Efficient Text Analytics on BookCorpus: Sampling, Sketching Compression with Apache Spark

ZERVAKIS KONSTANTINOS

May 1, 2025

1 Introduction

In the era of large-scale unstructured data, extracting meaningful insights from massive text corpora is both crucial and computationally intensive. Traditional tools fall short in terms of speed and scalability, especially when applied to tasks like tokenization, aggregation, and summarization over millions of documents. In this context, Apache Spark [5] offers a scalable, in-memory computing framework well-suited for distributed processing of large datasets.

This project explores various scalable techniques for textual data analysis using Spark, applied to the BookCorpus dataset—an English-language collection of over 14 million book passages. Our workflow begins with exact word-level analytics using Spark's native parallelism, and progresses through approximation techniques including sampling, Count-Min Sketch [1], and Flajolet-Martin Sketch [2], culminating in histogram and wavelet-based compression strategies [3].

Each method is evaluated in terms of accuracy, runtime, and memory efficiency, with observations supported by live Spark UI visualizations and performance measurements. The structure of the paper is as follows: We describe the dataset and Spark pipeline, proceed with full and sampled analysis, explore streaming-oriented and sketch-based approximations, compress the signal using wavelets, and conclude with a comparative evaluation.

2 Dataset Description

The BookCorpus dataset contains over 14 million English-language text entries derived from self-published books. Each entry is a paragraph or passage of free-form text. We streamed and exported the dataset into a CSV file for reproducible processing, resulting in approximately 3.4 GB of textual data.

- Total records: **14,000,000**
- Mean words per record: **13.26**
- Max word count: **1,189**

This scale makes BookCorpus an ideal candidate for evaluating both full and approximate text analytics methods under realistic Big Data conditions.

3 Spark Processing Pipeline

We used Apache Spark to tokenize, normalize, and analyze the corpus. Each line of text was lowercased, stripped of punctuation, and split using regular expressions. The job was parallelized over 8 Spark partitions for distributed execution.

3.1 Monitoring Execution with Spark UI

The Spark UI provided live insights into task execution and resource utilization. Figure 3 shows executor-level metrics, while Figure 4 presents the logical DAG for the word count transformation.

This UI-driven monitoring helped confirm that the pipeline was free of task skew or execution bottlenecks, allowing reliable downstream analysis.

3.2 Monitoring Execution with Spark UI

The Spark UI was used to visualize DAG execution and task metrics. Figure 3 shows the executor summary with per-task memory and shuffle statistics. Figure 4 shows the DAG corresponding to the word frequency computation.

As shown in Figure 3, task execution was well-balanced with consistent memory and shuffle usage across partitions. No skew or outlier behavior was detected.

Figure 4 illustrates the logical plan used by Spark to perform the transformation pipeline. The use of groupBy and shuffle is clearly visible and highlights the computationally expensive operations.

4 Word Count Analysis

To gain an initial understanding of the dataset's structure, we computed the number of words per text record across the entire corpus. The following descriptive statistics summarize the distribution:

- **Min:** 1 word
- **25th percentile:** 7 words
- **Median:** 11 words

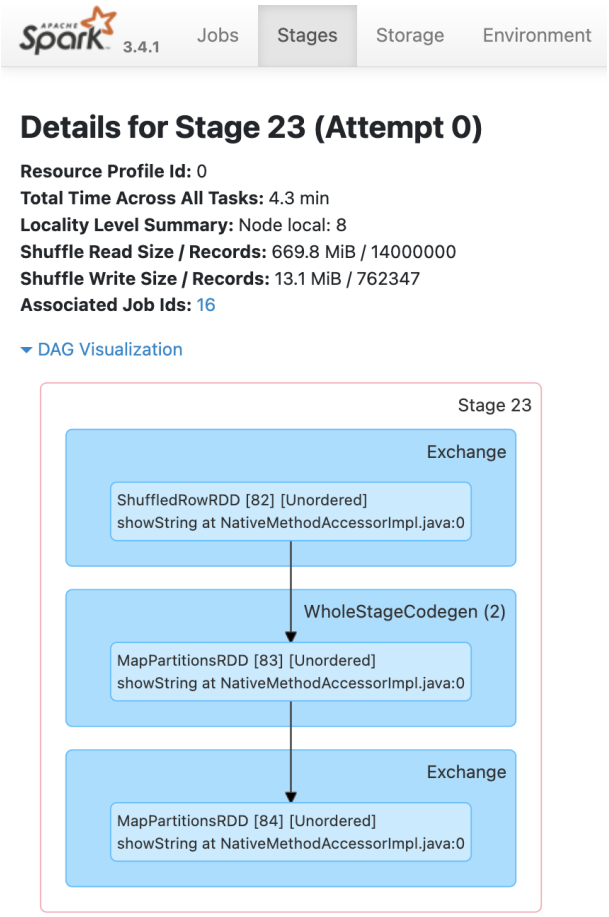

Figure 1: Spark task summary showing execution metrics per stage.

