Technische Universität München

How we Made it to 1st Place on the Leaderboard

Henrik Mühe
henrik.muehe@in.tum.de

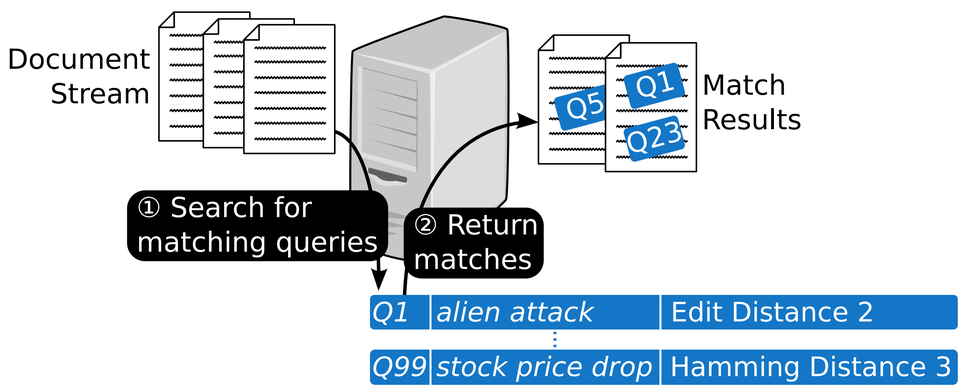
Florian Funke
florian.funke@in.tum.de

The Final Leaderboard

	Team	Small (sec)	Big (sec)	New (sec)
1	**Campers (TUM)**	0.081	1.938	**7.515**
2	RotaFortunae (Saint Petersburg University)	0.158	1.969	9.394
3	mofumofu (Tohoku University)	0.065	1.507	10.343
4	glhf	0.137	2.100	11.795
5	phoenix (Peking University)	0.585	2.320	12.794
6	StrongAccept (Tsinghua University)	0.396	3.019	12.848
	...			
51	weWillWin	18.021	N/A	N/A
52	null	22.463	N/A	N/A
53	ePetra	30.927	N/A	N/A
54	JoblessCoders	43.174	N/A	N/A
55	TangYuan	43.798	N/A	N/A

The Challenge

Implement a **stream system**:
The server must allow to register **fuzzy string matching** queries.
For each document streamed through the system, it must return the set of queries that matched.



The Metrics

Exact Matching: Test if two words are equal
Hamming Distance: Number of positions that differ between two words of the same size
Levenshtein Edit Distance: Minimum number of insert/delete/substitute operations between two words

The API

```
startQuery(qId,query,type,distance)
endQuery(qId)
matchDocument(docId, document)
getNextAvailableResult(docId*, n*, qId**)
```

Inherent Optimizations

① Document deduplication, trivial
② Query word clustering, **NOT** trivial

Example

Q1: justin bieber
Q2: justin timberlake
Q3: justin time

$$\neg matches(\text{"}justin\text{"}) \implies \neg matches(\{Q1, Q2, Q3\})$$

Algorithm

- For each query word, determine "skip words"
- Incrementally remove skip words, periodically recompute
- Skip vector for all deactivate words

Parallelism & Concurrency

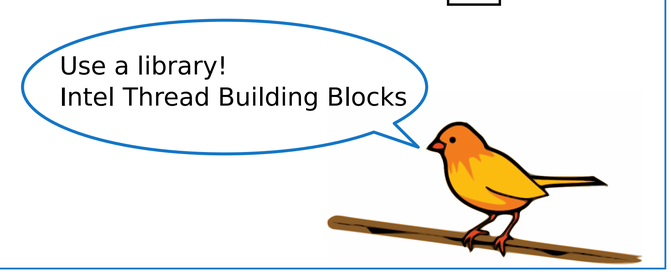
`matchDocument(char* doc)` → `strcpy()` → `spawn()` → `return`

Task: MatchDocument
- Tokenization and Preparation
- `spawn()`
- Exact Matching
- `join()`
- Result compilation
- `enqueue(Result)`

Task: Edit Distance

Task: Hamming Dist.

