star$ reproduces the full F. Hence F necessarily satisfies all identities that follow from iterated application of an associative binary operation on X. In particular, F is completely determined by the induced monoid structure $(\langle X \rangle_\star, \star)$.

Therefore, if F fails any of these associative compatibilities (for example, if there are $a, b, c \in X$ with $F(a, b, c) \neq (a \star b) \star c$ for every possible $\star$), then no such $\star$ can exist.

(b) Irreducible F forces auxiliary summaries, rounds, or approximation. Assume now that F is genuinely n-ary: there is no associative $\star$ on X that represents F by iterated binary combines. Suppose, for contradiction, that a distributed algorithm correctly computes F on all inputs by a single binary-tree reduction that only performs pairwise associative merges on values in the original domain X (i.e. nodes hold plain X-values, merges compute some binary associative $\star$ on X, and the root outputs an X-value). But then, by the argument in (a), the root output must be an iterated $\star$-product of the leaves,

hence equal to a function determined by $\star$. This contradicts the assumption that F is not representable by any such $\star$. Therefore no such plain X-valued binary-tree reduction can compute F on all inputs.

To compute F in a distributed setting one must therefore relax one of the restrictions:

I. *Extend the domain.* Introduce a larger associative domain $(S, \oplus)$ and an encoding $e : X \to S$ together with a decoder $\varphi : S \to X$ so that the associative merge of encodings of the inputs can be decoded back to F. Concretely, one requires

$$F(x_1, \ldots, x_n) \;=\; \varphi\Big(\bigoplus_{i=1}^{n} e(x_i)\Big).$$

Such an S must be able to carry whatever global information about the multiset $\{x_i\}$ that F depends on (for example, counts, order statistics, determinants, etc.). In general the minimal cardinality of S (or the minimal bit-length of encodings $e(x)$) is at least the number of equivalence classes of the relation $\sim$ on X^* defined by

$$u \sim v \iff \forall w \in X^* :\; F(u, w) = F(v, w),$$

since the merged summaries must not lose distinctions relevant to F.

II. *Use multi-round / centralized computation.* If one does not allow richer summaries, a distributed protocol can gather enough information by additional rounds of communication or by routing data to a central node, at the cost of latency or load imbalance. This avoids a single associative tree of plain X-merges but incurs extra communication/round complexity.

III. *Approximate F.* Permit a function $\widetilde{F}$ computable by associative merges on X-values that approximates F within some error tolerance. This trades exactness for efficiency.

The preceding argument is information-theoretic in spirit: if F depends on global properties of all n inputs that cannot be deduced from any strictly local (pairwise) associative combination of plain X-values, then any correct distributed computation must carry that global information explicitly (via summaries or extra communication). Hence the stated alternatives are necessary.

This completes the proof.

Remark 14. *The theorem formalizes the intuitive distributed-systems observation: associative binary merges on the same value domain enable cheap, one-pass tree reductions; genuinely n-ary operations that require access to global, nonpairwise information force one to pay in either summary size, rounds, or exactness. Classic examples include* Maj_n *(majority), medians/quantiles, and determinants.*

3.3. Consequences in Distributed Systems

i. **Parallelism:** Binary associative combines allow balanced tree computation in $\lceil \log_2 n \rceil$ rounds. Irreducible n-ary operations may require centralized aggregation or multi-round distributed protocols.

ii. **Communication:** Exact computation may require $\Omega(n)$ communication. Simulation typically requires transmitting auxiliary information (counts, sketches, partial summaries).

iii. **Tradeoffs:** One-shot n-ary aggregation minimizes rounds but centralizes load; summary-based merges enable parallelism but increase message sizes.

3.4. Implementation Strategies

(i) **Centralized Aggregator:** Gather all inputs at one node.

(ii) **Extended Domain Summaries:** Define $(S, \oplus)$ with associative merges, and a decoding $\phi : S \to X$ so that

$$\phi\left(\bigoplus_i s_i\right) \approx F(x_1, \ldots, x_n).$$

Example: quantile sketches, count vectors.

(iii) **Multi-Round Protocols:** Use distributed selection (e.g., median-of-medians).

(iv) **Approximation:** Accept approximate F for efficiency.

3.5. Implementation strategies: formal models and guarantees

We formalize the four implementation strategies sketched above and state precise correctness/robustness assertions. Let the raw domain be $\mathcal{X}$ (data items), and let $F : \mathcal{X}^n \to Y$ be the target aggregate we wish to compute (e.g. sum, median, quantile, majority).

