



## Lucene Performance Workshop

Thinking Lucene ▼ Think Lucid.

- 
- ▼ About the speaker and Lucid Imagination
  - ▼ Agenda
    - ▼ Lucene and performance
    - ▼ Lucid Gaze for Lucene: UI and API
    - ▼ Key statistics
    - ▼ Examples
    - ▼ Q & A session

## Lucene and performance

---

- ▼ Perceived performance issues can have different causes
- ▼ Classic JVM problems, classic solutions
  - ▼ heap size
  - ▼ garbage collection
  - ▼ stack size
  - ▼ HotSpot
- ▼ Lucene/Search-related issues: beyond JVM tuning
  - ▼ Indexing performance: indexing too slow, strange slowdowns during indexing
  - ▼ Search performance: search too slow in general, or for certain types of queries

- 
- ▼ Indexing:
    - ▼ Too many segments being created
    - ▼ Too many Token-s / TokenStream-s
    - ▼ Too many Documents / Fields
  - ▼ Searching:
    - ▼ Too many IndexReader-s / IndexSearcher-s
    - ▼ High RAM usage of IndexReader
    - ▼ Slow response times for certain queries
  - ▼ Application-level logging may not be up to the task
  - ▼ Profiler is too low-level and too intrusive
  - ▼ We need a lightweight probe to peek at vital Lucene statistics

# Lucid Gaze for Lucene (LG4L)

---

## Target audience and applications

- ▼ Tool for developers
- ▼ Performance monitoring
- ▼ Statistics collection
- ▼ Drop-in replacement for lucene-core-2.4.1.jar

## Statistics (per time unit):

---

- ▼ IndexReader / IndexWriter:
  - ▼ Number of documents and fields retrieved / created
  - ▼ Number of IndexWriter / IndexReader / Directory instances created
    - ▼ And the number of live instances!
  - ▼ Memory consumption of IndexWriter / IndexReader instances
- ▼ Analysis:
  - ▼ Number of Analyzers / TokenFilters / Tokenizers
  - ▼ Number of TokenStream-s and Token-s
- ▼ Search:
  - ▼ Number of searches and their average time
  - ▼ Number of opened IndexSearcher-s
- ▼ Storage:
  - ▼ Number of Lucene Directory instances created

## Lists and histograms

---

- ▼ Count and a list of `Analyzer`, `Tokenizer`, `TokenFilter` instances
- ▼ `Directory` implementations
- ▼ Top-N queries:
  - ▼ Queries with largest numbers of hits
  - ▼ Queries that took longest to execute
- ▼ All this data is available as log, persistent DB and through the API

## Retaining historical values of collected statistics

---

- ▼ In-memory
  - ▼ No files, no configuration hassles
  - ▼ Concise overview periodically written to log (optional)
  - ▼ Uses Java logging
- ▼ RRD (Round-Robin Database)
  - ▼ Persistent round-robin database
    - ▼ Single database of a constant size
    - ▼ E.g. hourly, daily, weekly, monthly, yearly statistics
  - ▼ Suitable for long-term monitoring
  - ▼ Many more metrics and statistics tracked
  - ▼ Can be accessed concurrently from other applications



## Java properties or gaze.properties

---

- ▼ List of properties supplied as `-Dlucid.gaze...`
- ▼ `gaze.properties` on classpath
- ▼ Configurability:
  - ▼ Turning on/off selected monitors
  - ▼ Producing debug output
  - ▼ Using in-memory or RRD log retention
  - ▼ Configuring RRD archives (to scale historical data over different periods)

## Facade with static methods: LuceneCore

---

- ▼ Programmatic access to all statistics groups
- ▼ Retrieve top-N queries
- ▼ Retrieving additional metrics (e.g. histograms of analyzers, tokenizers, directory implementations, tracking of IndexReader / IndexWriter instances and their memory consumption)
- ▼ Enabling / disabling monitors
- ▼ Resetting statistics (useful for creating snapshots)

