lprof : A Non-intrusive Request Flow Profiler for Distributed Systems

Xu Zhao, *Yongle Zhang*, David Lion, Muhammad FaizanUllah, Yu Luo, Ding Yuan, Michael Stumm

Oct. 8th 2014

UNIVERSITY OF TORONTO

Performance analysis tools are needed

- Poor performance of distributed systems leads to
  - Increase of user latency
  - Increase of data center cost

- Distributed system behavior is hard to understand
  - Concurrent requests being processed by multiple nodes

- To diagnose poor performance, tools are needed to
  - Reconstruct the request control flow
  - Understand system behavior

Existing tools are intrusive

- Instrument systems to infer request control flow
  - *E.g. MagPie, Project 5, X-Trace, Dapper, etc.*
  - Incur performance overhead
  - Instrumentations are often system specific

System logs contain rich information

- Rich information in logs is not coincidence
  - Developers rely on logs to perform manual debugging

- Distributed systems generate lots of logs
  - During normal execution

NetApp 7TB/month [Cloudera'13]

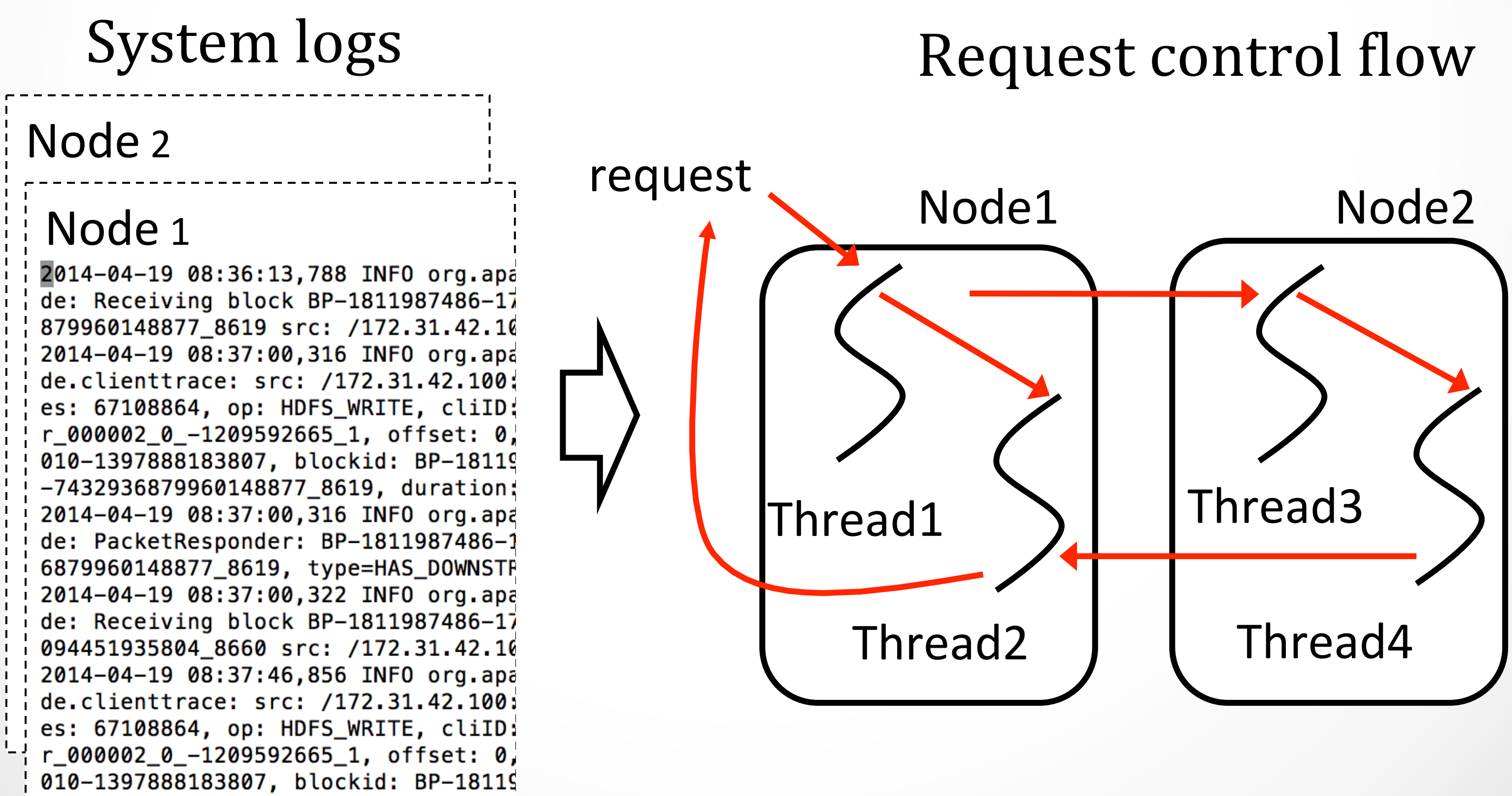
facebook 25TB/day [Rothschild'09]