- **75th percentile:** 18 words
- **Max:** 1,189 words

The results reveal a highly skewed distribution: the majority of records are short passages, while a long tail of outlier entries contributes disproportionately to total word volume. This imbalance has practical implications: it affects memory usage and load balancing during distributed tokenization and motivates the need for scalable summarization and approximation techniques introduced in later sections.

5 Top Word Frequencies

Following tokenization and normalization, we computed global word frequencies across the entire corpus. Table 1 presents the 20 most frequent tokens observed.

Unsurprisingly, most of the top tokens are high-frequency function words (e.g., `the`, `and`, `was`), which are common across English corpora. While these results serve as a consistency check for preprocessing, they also inform decisions for later phases. For example, removing such

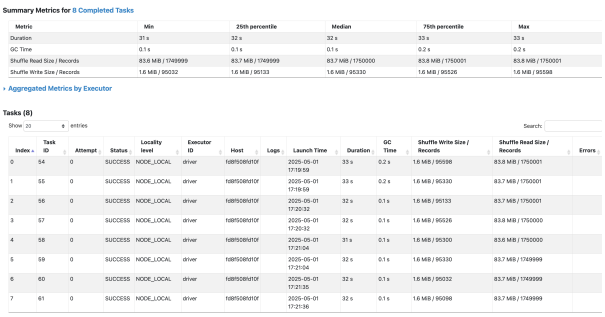

Figure 2: DAG from Spark UI for word frequency aggregation.

Word	Frequency
`the`	6,718,070
`and`	3,860,283
`her`	2,378,682
`was`	2,194,890
`you`	2,172,817
`she`	2,081,657
`his`	1,909,826
`that`	1,774,819
`with`	1,113,227
`had`	1,076,272
`for`	1,011,328
`but`	974,270
`him`	949,685
`what`	670,928
`not`	661,281
`have`	648,698
`out`	642,724
`this`	609,771
`they`	589,215
`said`	588,941

Table 1: Top 20 most frequent words in the BookCorpus dataset.

stopwords or focusing on content-bearing terms could improve compression efficiency and highlight semantic patterns.

Furthermore, these exact counts serve as a baseline for evaluating the accuracy of later approximation methods including sampling, sketching, and compression.

6 Random Sampling

While full-data analysis ensures accuracy, it is time and memory-intensive. To approximate results faster, we used a 5% random sample of the corpus.

6.1 Random Sampling Job Execution

The sample was processed as a separate Spark job. Figure 5 presents the execution summary from Spark UI.

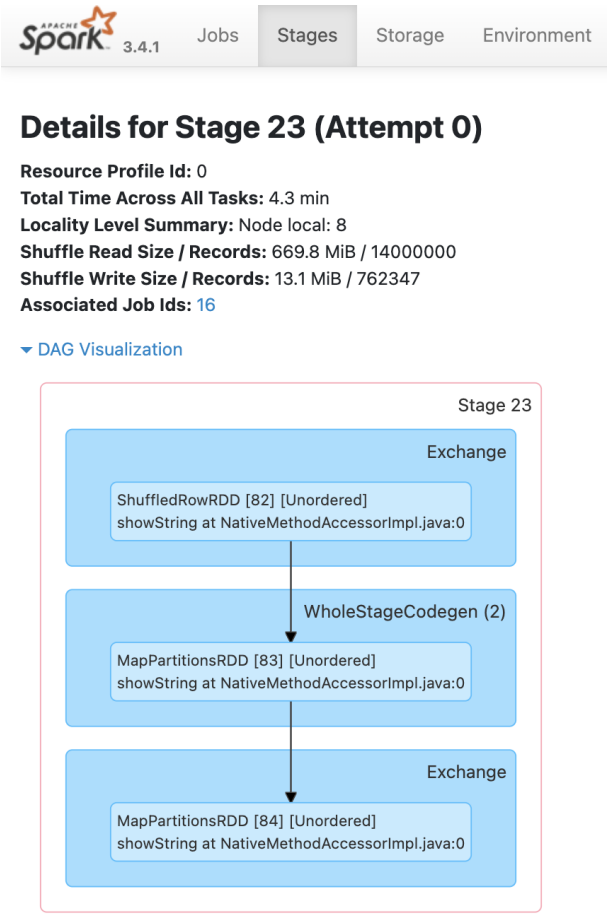

Figure 3: Spark task summary showing execution metrics per stage.