## Based on the contrib/benchmark suite

---

- ▼ Test impact of number of buffered docs
- ▼ Other interesting observations
  - ▼ Number of documents / fields
  - ▼ Number of tokens / token streams
  - ▼ Number of IndexReader / Directory instances
  - ▼ Number of IndexSearchers

# Example: console output

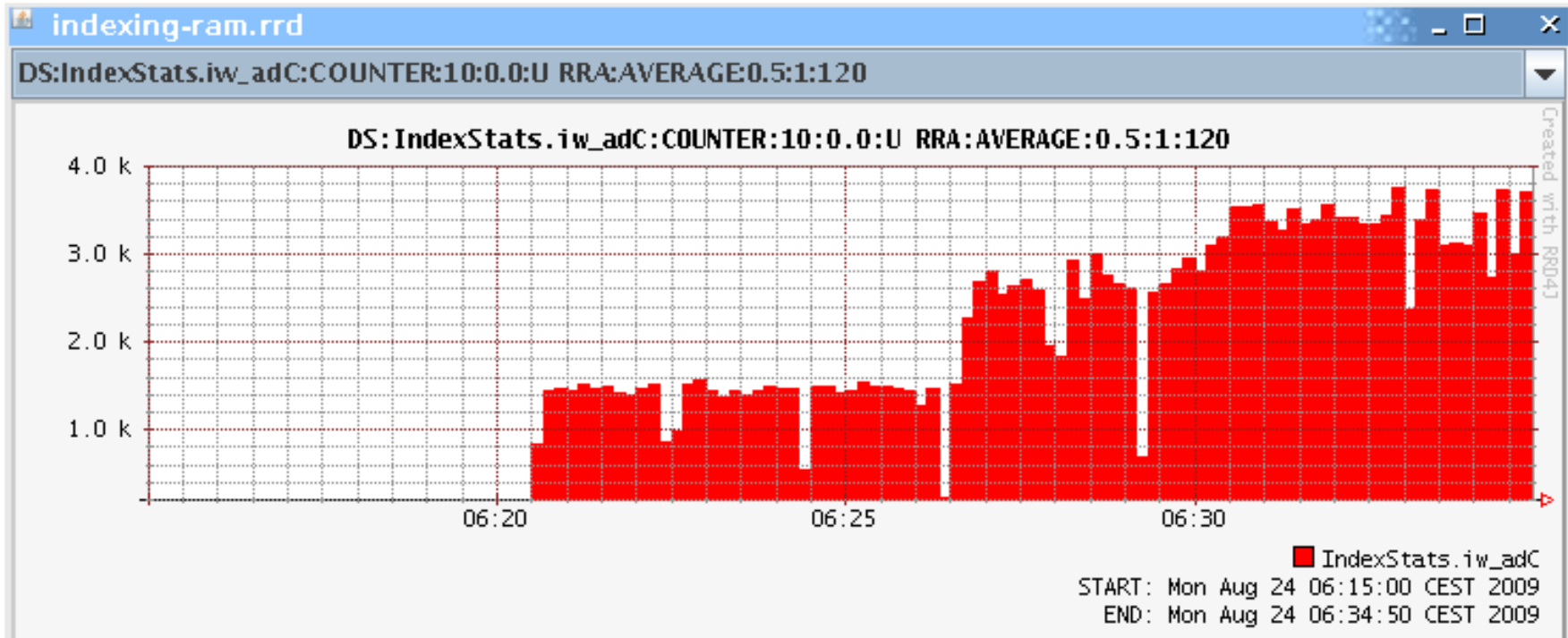
```
benchmark : mc
File Edit View Scrollback Bookmarks Settings Help
INFO: -----
INFO: * AnalysisStats:
INFO:   counters: {toks=134, tss=100511}
INFO:   metrics: {tns={StandardTokenizer=133}, tfs={StopFilter=133, LowerCaseFilter=133, StandardF
ilter=133}, ans={org.apache.lucene.analysis.standard.StandardAnalyzer=3}}
INFO: * DocumentStats:
INFO:   counters: {docs=2241218, fields=10910610}
INFO: * IndexStats:
INFO:   counters: {ir_isdC=4220546, iw_mrgT=544019957, ir_C=75, iw_C=0, ir_newC=3674, ir_tpC=25334
, iw_segs=0, iw_adC=20671, iw_segC=2291, iw_ram=0, iw_adT=1313002, iw_mrgC=223, ir_tC=2978, iw_buf=0,
ir_tdC=709940, iw_newC=2, ir_ram=4714772}
INFO: * SearchStats:
INFO:   counters: {dfC=251764, rwrC=11350, rwrT=39560, srchrC=22700, srchT=927905, srchC=22700}
INFO: * StoreStats:
INFO:   counters: {dirC=7354}
INFO:   metrics: {dir_t={CompoundFileReader=7348, FSDirectory=6}}
INFO: * Top largest: [1138,body:japan body:sony, 216,body:"world bank" -body:nigeria, 170,body:salomon
, 37,spanFirst(body:ford, 5), 31,body:comex, 16,spanNear([body:night, body:trading], 4, false), 15,bod
y:"ford credit"~5, 7,spanNear([spanFirst(body:ford, 10), body:credit], 10, false), 5,+body:"world bank
"^2.0 +body:nigeria, 1,body:"food needs"~3, 0,body:"sony japan"]
INFO: * Top slowest: [485038,body:"sony japan", 73176,body:comex, 24255,body:comex, 24255,body:comex,
24218,body:comex, 24218,body:comex, 24183,body:comex, 24131,body:comex, 24034,body:comex, 23941,body:c
omex, 23867,body:comex, 23806,body:comex]
```

