EUROPEAN JOURNAL OF PURE AND APPLIED MATHEMATICS
2026, Vol. 19, Issue 1, Article Number 7093
ISSN 1307-5543 – ejpam.com
Published by New York Business Global

Semigroups in Distributed Computations: Algebraic Foundations and Models

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

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

Abstract. This work develops algebraic foundations connecting semigroup theory with large-scale distributed computation. Classical constructions are revisited and extended by introducing metric and perturbed semigroups suited to modeling numerical processes. We present semigroup-based models for distributed aggregation, emphasizing Spark primitives and the limitations of binary reduction for inherently n-ary operations. Error propagation is treated through the framework of error semigroups, leading to robustness criteria that quantify resilience under perturbations. Case studies including Word Count, PageRank, and distributed matrix multiplication illustrate how algebraic structure governs both efficiency and reliability in computation.

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

Key Words and Phrases: Semigroup, metric semigroup, perturbed operation, error semigroup, distributed aggregation, robustness, Spark, large-scale computation.

1. Introduction

Semigroups—sets equipped with an associative binary operation—are among the most fundamental algebraic constructs. By retaining only the law of associativity while omitting identity and inverses, they generalize groups and monoids. Their simplicity conceals wide applicability: semigroup operations arise whenever data is combined in a manner independent of bracketing. Algorithmic perspectives on semigroup bases have been developed to systematically generate and prune semigroup elements [1, 2].

In computation, associativity is the algebraic principle that enables parallelism at scale. Distributed frameworks such as Apache Spark and MapReduce rely on associative `reduce`/`aggregate` primitives: partitioned data can be combined locally, merged in parallel, and consolidated globally without ambiguity. This algebraic guarantee underpins tree-based reductions and supports fault tolerance in large-scale analytics.

*Corresponding author.
DOI: https://doi.org/10.29020/nybg.ejpam.v19i1.7093

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

When applied to numerical and data-driven systems, however, semigroup operations raise new challenges. Approximate associativity is a central concern, as floating-point addition and similar numerical operations are only approximately associative, leading to questions of error propagation and stability in distributed reductions. Approximate laws of associativity also appear in operator algebras [3], and connections with semigroups of operators in functional analysis are treated in Engel and Nagel [4]. Another challenge involves minimal generators, since discretized operator semigroups often contain redundant elements, and algorithms for pruning to minimal generating sets make these collections tractable in both algebraic theory and computational practice [1]. A further difficulty concerns error and resilience, as distributed computations must tolerate roundoff, faults, and skew. Incorporating metric and probabilistic error models into semigroups provides a principled way to quantify robustness.

This paper constitutes Part I of the study. Here, we develop a framework for *approximate semigroups* modeling numerical error in parallel aggregation, algorithms for pruning and computing minimal generating sets (see [1, 2]) in discretized semigroups with illustrations in operator-semigroup examples, and a case-based analysis of error accumulation and robustness, culminating in three representative scenarios that examine distributed aggregation under perturbations.

Novelty. These developments differ from prior work in approximate algebra or semiring-based computation in a crucial way: numerical and system-induced errors are embedded *directly in the algebraic law*, rather than treated externally as noise or post-hoc correction [5–7]. This creates a new algebraic–computational correspondence in which the reliability and efficiency of parallel reductions follow from associativity properties of the underlying semigroup [8, 9], and performance, robustness, and correctness can be reasoned about symbolically while remaining empirically testable on distributed frameworks such as MapReduce [10] and Spark [11, 12]. Formal stability analysis of approximate operations is grounded in classical numerical analysis [6, 7], ensuring consistency of associative approximations in large-scale analytics.

This integrated viewpoint enables both theoretical analysis and practical guidance for the design of scalable, error-aware aggregation primitives. Error-bounded aggregation schemes provide quantitative guarantees on tolerance levels during parallel execution and can be implemented efficiently using semigroup-based pruning algorithms [1, 2].

Section 2 reviews algebraic preliminaries and metric extensions. Sections 3 to 6 presents the main results in this work. In Section 3 we introduce approximate semigroups and perturbed operations. Section 4 develops pruning algorithms and minimal generators. Section 5 formulates error-semigroup models and analyzes robustness, organized into three cases. Section 6 concludes with outlook and future directions, leading into Part II of this study.

2. Preliminaries

2.1. Basic algebraic notions

Basic analytic preliminaries follow standard references such as Lang [13]. Extensions toward semirings and related algebraic systems may be found in Golan [5].

Definition 1 (Semigroup)**.** *A* semigroup *is a pair* $(S, \cdot)$ *where* S *is a nonempty set and* $\cdot : S \times S \to S$ *is an associative binary operation:*

$$(a \cdot b) \cdot c = a \cdot (b \cdot c) \quad \textit{for all } a, b, c \in S.$$

Example 1. $(\mathbb{N}, +)$*,* $(\mathbb{Z}, \max)$*, and* $(\mathbb{Z}, \min)$ *are semigroups.*

Definition 2 (Monoid)**.** *A* monoid *is a semigroup* $(M, \cdot)$ *that contains a neutral element* $e \in M$ *satisfying* $e \cdot a = a \cdot e = a$ *for all* $a \in M$*.*

Definition 3 (Homomorphism, automorphism)**.** *A map* $\varphi : (S, \cdot) \to (T, *)$ *is a* semigroup homomorphism *if* $\varphi(a \cdot b) = \varphi(a) * \varphi(b)$ *for all* $a, b \in S$*. An* automorphism *is a bijective homomorphism from* S *to itself. The set of automorphisms of* S *forms a group under composition; we denote it by* $Aut(S)$*.*

Remark 1. *When* S *carries additional structure (topological, metric, Banach-space structure), we typically demand homomorphisms preserve that structure (continuous homomorphisms, bounded linear homomorphisms, etc.).*

2.2. Classical and Operator-theoretic Examples

Example 2. *(i)* $(\mathbb{N}, +)$ *is a commutative monoid (identity* 0*).*

(ii) The set of $n \times n$ *real matrices* $M_n(\mathbb{R})$ *with multiplication is a (noncommutative) semigroup; adding the identity gives a monoid.*

(iii) For a Banach space X*,* $\mathcal{L}(X)$ *denotes the bounded linear operators on* X*; composition makes* $\mathcal{L}(X)$ *a monoid with identity* I*.*

2.3. Metric semigroups and approximately associative algebra

To model numerical rounding and inexact computation, we introduce metrics on semigroups and quantify associativity defects.

Definition 4 (Metric semigroup)**.** *A* metric semigroup *is a triple* $(S, \cdot, d)$ *such that:*

(i) $(S, \cdot)$ *is a semigroup;*

(ii) (S, d) *is a metric space;*

(iii) the multiplication map

$$m : S \times S \to S, \quad (x, y) \mapsto x \cdot y,$$

is jointly continuous with respect to the product metric on $S \times S$.

