Clustering and noise reduction

Clustering has always been one of the necessary ways to reduce noise. Instead of engineers seeing a stream of errors every day, a properly clustered system presents a manageable set of distinct issues to solve. This study compares two fundamentally different approaches to the clustering problem: Sentry's hash-based grouping algorithm, and Foam's streaming log template mining system built upon the Drain3 algorithm.

10.9×
Median issue reduction

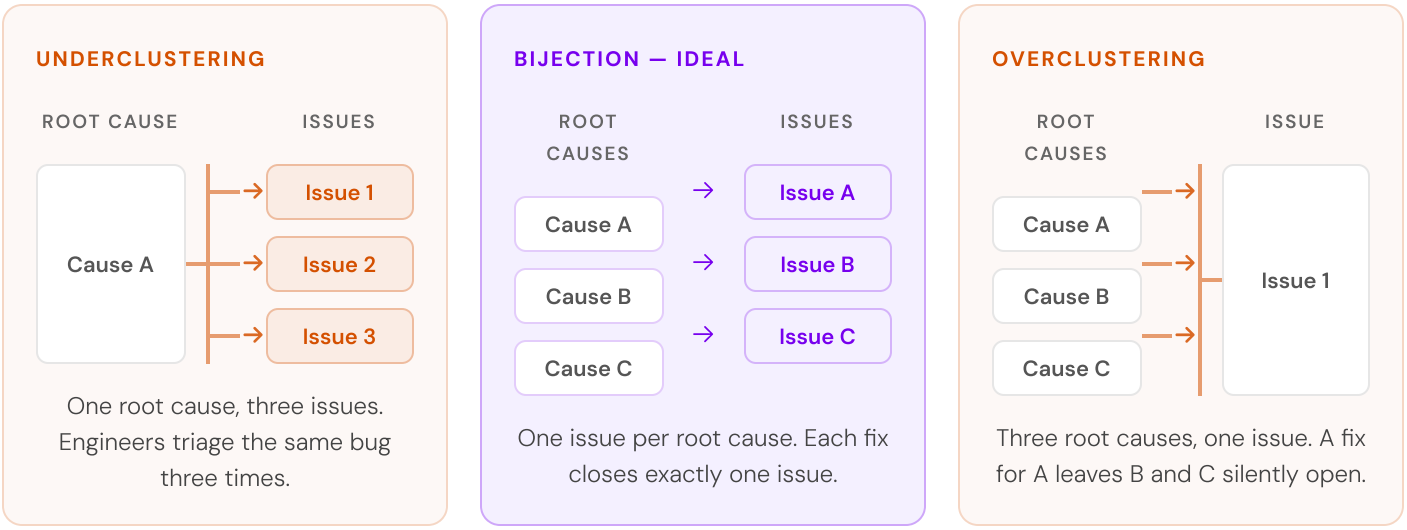
95.5%
Fewer issues to triage

11
Production environments

Across 11 production environments, we compared Sentry's grouping output against Foam's clustering applied to the same underlying error data. Sentry's pipeline produced 126,651 issues; Foam reduced these to 5,671 distinct clusters.

> **The ideal clustering algorithm is a bijection between issues and root causes: one issue per root cause, one root cause per issue.**

The central challenge with clustering algorithms is avoiding both failure modes: underclustering, where a single root cause fragments into multiple issues, and overclustering, where multiple root causes collapse under a single issue.



How Sentry groups events

When Sentry receives an error event, it normalizes the stack trace by stripping system frames, prioritizing module and function names as the stable identifiers within a frame. It computes an MD5 hash of the normalized stack trace:

$$\texttt{GroupHash} = \texttt{MD5}(f_1 \parallel f_2 \parallel \cdots \parallel f_n)$$

where each f_i is a contributing in-app frame, represented as `module::function_name`

Sentry checks whether any existing issue carries the same hash. If no match is found, it invokes an ML agent which embeds the normalized stack trace and uses cosine similarity against existing issues, combining them if similarity meets a certain threshold. [1] [2]

When an event has a stable stack trace, Sentry groups on that normalized trace and ignores the exception message entirely. This approach has a structural limitation: code refactors change the hash. The same bug, relocated to a different function or file, produces a new Sentry issue, even though the root cause is identical.

The structural limitation: Fingerprinting groups by a property of the failure, the call sequence, rather than a property of its cause. A bug relocated across refactors becomes a new issue. Errors from unrelated bugs that share a common library call path collapse into one.