- Hierarchically expose parallelism
- No unnecessary synchronization



Low-Overhead Filtering

Use filters to determine if two words *can* be within Edit/Hamming distance *d*.
Challenge: Filter must be substantially faster than invoking the metric itself
Our filters:

① **Length:** $||word_a| - |word_b|| \leq d \implies$ possible match

② **Frequency Histogram (full size and folded):**
"abb" Delta Δ "abbab"

a=1	1	a=2
b=2	1	b=3
…	…	…
z=0	0	z=0
	∑=2	

$$\Delta \leq 2d - ||word_a| - |word_b||$$

Blazing-Fast Metric Computation

Hamming Distance

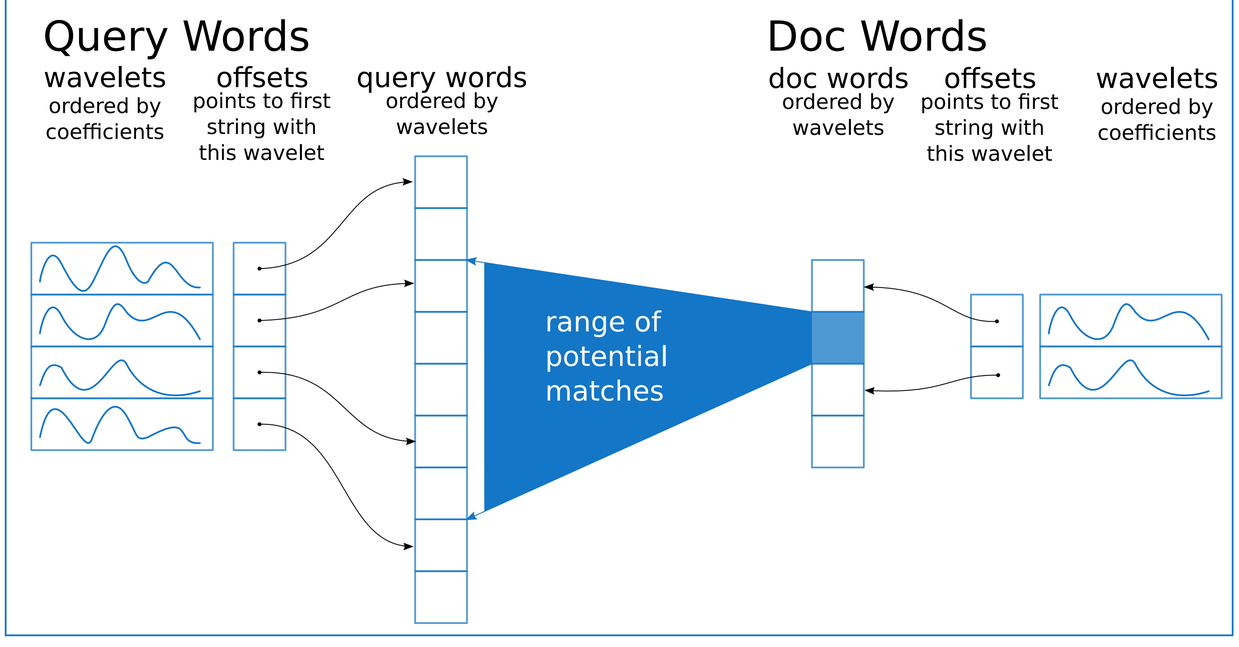
- SIMD instructions
- Improved version over streaming string SSE4

```
static inline unsigned similarity_hamming(__m128i a,__m128i b)
{
    __m128i mask=_mm_set1_epi8(254);
    union { __m128i a; uint64_t b[2]; } x;
    x.a=_mm_adds_epu8(_mm_xor_si128(a,b),mask);
    return (_mm_popcnt_u64(x.b[0])+_mm_popcnt_u64(x.b[1]))-(128-16);
}
```

Edit Distance

- Fastest available bit-parallel algorithm by Myers
- Enhanced with SIMD instructions to eliminate loop

Haar Wavelet Index



Match Caching

Observation

People make the same typos again and again, e.g. "calendar" vs "calender".

Idea

- Cache misspelled (document) words with matching query words (for each metric and each distance).
- Probe document's hash table with misspelled word.
- Often saves the iteration through entire document.