Example 3 (Real numbers under addition)**.** *Let* $S = (\mathbb{R}, +)$*, and let* $d(x, y) = |x - y|$ *be the usual Euclidean metric. Then* $(\mathbb{R}, +)$ *is a semigroup,* $(\mathbb{R}, d)$ *is a metric space, and the map*

$$m(x, y) = x + y$$

is jointly continuous. Thus $(\mathbb{R}, +, d)$ *is a metric semigroup.*

Example 4 (Continuous functions under pointwise multiplication)**.** *Let* $S = C([0, 1], \mathbb{R})$ *be the set of continuous real-valued functions on* $[0, 1]$*, with pointwise multiplication*

$$(f \cdot g)(t) = f(t)g(t).$$

Equip S *with the metric*

$$d(f, g) = \sup_{t \in [0,1]} |f(t) - g(t)|.$$

Then $(S, \cdot)$ *is a semigroup,* (S, d) *is a metric space, and multiplication is jointly continuous in the uniform metric. Hence* $(S, \cdot, d)$ *is a metric semigroup.*

Definition 5 (Associator defect)**.** *Given a metric semigroup* $(S, \cdot, d)$*, define the* associator defect *function*

$$\alpha : S \times S \times S \to [0, \infty), \qquad \alpha(a, b, c) := d\big((a \cdot b) \cdot c,\ a \cdot (b \cdot c)\big).$$

Definition 6 (ε-semigroup)**.** *Let* $(S, \cdot, d)$ *be a metric semigroup. For* $\varepsilon \geq 0$*, we call* S *an* ε-semigroup *if* $\sup_{a,b,c \in S} \alpha(a, b, c) \leq \varepsilon$*. If* $\varepsilon = 0$ *we recover an ordinary semigroup.*

Example 5 (Truncated addition on [0,1)**.** *] Let* $S = [0, 1]$ *with the usual metric and define the operation*

$$x \cdot y := \min\{x + y,\ 1\}.$$

Associativity fails near the truncation boundary. Indeed, for $x = y = z = 0.6$*,*

$$(x \cdot y) \cdot z = 1 \cdot 0.6 = 1, \qquad x \cdot (y \cdot z) = 0.6 \cdot 1 = 1,$$

so the error is 0 *in this case. But taking* $x = 0.8$*,* $y = 0.5$*,* $z = 0.4$*,*

$$(x \cdot y) \cdot z = 1, \qquad x \cdot (y \cdot z) = 0.8 \cdot 0.9 = 1,$$

still gives no error. To obtain a genuine deviation, note that the maximum associativity defect is

$$\alpha(x, y, z) = d\big((x \cdot y) \cdot z,\ x \cdot (y \cdot z)\big) \leq 1 - (x + y + z - 1)_+,$$

and in fact one checks that

$$\sup_{x,y,z \in [0,1]} \alpha(x, y, z) = 1.$$

Thus $([0, 1], \cdot, d)$ *is a* 1*-semigroup (but not an* ε*-semigroup for any* $\varepsilon < 1$*).*

Remark 2. *In finite-precision arithmetic, associator defects arise from rounding. Modeling floating-point addition as an ε-semigroup with ε proportional to machine epsilon is useful for error propagation analysis.*

Definition 7 (Lipschitz Homomorphism)**.** *Let $(S, \cdot, d_S)$ and $(T, *, d_T)$ be semigroups (or monoids, groups) equipped with metrics. A map $\varphi : S \to T$ is called a* Lipschitz homomorphism *if*

$$\varphi(x \cdot y) = \varphi(x) * \varphi(y) \quad \text{for all } x, y \in S,$$

and there exists a constant $L \geq 0$ such that

$$d_T(\varphi(x), \varphi(y)) \leq L\, d_S(x, y), \qquad \text{for all } x, y \in S.$$

The smallest such L is called the Lipschitz constant *of φ.*

Remark 3. *A Lipschitz homomorphism simultaneously respects the algebraic structure and controls distortion of distances. This property is useful in geometric group theory, functional analysis, and distributed computation, as it ensures algebraic operations remain numerically stable under metric approximations.*

Example 6. *Consider $(\mathbb{Z}, +)$ with the usual metric $d(m, n) = |m - n|$ and $(\mathbb{R}, +)$ with the Euclidean metric. The inclusion map*

$$\iota : \mathbb{Z} \hookrightarrow \mathbb{R}, \qquad \iota(n) = n,$$

is a homomorphism of additive groups, and satisfies

$$|\iota(m) - \iota(n)| = |m - n| = d_{\mathbb{Z}}(m, n).$$

Hence ι is a Lipschitz homomorphism with constant $L = 1$.

Theorem 1. *Let (S, d_S) and (T, d_T) be metric semigroups, and let $\varphi : S \to T$ be a semigroup homomorphism. If φ is Lipschitz, i.e., there exists $L > 0$ such that*

$$d_T\big(\varphi(x), \varphi(y)\big) \leq L\, d_S(x, y) \quad \text{for all } x, y \in S,$$

then φ is uniformly continuous.

Proof. Recall that φ is uniformly continuous if for every $\varepsilon > 0$ there exists $\delta > 0$ such that

$$d_S(x, y) < \delta \implies d_T(\varphi(x), \varphi(y)) < \varepsilon \quad \text{for all } x, y \in S.$$

Let $\varepsilon > 0$ be given. Define

$$\delta := \frac{\varepsilon}{L}.$$

Now, for any $x, y \in S$ with $d_S(x, y) < \delta$, the Lipschitz property of φ gives

$$d_T(\varphi(x), \varphi(y)) \leq L\, d_S(x, y) < L\delta = L \cdot \frac{\varepsilon}{L} = \varepsilon.$$

Since $\varepsilon > 0$ was arbitrary, φ is uniformly continuous.

Remark 4. *This result, while elementary, has important implications for algebraic modeling in distributed computation frameworks such as Apache Spark. In such settings, semigroup homomorphisms model data transformations that must respect associativity. The Lipschitz condition ensures that the map does not amplify numerical or partitioning errors disproportionately. Uniform continuity then guarantees stability across the entire semigroup, independent of scale, making this analytic property particularly suitable for error analysis in large-scale, parallel computations.*

Theorem 2 (Stability under Lipschitz homomorphisms)**.** *Let $(S, \cdot, d_S)$ be an ε-semigroup and $f : S \to T$ a semigroup homomorphism into metric semigroup $(T, *, d_T)$ which is L-Lipschitz ($d_T(f(x), f(y)) \leq L d_S(x, y)$). Then $f(S)$ is an $(L\varepsilon)$-semigroup in T.*

Proof. For $a, b, c \in S$,

$$d_T\big((f(a)*f(b))*f(c), f(a)*(f(b)*f(c))\big) = d_T\big(f((a\cdot b)\cdot c), f(a\cdot(b\cdot c))\big) \leq L\,\alpha(a,b,c) \leq L\varepsilon.$$

2.4. Derivations and automorphism groups (Lie-theoretic viewpoint)

When A is an algebra, derivations generate one-parameter automorphism groups (in well-behaved settings - Derivations are to automorphism groups what infinitesimal generators are to semigroups of operators.). This observation provides a unifying viewpoint for operator-theoretic examples later.

Definition 8 (Derivation)**.** *Let $\mathcal{A}$ be an (associative) algebra over $\mathbb{R}$ or $\mathbb{C}$. A linear map $D : \mathcal{A} \to \mathcal{A}$ is a* derivation *if*

$$D(xy) = D(x)y + xD(y)$$

for all $x, y \in \mathcal{A}$.

Proposition 1. *If D is a bounded derivation on a Banach algebra $\mathcal{A}$, then the series*

$$e^{tD} := \sum_{k=0}^{\infty} \frac{t^k}{k!} D^k$$

converges in operator norm for all $t \in \mathbb{R}$ and each e^{tD} is an algebra automorphism of $\mathcal{A}$. Moreover $t \mapsto e^{tD}$ is a one-parameter automorphism group.

Proof. Boundedness of D implies the exponential series converges in operator norm. The derivation property ensures e^{tD} is multiplicative (standard). Group property follows from properties of exponentials.

