NanoLog: A Nanosecond Scale Logging System

武新楠

M201877271

HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Overview

- **Implemented a fast C++ Logging System**
  - NanoLog是一种新的日志系统,比现有的解决方案快1-2个数量级
  - 像Log4j2一样,保持了类似printf的API
  - 吞吐量高达每秒8000万个日志消息
- **Shifts work out of the runtime hot-path**
  - 将大部分工作从程序运行时转移到程序编译和执行后阶段
  - 通过编译时提取静态内容来减少用户日志调用
  - 重写日志代码仅在运行时发出动态信息
  - 依赖后处理器来重新整理日志

What is Logging?

- **printf-like messages with dynamic information**
  - Written at development time; output at runtime
  - Developer specifies severity, log message + dynamic data
  - Logging system inserts invocation date/time, file location

main.cc

```
1  void main() {
2    LOG(NOTICE, "The task began");
3    float result = doSomething();
4
5    LOG(NOTICE,
6      "The result was %4.2f", result);
```

Date/Time File Location Severity

```
2017-11-17 21:35:16 main.cc:3 NOTICE[4]: The task began
2017-11-17 21:38:29 main.cc:5 NOTICE[4]: The result was 12.00
```

Context (Implicit) Message (explicit)

What makes logging slow?

```
1473057128.133777014 src/LogCleaner.cc:826 in TombstoneRatioBalancer
NOTICE: Using tombstone ratio balancer with ratio = 0.400000
```

- **Compute: Complex Formatting**
  - Loggers need to provide context (i.e. file location, time, severity, etc)
  - The message above has **7 arguments** and takes **850ns** to compute

- **I/O Bandwidth: Disk IO**
  - On a 250MB/s disk, the **129 byte** message above takes **500ns** to output!

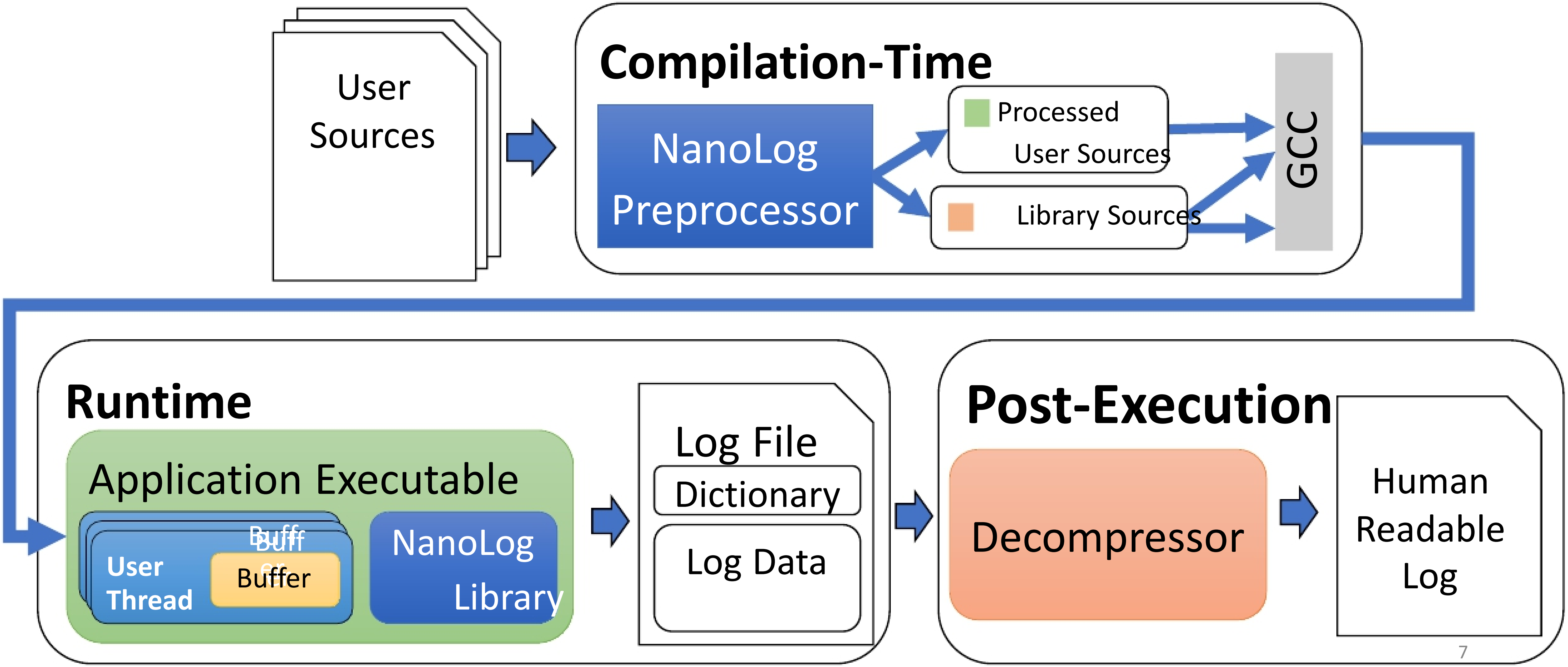
Solutions

```
1473057128.133777014 src/LogCleaner.cc:826 in TombstoneRatioBalancer
NOTICE: Using tombstone ratio balancer with ratio = 0.400000
```