Existing log analyzers are limited

- Cannot infer request control flow
  - Machine learning based log analyzers
    - *E.g. [Xu'09], DISTALYZER, Synoptic, etc.*
    - Only detect system anomalies
  - Commercial tools
    - *E.g. splunk, VMWare LogInsight*
    - Require users to perform key-word based searches

lprof: a non-intrusive profiler

- Infers request control flow from system logs
  - Along with timing information
  - Group logs printed by the same request on multiple nodes
  - *Use information generated by static analysis*

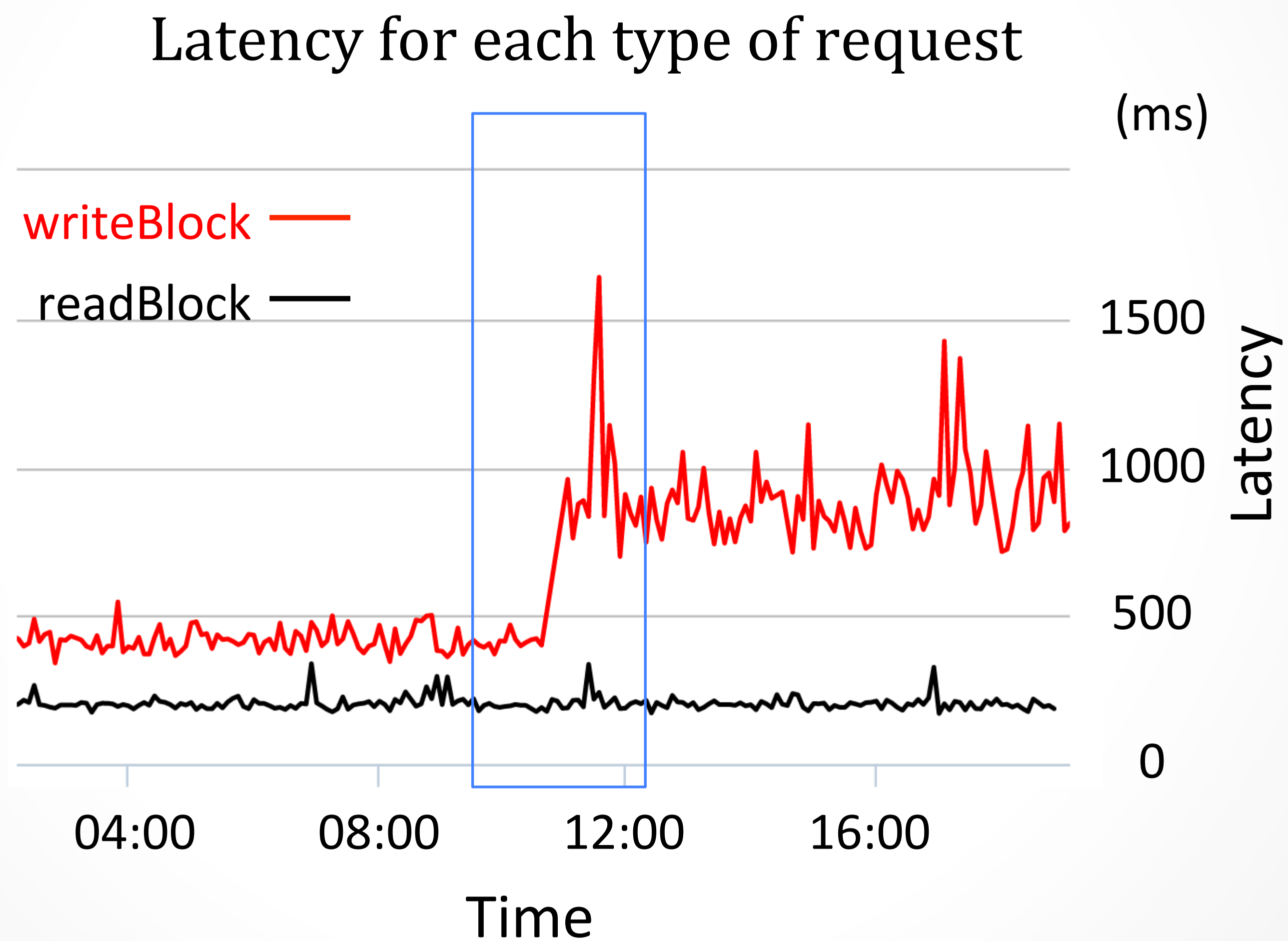


Outline

- Introduction
- Case Study
- Design
- Evaluation