Now we present the detailed proof of every claim in the above proof:

Theorem 3. *Let A be a (complex) Banach algebra and let $D \in \mathcal{B}(A)$ be a bounded derivation, i.e. D is bounded linear and*

$$D(ab) = D(a)b + aD(b) \quad \text{for all } a, b \in A.$$

For $t \in \mathbb{R}$, define

$$\alpha_t \;=\; e^{tD} \;:=\; \sum_{n=0}^{\infty} \frac{t^n}{n!}\, D^n \;\in\; \mathcal{B}(A).$$

Then:

(i) The series defining α_t converges in operator norm, and $t \mapsto \alpha_t$ is a norm-continuous one-parameter group in $\mathcal{B}(A)$ with $\alpha_{s+t} = \alpha_s \alpha_t$, $\alpha_0 = \mathrm{id}_A$ and $\alpha_t^{-1} = \alpha_{-t}$.

(ii) α_t is an algebra automorphism: for all $a, b \in A$,

$$\alpha_t(ab) \;=\; \alpha_t(a)\, \alpha_t(b).$$

(iii) The map $t \mapsto \alpha_t$ is C^1 in operator norm and, for every $x \in A$,

$$\frac{d}{dt}\, \alpha_t(x) \;=\; D\big(\alpha_t(x)\big) \;=\; \alpha_t\big(Dx\big).$$

If A is unital, then $D(1) = 0$ and hence $\alpha_t(1) = 1$ for all t.

Proof. (1) Convergence and group property. Since D is bounded, $\|D^n\| \le \|D\|^n$ for all $n \ge 0$. Hence, for each $t \in \mathbb{R}$,

$$\sum_{n=0}^{\infty} \left\| \frac{t^n}{n!} D^n \right\| \;\le\; \sum_{n=0}^{\infty} \frac{|t|^n}{n!}\, \|D\|^n \;=\; e^{|t|\|D\|} < \infty,$$

so the exponential series converges absolutely in operator norm. Thus $\alpha_t \in \mathcal{B}(A)$ and $\|\alpha_t\| \le e^{|t|\|D\|}$.

The *group property* follows from the standard series computation (justified by absolute convergence of the double series in operator norm):

$$\alpha_s \alpha_t = \Big(\sum_{m=0}^{\infty} \frac{s^m}{m!} D^m\Big)\Big(\sum_{n=0}^{\infty} \frac{t^n}{n!} D^n\Big) = \sum_{k=0}^{\infty} \Big(\sum_{m=0}^{k} \frac{s^m t^{\,k-m}}{m!(k-m)!}\Big) D^k = \sum_{k=0}^{\infty} \frac{(s+t)^k}{k!} D^k = \alpha_{s+t}.$$

Taking $s = 0$ gives $\alpha_0 = \mathrm{id}_A$, and $\alpha_t^{-1} = \alpha_{-t}$ follows immediately.

(2) Multiplicativity. We use the higher Leibniz rule for powers of a derivation.

Lemma 1 (Higher Leibniz rule)**.** *For every $n \in \mathbb{N}$ and $a, b \in A$,*

$$D^n(ab) \;=\; \sum_{k=0}^{n} \binom{n}{k} D^k(a)\, D^{\,n-k}(b).$$

Proof. [Proof of the lemma] By induction on n. The case $n = 1$ is exactly the derivation property. Suppose the identity holds for some n. Then

$$\begin{aligned} D^{n+1}(ab) &= D\left(\sum_{k=0}^{n}\binom{n}{k}D^k(a)D^{n-k}(b)\right) = \sum_{k=0}^{n}\binom{n}{k}\Big(D^{k+1}(a)\,D^{n-k}(b) + D^k(a)\,D^{n-k+1}(b)\Big) \\ &= \sum_{k=1}^{n+1}\binom{n}{k-1}D^k(a)\,D^{n+1-k}(b) \;+\; \sum_{k=0}^{n}\binom{n}{k}D^k(a)\,D^{n+1-k}(b) \\ &= \binom{n}{n}D^{n+1}(a)D^0(b) + \sum_{k=1}^{n}\left(\binom{n}{k-1}+\binom{n}{k}\right)D^k(a)\,D^{n+1-k}(b) + \binom{n}{0}D^0(a)D^{n+1}(b) \\ &= \sum_{k=0}^{n+1}\binom{n+1}{k}\,D^k(a)\,D^{n+1-k}(b), \end{aligned}$$

using Pascal's identity $\binom{n}{k-1}+\binom{n}{k}=\binom{n+1}{k}$. This proves the claim for $n+1$.

Now compute, using the lemma and absolute convergence of the involved series in operator norm (which justifies rearranging terms and applying the Cauchy product):

$$\begin{aligned} \alpha_t(ab) &= \sum_{n=0}^{\infty}\frac{t^n}{n!}D^n(ab) = \sum_{n=0}^{\infty}\frac{t^n}{n!}\sum_{k=0}^{n}\binom{n}{k}D^k(a)D^{n-k}(b) \\ &= \sum_{n=0}^{\infty}\sum_{k=0}^{n}\frac{t^n}{k!(n-k)!}D^k(a)D^{n-k}(b) = \sum_{k=0}^{\infty}\sum_{\ell=0}^{\infty}\frac{t^{k+\ell}}{k!\,\ell!}D^k(a)D^{\ell}(b) \\ &= \Big(\sum_{k=0}^{\infty}\frac{t^k}{k!}D^k(a)\Big)\Big(\sum_{\ell=0}^{\infty}\frac{t^\ell}{\ell!}D^\ell(b)\Big) = \alpha_t(a)\,\alpha_t(b). \end{aligned}$$

Thus α_t is multiplicative. Since α_t is invertible (by (1)), it is an automorphism.

(3) Differentiability and generator. Fix $x \in A$. The series defining $t \mapsto \alpha_t(x)$ converges uniformly on compact intervals by the Weierstrass M-test (using $\|D^n x\| \le \|D\|^n\|x\|$), so termwise differentiation is valid. Therefore

$$\frac{d}{dt}\,\alpha_t(x) = \sum_{n=1}^{\infty}\frac{t^{n-1}}{(n-1)!}D^n x = D\left(\sum_{n=0}^{\infty}\frac{t^n}{n!}D^n x\right) = D\big(\alpha_t(x)\big).$$

Because D commutes with every power D^n, and hence with e^{tD}, we also have

$$\frac{d}{dt}\,\alpha_t(x) \;=\; \alpha_t(Dx).$$

If A is unital, the derivation identity on $1 = 1\cdot 1$ gives $D(1) = D(1)\cdot 1 + 1\cdot D(1) = 2D(1)$, hence $D(1) = 0$, and so

$$\alpha_t(1) \;=\; \sum_{n=0}^{\infty}\frac{t^n}{n!}D^n(1) \;=\; 1.$$

This completes the proof.

This completes the foundational material. Applications and models follow next.

3. Semigroup Models in Distributed Systems

3.1. Distributed computation and aggregation primitives

Early descriptions of Spark are given in Zaharia et al. [11], with the resilient distributed dataset abstraction introduced in [12]. Distributed frameworks (MapReduce, Spark) rely on associative operations to aggregate results computed on subsets. We formalize common primitives.

Definition 9 (MapReduce)**.** MapReduce *is a programming model for processing and generating large data sets with a parallel, distributed algorithm on a cluster. It consists of two main steps:*

(i) ***Map step:*** *A user-defined* `map` *function applies in parallel to input data, producing intermediate key–value pairs:*

$$\texttt{map} : \mathit{Input} \longrightarrow \{(k, v)\}.$$

(ii) ***Reduce step:*** *A user-defined* `reduce` *function aggregates all intermediate values associated with the same key:*

$$\texttt{reduce} : (k, \{v\}) \longrightarrow \mathit{Output}.$$

The runtime system handles data distribution, parallelization, fault tolerance, and load balancing automatically.