Sentry describes its fallback behavior directly: "When stack traces aren't available or sufficient, the system falls back to examining exception types and values, and as a last resort, uses normalized versions of error messages." The exception message is the last signal consulted, not the first.

How Foam groups events

Foam operates on exception messages, not the stack trace. The core of this system is Drain3: an open-source streaming log template miner. [3] Exception messages inherently have constant parts (describing what went wrong) and variable parts (describing specific context such as user IDs or URLs). Drain3 distinguishes between the two by replacing variable tokens with a wildcard denoted as `<*>`. This ensures that issues represent a class of errors rather than a single unique occurrence.

Drain3 maintains a fixed-depth parse tree. Incoming exception spans are first routed by exception message length, then by their tokens. At the leaf nodes sit candidate cluster IDs. Each candidate is scored by positional token similarity against the incoming message:

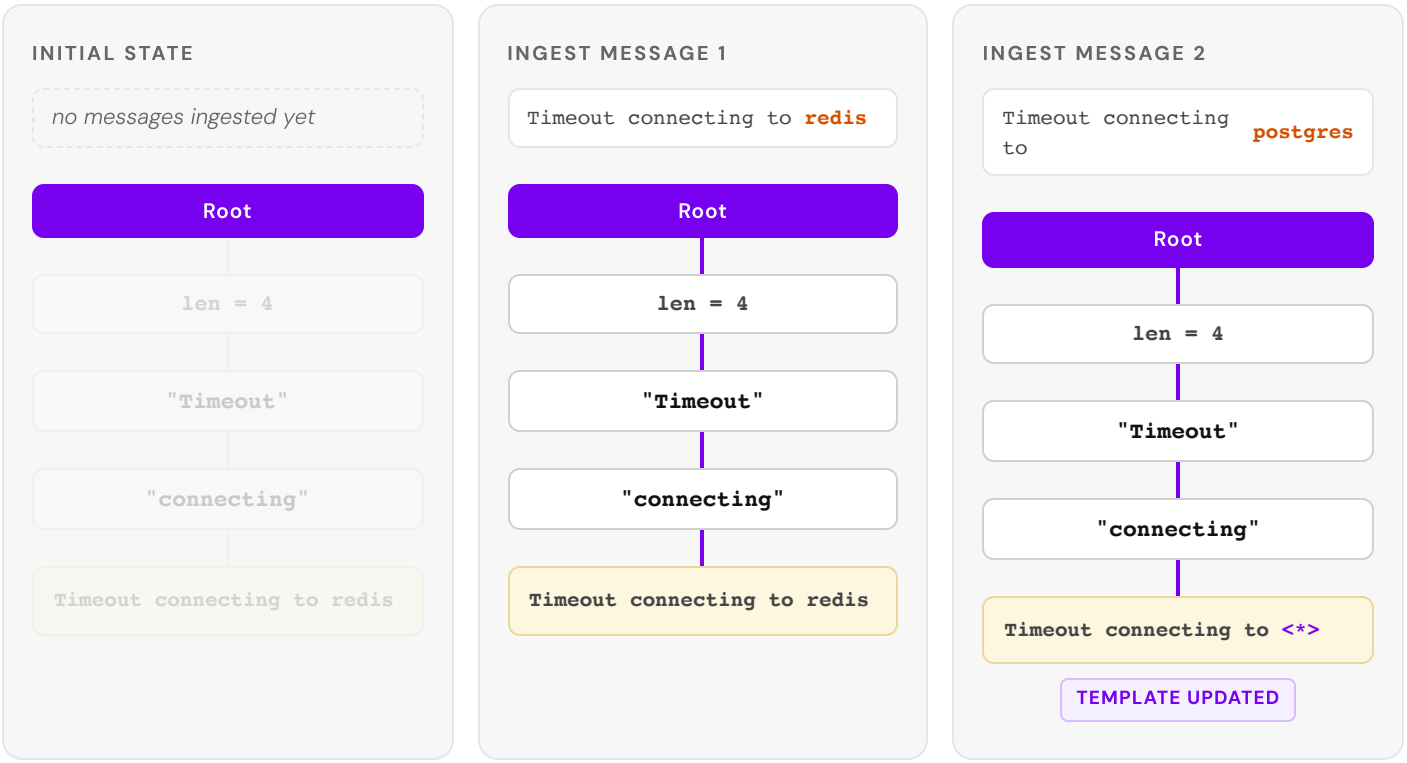
$$\mathrm{sim}(t,\ m) = \frac{\left|\{i\ :\ t_i = m_i,\ t_i \neq \texttt{<*>}\}\right|}{|t|}$$

where t is the existing template token sequence and m is the incoming message

If no candidate exceeds the similarity threshold, Drain3 creates a new cluster. Because Drain3 is a streaming parser, templates evolve as more exceptions are ingested. Cluster definitions become a product of the data rather than being statically configured.