Definition 12 (Centralized aggregator)**.** *A* centralized aggregator *computes $F(x_1, \ldots, x_n)$ by transmitting all x_i to a single node and evaluating F there. This is correct by definition but incurs communication cost $\Theta(\sum_i |x_i|)$ and has single-point-of-failure risk.*

Definition 13 (Associative summary model)**.** *An* associative summary model *for F is a triple $(S, \oplus, \phi)$ where $(S, \oplus)$ is a (finite or infinite) semigroup (typically a monoid with unit 0_S), and $\phi : S \to Y$ is a decoding map such that for each block partition $D_1, \ldots, D_k$ of the dataset,*

$$\phi\Big(\bigoplus_{j=1}^{k} s(D_j)\Big) \approx_{\mathcal{E}} F(x_1, \ldots, x_n),$$

where $s(D_j) \in S$ is a per-block summary computed from the items in D_j, and "$\approx_{\mathcal{E}}$" is a correctness relation that may be exact equality or an error guarantee drawn from an error model $\mathcal{E}$.

Theorem 9 (Exact summarization via monoid homomorphism)**.** *If there exists a monoid $(S, \oplus)$ and a map $s : \mathcal{X} \to S$ such that the induced map $\tilde{s} : \mathcal{X}^n \to S$ defined by $\tilde{s}(x_1, \ldots, x_n) = s(x_1) \oplus \cdots \oplus s(x_n)$ satisfies $F = \phi \circ \tilde{s}$ for some decoding $\phi : S \to Y$, then F is exactly computable in one parallel reduction using $\oplus$. Moreover the computation is independent of partitioning and reduction order.*

Proof. Immediate: because $\oplus$ is associative, any parenthesisation or partitioning of the $s(x_i)$ yields the same $\tilde{s}$, and therefore $\phi(\tilde{s}) = F(x_1, \ldots, x_n)$.

Note. The following version of Theorem 6.9 restates the result with explicit independence conditions (associativity and commutativity) made explicit. It is equivalent to the preceding statement, but formulated to highlight the independence of partitioning and ordering in the computation.

Theorem 10 (Second Exact summarization via monoid homomorphism)**.** *Let $(S, \oplus)$ be a monoid with identity element e. Let X and Y be sets, and let $s : X \to S$ and $\varphi : S \to Y$ be maps. Define the* fold *(induced map)*

$$\widetilde{s}(x_1, \ldots, x_n) := s(x_1) \oplus s(x_2) \oplus \cdots \oplus s(x_n),$$

where the n-fold $\oplus$-product is evaluated using any parenthesisation (with e used for the empty case). If

$$F(x_1, \ldots, x_n) = \varphi\big(\widetilde{s}(x_1, \ldots, x_n)\big)$$

for all $(x_1, \ldots, x_n) \in X^n$, then F can be computed exactly by a single parallel reduction using $\oplus$.

Moreover, the result of the computation is independent of how the intermediate values are grouped or combined (i.e. independent of partitioning or reduction order). If $\oplus$ is commutative, the result is also independent of the order (permutation) of the inputs.

Proof. Since $(S, \oplus)$ is a monoid, $\oplus$ is associative and has an identity element e. Hence for any sequence $(s_1, \ldots, s_n) \in S^n$, the n-fold product

$$s_1 \oplus s_2 \oplus \cdots \oplus s_n$$

is unambiguous with respect to parenthesisation. That is, for any binary-tree arrangement (parallel grouping) of the same terms, associativity ensures the same result.

Define recursively

$$\mathrm{fold}_\oplus(s_1) = s_1, \qquad \mathrm{fold}_\oplus(s_1, \ldots, s_{k+1}) = \mathrm{fold}_\oplus(s_1, \ldots, s_k) \oplus s_{k+1}.$$

Associativity implies that $\mathrm{fold}_\oplus$ is invariant under parenthesisation, so any parallel reduction tree that combines the $s(x_i)$'s with $\oplus$ yields exactly $\widetilde{s}(x_1, \ldots, x_n)$.

Applying φ then gives

$$\varphi\big(\widetilde{s}(x_1, \ldots, x_n)\big) = F(x_1, \ldots, x_n),$$

establishing that the parallel reduction computes F exactly.

Finally, independence of grouping follows from associativity, while independence of permutation requires the stronger condition that $\oplus$ be commutative. Hence, if $(S, \oplus)$ is a commutative monoid, the result of the reduction is independent of both partitioning and ordering.