Example. *Counting word frequencies in a large corpus:*

- ***Map:*** *For each word occurrence, emit* $(\mathit{word}, 1)$*.*
- ***Reduce:*** *Sum the values for each key (word).*

3.1.1. Apache Spark

Apache Spark is an open-source distributed computing framework designed for large-scale data processing. Unlike MapReduce, which materializes intermediate results to disk after each step, Spark keeps data in memory as much as possible, which makes it significantly faster for iterative algorithms and interactive analytics.

Spark provides high-level APIs in Java, Scala, Python, and R. Its core abstraction is the *Resilient Distributed Dataset (RDD)*, an immutable distributed collection of objects that can be operated on in parallel. Spark generalizes MapReduce by supporting not only `map` and `reduce`, but also transformations such as `filter`, `flatMap`, `groupByKey`, `reduceByKey`, and actions like `collect` or `save`.

Example 7 (Word count in Spark)**.**

```
rdd = sc.textFile("file.txt")
counts = (rdd.flatMap(lambda line: line.split())
              .map(lambda word: (word, 1))
              .reduceByKey(lambda a,b: a+b))
counts.collect()
```

Remark 5. ***Summary:*** *MapReduce is a simple two-stage model (map → reduce), disk-based, very robust but slower. Spark generalizes and extends MapReduce with in-memory computation, rich Application Programming Interfaces (APIs), iterative processing, and is widely used in modern large-scale data analytics.*

Definition 10 (Aggregation monoid)**.** *An* aggregation monoid *is a monoid* $(M, *, e)$ *intended to model the local and global combination of partial results in a distributed system.*

Definition 11 (RDD aggregation abstraction)**.** *Let* $(M, *, e)$ *be a monoid and let* $\mathcal{P}$ *be a partition of dataset* D *into blocks* $D_1, \ldots, D_k$*. A function* $\texttt{agg} : \mathcal{P} \to M$ *is an* RDD aggregation routine *if*

$$\texttt{agg}(D_1 \cup \cdots \cup D_k) = \texttt{agg}(D_1) * \cdots * \texttt{agg}(D_k).$$

Remark 6. *This property ensures parallel reducibility: each block can be processed independently, and partial aggregates combined by* $*$*.*

3.2. Concrete examples from Spark

We summarize common patterns and show how they fit the monoid/semigroup picture.

Example 8 (Summation / averages)**.** *Summation* $(\mathbb{R}, +, 0)$ *and pairwise aggregation used for computing averages (sum, count) are monoids. For average, one uses the pair monoid* $(\mathbb{R} \times \mathbb{N}, \oplus, (0, 0))$ *with*

$$(s_1, n_1) \oplus (s_2, n_2) = (s_1 + s_2, n_1 + n_2).$$

Example 9 (Word Count)**.** *Word count maps documents to frequency maps in* $\mathbb{N}^{\mathcal{V}}$ *and uses pointwise addition. This is a commutative monoid.*

Consider a vocabulary $\mathcal{V} = \{w_1, w_2, \ldots, w_m\}$*. Each document* d *can be represented as a vector in* $\mathbb{N}^{\mathcal{V}}$*, where the coordinate corresponding to* w_i *is the number of times* w_i *occurs in* d*. Formally, define*

$$\phi : \textit{Documents} \to \mathbb{N}^{\mathcal{V}}, \qquad \phi(d)(w) = \textit{number of occurrences of } w \textit{ in } d.$$

Given two documents d_1, d_2*, their word count vectors satisfy*

$$\phi(d_1 \cup d_2) \;=\; \phi(d_1) + \phi(d_2),$$

where $+$ *is pointwise addition in* $\mathbb{N}^{\mathcal{V}}$*.*

Algebraic structure. *The set* $\mathbb{N}^{\mathcal{V}}$ *with pointwise addition is a* commutative monoid*, with identity element the zero vector (corresponding to the empty document). This captures the essence of the MapReduce "reduce" step: aggregating counts by addition.*

Illustration. *Suppose* $\mathcal{V} = \{\texttt{cat}, \texttt{dog}, \texttt{fish}\}$*. Two documents are given by*

$$d_1 = \text{"cat dog cat"}, \qquad d_2 = \text{"dog fish"}.$$

Then

$$\phi(d_1) = (2, 1, 0), \qquad \phi(d_2) = (0, 1, 1).$$

Aggregating,

$$\phi(d_1) + \phi(d_2) = (2, 1, 0) + (0, 1, 1) = (2, 2, 1).$$

Thus the combined corpus has frequency map

$$\{\texttt{cat} : 2,\ \texttt{dog} : 2,\ \texttt{fish} : 1\}.$$

This example makes explicit how word count is naturally modeled by the commutative monoid $(\mathbb{N}^{\mathcal{V}}, +)$.

Example 10 (Set union)**.** *Union of sets* $(\mathcal{P}(U), \cup, \varnothing)$ *models deduplication and joins with idempotence.*

Theorem 4 (Monoid structure of Spark aggregations)**.** *Let* $(M, *, e)$ *be a commutative monoid. Suppose a dataset* $D = \{x_1, \ldots, x_n\}$ *is partitioned into blocks* $D_1, \ldots, D_k$*. If each block is mapped to an element of* M *by a homomorphism* $f : D_i \to M$*, and the results are combined using* $*$*, then the final aggregate*

$$f(D) = f(D_1) * f(D_2) * \cdots * f(D_k)$$

is independent of the partitioning of D*.*

Proof. Because $(M, *, e)$ is a monoid, $*$ is associative and e is the neutral element. Thus, the evaluation of

$$f(D_1) * f(D_2) * \cdots * f(D_k)$$

is well-defined independent of the grouping of terms. Since the $f(D_i)$ cover disjoint parts of D, the final result is exactly $f(D)$, and this does not depend on how D was partitioned.

Remark 7. *This theorem explains why Spark* `reduce` *and* `aggregate` *operations require associativity (semigroup law) and often an identity (monoid law). Summation, word count, and set union all instantiate this principle, ensuring correctness across arbitrary parallel partitionings.*

Theorem 5 (Semigroup structure of Spark reductions)**.** *Let* $(S, *)$ *be a semigroup (associative binary operation). Suppose a dataset* $D = \{x_1, \ldots, x_n\}$ *is partitioned into blocks* $D_1, \ldots, D_k$*. If each block is mapped to an element of* S *by a homomorphism* $f : D_i \to S$*, and the results are combined using* $*$*, then the final reduction*

$$f(D) = f(D_1) * f(D_2) * \cdots * f(D_k)$$

is independent of the way the blocks are grouped, though not defined for the empty dataset.

Remark 8. *This theorem formalizes a standard principle from functional programming, namely that parallelizable reductions are precisely those arising from semigroup (or monoid) homomorphisms [14, 15]. In distributed systems such as MapReduce [10] and Spark [12], this condition is explicitly required: the "reduce" operator must be associative to ensure that the result is independent of partitioning or evaluation order.*

Proof. By associativity of $*$, the expression

$$f(D_1) * f(D_2) * \cdots * f(D_k)$$

is independent of parenthesization. Thus any order of combining block-results yields the same aggregate. The caveat is that a semigroup does not provide a neutral element, so the construction presupposes that D is nonempty. For empty datasets, a unit (monoid extension) is required.