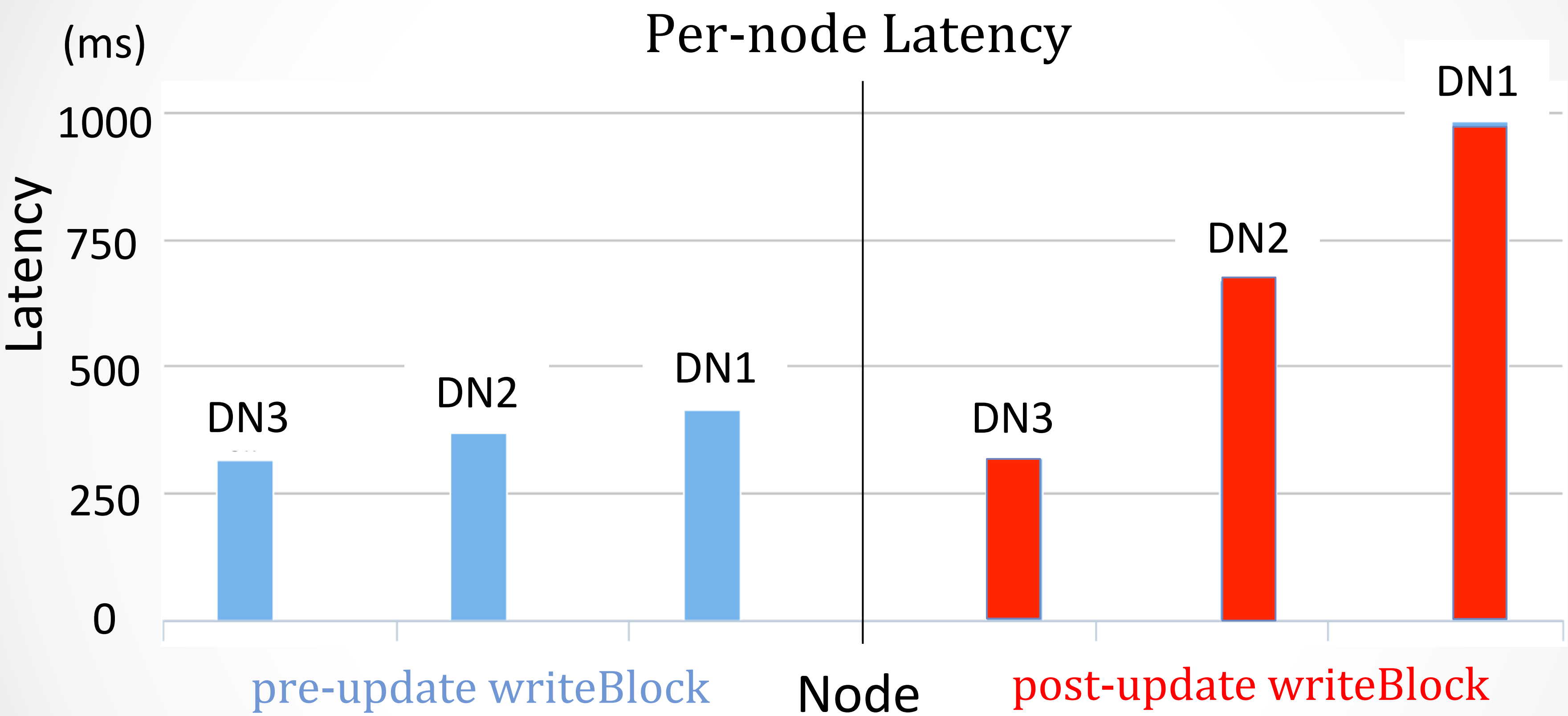
A real-world example

- Performance regression – HDFS-4049



- writeBlock is suspecious

Zoom into per-node latency

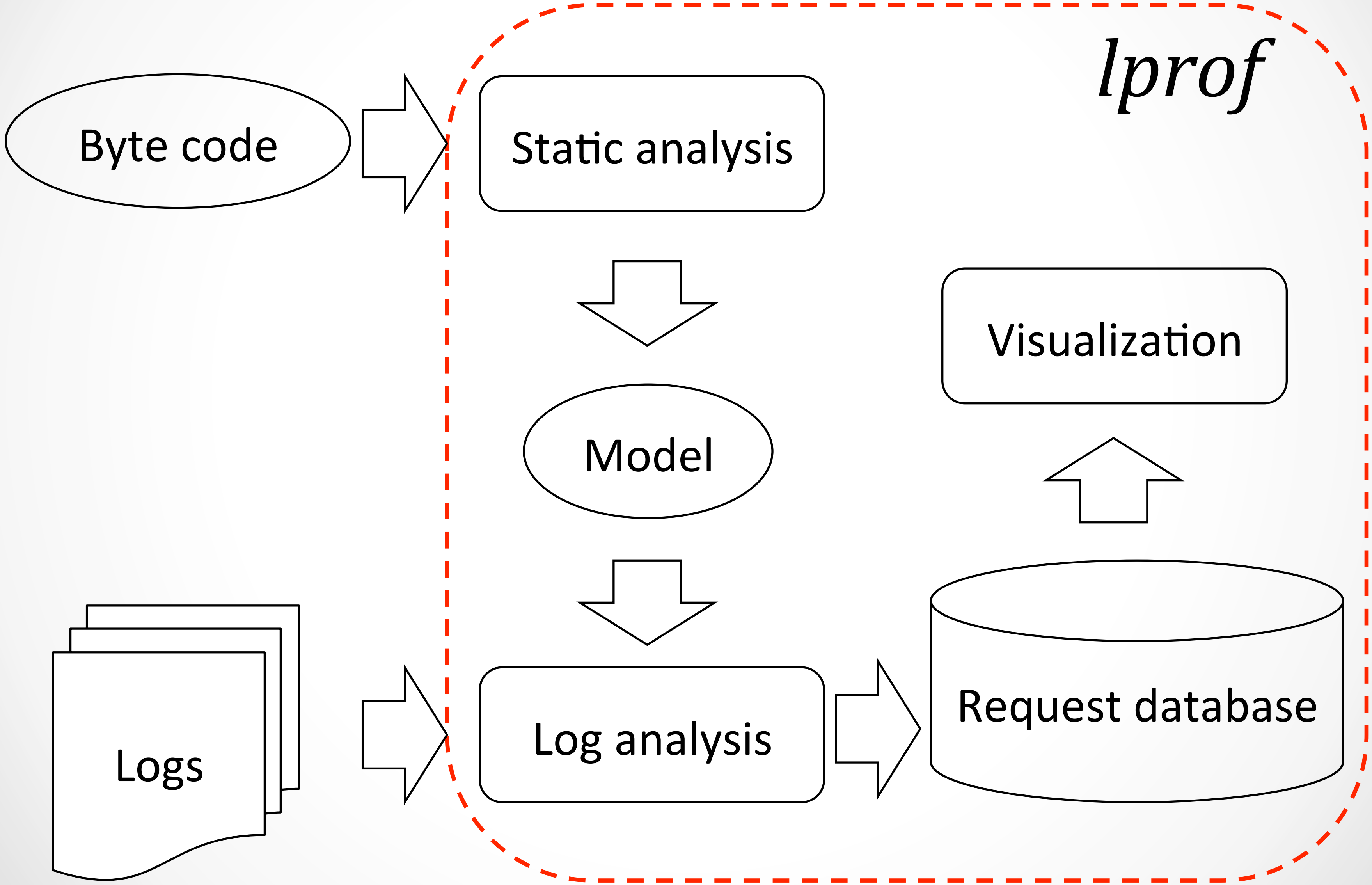


- Intra-node latency doesn't increases while inter-node does
- Conclusion: unnecessary network communication