DRAIN3 PARSE TREE

A fixed-depth prefix tree. Incoming messages are routed by token count (depth 1), then first token (depth 2), then second token (depth 3), arriving at candidate log clusters at the leaf. When two messages share a path, the differing token becomes a wildcard.



Root · Depth 1 · token count · Depth 2 · first token · Depth 3 · second token · Leaf · log template

Active routing path

Deduplication layer

Drain3 matches raw incoming messages against existing templates but has no mechanism to compare two templates against each other. A template created early in one batch and a similar template created independently in another may never be reconciled. Additionally, Drain3's similarity function requires equal-length sequences. To address these limitations, Foam adds a deduplication layer using the overlap coefficient. This algorithm scans existing issues sharing the same initial three tokens and computes the overlap coefficient: the shared token count divided by the size of the smaller set. If the threshold is reached, the issues are merged.

$$\frac{|A \cap B|}{\min(|A|, |B|)}$$

where A and B are the token sets of two candidate clusters

Side-by-side comparison

	FOAM	SENTRY
Loss of data	No sampling — all exceptions are clustered	Sampling applied
Clusters on	Exception messages	Stack traces and exception type; exception message as last resort
Similarity metric	Token sequence similarity + overlap coefficient	Cosine distance on embeddings
Template discovery	Templates evolve as more exceptions are ingested	Engineers define rules per project, requires maintenance as the codebase changes

Reduction ratios across production environments

The ratio of Sentry issues to Foam issues across environments is not linear; the spread reveals how the two systems respond to different error patterns. This data was collected during Foam's transition from Sentry-based grouping to its own clustering technology, allowing a direct comparison of how many issues each system produces from the same underlying errors.

INDUSTRY	SENTRY ISSUES	FOAM ISSUES	REDUCTION
Billing infrastructure	15,858	1,320	12×
AI infrastructure	259	93	2.8×
Education	2,220	204	10.9×
AI detection	8,101	667	12.1×
Field operations	2,345	140	16.8×
AI content	694	200	3.5×

INDUSTRY	SENTRY ISSUES	FOAM ISSUES	REDUCTION
Accounting	485	201	2.4×
AI evaluation	30,410	863	35.2×
Developer tools	1,216	327	3.7×
HR technology	2,179	413	5.3×
AI search	62,884	1,243	50.6×
Total	**126,651**	**5,671**	**22.3×**

50.6×

AI Search — High end

Services primarily error on external dependencies: MCPs, browser extensions, search APIs. Small changes in extension versions create separate Sentry issues; Foam wildcards the variable parts and collapses them. Code refactors amplify this — the same bug relocated twice produced three distinct Sentry issues.

10.9×

Education — Median

Clusters well on issues that don't originate from the same call stack but share the same root cause. A misconfigured asyncio event loop propagated through every span in a trace, generating 6 Sentry issues. Foam collapsed these to 3 issues.

2.4×

Accounting — Low end

Integration with a multitude of financial APIs, each with its own error format, produces a high number of distinct errors. Where errors are truly distinct, both systems produce separate issues. A low ratio is not underclustering but rather an accurate reflection of the error diversity.

Conclusion

Clustering errors perfectly is an unsolved task. The idealism of a bijection between issues and root causes is enviable, but real systems must make tradeoffs between the two failure modes.

Foam's approach, which is operating on exception messages via a streaming template miner rather than on stack trace hashes, yields a 10.9× median reduction across 11 production environments and 95.5% fewer issues for engineers to triage. The deduplication layer handles

merging that Drain3 alone cannot. The result is a clustering system that improves automatically as more data is ingested, without requiring manual rule maintenance as the codebase changes.

References

[1] Elser, T., Ferge, J. "How Sentry Decreased Issue Noise with AI." Sentry blog. Documents Sentry's deployed AI-enhanced issue grouping. blog.sentry.io/how-sentry-decreased-issue-noise-with-ai

[2] Sentry. "Enhancing Issue Grouping." Sentry blog. blog.sentry.io/enhancing-issue-grouping

[3] Drain3 — Streaming log template miner by IBM Research, based on the Drain algorithm. github.com/logpai/Drain3