Remark 15. *This theorem formalizes the correctness of one-round map–reduce style summarization: as long as the intermediate combination operation $\oplus$ forms a monoid, all parallel aggregation paths produce the same exact result.*

Example 11 (Majority via counts)**.** *For Boolean inputs $\mathcal{X} = \{0, 1\}$ and majority $F = m_n$, take $S = \mathbb{N}$ with $\oplus$ equal to integer addition, $s(x) = x$, and $\phi(k) = 1$ iff $k > \lfloor n/2 \rfloor$. Then $m_n = \phi \circ \tilde{s}$ and majority is computable by associative summaries (counts).*

Definition 14 (Approximate summaries and error bounds)**.** *An associative summary model $(S, \oplus, \phi)$ is* (δ, α)-accurate *with respect to a metric d_Y on Y if for every dataset $\mathbf{x} = (x_1, \ldots, x_n)$,*

$$d_Y\big(\phi(\tilde{s}(\mathbf{x})),\, F(\mathbf{x})\big) \le \delta(n) \quad \textit{with probability at least } 1 - \alpha,$$

or deterministically if $\alpha = 0$. (Here δ may be a function of n and of internal sketch parameters.)

Theorem 11 (Robustness transfer to (C, γ)-robustness)**.** *Let $(S, \oplus, \varphi)$ be an aggregator (notation as in the main text) and let $\mathcal{E}$ denote the space of implementation error vectors for the protocol. Assume the following.*

*(a) (*Error bookkeeping*) There is a nonnegative map $\Pi : \mathcal{E} \to \mathbb{R}_{\ge 0}$ (the bookkeeping or magnitude map) and a function $B : \mathbb{N} \to \mathbb{R}_{\ge 0}$ (the worst-case error growth) such that for every error $\tilde{e}$ that can occur under the implemented protocol on n participants we have*

$$\Pi(\tilde{e}) \le B(n).$$

*(b) (*Hölder/Lipschitz continuity*) The decoding map $\varphi : S \times \mathcal{E} \to Y$ (seen as a perturbation in the error coordinate) satisfies a Hölder-type bound: there exist constants $L > 0$ and $\gamma > 0$ such that for all $s \in S$ and all $e \in \mathcal{E}$,*

$$d_Y\big(\varphi(s, e), \varphi(s, 0)\big) \le L\,[\Pi(e)]^{\gamma}.$$

Then the distributed aggregator is (C, γ)-robust with

$$C \;=\; L\,[B(n)]^{\gamma},$$

in the following sense: for every admissible implementation error $\tilde{e}$ we have

$$d_Y\big(\varphi(s, \tilde{e}), \varphi(s, 0)\big) \le C.$$

In particular, if $\gamma = 1$ (Lipschitz case) then $C = L \cdot B(n)$.

Proof. Fix n and let $\tilde{e} \in \mathcal{E}$ be any error produced by the implemented protocol on n participants. By assumption (a) the bookkeeping map satisfies $\Pi(\tilde{e}) \leq B(n)$. By the Hölder/Lipschitz hypothesis (b) we have, for the same $s \in S$,

$$d_Y\big(\varphi(s,\tilde{e}),\varphi(s,0)\big) \leq L\,[\Pi(\tilde{e})]^{\gamma}.$$

Combining the two displayed inequalities yields

$$d_Y\big(\varphi(s,\tilde{e}),\varphi(s,0)\big) \leq L\,[B(n)]^{\gamma} = C,$$

which is precisely the claimed (C,γ)-robustness bound.

Remark 16 (Comments and variants)**.**

- ***Deterministic versus probabilistic bounds.*** *The statement above assumes the bound $\Pi(\tilde{e}) \leq B(n)$ holds deterministically for every execution of the protocol. If instead $\Pi(\tilde{e}) \leq B(n)$ only holds with high probability (say $1-\beta$), then the same inequality for d_Y holds with probability at least $1-\beta$. Similarly, one may replace the deterministic bound $B(n)$ by an expectation or a tail bound and obtain corresponding expected or high-probability robustness guarantees.*

- ***Role of the exponent γ.*** *The assumption (b) is a Hölder condition. If $\gamma = 1$ we recover the usual Lipschitz case and the robustness constant simplifies to $C = L \cdot B(n)$. If $0 < \gamma < 1$ the dependence on the bookkeeping bound becomes sublinear in $B(n)$, i.e. $C = L[B(n)]^{\gamma}$.*