Outline

- Introduction
- Case Study
- Design
- Evaluation

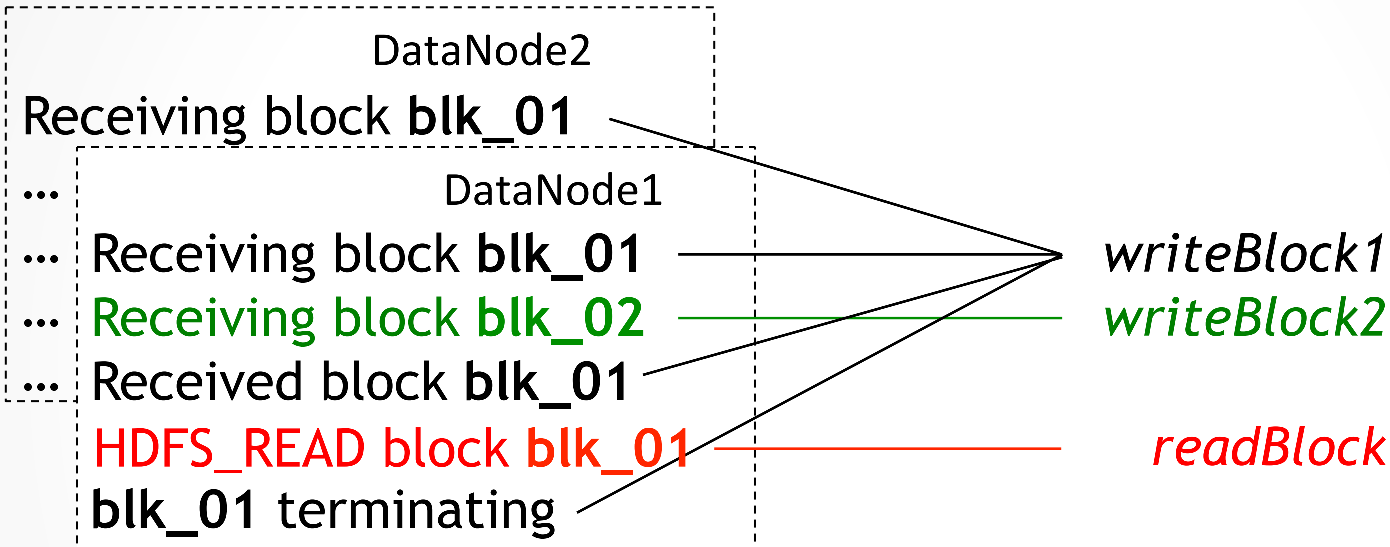
Overview



lprof
Byte code
Static analysis
Model
Visualization
Logs
Log analysis
Request database

Challenges

- Goal: to stitch log messages with respective requests



log snippet from HDFS data nodes

- Logs are interleaved
  - From different request types
  - From different request instances of the same type
- Perfect identifiers doesn't always exist
- Distributed across multiple nodes

Code snippet in HDFS

```
dataXCeiver() {
  switch(opCode){
    case WRITE_BLOCK:
      blk_id = getBlock();
      writeBlock(blk_id, …);
      break;
    case READ_BLOCK:
      blk_id = getBlock();
      readBlock(blk_id, …);
      break;
  }
}
```

```
writeBlock(blk_id, …) {
  log("Receiving block " + blk_id);
  new PacketResponder().start();
}
readBlock(blk_id, …) {
  log("HDFS_READ block " + blk_id);
}
PacketResponder.run() {
  log("Received block " + blk_id);
  …
  log(blk_id + " terminating");
}
```

- Top level method - starting method to process a request
- Request identifier - logged variable not modified in one request
- Log temporal order - possible order between log statements
- Communication behavior - communication between threads