# Example: RRD Inspector

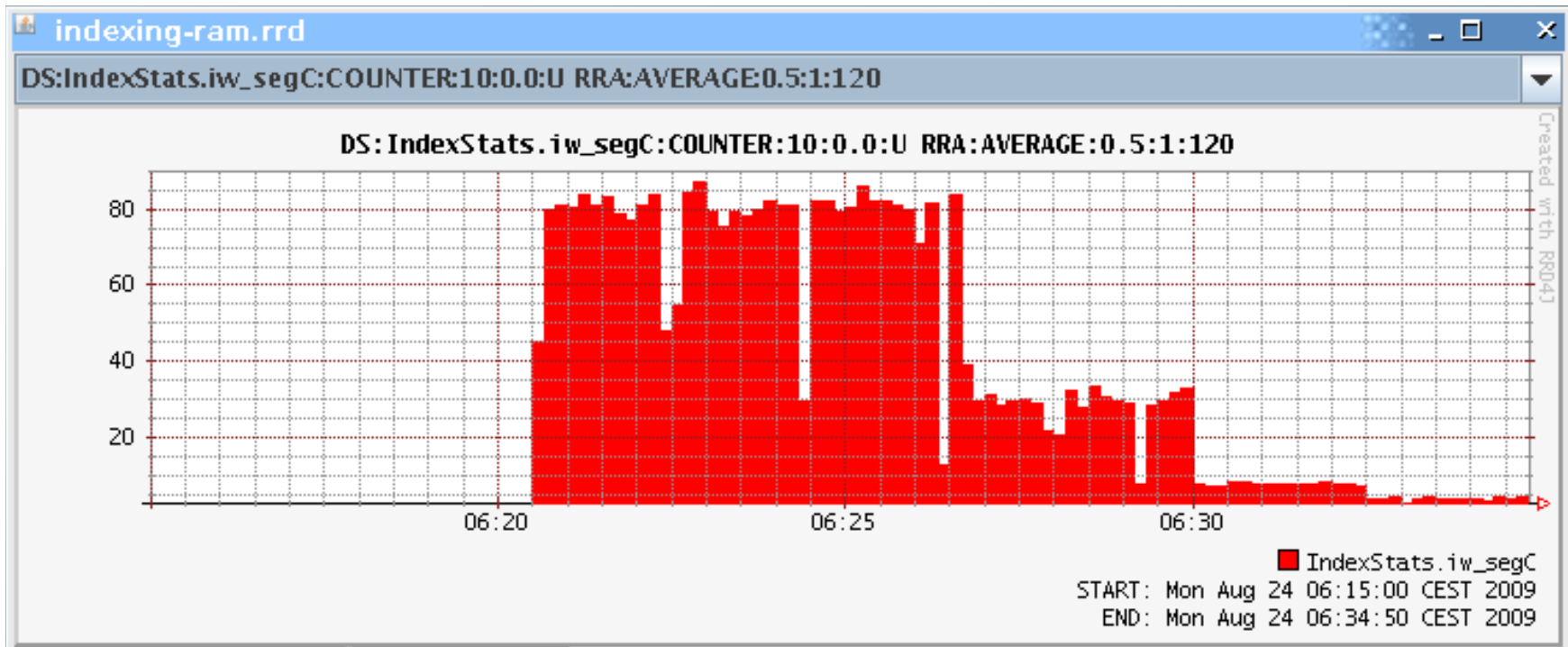
The screenshot shows the RRD File Inspector application window. The title bar reads "RRD File Inspector". The menu bar contains "File" and "Help". The left pane shows a tree view of the file "indexing-ram.rrd" with various data sources (DS) and retention aggregation (RRA) files. The right pane has four tabs: "General info", "Datasource info", "Archive info", and "Archive data". The "Archive info" tab is selected, displaying a table with the following data:

description	value
consolidation	AVERAGE
xff	0.5
steps	1
rows	120
accum. value	NaN
NaN steps	0
start	1251087300 [Mon Aug 24 06:15:00 CEST 2009]
end	1251088490 [Mon Aug 24 06:34:50 CEST 2009]

# RRD Inspector (2)



# RRD Inspector (3)



# Performance impact of performance monitoring

## Overhead of using LG4L

---

- ▼ Benchmarks (in contrib/benchmark) slower by ~10-15% on average, memory consumption higher by ~10%
- ▼ Remember: you can turn off some monitors!



# Conclusions

- 
- ▼ Lucene can perform fantastically
    - ▼ ... but it can't outmaneuver sub-optimal design or weak configuration
  - ▼ LG4L helps to understand the causes of poor performance
    - ▼ Insight into high-level statistics that relate to Lucene API
    - ▼ Round-robin database for tracking historical data
    - ▼ LG4L is lightweight!

# Q & A

## Download and documentation:

▼ <http://www.lucidimagination.com/Downloads/LucidGaze-for-Lucene>

# Example: LG4L with Solr

```
INFO: * AnalysisStats:
INFO:   counters: {toks=258}
INFO:   metrics: {tns={=2, WhitespaceTokenizer=8},
tfs={WordDelimiterFilter=8, StopFilter=8, SynonymFilter=8, LowerCaseFilter=8,
EnglishPorterFilter=8},
ans={org.apache.solr.schema.IndexSchema$SolrQueryAnalyzer=1,
org.apache.solr.analysis.TokenizerChain=6,
org.apache.solr.schema.FieldType$DefaultAnalyzer=13,
org.apache.lucene.analysis.WhitespaceAnalyzer=1,
org.apache.solr.schema.IndexSchema$SolrIndexAnalyzer=1}}
INFO: * DocumentStats:
INFO:   counters: {docs=1, fields=20}
INFO: * IndexStats:
INFO:   counters: {ir_isdC=1, ir_C=4, iw_C=0, ir_newC=7, ir_tpC=12,
iw_segs=0, iw_buf=0, ir_tdC=10, ir_ram=1343504, iw_ram=0}
INFO: * SearchStats:
INFO:   counters: {dfC=11, rwrC=3, rwrT=30265, srchrC=14, srchT=90133076,
srchC=6}
INFO: * StoreStats:
INFO:   counters: {dirC=8}
INFO:   metrics: {dir_t={FSDirectory=8}}
```

**... but of course you should use LucidGaze for Solr instead!**