- ***Necessity of the continuity condition.*** *Without a uniform continuity/Hölder bound of the form (b), a small bookkeeping magnitude $\Pi(e)$ does not in general control the change $d_Y(\varphi(s,e),\varphi(s,0))$; hence (b) (or a suitable variant) is necessary to transfer an error-magnitude bound into a metric bound on the decoded outputs.*

- ***Dependence on s.*** *We asked for a uniform Hölder bound in s. If only a local bound $L(s)$ is available, then the conclusion becomes $d_Y \leq L(s)[B(n)]^{\gamma}$, i.e. the robustness constant depends on the particular s under consideration.*

- ***Tightness.*** *The inequality is tight in the sense that if both (a) and (b) hold as equalities for some $\tilde{e}$ then $d_Y(\varphi(s,\tilde{e}),\varphi(s,0)) = L[B(n)]^{\gamma}$.*

Corollary 1 (High-probability (probabilistic) robustness)**.** *Under the hypotheses of Theorem 11, suppose instead that the bookkeeping bound holds only with high probability, i.e. there exists a function $B(n)$ and a failure probability $\beta \in (0,1)$ such that for the implemented protocol on n participants*

$$\Pr\big(\Pi(\tilde{e}) \leq B(n)\big) \;\geq\; 1-\beta.$$

Assume the Hölder/Lipschitz condition

$$d_Y\big(\varphi(s,e),\varphi(s,0)\big) \leq L\,[\Pi(e)]^{\gamma}$$

from Theorem 11 holds uniformly for all admissible s and e. Then the distributed aggregator is (C, γ)-robust with probability at least $1 - \beta$, where

$$C \;=\; L\,[B(n)]^{\gamma}.$$

Equivalently, with probability at least $1 - \beta$ over the protocol execution,

$$d_Y\big(\varphi(s, \tilde{e}), \varphi(s, 0)\big) \le C.$$

Proof. Let $\tilde{e}$ denote the (random) implementation error vector produced by the protocol. By the assumed high-probability bookkeeping bound, $\Pr\big(\Pi(\tilde{e}) \le B(n)\big) \ge 1 - \beta$. On the event $\{\Pi(\tilde{e}) \le B(n)\}$ we apply the Hölder/Lipschitz inequality to obtain

$$d_Y\big(\varphi(s, \tilde{e}), \varphi(s, 0)\big) \le L\,[\Pi(\tilde{e})]^{\gamma} \le L\,[B(n)]^{\gamma} = C.$$

Therefore the stated robustness bound holds with probability at least $1 - \beta$.

Remark 17. *If one needs a uniform high-probability guarantee simultaneously for several different protocol instances or for a family of events, standard concentration/union-bound techniques apply. For example, to ensure the bound holds for m (possibly dependent) events each with failure probability at most β', a union bound yields a simultaneous guarantee with failure probability at most $m\beta'$, so choose $\beta' = \beta/m$ to obtain an overall failure probability β.*

Definition 15 (Multi-round protocols)**.** *A* multi-round protocol *is an algorithm that alternates between (local) associative summarization and (global or partial) selection steps. Formally it is a composition of maps*

$$\mathcal{X}^n \xrightarrow{s_1} S_1^{k_1} \xrightarrow{T_1} \cdots \xrightarrow{s_r} S_r^{k_r} \xrightarrow{T_r} Y,$$

where each s_i produces associative per-block summaries and T_i are selection/merge transformations (possibly non-associative) that reduce the candidate set. Examples include "median-of-medians".

Definition 16 (Approximation strategy)**.** *An* approximation *strategy returns $\tilde{y}$ such that $d_Y(\tilde{y}, F(\mathbf{x})) \le \varepsilon$ with high probability and at lower cost (communication or time) than exact methods. Error bounds of sketches and randomized summaries fall in this class.*

Theorem 12 (Communication Lower Bound for Irreducible Aggregators)**.** *Let $F : X^n \to Y$ be an n-ary function that depends on all its arguments and cannot be expressed as*

$$F(x_1, \ldots, x_n) = \varphi(s(x_1) \oplus \cdots \oplus s(x_n))$$

for any associative binary operator $\oplus$ and summary map $s : X \to S$. Then, in any distributed model where each participant holds one input x_i and communication is by message passing, computing F exactly requires $\Omega(n)$ bits of total communication in the worst case.