- **Compute: Defer formatting to an offline process**
  - Developers only look at a small portion of the log
  - Seems wasteful to output a human-readable log eagerly

- **I/O Bandwidth: De-duplicate Static Information**
  - Log messages contain a lot of information known at compile-time
    - i.e. file location, line #, function, severity, format string
  - Logging only the dynamic information in binary saves I/O
    - Shrinks the 129 byte message above to just 16 bytes

NanoLog System Architecture



Compile-time Optimizations(编译过程)

User Source (main.cc)

```
#include "NanoLog.h"

main() {
  NANO_LOG("Hello World %d", rand());
}
```

(a) Inject optimized log code

Post-Processed User Source (main.ii)

```
#include "NanoLog.h"

# 1 "generated.h"
inline void
record_main_4(const char *str, int arg1) {
  extern int uniqueId_main_cc_4;

  NanoLog::push(uniqueId_main_cc_4);
  NanoLog::push(time());
  NanoLog::push(arg1);
}

# 2 "main.cc"
main() {
  record_main_4(
        "Hello World %d", rand());
}
```

(b) Extract static log info into a dictionary

NanoLog Library (Dictionary Entry)

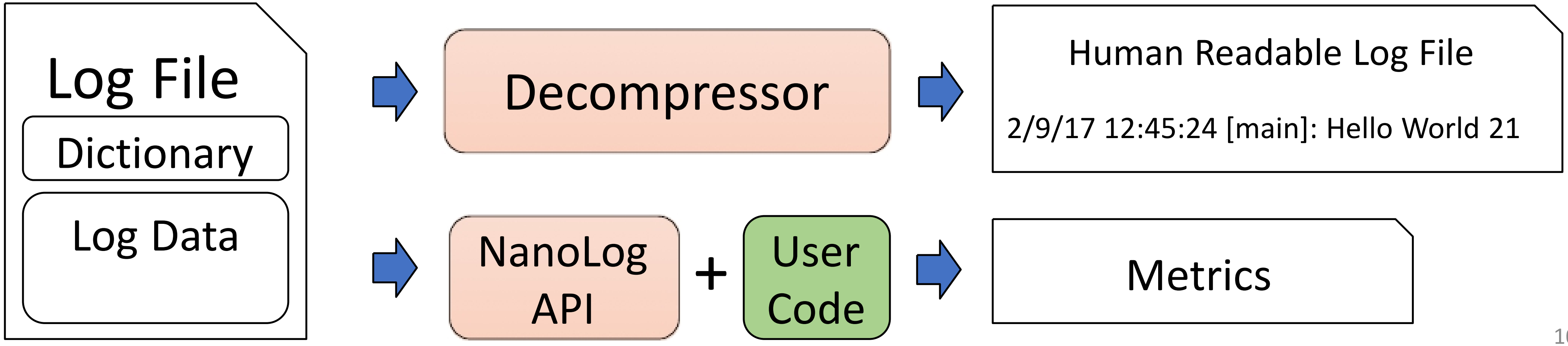
```
// Maps log instance to id
extern int uniqueId_main_cc_4 = 0;

writeDictionaryEntry(out) {
  // Save static information
  const char* fmtString = "Hello World %d";
  const char* filename  = "main.cc";
  const int linenum     = 4;
  enum severity         = "NOTICE"

  out << uniqueId_main_cc_4;
  out << fmtString;
  ...
}
```