Remark 9. *This explains Spark distinction between* `reduce` *(which works for nonempty datasets under a semigroup law) and* `aggregate` *or* `fold` *(which require a global identity element). Thus semigroups capture the algebraic essence of parallelizable reductions, while monoids extend the framework to handle empty inputs.*

Corollary 1 (Concrete Spark examples as semigroup/monoid reductions)**.** *The following common Spark aggregations are instances of the semigroup/monoid reduction principle of the previous theorem:*

(i) ***Summation / average (pair monoid).*** *The sum on $\mathbb{R}$ is a commutative monoid $(\mathbb{R}, +, 0)$. Averaging is implemented by the pair monoid $(\mathbb{R} \times \mathbb{N}, \oplus, (0, 0))$ with*

$$(s_1, n_1) \oplus (s_2, n_2) := (s_1 + s_2,\ n_1 + n_2).$$

The block map sends a block of values to $(sum, count)$, and global average is obtained from the combined pair.

(ii) ***Word count (commutative monoid).*** *Each document maps to a frequency vector in $\mathbb{N}^{\mathcal{V}}$. Pointwise addition gives the commutative monoid $(\mathbb{N}^{\mathcal{V}}, +, \mathbf{0})$, so partial frequency maps combine by the monoid law.*

(iii) ***Set union (idempotent monoid).*** *For a universe U, the power set $(\mathcal{P}(U), \cup, \varnothing)$ is an idempotent commutative monoid; partial sets union in any order.*

Proof. Each item fits the hypothesis of the semigroup reduction theorem (or its monoid variant) by specifying:

(i) *Summation / average:* The block map f sends a block B to $(\sum_{x \in B} x,\ |B|)$. The operation $\oplus$ is associative and has identity $(0, 0)$, so iterative combination of block results yields the global sum/count pair; the final average is the first coordinate divided by the second (when the count is nonzero).

(ii) *Word count:* The block map f sends a block to its frequency vector in $\mathbb{N}^{\mathcal{V}}$. Pointwise addition is associative and has the zero vector as identity; combining blocks by addition yields exact global frequencies.

(iii) *Set union:* The block map f sends each partition to the subset of seen items. Union is associative with identity $\varnothing$, so combining block results by $\cup$ gives the global set.

In each case the associativity (and in practice the existence of an identity) guarantees correctness of parallel reduction independent of grouping or pairing order.

Remark 10. *This corollary clarifies Spark API design:* `reduce` *requires only an associative binary operator (a semigroup), and therefore is defined for nonempty RDDs;* `aggregate` *and* `fold` *additionally require a neutral element (monoid) so they can handle empty partitions and provide fault-tolerant defaults. When implementers provide a monoid (identity + associative combiner), Spark can safely parallelize and reorder computations without affecting results.*

3.3. Semigroup viewpoint on joins and multi-way operations

Joins can often be represented as binary operations on tuples; however, multi-way joins sometimes have natural n-ary formulations that emphasize symmetry and constraints. We discuss genuine n-ary operations next.

Theorem 6 (Semigroup structure for multi-way joins)**.** *Let $R_1, \ldots, R_n$ be finite relations over a common attribute set U, with possible overlap. Define the n-ary operation*

$$J(R_1, \ldots, R_n) \;:=\; R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n,$$

where $\bowtie$ denotes the natural join. Then:

(i) J is well-defined and associative in the sense that

$$J(R_1, \ldots, R_n) \;=\; (((R_1 \bowtie R_2) \bowtie R_3) \cdots \bowtie R_n),$$

up to isomorphism of attributes.

(ii) The binary semigroup $(\mathcal{R}, \bowtie)$ embeds the n-ary operator J, but the embedding can incur exponential blowup in intermediate relation size.

Proof. (i) Follows from the associativity of natural join: the order of grouping does not affect the final result, only the schema alignment. (ii) The embedding is trivial syntactically, but cost models in distributed databases show that binary decomposition can introduce intermediate results larger than the final output (the "intermediate result explosion" phenomenon). Thus, algebraically reducible but computationally inefficient.

Example 11 (Three-way join as native ternary operation)**.** *Consider relations $R(A, B)$, $S(B, C)$, $T(C, A)$. The ternary join*

$$J(R, S, T) \;=\; \{(a, b, c) : (a, b) \in R,\, (b, c) \in S,\, (c, a) \in T\}$$

is symmetric in R, S, T. While representable as $(R \bowtie S) \bowtie T$, the binary sequence typically materializes large intermediate joins, whereas the native ternary view emphasizes cyclic constraints and avoids redundant computation.

Remark 11. *This formalism motivates studying* n-ary semigroups of relations*: sets of relations closed under native n-ary join operators. In distributed frameworks (e.g. Spark SQL), query planners attempt to approximate this by optimizing join orders, but the algebraic inefficiency remains unless the n-ary operator is treated as primitive.*

3.4. Genuine n-ary operations and examples

Many operations naturally take n inputs simultaneously and are not simply iterates of a binary operator. We collect examples relevant to computation and algebra.

Definition 12 (Genuine n-ary operation)**.** *An n-ary operation $F : X^n \to X$ is* genuine *(not reducible) if there does not exist a binary operation $\star : X \times X \to X$ and a bracketing/associative scheme such that $F(x_1, \ldots, x_n)$ equals an iterated combination of $\star$ for all $x_1, \ldots, x_n \in X$.*

Example 12 (Median)**.** *The median of three real numbers $m(x, y, z)$ is a ternary function that cannot be represented for all inputs by iterating a fixed binary operation in a consistent way that preserves median behavior.*

Example 13 (Majority)**.** *The Boolean majority* Maj_n *is inherently n-ary: it depends on the global count of inputs and is not expressible as iterated application of a fixed binary operator while preserving the majority semantics in all cases.*

Example 14 (Determinant)**.** *The determinant of n column vectors in $\mathbb{R}^n$ is multilinear and alternating; while one can compute it via binary Laplace expansions, its algebraic nature is genuinely n-ary and the direct multilinear form is the natural viewpoint.*

3.5. When n-ary operations appear in distributed systems

- Multi-way joins: combining several relations at once might be more efficient when done in a single coordinated step, and modeling this as an n-ary operator clarifies concurrency and locking semantics.
- Aggregators with global constraints: normalization (producing a probability vector) requires knowledge of the global sum of all inputs — naturally n-ary.
- Fault-tolerant majority voting: consensus functions (Byzantine-tolerant aggregation) use n-ary majority-like operations.

4. Automorphism groups and dynamics on operator algebras

We collect operator-theoretic automorphism examples which will be used in the subsequent sections and later extensions.

Definition 13 (One-parameter automorphism group)**.** *Let $\mathcal{A}$ be an algebra (Banach or C*-algebra). A family $\{\Theta_t\}_{t\in\mathbb{R}} \subset Aut(\mathcal{A})$ is a* one-parameter automorphism group *if $\Theta_0 = id$, $\Theta_{s+t} = \Theta_s \circ \Theta_t$, and $t \mapsto \Theta_t(a)$ is continuous for each $a \in \mathcal{A}$.*

Example 15 (Conjugation by a C_0-group)**.** *Let $U(t)$ be a strongly continuous group on Banach space X. Define $\Theta_t(S) = U(t)SU(-t)$ for $S \in \mathcal{L}(X)$. Then $(\Theta_t)_{t\in\mathbb{R}}$ is a one-parameter automorphism group of $\mathcal{L}(X)$.*

Example 16 (Adjoint action from skew-adjoint generator)**.** *If B is skew-adjoint on a Hilbert space H, then e^{tB} is unitary and $\Theta_t = \mathrm{Ad}(e^{tB})$ gives a one-parameter automorphism group of $\mathcal{L}(H)$.*

Example 17 (Exponentiated derivations)**.** *If D is a (bounded) derivation of an algebra $\mathcal{A}$, then $e^{tD} \in Aut(\mathcal{A})$ for all t and forms a one-parameter automorphism group.*

Remark 12 (Lie-theoretic principle)**.** *In all the preceding examples $\Theta_t = e^{tD}$ for a derivation D (possibly implemented as commutator with an unbounded generator in the Banach/Hilbert setting). This precisely parallels Lie theory: derivations are infinitesimal generators and exponentiation yields automorphism flows.*

5. Algebraic Error Analysis and Case Studies

5.1. Modeling errors algebraically

Distributed computation introduces two main sources of deviation from ideal algebraic behavior:

(a) *Numerical rounding* and finite-precision arithmetic (machine epsilon effects).

