Performance Profiling Guide

Version: 1.0
Date: 2026-01-07

Overview

The Contest Log Analyzer includes built-in performance profiling capabilities to help identify bottlenecks and measure execution times of critical operations. This feature is controlled by the `CLA_PROFILE` environment variable.

Enabling Profiling

Windows (setup.bat)

Profiling is already enabled in `setup.bat`:

```
set CLA_PROFILE=1
set CONTEST_INPUT_DIR=CONTEST_LOGS_REPORTS
set CONTEST_REPORTS_DIR=C:\Users\mbdev\Hamradio\CLA
activate cla
```

Manual Activation

To enable profiling manually:

Windows:

```
set CLA_PROFILE=1
```

Linux/Mac:

```
export CLA_PROFILE=1
```

Docker/Web Application

Add to your `docker-compose.yml`:

```
environment:
  - CLA_PROFILE=1
  - CONTEST_INPUT_DIR=/app/CONTEST_LOGS_REPORTS
  - CONTEST_REPORTS_DIR=/app/reports
```

Note: The `docker-compose.yml` in this project has been updated to include `CLA_PROFILE=1` by default. You'll need to **restart the Docker container** for the profiling to become active:

```
docker-compose down
docker-compose up --build
```

Disabling Profiling

To disable profiling (for normal production use):

Windows:

```
set CLA_PROFILE=0
```

Linux/Mac:

```
unset CLA_PROFILE
# or
export CLA_PROFILE=0
```

Reading Profiling Output

When `CLA_PROFILE=1`, timing information is logged to the console with a stopwatch emoji ( ) prefix:

Example CLI Output

```
INFO -    [Pre-flight Validation] completed in 0.342s
INFO -    [CTY File Resolution] completed in 0.089s
INFO -    [Individual Log Loading - K3LR.log] completed in 2.156s
INFO -    [Cabrillo Data Ingestion] completed in 1.823s
INFO -    [Individual Log Loading - N2IC.log] completed in 1.987s
INFO -    [Cabrillo Data Ingestion] completed in 1.654s
INFO -    [Log Batch Loading (Total)] completed in 4.623s
INFO -    [Master Time Index Creation] completed in 0.045s
INFO -    [Time-Series Score Pre-calculation] completed in 0.312s
INFO -    [Finalize Loading (Total)] completed in 0.891s
INFO -    [Report Generation (All Reports)] completed in 8.567s
```

Example Web Application Output

```
INFO -    [Web - Log Loading] completed in 5.234s
INFO -    [Log Batch Loading (Total)] completed in 5.012s
INFO -    [Finalize Loading (Total)] completed in 0.723s
INFO -    [Web - Report Generation] completed in 9.123s
INFO -    [Report Generation (All Reports)] completed in 9.045s
INFO -    [Web - Dashboard Aggregation] completed in 0.456s
INFO -    [Web Analysis Pipeline (Total)] completed in 15.234s
```

Profiled Sections

The following code sections are instrumented for profiling:

LogManager (`contest_tools/log_manager.py`)

- **Log Batch Loading (Total)** - Complete batch loading process
- **Pre-flight Validation** - Header parsing and consistency checks
- **CTY File Resolution** - CTY.DAT file selection
- **Individual Log Loading** - Per-log processing (one entry per log)
- **Finalize Loading (Total)** - Master time index and CSV exports
- **Master Time Index Creation** - Cross-log time synchronization
- **Time-Series Score Pre-calculation** - Hourly score computation

ContestLog (`contest_tools/contest_log.py`)

- **Cabrillo Data Ingestion** - File parsing and validation per log

ReportGenerator (`contest_tools/report_generator.py`)

- **Report Generation (All Reports)** - Complete report generation cycle

Web Pipeline (`web_app/analyzer/views.py`)

- **Web Analysis Pipeline (Total)** - Complete web request processing
- **Web - Log Loading** - Log parsing in web context
- **Web - Report Generation** - Report generation in web context
- **Web - Dashboard Aggregation** - Dashboard data preparation

Performance Optimization Workflow

1. **Establish Baseline**

   ```
   # Run analysis with profiling enabled
   python main_cli.py K3LR.log N2IC.log --report all
   ```

2. **Identify Bottlenecks** Look for sections with the highest execution times in the output.

3. **Focus Optimization Efforts**

   - Times > 5s: High priority for optimization
   - Times 1-5s: Medium priority
   - Times < 1s: Low priority (unless called frequently)

4. **Common Bottlenecks**

   - **Pre-flight Validation**: May indicate redundant file I/O
   - **Cabrillo Data Ingestion**: Check for slow DataFrame operations
   - **Report Generation**: May benefit from parallel execution
   - **Time-Series Score Pre-calculation**: Consider caching strategies

Interpreting Results

Good Performance Indicators

- Pre-flight Validation: < 0.5s per log
- Individual Log Loading: < 3s per log (depends on log size)
- Finalize Loading: < 1s total
- Report Generation: < 10s for 'all' reports

Performance Red Flags

- Pre-flight Validation: > 2s (indicates I/O issues)
- Individual Log Loading: > 10s per log (parsing inefficiency)
- Report Generation: > 30s (may need parallelization)

Technical Details

Implementation

The profiling system uses two components from `contest_tools/utils/profiler.py`:

1. **`@profile_section(name)` decorator** - For instrumenting functions
2. **`ProfileContext(name)` context manager** - For instrumenting code blocks

Both check the `CLA_PROFILE` environment variable and only add timing overhead when profiling is enabled.

Performance Impact

When `CLA_PROFILE=0` or unset, the profiling code has **zero performance impact** - the decorators and context managers detect the disabled state and short-circuit immediately without any timing operations.

When `CLA_PROFILE=1`, the overhead is negligible (< 0.1ms per timed section) due to the use of `time.perf_counter()`, Python's high-resolution timer.

Troubleshooting

No Profiling Output Appears

1. Verify environment variable is set: `echo %CLA_PROFILE%` (Windows) or `echo $CLA_PROFILE` (Linux/Mac)
2. Ensure value is exactly `1` (not `true`, `yes`, or other values)
3. Check that logging level is INFO or lower

Profiling Slows Down Analysis

This should not happen - profiling overhead is < 0.1%. If you experience slowdowns:

1. Disable profiling: `set CLA_PROFILE=0`
2. Report the issue (this indicates a bug in the profiling code)

Future Enhancements

Planned improvements to the profiling system:

- JSON output format for automated performance tracking
- Aggregated statistics across multiple runs
- Memory profiling in addition to time profiling
- Integration with web dashboard for real-time monitoring
- Per-report timing breakdown

See Also:

- `Docs/ProgrammersGuide.md` - Architecture and design patterns
- `contest_tools/utils/profiler.py` - Profiling implementation