Request analysis

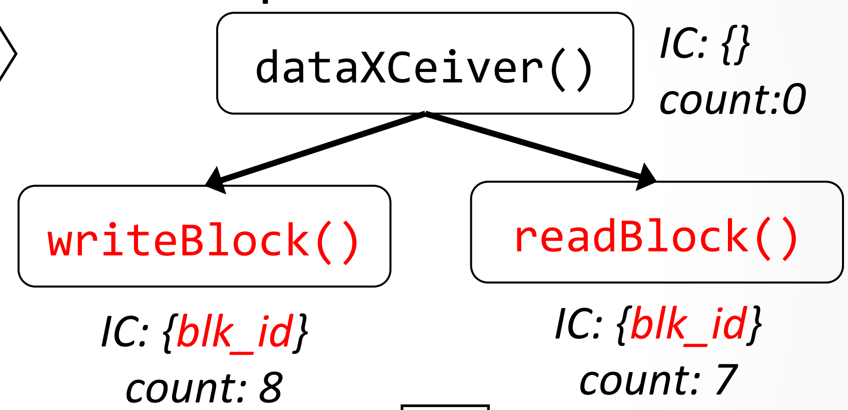
- Find top level method
- Find request identifiers
- Intuition
  - Request identifiers already exist for manual debugging
  - Not modified within one request
  - Once modified, outside of the request

Request analysis example

- Bottom-up analysis on call graph
  - Logged variables – identifier candidates (IC)
  - Number of times they got printed – count

```
dataXCeiver () {
  switch(opCode){
    case WRITE_BLOCK:
      blk_id = getBlock();
      writeBlock(blk_id, …);
      break;
    case READ_BLOCK:
      blk_id = getBlock();
      readBlock(blk_id, …);
      break;
  }
}
```

Call Graph



top level method	*identifiers*
writeBlock	blk_id
readBlock	blk_id

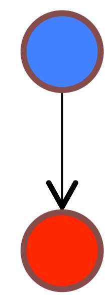
- Once count decreases, pick top level method and identifier

Temporal order analysis

- Control flow analysis in each top level method

temporal order

```
PacketResponder.run() {
  log("Received block " + blk_id);
  …
  log(blk_id + " terminating");
}
```



Communication pair analysis

- Communication between request top level methods
  - Intra-node: thread creation, shared objects
  - Network: socket, RPC
    - Pair serializing and de-serializing methods

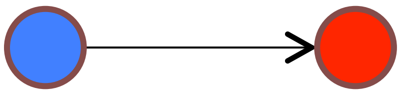
Summary of static analysis output

- Top level method & request identifier

top level method	*identifiers*
`writeBlock()`	blk_id
`readBlock()`	blk_id

- Log temporal order

`PacketResponder.run()`



- Communication pair

`writeBlock()` → `writeBlock()`

type: Network

id: blk_id

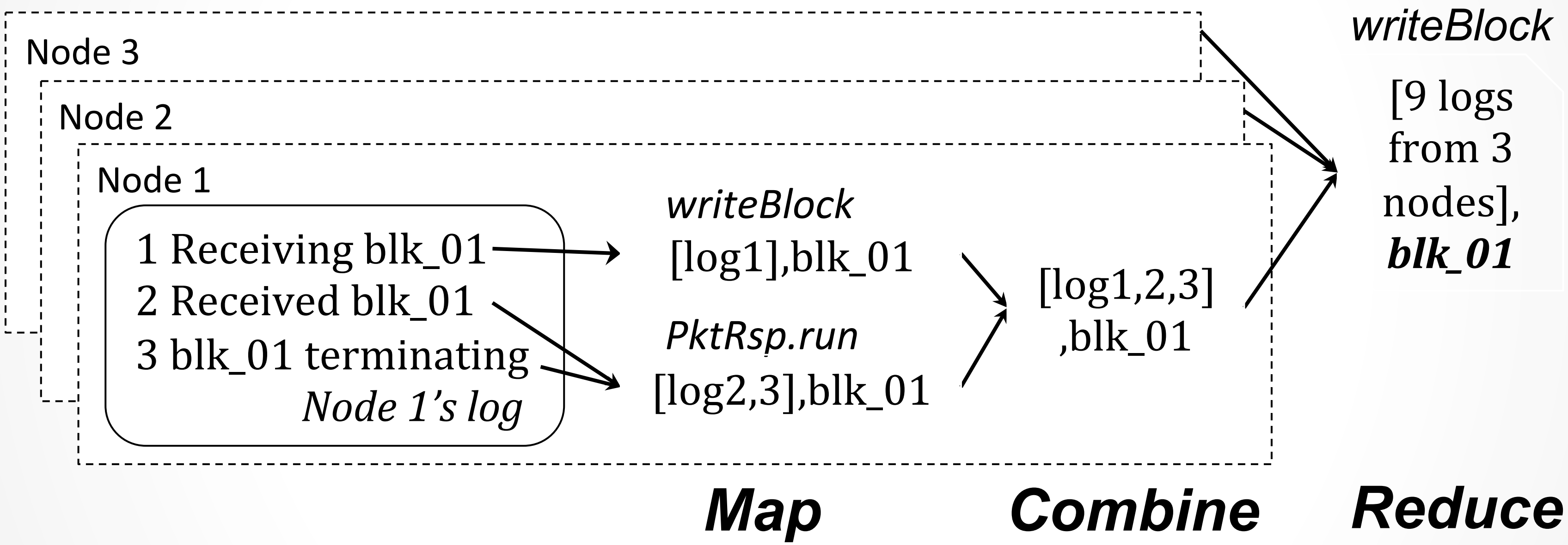
`writeBlock()` → `PacketResponder.run()`