(b) *Systemic errors* such as data loss, message delays, and partial failures.

We propose algebraic models incorporating both.

5.2. Perturbed operations and error maps

Definition 14 (Perturbed operation)**.** *Let $(S, *)$ be an ideal semigroup (the intended exact operation). A* perturbed operation *is a binary map $\circ : S \times S \to S$ such that*

$$x \circ y = \Phi(x * y; \eta(x, y))$$

where $\eta(x, y)$ denotes an error state and Φ is a small perturbation of the identity depending on η (in many models Φ is simply multiplication by a near-identity scalar or addition of a small vector).

Example 18 (Floating-point addition)**.** *For IEEE-754 arithmetic one can model $x \oplus y = (x + y)(1 + \delta)$ where $|\delta| \leq u$ (relative error model), so the error map $\eta(x, y)$ is the scalar δ.*

5.3. Error semigroups (product with error component)

Definition 15 (Error semigroup)**.** *Let $(S, *)$ be a semigroup and $(E, \oplus)$ a semigroup modeling errors. Define the set $S \times E$ with multiplication*

$$(s_1, e_1) \star (s_2, e_2) := (s_1 * s_2,\ e_1 \oplus e_2 \oplus \epsilon(s_1, s_2))$$

where $\epsilon(s_1, s_2) \in E$ captures operation-specific errors. Then $(S \times E, \star)$ is called an error semigroup*.*

Remark 13. *This construction separates algebraic content $s \in S$ and error bookkeeping $e \in E$. When $(E, \oplus)$ is abelian (e.g. additive real errors), one can quantify growth and bound total errors.*

5.4. Robustness: formal criteria

Definition 16 (Robustness of distributed aggregation)**.** *Given an aggregation monoid $(M, *, e)$ and an error semigroup model $(M \times E, \star)$, a distributed aggregation procedure is* (C, γ)-robust *if for every partitioned dataset whose exact aggregate is $m \in M$ and whose perturbed aggregate is $(\tilde{m}, \tilde{e}) \in M \times E$ produced by the protocol, there exists a canonical mapping $\Pi : E \to \mathbb{R}_{\geq 0}$ such that*

$$d_M(\tilde{m}, m) \leq C \cdot \Pi(\tilde{e})^{\gamma},$$

for fixed constants $C > 0$ and $\gamma \in (0, 1]$, and d_M a chosen metric on M compatible with the error model.

Remark 14. *The precise choice of Π depends on the error semantics (additive versus multiplicative). For relative floating-point errors, Π may be the absolute value or log-scale norm of accumulated multiplicative perturbations.*

Proposition 2 (Semigroup property of error models)**.** *Let $(S, *)$ and $(E, \oplus)$ be semigroups, and let $\epsilon : S \times S \to E$ be any error map. Define the operation*

$$(s_1, e_1) \star (s_2, e_2) := (s_1 * s_2,\ e_1 \oplus e_2 \oplus \epsilon(s_1, s_2)).$$

Then $\star$ is associative (with the standard parenthesization), hence $(S \times E, \star)$ is a semigroup.

Proof. For $(s_1, e_1), (s_2, e_2), (s_3, e_3) \in S \times E$, consider

$$\big((s_1, e_1) \star (s_2, e_2)\big) \star (s_3, e_3) = ((s_1 * s_2) * s_3,\ (e_1 \oplus e_2 \oplus \epsilon(s_1, s_2)) \oplus e_3 \oplus \epsilon(s_1 * s_2, s_3)),$$

and

$$(s_1, e_1) \star \big((s_2, e_2) \star (s_3, e_3)\big) = (s_1 * (s_2 * s_3),\ e_1 \oplus (e_2 \oplus e_3 \oplus \epsilon(s_2, s_3)) \oplus \epsilon(s_1, s_2 * s_3)).$$

Associativity of $*$ in S and of $\oplus$ in E ensures that these expressions are equal when parenthesized in the order above. Hence $(S \times E, \star)$ is a semigroup. Note that commutativity of $\oplus$ is **not required** for this proposition.

Theorem 7 (Error growth under aggregation)**.** *Let $(M, *, e)$ be a commutative monoid, $(E, \oplus, 0)$ an* abelian *monoid, and $\epsilon : M \times M \to E$ an error map such that $\Pi(\epsilon(x, y)) \leq \delta$ for all $x, y \in M$ and some $\delta > 0$. Then for any dataset of size n, the error component $\tilde{e}$ in the perturbed reduction $(\tilde{m}, \tilde{e})$ satisfies*

$$\Pi(\tilde{e}) \leq n\delta.$$

Proof. Each binary aggregation introduces an error $\epsilon(x, y)$ with $\Pi(\epsilon(x, y)) \leq \delta$. Since $(E, \oplus)$ is abelian, all individual error terms can be freely reordered, and the total reduction of a dataset of size n involves exactly $n-1$ binary combinations. Therefore, the accumulated error satisfies $\Pi(\tilde{e}) \leq (n-1)\delta \leq n\delta$.

Remark 15 (Commutativity assumptions)**.** *Proposition 2 establishes a semigroup structure without assuming commutativity of $(E, \oplus)$. However, the explicit error bound in Theorem 7 relies on $(E, \oplus)$ being abelian to freely reorder error terms. This distinction clarifies where commutativity is necessary for quantitative error estimates.*

Corollary 2 (Robustness with linear error growth)**.** *In the setting of the theorem, any distributed aggregation is $(C, 1)$-robust with $C = n$, i.e.*

$$d_M(\tilde{m}, m) \leq n\delta,$$

provided d_M is Π-Lipschitz with constant 1*.*

5.5. Literature context and novelty

We close this subsection by situating the above three results in the existing literature and by clarifying their novelty.

(1) Semigroup property of error models. The construction of a product semigroup $(S \times E, \star)$ that keeps "algebraic content" and "error bookkeeping" separate is a straightforward algebraic device: product-like constructions are standard in semigroup theory (cf. [8, 9]), and the idea of carrying auxiliary data in a product is ubiquitous (e.g. semidirect and wreath-product constructions). What is new in our presentation is the explicit modelling of computational perturbations via an error map $\epsilon : S \times S \to E$ that records operation-specific errors arising from perturbed binary operations (see Definitions of "perturbed operation" and "error semigroup"). Thus Result 1 (Proposition 2) is not a deep algebraic theorem in isolation, but it is a novel and useful *modelling* device: it packages numerical or systems-level perturbations together with algebraic aggregation in a single semigroupic object, making later quantitative estimates and categorical constructions uniform across a broad range of aggregation semigroups.