Proof. [Sketch] In the communication complexity model, each input x_i must influence the final output for correctness. Since F is irreducible, no smaller associative summaries exist that can combine partial information without loss. Thus, each participant must transmit at least one bit of information about its input to the coordinator. By standard lower-bound arguments for symmetric and non-decomposable functions (see [18, 19]), the total communication cost is $\Omega(n)$.

3.6. Summary

Irreducibility implies no pure binary-tree parallelization on the same domain is possible. One must either centralize, extend the domain with summaries, run multi-round protocols, or approximate. For distributed frameworks, this justifies studying n-ary algebra *natively*, rather than forcing binary encodings.

4. Conclusion and Outlook

This second part has extended the algebraic framework of distributed computation developed in Part I by pursuing three complementary directions: categorical refinements, robustness principles, and practical strategies. On the categorical side, we clarified how n-ary operations embed into binary structures and when such reductions are algebraically exact but operationally inefficient. On the robustness side, we established stability transfer theorems under Lipschitz homomorphisms, showing how error control persists through natural morphisms of metric semigroups. On the practical side, we analyzed implementation models for genuinely n-ary operators, ranging from centralized aggregation to approximation protocols, and demonstrated their algebraic limits through case studies such as polynomial aggregation, majority, and determinants.

Together with Part I, these results provide a consolidated algebraic view of distributed computation: Part I furnished the definitions and foundational error models, while Part II advanced categorical structure, robustness, and concrete implementation strategies. The central theme that emerges is the tension between algebraic reducibility and distributed efficiency: while n-ary laws often embed algebraically into binary ones, the distributed cost model (synchronization, communication, memory, and fault tolerance) makes it advantageous to treat higher-arity operators as first-class computational primitives.

Outlook. The natural continuation, to be pursued in Part III, is to apply these algebraic insights to optimization problems within distributed frameworks such as Spark and MapReduce, and to develop new semigroup-theoretic questions motivated by parallel data processing. These include the study of algebraic cost models, communication lower bounds for irreducible n-ary functions, and the design of approximate semigroup structures tailored for large-scale probabilistic algorithms. In this way, the sequence of papers aims to establish a coherent semigroup-theoretic foundation for the analysis, design, and optimization of distributed computational systems.

Funding

The authors extend their appreciation to Prince Sattam bin Abdulaziz University, Saudi Arabia for funding this research work through the project number (PSAU/2025/01/35396).

Acknowledgements

The authors are thankful to the learned editor and reviewers for their valuable suggestions and comments which helped in bringing this paper to its present form.

References

[1] Marshal I. Sampson and Reny George. Semigroups in distributed computation: Foundation and models. *European Journal of Pure and Applied Mathematics*, 2025. Accepted, Nov. 2025.

[2] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Michael McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 15–28. USENIX Association, 2012.

[3] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[4] Matei Zaharia et al. Apache spark: Cluster computing with working sets. In *USENIX HotCloud*, 2010.

[5] A. H. Clifford and G. B. Preston. *The Algebraic Theory of Semigroups*, volume I–II. American Mathematical Society, 1961.

[6] J. M. Howie. *Fundamentals of Semigroup Theory*. Clarendon Press, Oxford, 1995.

[7] Serge Lang. *Algebra*. Graduate Texts in Mathematics. Springer, 2002.

[8] Jonathan S. Golan. *Semirings and Their Applications*. Springer, 1999.

[9] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2 edition, 2002.

[10] James H. Wilkinson. *Rounding Errors in Algebraic Processes*. Prentice Hall, 1963.

[11] Klaus-Jochen Engel and Rainer Nagel. *One-Parameter Semigroups for Linear Evolution Equations*. Springer, 2000.

[12] Gert K. Pedersen. *C^*-Algebras and Their Automorphism Groups*. Academic Press, 1979.

[13] V. Filippov. n-lie algebras. *Siberian Mathematical Journal*, 1985.

[14] Yoichiro Nambu. Generalized hamiltonian dynamics. *Physical Review D*, 1973.

[15] Leon Takhtajan. Foundations of the generalized nambu mechanics. *Communications in Mathematical Physics*, 1994.

[16] E. Offiong Akak, Marshal I. Sampson, Zsolt Lipcsey, and Rafiat B. Abubakar. Algorithm for semigroup bases and its implications on the basis of cyber groups. *IJMSO*, 2025. To appear.

[17] Serge Lang. *Algebra*, volume 211 of *Graduate Texts in Mathematics*. Springer, 3 edition, 2002.

[18] Andrew C.-C. Yao. Some complexity questions related to distributed computing. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–213, 1979.

[19] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, Cambridge, 1997.