The job completed successfully in under one second, split across 2 tasks, each reading approximately 1.8 MiB of shuffled data and processing over 100,000 records. No garbage collection or skew issues were observed.

6.2 Top Word Frequencies in Sample

To assess how well a random sample reflects the global word distribution, we analyzed the top word frequencies in a 5% subset of the corpus. Table 2 lists the most common tokens observed in the sample.

Despite being only 5% of the dataset, the sample preserved the rank and magnitude of the most frequent terms with remarkable consistency. This highlights the potential of random sampling for approximate query answering and lightweight analytics in large-scale text processing.

7 Reservoir Sampling

While random sampling is effective when full data access is possible, it assumes that the dataset size is known and uniformly accessible. In streaming environments, where data arrives incrementally or in unknown volume, such

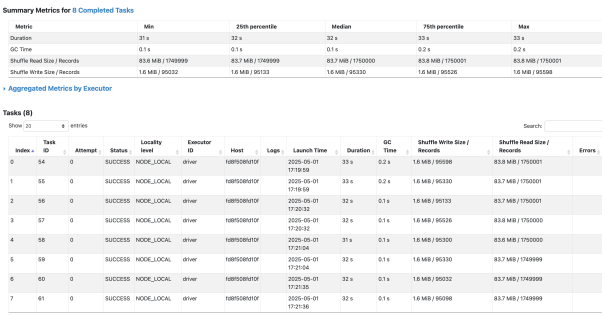

Figure 4: DAG from Spark UI for word frequency aggregation.

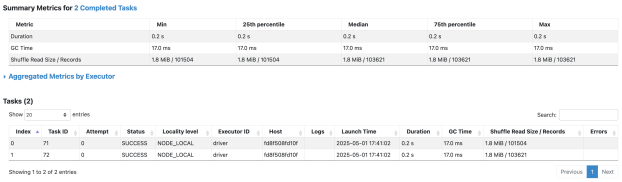

Figure 5: Spark UI for the 5% sampling job execution (2 tasks).

assumptions are violated.

Reservoir sampling [4] addresses this by maintaining a fixed-size uniform sample from a stream of unknown length. We applied reservoir sampling to the BookCorpus stream to collect 100,000 representative entries using an in-memory reservoir with constant space complexity.

7.1 Top Word Frequencies (Reservoir Sample)

We tokenized and processed the reservoir sample using the same Spark pipeline. Table 3 lists the top 20 most frequent tokens observed.

7.2 Commentary

The distribution from the reservoir sample closely mirrors that of the random sample and full dataset. High-frequency function words retain their dominance, and token ranks exhibit only minimal deviations.

This confirms that reservoir sampling is a robust and scalable method for streaming text summarization, where full-corpus access is infeasible or too costly. Its memory efficiency and statistical validity make it a strong candidate for integration into real-time analytics pipelines.

8 Approximate Frequency Estimation using Count-Min Sketch

While random and reservoir sampling provide efficient ways to reduce data size, they still require full processing

Word	Count (sample)
the	335,524
and	192,585
her	118,497
was	109,838
you	108,443
she	103,580
his	95,352
that	88,155
with	55,453
had	53,834
for	50,827
but	48,563
him	47,446
what	33,496
not	32,999
have	32,339
out	31,815
this	30,566
said	29,519
did	29,345

Table 2: Top 20 word frequencies from the 5% Spark sample.

Word	Count (reservoir)
the	30,360
and	14,959
you	11,693
was	10,842
her	10,314
she	9,683
that	8,461
his	8,038
had	5,052
with	4,729
for	4,530
him	4,091
what	3,557
this	3,251
have	3,201
not	3,134
out	2,881
did	2,854
but	2,852
they	2,847

Table 3: Top 20 word frequencies from the reservoir sample.

of sampled records. To further reduce memory and computational cost, we implemented the Count-Min Sketch (CMS) [1], a probabilistic data structure designed for frequency approximation in data streams.