(2) Error growth bound. Linear accumulation bounds for local rounding or perturbation errors (e.g. bounds of the form $\leq n\delta$) are classical in numerical analysis; they appear in landmark works on floating-point analysis and stability theory (see Wilkinson [7] and Higham [6]). Our Theorem 7 (Error growth under aggregation) generalizes this classical linear bound to an abstract semigroup setting: by modelling perturbations inside the error semigroup $(E, \oplus)$ and assuming a uniform per-step bound on $\Pi(\epsilon(x, y))$, we obtain a dimension-free, algebraically transparent bound on accumulated error that applies to any commutative aggregation monoid, not only to real addition. In this sense the result reinterprets and extends the classical numerical analysis bound by lifting it to a semigroup-level abstraction that is applicable to distributed reductions beyond scalar-summation.

(3) Robustness criterion. The (C, γ)-robustness definition and the ensuing corollary are, to the best of our knowledge, original. While distributed computing and numerical-analysis literatures discuss fault tolerance, stochastic error models, and numerical stability (e.g. MapReduce/Spark documentation requires associativity of reductions for determinism; see [10, 12]), they do not formulate a metric-style robustness notion tied to an explicitly defined error semigroup and a canonical 'projection' $\Pi : E \to \mathbb{R}_{\geq 0}$ that links bookkeeping values to metric error on the algebraic output. Thus Result 3 (Corollary 2)provides a rigorous bridge between algebraic models of aggregation and quantitative stability guarantees. It can be used to compare algorithms, pick resilient reductions, and reason composably about error accumulation in distributed environments.

5.6. Summarized deduction from the Three Results.

In summary:

- Result 1 is an algebraically standard but modeling-wise novel construction (error semigroup via ϵ).
- Result 2 is classical in spirit (linear accumulation) but novel in its abstraction to arbitrary aggregation monoids through semigroup/error-semigroup language.
- Result 3 is a genuinely new formal robustness criterion that unifies the algebraic and metric viewpoints and appears not to be present in the existing MapReduce / Spark / numerical-stability literatures in this exact form.

6. Case studies

We now present three focused case studies illustrating the algebraic perspective.

6.1. Case study I: Word Count (exact aggregation)

Word count uses integer addition and is exact in the absence of system failures. The algebraic model is the commutative monoid $(\mathbb{N}^{\mathcal{V}}, +, \nvdash)$; there is no inherent numerical error. Systemic errors (missed partitions) are modeled by error semigroups with an E

capturing stochastic omissions; robustness analysis then focuses on probabilistic recovery and checkpointing rather than numeric stability.

Proposition 3 (Word Count as a commutative monoid)**.** *Let $\mathcal{V}$ be a vocabulary and let $\mathbb{N}^{\mathcal{V}}$ denote the set of frequency vectors over $\mathcal{V}$. Define the operation $+$ pointwise by*

$$(f+g)(w) \;=\; f(w)+g(w), \quad w \in \mathcal{V},$$

with identity $\nvDash$ the zero function. Then $(\mathbb{N}^{\mathcal{V}}, +, \nvDash)$ is a commutative monoid.

Proof. For $f, g, h \in \mathbb{N}^{\mathcal{V}}$:

- Associativity: $((f+g)+h)(w) = (f(w)+g(w))+h(w) = f(w)+(g(w)+h(w)) = (f+(g+h))(w)$.
- Identity: $(f+\mathbf{0})(w) = f(w)+0 = f(w)$.
- Commutativity: $(f+g)(w) = f(w)+g(w) = (g+f)(w)$.

Hence $(\mathbb{N}^{\mathcal{V}}, +, \mathbf{0})$ is a commutative monoid.

Remark 16 (Error modeling in distributed Word Count)**.** *In practice, Word Count in systems such as Spark is exact under ideal conditions. However, failures or missed partitions can be modeled by extending the monoid to an* error semigroup*, where*

$$(f, E_1) \oplus (g, E_2) \;=\; (f+g,\; E_1 \cup E_2),$$

with E tracking omitted or uncertain partitions. Robustness analysis then studies probabilistic recovery mechanisms (e.g. checkpointing, lineage recomputation) rather than numerical stability, since the underlying operation is discrete and error-free.

6.2. Case study II: PageRank (iterative operator semigroup)

Definition 17 (PageRank)**.** *Let $G = (V, E)$ be a directed graph with n nodes. Denote by $P \in \mathbb{R}^{n \times n}$ the row-stochastic transition matrix, where P_{ij} is the probability of moving from node i to node j.*

The PageRank vector *$x \in \mathbb{R}^n$ is the unique probability vector satisfying*

$$x = \alpha P^{\top} x + (1-\alpha)v,$$

where $0 < \alpha < 1$ is a damping factor and $v \in \mathbb{R}^n$ is a personalization vector with $\sum_i v_i = 1$.

Equivalently, x is the stationary distribution of the Markov chain defined by $P_\alpha = \alpha P + (1-\alpha)\mathbf{1}v^{\top}$.

PageRank iterates a stochastic linear operator P on $\mathbb{R}^n$.

$$x^{(k+1)} = P x^{(k)}.$$

While P^k forms a discrete semigroup under composition, numerical implementations compute iterates approximately. Modeling one iteration by the exact map plus a bounded round-off error yields an error propagation estimate based on spectral properties of P. In particular, when the second eigenvalue λ_2 satisfies $|\lambda_2| < 1$, perturbation theory ensures stability of the dominant eigenvector under small per-iteration errors.

Proposition 4 (PageRank operator as a semigroup)**.** *Let $P \in \mathbb{R}^{n\times n}$ be a row-stochastic matrix (transition matrix of a Markov chain). Then $\{P^k : k \in \mathbb{N}\}$ forms a discrete semigroup under matrix multiplication.*

Proof. Matrix multiplication is associative, so for $k, m \in \mathbb{N}$,

$$(P^k)(P^m) = P^{k+m} \in \{P^t : t \in \mathbb{N}\}.$$

Hence closure and associativity hold, and the set forms a semigroup.

Theorem 8 (Perturbed power iteration stability)**.** *Let P be as above with eigenvalues $1 = \lambda_1 > |\lambda_2| \geq \cdots \geq |\lambda_n|$. Suppose the PageRank iteration is computed with bounded additive error,*

$$x^{(k+1)} = Px^{(k)} + \varepsilon^{(k)}, \quad \|\varepsilon^{(k)}\| \leq \delta.$$

Then the iterates converge to a vector x^* satisfying

$$\|x^* - v\| \ \leq \ \frac{\delta}{1 - |\lambda_2|},$$

where v is the true dominant eigenvector of P.

Proof. The exact iteration $x^{(k+1)} = Px^{(k)}$ converges to v since $|\lambda_2| < 1$. With perturbation, the error at step k satisfies

$$e^{(k+1)} = Pe^{(k)} + \varepsilon^{(k)}.$$

Unrolling,

$$e^{(k)} = P^k e^{(0)} + \sum_{j=0}^{k-1} P^j \varepsilon^{(k-1-j)}.$$

Since $\|P^j\| \leq |\lambda_2|^j$, we bound

$$\|e^{(k)}\| \leq \|P^k e^{(0)}\| + \delta \sum_{j=0}^{k-1} |\lambda_2|^j \leq \|e^{(0)}\||\lambda_2|^k + \frac{\delta}{1 - |\lambda_2|}.$$

As $k \to \infty$, the initial error vanishes, yielding the bound.