type: Thread Creation

id: blk_id

Distributed log stitching

Implemented as a MapReduce job

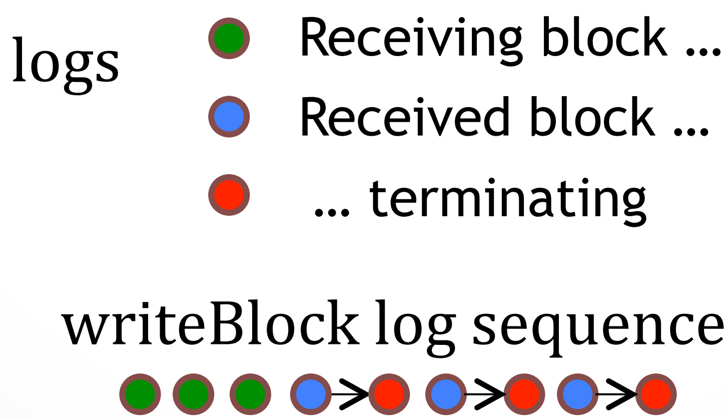


Outline

- Introduction
- Case Study
- Design
- Evaluation

Evaluation methodology

- Evaluated on logs from 4 distributed systems
  - HDFS, Yarn, HBase, Cassandra
  - Logs generated on 200 Amazon EC2 nodes
  - HiBench, YCSB workload
- Authors manually verified each unique log sequence



Request attribution accuracy

accuracy for all the log messages

System	Correct	Incomplete	Failed	Incorrect
HDFS	97.0%	0.1%	2.6%	0.3%
Yarn	79.6%	19.2%	1.2%	0.0%
Cassandra	95.3%	0.1%	4.6%	0.0%
HBase	90.6%	2.5%	3.4%	3.5%
Average	90.4%	5.7%	3.0%	1.0%

Real-world performance anomalies

- Randomly selected 23 anomalies
    - Reproduced each one to collect logs
- lprof is helpful for identifying the root cause for 65%
- Reasons for the cases lprof cannot help
    - Abnormal requests don’t print any logs
    - The abnormal request only print 1 log
        - But latency is needed for debugging

Related work

- Intrusive tools
  - *E.g. MagPie, Project 5, X-Trace, Dapper, etc.*
- Existing log analyzers
  - *E.g. [Xu'09], DISTALYZER, Synoptic, etc.*
- The Mystery Machine [Chow'14]
  - Infers request flow across software layers
  - Analyzes critical path and slack
  - But requires instrumenting IDs into logs

Conclusions

- lprof: a profiler for distributed system
  - Infers request control flow along with timing information
  - Non-intrusive because entirely from system logs
  - Analyzes logs with information generated by static analysis

- lprof leverages the natural way developers do logging

Demo

UNIVERSITY OF TORONTO

Limitations

- lprof benefits from good logging practice
  - lprof cannot help when there's no log
  - Timestamp is required for latency analysis
  - Good identifier can improve the accuracy

- lprof cannot infer request across software layers

- lprof currently works on Java byte code