

rsy: A Mathematical Note on Parallel File Synchronization

Pranav Gundu

Revision 1.1.3

Abstract

This note models **rsy**, a Rust file synchronization tool for local copies, SSH transfers, and daemon-style **rsync://** targets. The emphasis is mathematical: file-list construction is treated as a parallel traversal problem, filtering as an ordered predicate system, cold copies as a kernel-assisted byte movement problem, and warm synchronizations as a rolling-checksum delta problem. The current release pipeline is also expressed as a dependency relation: continuous deployment is blocked unless continuous integration first proves formatting, linting, building, and testing.

1 Objects and Cost Model

Let the source tree be a finite rooted tree

$$\mathcal{T} = (V, E), \quad V = \mathcal{F} \cup \mathcal{D} \cup \mathcal{L},$$

where $\mathcal{F}$ is the set of regular files, $\mathcal{D}$ the set of directories, and $\mathcal{L}$ the set of symbolic links. Let

$$F = |\mathcal{F}|, \quad D = |\mathcal{D}|, \quad L = |\mathcal{L}|, \quad N = F + D + L.$$

Each regular file $f_i \in \mathcal{F}$ has size b_i , mode m_i , modification time τ_i , and relative path p_i . Define total byte mass

$$B = \sum_{i=1}^F b_i.$$

The observed runtime is decomposed as

$$T = T_{walk} + T_{filter} + T_{meta} + T_{copy} + T_{delta} + T_{transport} + T_{ui}. \quad (1)$$

For a local cold copy, $T_{transport} = 0$ and $T_{delta} \approx 0$. For SSH and daemon targets, $T_{transport}$ includes remote process setup, framing, and stream I/O. For warm destinations, T_{copy} may collapse to zero for unchanged files.

Definition 1 (Skip predicate). *For source file s and destination file d , the default unchanged predicate is*

$$U_{mtime}(s, d) \iff |s| = |d| \wedge \tau(s) = \tau(d). \quad (2)$$

*With **-checksum**, the predicate becomes*

$$U_{hash}(s, d) \iff H(s) = H(d), \quad (3)$$

*where H is the file hash. With **-update**, a destination-newer predicate*

$$N_{dst}(s, d) \iff \tau(d) > \tau(s) \quad (4)$$

also causes preservation of d .

2 Path Classification

The command parser maps each invocation to exactly one mode. In order, destination daemon syntax is recognized, then destination SSH syntax, then source SSH syntax, otherwise local mode.

Form	Mode	Interpretation
<code>rsy SRC/ DST/</code>	Local	local namespace copy
<code>rsy SRC/ host:/path/</code>	SSH push	local sender, remote receiver
<code>rsy host:/path/ DST/</code>	SSH pull	remote sender, local receiver
<code>rsy SRC/ rsync://host/module/path</code>	Daemon push	daemon transport
<code>rsy SRC/ host::module/path</code>	Daemon push	daemon module syntax

Trailing slash semantics match `rsync`: `SRC/` copies the contents of `SRC`, while `SRC` copies the directory itself under the destination.

3 Filtering as an Ordered Predicate System

A filter list is an ordered sequence

$$R = (r_1, r_2, \dots, r_k), \quad r_j = (kind_j, pattern_j),$$

where $kind_j \in \{include, exclude\}$. A file path p is accepted by the first matching rule:

$$Allow(p) = \begin{cases} 1, & \exists j : Match(r_j, p) \wedge kind_j = include \wedge \forall \ell < j, \neg Match(r_\ell, p), \\ 0, & \exists j : Match(r_j, p) \wedge kind_j = exclude \wedge \forall \ell < j, \neg Match(r_\ell, p), \\ 1, & \text{no rule matches.} \end{cases} \quad (5)$$

Directory-only rules extend to descendants by testing ancestors of p .

Proposition 1 (First-match determinism). *For a fixed ordered rule list R and path p , $Allow(p)$ is deterministic and independent of later matching rules after the first match.*

Proof. Let $j^* = \min\{j : Match(r_j, p)\}$. If no such j^* exists, the default is accept. Otherwise the definition depends only on $kind_{j^*}$, so all r_j for $j > j^*$ are irrelevant. $\square$

4 Traversal and Parallel Work

Let P be the number of worker threads. For independent file work with per-file costs w_i , ideal scheduling gives

$$T_{parallel} \geq \max \left(\max_i w_i, \frac{1}{P} \sum_{i=1}^F w_i \right). \quad (6)$$

In practice,

$$T_{parallel} = \frac{1}{P_{eff}} \sum_i w_i + T_{sched} + T_{io-contention}, \quad (7)$$

where $1 \leq P_{eff} \leq P$. `rsy` defaults to approximately $2 \times$ logical CPU count with a minimum of four workers because many file operations are I/O-bound rather than pure CPU-bound.

When `-delete` is absent, destination inventory is not needed. If N_s and N_d are source and destination node counts, avoiding the destination walk saves approximately

$$\Delta T_{delete=false} \approx c_w N_d + c_m N_d, \quad (8)$$

where c_w is traversal cost and c_m is metadata inspection cost.

5 Cold Copy Model

For a new file, no destination basis exists. The optimal path is therefore whole-file copy, not delta construction. Model the cost for file i as

$$C_i^{cold} = \alpha_{open} + \alpha_{meta} + \beta_i b_i + \alpha_{mtime}. \quad (9)$$

Here β_i depends on the filesystem and kernel primitive. On APFS, `clonefile()` may make $\beta_i \approx 0$ for same-volume copy-on-write; on Linux, `FICLONE` reflinks may do the same on supporting filesystems. Otherwise β_i is normal copy bandwidth cost.