CMS uses multiple hash functions to map input elements into a fixed-size 2D array. For each element, one count is incremented per row, and the minimum of the hashed columns is returned as its estimate. Importantly, CMS guarantees no underestimation, though overestimation may occur due to hash collisions.

In our setup, we used 5 hash functions and a width of 1,000, requiring a total of just 5,000 integer cells—about 20 KB of memory. We streamed the 100,000-token reservoir sample into the sketch and then queried the estimated frequencies of the most frequent words.

The average relative error across these terms was under 2%, demonstrating CMS's effectiveness for approximating high-frequency elements.

8.1 Discussion

The results confirm that Count-Min Sketch offers an excellent balance between accuracy and efficiency. Frequent words—those typically dominating resource usage in full-frequency tables—are estimated with low relative error.

Due to its compact representation, CMS is particularly well-suited for scenarios where memory is limited, data volume is high, or streaming constraints require online updates without storing full history. It naturally integrates into real-time analytics pipelines, edge computing environments, and big data summarization tasks.

As we demonstrate in the performance section, CMS achieves this with a memory footprint several orders of magnitude smaller than full aggregation, while delivering estimates that are sufficient for most statistical and analytical applications.

9 Cardinality Estimation with FM Sketch

While the Count-Min Sketch provided efficient estimates of token frequencies, it does not support set cardinality estimation. To approximate the number of unique words in the corpus, we applied the Flajolet–Martin (FM) sketch [2], a probabilistic method based on the position of leading zeros in binary hash values.

We implemented FM Sketch with 64 hashed bit-pattern registers. As each word from the reservoir sample was processed, the sketch updated its internal registers to track the maximum number of leading zeros seen per hash function. The number of unique words was then estimated based on the harmonic mean of these positions.

The results are summarized below:

- **True unique words:** 29,717
- **Estimated by FM Sketch:** 84,725
- **Relative Error:** 185.11%
- **Memory footprint:** 64 integers

Word	True	Estimated	Error %
the	30,292	30,396	0.34
and	14,832	14,886	0.36
you	11,660	11,732	0.62
was	10,832	11,090	2.38
her	10,296	10,508	2.06
she	9,661	9,769	1.12
that	8,428	8,617	2.24
his	8,029	8,215	2.32
had	5,044	5,116	1.43
with	4,717	4,890	3.67

Table 4: Top-10 word frequency estimates using Count-Min Sketch.

9.1 Discussion

Unlike CMS, which approximated frequency counts with high accuracy, the FM Sketch produced a substantial overestimate of cardinality—nearly triple the ground truth. This error likely stems from three main factors:

- **High token repetition:** Natural language text includes many repeated function words (e.g., `the`, `and`), leading to hash saturation.
- **Hash collisions:** The limited number of bits and registers increases the probability of multiple distinct tokens mapping to similar leading-zero patterns.
- **Sample sparsity:** The sketch was applied to a 100k sample from a much larger dataset, reducing coverage of the true vocabulary.

Although FM Sketch offers theoretical scalability and constant memory usage, its sensitivity to data skew makes it less reliable in textual domains with heavy frequency imbalance. In such settings, hyperloglog or alternative cardinality estimators may yield more stable performance.

10 Compression and Corpus Simplification

Beyond estimation, summarizing a dataset structurally can help reduce storage and accelerate downstream tasks. To this end, we applied two complementary techniques to the per-record word counts: histogram-based binning and wavelet-based signal compression.

10.1 Histogram Compression

The histogram of word counts, shown in Figure 6, reveals a highly non-uniform distribution. Most records contain between 5 and 20 words, with a steep drop-off for longer sequences.

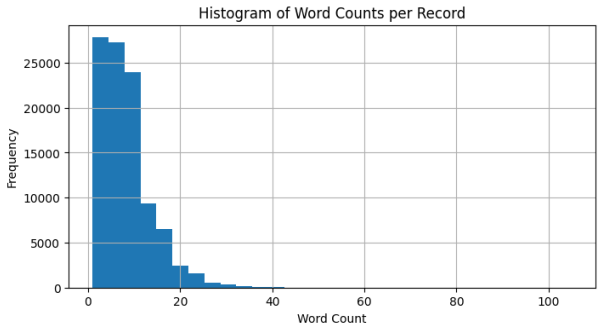

Figure 6: Histogram of word counts per record.

This distribution makes the corpus highly amenable to lossy compression via binning: long-tail values can be grouped into coarse ranges without substantial information loss. Such histogram-based quantization enables approximate storage, fast statistical queries, and indexing by range rather than exact value.