compile

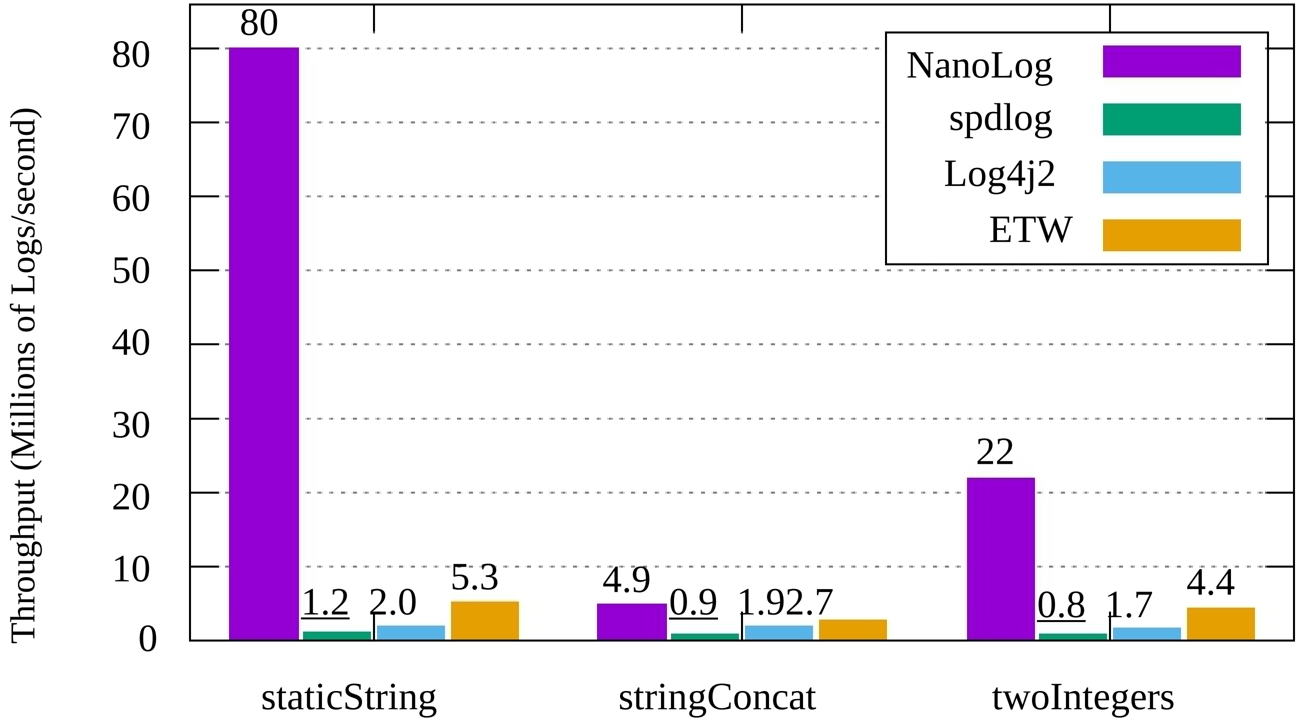
Application Executable

Decompressor(解压缩器)

- **Reconstitutes the log data by combining static + dynamic info**
- **Stand-Alone**
  - Reads the log file and produces a full human-readable log file.
  - Ultimately pays the formatting + output costs of the full log
- **Programmatic API (for fast aggregation)**
  - Process the log messages one by one without formatting.
  - More efficient than processing data in ASCII



Microbenchmark - Throughput



Evaluation

- Repeated logged 1 message with no inter-log delay in a thread
- Varied the number of logging threads to maximize throughput

Conclusions

- NanoLog always faster
- Even better with fewer dynamic arguments.

ID	ExampleOutput	NLMsgSize
staticString	Startingbackupreplicagarbagecollectorthread	3-4Bytes
stringConcat	Openedsessionwithcoordinatoratbasic+udp:host=192.168.1.140,port=12246	43Bytes
twoIntegers	bufferhasconsumed1032024bytesofextrastorage,currentallocation:1016544bytes	10Bytes