Remark 17 (Distributed PageRank in Spark)**.** *In distributed systems like Apache Spark, each iteration corresponds to a joinâ€“aggregate operation on the graph edges. The algebraic semigroup* $(\mathbb{R}^n, P)$ *models the iteration, while real-world computation introduces per-partition round-off and communication errors. The above theorem formalizes that as long as* $|\lambda_2| < 1$*, the PageRank vector is robust against such bounded local errors, with an error floor controlled by* $\frac{1}{1-|\lambda_2|}$*. This highlights how spectral properties dictate the resilience of large-scale iterative algorithms.*

Example 19 (Small-scale PageRank iteration)**.** *Consider a graph with* 3 *nodes and transition matrix*

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

This is row-stochastic and forms a semigroup $\{P^k : k \in \mathbb{N}\}$ *under matrix multiplication.*

Let the initial rank vector be $x^{(0)} = (1/3, 1/3, 1/3)^\top$*, and suppose each iteration is computed with bounded additive error* $\|\varepsilon^{(k)}\|_\infty \leq 0.01$*.*

Compute the first two iterates:

$$x^{(1)} = Px^{(0)} + \varepsilon^{(0)} \approx \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \end{pmatrix} + \varepsilon^{(0)}, \quad x^{(2)} = Px^{(1)} + \varepsilon^{(1)}.$$

Since the second eigenvalue of P *satisfies* $|\lambda_2| = 1$*, this simple cyclic graph does not contract errors (eigenvalue 1 multiplicity ¿ 1), illustrating that the error bound*

$$\|x^{(k)} - v\| \leq \frac{\delta}{1 - |\lambda_2|}$$

becomes ineffective here.

If instead we modify P *to include damping* $P_\alpha = \alpha P + (1-\alpha)\mathbf{1}\mathbf{1}^\top/3$ *with* $\alpha = 0.85$*, then* $|\lambda_2| = 0.85 < 1$ *and the theorem ensures convergence to a unique* v*, with maximum error* $\delta/(1 - 0.85) \approx 0.067$ *per entry. This demonstrates how damping stabilizes distributed PageRank computations under numerical errors.*

6.3. Case study III: Distributed matrix multiplication

Block-matrix multiplication in distributed frameworks decomposes matrix product into partial products aggregated via summation. The algebraic model is $M_n(\mathbb{R})$ with multiplication; rounding errors in partial products accumulate additively (in an error semigroup with $E = \mathbb{R}^{n\times n}$ under addition). The classical backward/forward error analyses (Higham, Wilkinson) can be recast in the semigroup framework to estimate norms of the error component.

Definition 18 (Distributed Block-Matrix Multiplication)**.** *Let* $A, B \in M_n(\mathbb{R})$ *be two matrices. In distributed computation, the product* $C = AB$ *is computed by partitioning* A

and B into blocks (A_i) and (B_j), forming partial products $C_{ij} = A_i B_j$, and aggregating

$$C = \sum_{i,j} C_{ij}.$$

The aggregation is associative and modeled algebraically by the semigroup $(M_n(\mathbb{R}), +)$, while rounding errors from partial products accumulate additively.

Remark 18 (Error Semigroup Perspective)**.** *The cumulative rounding errors can be represented in an* error semigroup *$(E, +)$, where $E = M_n(\mathbb{R})$, and each partial error contributes additively. Classical backward and forward error analysis (Higham, Wilkinson) can be reformulated in this semigroup framework to estimate norms of the total error.*

Example 20 (Distributed Block-Matrix Product)**.** *Suppose $A, B \in M_4(\mathbb{R})$ and we split each into 2×2 blocks:*

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}.$$

Compute each partial product $C_{ij} = A_i B_j$ on separate nodes, then aggregate:

$$C = C_{11} + C_{12} + C_{21} + C_{22}.$$

Each C_{ij} may have a small rounding error ΔC_{ij}, and the total error is

$$\Delta C = \sum_{i,j} \Delta C_{ij} \in (E, +),$$

giving a concrete illustration of the error semigroup.

7. Conclusion and Outlook

This first part of the study has established the algebraic foundations for analyzing distributed computation through semigroup theory. We have shown that the essential structure underlying distributed processes is algebraic associativity, and that semigroups provide a natural framework for reasoning about both correctness and robustness. The development proceeded along several complementary directions. We defined categorical and metric structures for semigroups, providing examples to illustrate their role in modeling distributed data operations. We performed a detailed analysis of transversal decompositions (Cases I–III), clarifying how algebraic constraints interact with computational partitioning. Additionally, we introduced perturbed operations and error semigroups, offering a systematic framework for quantifying error accumulation and formulating robustness criteria in distributed settings.

Taken together, these contributions demonstrate that semigroup theory can encode both the algebraic correctness and the error-propagation properties of large-scale computations.

The categorical viewpoint establishes a flexible formalism for reasoning about decompositions, while the error-semigroup construction formalizes resilience and stability under imperfect operations. In particular, the ε-semigroup model bridges algebraic abstraction with measurable system behavior, enabling a unified treatment of symbolic and numerical errors.

This paper has thus concentrated on establishing the core tools and illustrating their effectiveness with structural results and representative cases. The natural continuation, to be developed in a forthcoming companion paper, will extend the categorical analysis to embeddings, universal properties, and adjunctions in semigroup categories, provide a systematic treatment of robustness under Lipschitz homomorphisms and error-controlled morphisms, develop implementation strategies in distributed systems—including associative summaries, multi-round protocols, and approximation schemes with rigorous error bounds—and present case studies on large-scale problems such as PageRank, distributed matrix factorization, and quantile computation, building on the algebraic models established here.

Future work will also explore categorical formulations and prototype implementations in open-source frameworks, enabling symbolic reasoning about fault tolerance and precision directly within distributed pipelines. Thus, while this first part lays the algebraic and conceptual groundwork, the second part will develop advanced categorical tools, computational protocols, and practical implementations. Together, these two parts aim to present a comprehensive account of semigroups as a mathematical foundation for distributed computation.

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 first author would like to thank Professor Zsolt Lipcsey whose encouragement helped to strengthen focused research skills over the years in new areas of application of theoretical knowledge in the semigroup. Recent applications in areas like Classes of Semigroups, Distributed Computation, Agriculture and Marine biology reveals the extent of grooming received from him. The authors are also 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] M. I. Sampson, Z. Lipcsey, A. E. Offiong, F. A. Efiong, and M. A. Essien. Algorithm for semigroup bases i. *International Journal of Mathematical Analysis and Modelling*, 7(1):144–152, 2025.

[2] M. I. Sampson. *Generating Systems of Semigroups and Independence*. PhD thesis, University of Calabar, Calabar, Nigeria, 2022.
[3] G. K. Pedersen. *C^*-Algebras and Their Automorphism Groups*. Academic Press, 1979.
[4] K.-J. Engel and R. Nagel. *One-Parameter Semigroups for Linear Evolution Equations*. Springer, 2000.
[5] J. S. Golan. *Semirings and Their Applications*. Springer, 1999.
[6] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2 edition, 2002.
[7] J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Prentice Hall, 1963.
[8] J. M. Howie. *Fundamentals of Semigroup Theory*. Oxford University Press, 1995.
[9] A. H. Clifford and G. B. Preston. *The Algebraic Theory of Semigroups*, volume 1. American Mathematical Society, 1961.
[10] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
[11] M. Zaharia, M. Chowdhury, M. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud)*, 2010.
[12] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012.
[13] S. Lang. *Algebra*. Springer, revised 3rd edition, 2002.
[14] R. Bird. An introduction to the theory of lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design*. Springer, 1987.
[15] J. Gibbons. Origami programming. In J. Gibbons, editor, *The Fun of Programming*. Palgrave, 2015.