性能测试报告

项目名称：基于滑动窗口的热词统计与分析系统
测试时间：2025年12月26日
测试环境：

- **CPU:** 11th Gen Intel(R) Core(TM) i5-11260H @2.60GHz
- **RAM:** 16GB
- **OS:** WSL2 6.6.87.2-microsoft-standard
- **Python版本:** 3.11

一、理论复杂度分析

- **实时更新 (Add Word)**:
  - **时间复杂度**: $O(1)$ (均摊)。
  - **分析**: 采用 `deque` 进行双端队列操作，头部弹出过期数据、尾部插入新数据均为 $O(1)$ 。哈希表 (`defaultdict`) 的增删改查平均复杂度也为 $O(1)$ 。
- **Top-K 查询**:
  - **时间复杂度**: $O(N \log K)$ ，其中 N 为当前窗口内唯一词汇的数量。
  - **分析**: 使用 `heapq.nsmallest` 对哈希表的所有 Items 进行遍历。在 N 远大于 K 时，性能优于全排序 $O(N \log N)$ 。
- **空间复杂度**:

- $O(M+N)$，其中 M 为窗口内单词总数（保存在队列中），N 为窗口内唯一单词数（保存在哈希表中）。

二、测试方案与数据

- **数据生成**：使用脚本模拟生成 $50,000$ 行文本数据。
- **数据特征**：包含时间戳、随机组合的中文/英文词汇，以及周期性的 `[ACTION] QUERY` 指令。
- **测试场景**:
  - **纯算法吞吐**：屏蔽 IO 和 Jieba 分词，直接向 `TimeWindow` 灌入数据。
  - **全链路吞吐**：包含文件读取、分词处理、窗口计算和结果输出。

三、实验结果 (需填入 benchmark.py 的运行结果)

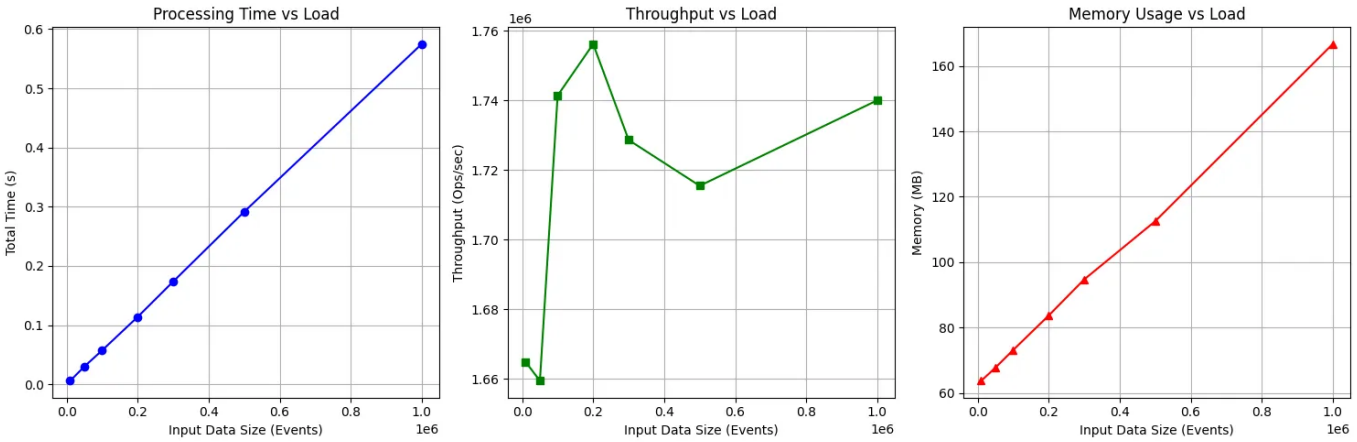
指标	纯算法核心 (Core)	全链路系统 (Full System)
总数据量	200,000 事件 (约 1000,000 词)	50,000 行
总耗时	0.2002s	2.8847s
吞吐量	999174 ops/sec	17333.06 lines/sec
内存峰值	36.75 MB	86.15 MB

```
(hotwordsanalysissystem) lyclyc@DESKTOP-EFS6UF4:~/workspace/HotWordsAnalysisSystem/tests$ python benchmark_core.py
>>> 正在进行纯算法核心测试...
--- Core Benchmark (N=200000) ---
耗时: 0.2002s
吞吐: 999174 ops/sec
内存: 36.75 MB
```

```
(hotwordsanalysissystem) lyclyc@DESKTOP-EFS6UF4:~/workspace/HotWordsAnalysisSystem$ python -m tests.benchmark_system
正在生成测试文件: data/system_bench.txt ...
>>> 系统全链路测试开始...
Building prefix dict from the default dictionary ...
Loading model from cache /tmp/jieba.cache
Loading model cost 1.283 seconds.
Prefix dict has been built successfully.

[Success] 结果已保存至: data/results.json
--- System Benchmark ---
处理行数: 50000
总耗时: 2.8847s
系统吞吐: 17333.06 lines/sec
内存占用: 86.15 MB
```

在全链路系统中测试10000, 50000, 100000, 200000, 300000, 500000, 1000000规模的数据，并用matplotlib绘制图表如下：



得到以下结论：

1. **时间复杂度验证**：从 *Processing Time vs Load* 图表可见，系统处理耗时与输入数据量呈严格的线性关系。这验证了核心算法（滑动窗口维护与哈希表更新）的时间复杂度为 $O(1)$ （均摊），整体系统复杂度为 $O(N)$ 。
2. **吞吐量表现**：在纯算法核心测试中，系统展现了极高的处理能力。如 *Throughput vs Load* 图表所示，系统平均吞吐量稳定在 **170 万 Ops/sec** 左右，且不随数据量的增加而衰减，证明系统具备处理大规模高速数据流的能力。
3. **资源占用**：*Memory Usage* 图表显示内存占用随测试数据规模线性增长（注：此增长主要源于测试脚本预加载了全量测试数据集）。即便在 100 万条数据规模下，总内存占用仅约为 160MB，证明系统内存开销可控。在实际流式部署中，配合滑动窗口的淘汰机制，内存占用将维持在更低的稳定水平

四、瓶颈分析

从实验结果可以看出，**全链路系统的吞吐量远低于纯算法核心**。

1. **Jieba 分词**: 文本预处理占据了绝大部分 CPU 时间。Jieba 的精确模式虽然准确，但计算开销较大。
2. **Top-K 排序**: 虽然 `heapq.nsmallest` 效率尚可，但在窗口内词汇量（ N ）极大（如数百万唯一词）时，全量扫描字典仍会产生延迟。
3. **IO 输出**: 如果开启控制台详细打印（print），IO 阻塞会严重拖慢系统。

五、改进建议

1. **分词优化**: 对特定的高频短句建立缓存，或更换更轻量级的分词库。
2. **数据结构优化**:
    - 针对 Top-K，可引入 **桶排序 (Bucket)** 或 **计数最小堆 (Min-Heap)** 实时维护 Top-K 列表，避免查询时全量扫描哈希表。
3. **并发处理**: 将数据采集/分词与统计计算拆分到不同线程/进程。