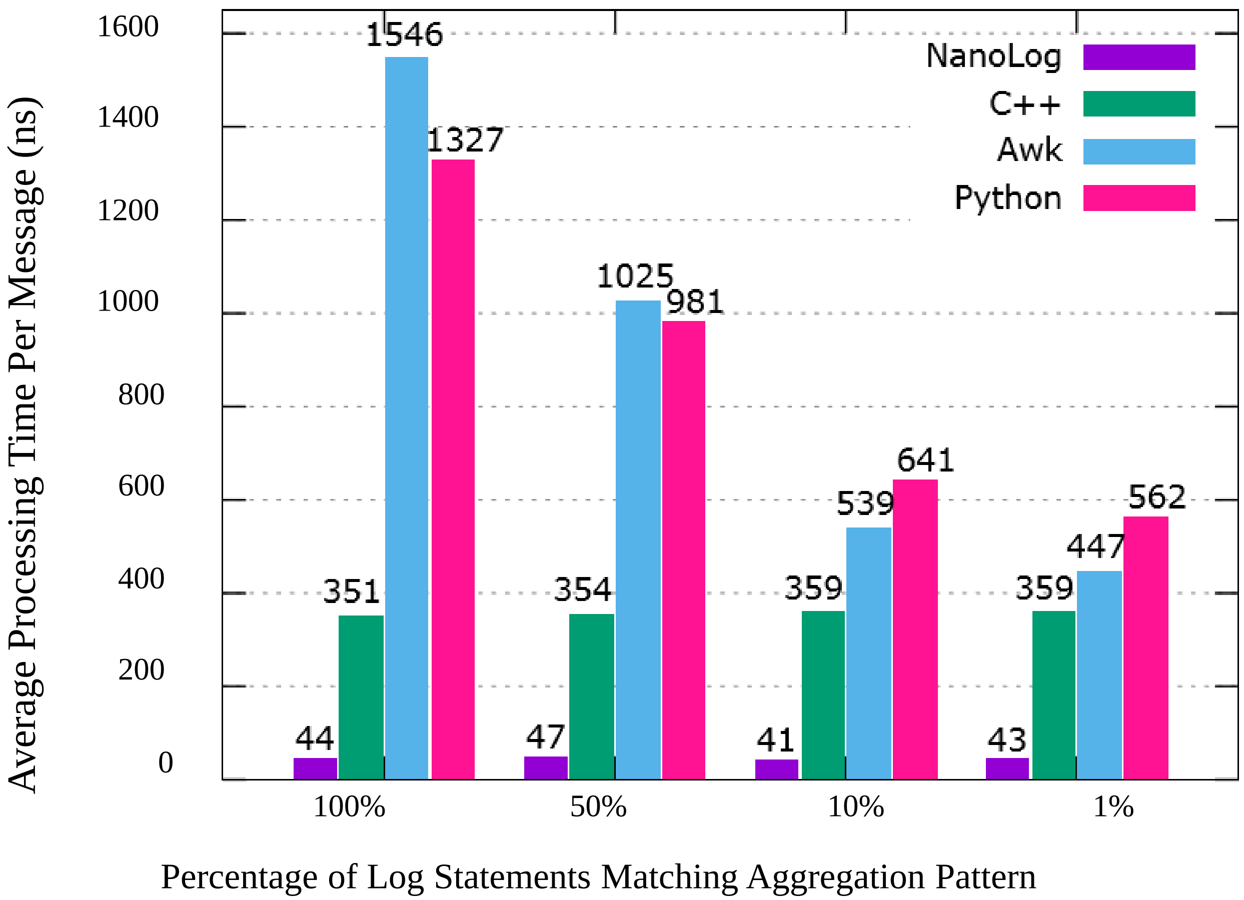
underlined indicates dynamic data

Microbenchmark - Unloaded Latency

- **Goal: Measure minimum processing delay**
- **Setup:**
  - 100,000 iterations of logging 1 log message with a 650ns delay between each to eliminate blocking due to I/O
  - Percentile times reported below in nanoseconds

System	NanoLog	spdlog	Log4j2	ETW		NanoLog	spdlog	Log4js	ETW
LogMsgs	MedianLatency(ns)					99.9PercentileLatency(ns)			
staticString	**8**	230	192	180		**33**	473	1868	726
stringConcat	**8**	436	230	208		**33**	1614	6171	2954
twoIntegers	**7**	674	160	200		**44**	1335	1992	761

NanoLog For Analytics



Mock Analytics Workload

1. Find all instances of "Hello World # %d"
2. Interpret "%d" as an integer and compute the min/mean/max

Configurations

- **NanoLog:** Used the compact, binary log
- **C++**: Used *atoi()* on the full log
- **Python, Awk**: Use regex on the full log

NanoLog Analytics Benefits:

- Smaller log files (~747MB vs. 7.6GB)
- Binary format (no ASCII parsing req.)

Conclusion

- **NanoLog’s Techniques**

  总之,NanoLog是一种新的日志系统,比现有的日志系统快1-2个数量级。它通过使用预处理器在编译时删除静态日志信息，注入优化代码,仅记录运行时的动态信息，并将日志消息的格式化推迟到程序执行后来加速日志记录。

- **Benefits:**

该系统的好处在于他运行时非常高效，并且可以在执行后提供快速聚合。唯一需要注意的是，它可能不会立即以人类可读的格式显示日志。

Thanks!