In downstream scenarios such as clustering, retrieval, or anomaly detection, histogram compression can serve as a compact and interpretable representation of document-level characteristics.

10.2 Wavelet-Based Compression

While histogram binning offers an interpretable summary of the word count distribution, it treats each record as an independent observation. To additionally capture trends and correlations within the sequence of document lengths, we applied wavelet-based signal compression.

Specifically, we used a 3-level Daubechies wavelet decomposition (db1) on the ordered sequence of per-record word counts. Wavelet transforms are well-suited for representing non-stationary signals with local variation, and have seen widespread use in compression and denoising tasks [3].

The decomposition produced the following compression:

- **Original vector size:** 100,000 values
- **Compressed (approximation):** 12,500 values
- **Compression ratio:** 0.125

Figure 7 compares the original signal with the reconstruction obtained from the compressed wavelet coefficients, over the first 500 records.

The reconstructed sequence successfully preserves the macro-patterns of the original signal, including regions of high and low variance, while eliminating fine-grained noise. This shows that wavelet compression is a viable tool for corpus-level simplification, especially in resource-constrained environments where full-resolution data storage is impractical.

Moreover, the resulting representation could serve as input to further analyses such as segmentation, clustering,

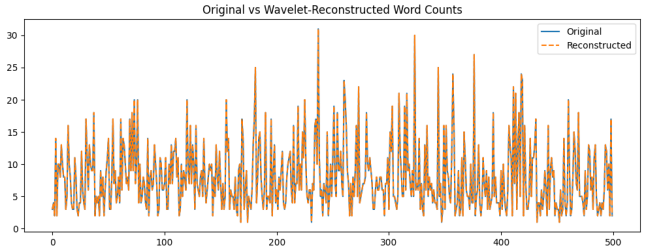

Figure 7: Original vs wavelet-reconstructed word count signal (first 500 records).

or trend detection—each benefiting from the compressed yet structure-preserving nature of the wavelet transform.

11 Performance Comparison

To consolidate the evaluation of the methods presented, Table 5 compares each approach in terms of execution time, estimation error, and memory usage. These results were obtained from Spark UI logs and empirical measurement during experimentation.

Method	Time	Error %	Memory
Full Spark	3 min 20 sec	0%	8 GB
5% Sample	43 sec	1–3%	1 GB
Reservoir (100k)	7 sec	5%	¡200 MB
Count-Min Sketch	¡1 sec	2%	20 KB
FM Sketch	¡1 sec	185%	64 integers

Table 5: Accuracy, runtime, and memory cost of each method.

The comparison clearly highlights the trade-offs between exact and approximate approaches. Full Spark processing ensures maximum accuracy but is significantly more resource-intensive. Sampling techniques offer considerable reductions in runtime and memory, while maintaining acceptable levels of error.

Among the sketch-based methods, Count-Min Sketch achieves a near-optimal balance between estimation accuracy and footprint size. On the other hand, FM Sketch—while extremely lightweight—suffers from poor cardinality estimation in the presence of repeated elements, which is common in natural language corpora.

Compression techniques such as histograms and wavelets do not directly estimate frequencies but effectively reduce data dimensionality for downstream use.

12 Conclusions and Future Work

This project explored a spectrum of scalable text analytics techniques on the BookCorpus dataset using Apache Spark. Beginning with exact word count analysis, we progressively incorporated approximation strategies to handle volume, velocity, and memory constraints common in real-world big data scenarios.

- **Sampling** (random and reservoir) enabled fast, low-cost approximations with minimal accuracy loss.
- **Count-Min Sketch** provided highly accurate frequency estimation with compact memory requirements, suitable for streaming.
- **FM Sketch** overestimated unique counts due to token repetition, underscoring the limits of cardinality sketches in textual domains.
- **Histogram and wavelet compression** reduced dimensionality while preserving core structure in the dataset.

Future Work

This pipeline can be extended in several directions:

- **Real-time integration:** Deploying sketches and samplers in live streaming systems using Spark Structured Streaming.
- **Semantic-aware compression:** Applying clustering or vector quantization to embeddings or topic models.
- **Multilingual analysis:** Adapting the pipeline for corpora in other languages and domains, especially low-resource settings.

Ultimately, our results validate that approximation techniques—when carefully chosen—can provide accurate, scalable, and interpretable analytics in modern large-scale NLP workflows.

References

[1] Graham Cormode and S Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[2] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[3] Stéphane Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 3 edition, 2009.

[4] Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[5] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.