Summing over all cold files gives

$$T_{cold} = F(\alpha_{open} + \alpha_{meta} + \alpha_{mtime}) + \sum_{i=1}^F \beta_i b_i. \quad (10)$$

Corollary 1 (Small-file sensitivity). *If b_i is small for most files, fixed metadata terms dominate. Parallelizing file work reduces wall time more strongly than increasing sequential byte bandwidth.*

6 Rolling-Checksum Delta Model

For a warm file with a destination basis, `rsy` partitions the basis into blocks of length q . For basis block j , it stores a weak rolling sum r_j and strong sum h_j :

$$\sigma_j = (r_j, h_j, offset_j), \quad j = 0, \dots, \left\lceil \frac{b}{q} \right\rceil - 1. \quad (11)$$

The sender scans the source with a rolling window. A weak match proposes candidates; a strong match validates a copy token. The output token stream is

$$\Theta = (\theta_1, \dots, \theta_m), \quad \theta_j \in \{Data(bytes), Copy(offset, len)\}. \quad (12)$$

Let $\rho \in [0, 1]$ be the changed-byte fraction. Ignoring headers, the wire payload is approximately

$$W_{delta} \approx \rho B + c_{tok} \frac{B}{q}. \quad (13)$$

Whole-file transfer has payload $W_{whole} = B$. Delta is wire-efficient when

$$\rho B + c_{tok} \frac{B}{q} < B \iff \rho < 1 - \frac{c_{tok}}{q}. \quad (14)$$

Lemma 1 (Block-size tradeoff). *Smaller q increases match granularity but increases signature and token overhead. Larger q reduces overhead but may turn small edits into larger literal regions.*

7 Atomicity and In-Place Updates

By default, `rsy` writes patched output to a temporary file and renames it into place. Let A be the event that interruption occurs during transfer. Atomic replacement ensures that, under normal filesystem rename semantics,

$$A \Rightarrow dst \in \{old, new\}, \quad (15)$$

not an arbitrary prefix. With `-inplace`, the temporary file is avoided, but interruption can leave a partial destination. Thus `-inplace` trades safety for reduced replacement overhead and lower temporary space usage.

8 Path Safety

The receiver cleans relative paths lexically. A path p is admissible only if

$$\neg \text{Absolute}(p) \wedge \neg \text{ContainsParentTraversal}(p) \wedge \text{Confined}(\text{base}, p). \quad (16)$$

For symbolic links, the target t must also be relative and non-escaping:

$$\text{SafeLink}(t) \iff \neg \text{Absolute}(t) \wedge \text{depth}(t[0..k]) \geq 0 \ \forall k. \quad (17)$$

9 Command Surface

The core transfer options are summarized below.

Option	Mathematical or operational effect
-a, -archive	preserve times, permissions, owner, group, symlinks
-delete	remove $\text{dst} \setminus \text{src}$ after filtered source inventory
-u, -update	apply $N_{\text{dst}}(s, d)$ skip predicate
-ignore-existing	skip all d where $\text{exists}(d) = \text{true}$
-W, -whole-file	force W_{whole} rather than W_{delta}
-inplace	patch destination directly
-c, -checksum	use U_{hash} rather than U_{mtime}
-min-size, -max-size	restrict files by b_i bounds
-exclude-from	import ordered exclude predicates
-log, -no-tui, -stats	choose output/reporting channel

10 Release Pipeline as a Dependency Graph

Let C be the CI job, B_m the release build matrix, N npm publishing, and R crate publishing. The current workflow enforces

$$C \rightarrow B_m \rightarrow \{N, R\}. \quad (18)$$

The CI predicate is

$$C = \text{Fmt} \wedge \text{Clippy} \wedge \text{Build}_{\text{release}} \wedge \text{Test}_{\text{release}}. \quad (19)$$

Therefore publishing is possible only if

$$\text{Publish} = C \wedge B_m. \quad (20)$$

This is the precise meaning of “CI must pass before CD.”

11 Benchmarks

The benchmark suite uses Criterion and generated corpora. Because corpora and destination trees are created by temporary directories, disk placement is controlled by the environment variable TMPDIR:

```
mkdir -p "/Volumes/PG Drive/rsy-bench-tmp"
TMPDIR="/Volumes/PG Drive/rsy-bench-tmp" cargo bench
```

Representative README results show speedups in three regimes:

Scenario	rsy	Comparison	Dominant term
10k files $\times$ 4KB cold	126 ms	3.83x faster than rsync	metadata parallelism
500 files $\times$ 1MB cold	35 ms	7.77x faster than rsync	parallel byte movement
10 files $\times$ 10MB, 10% changed	1.4 ms	31.43x faster than rsync	delta token efficiency

12 Verification

The binary unit suite currently exercises 69 tests across checksum behavior, rolling delta reconstruction, parser mode detection, filter precedence, filter-file loading, path confinement, protocol bounds, dry-run behavior, deletion, size filtering, update preservation, and whole-file replacement. The release CI gate runs formatting, clippy, release build, and release tests before any tagged CD build can start.

13 Conclusion

The design of `rsy` is compactly captured by three inequalities. Cold-copy performance improves when metadata and kernel-copy constants shrink:

$$F\alpha_{meta} + \beta B \ll F\alpha'_{meta} + \beta' B.$$

Delta transfer wins when the changed fraction satisfies

$$\rho < 1 - \frac{c_{tok}}{q}.$$

Release safety improves because deployment is no longer independent of CI:

$$CD \Rightarrow CI.$$

Together these properties describe the current implementation: fast when copying, conservative when writing, selective when updating, and gated when releasing.