Chukwa: a scalable log collector

Ari Rabkin and Randy Katz
UC Berkeley

USENIX LISA 2010

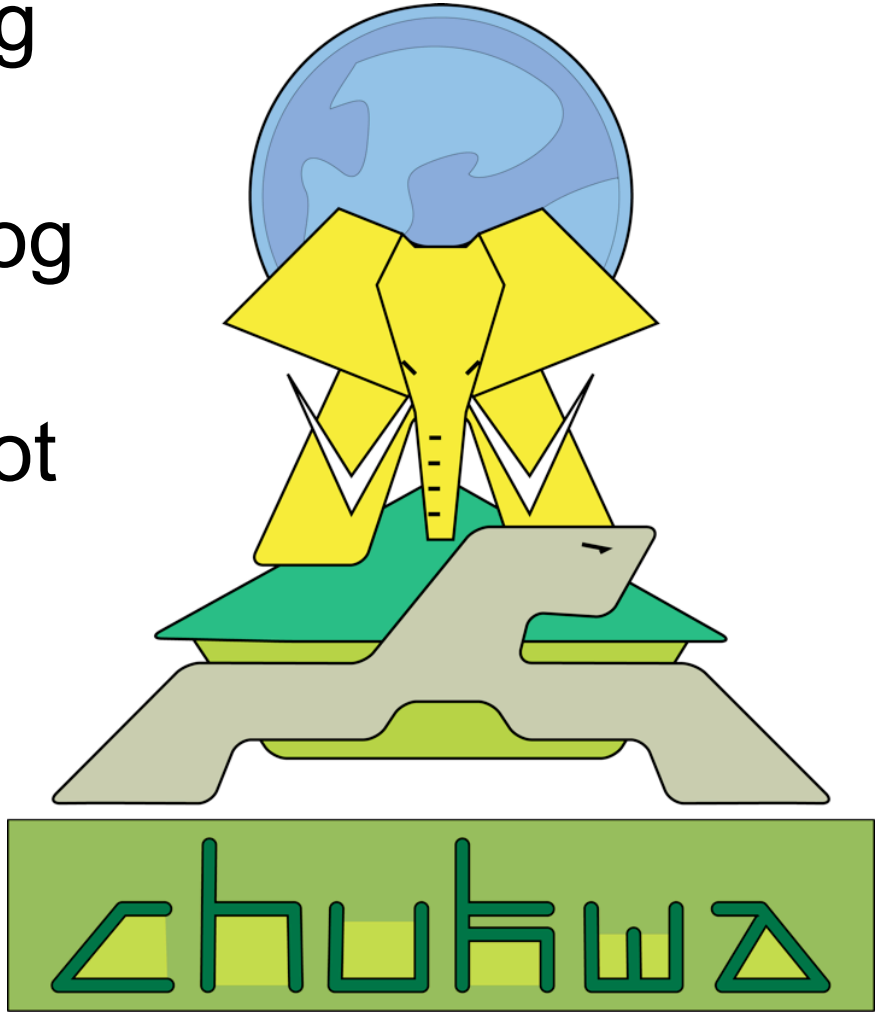
With thanks to… Eric Yang, Jerome Boulon, Bill Graham, Corbin Hoenes, and all the other Chukwa developers, contributors, and users

Why collect logs?

- Many uses
  - Need logs to monitor/debug systems
  - Machine learning is getting increasingly good at detecting anomalies automatically.
  - Web log analysis is key to many businesses
- Easier to process if centralized

Three Bets

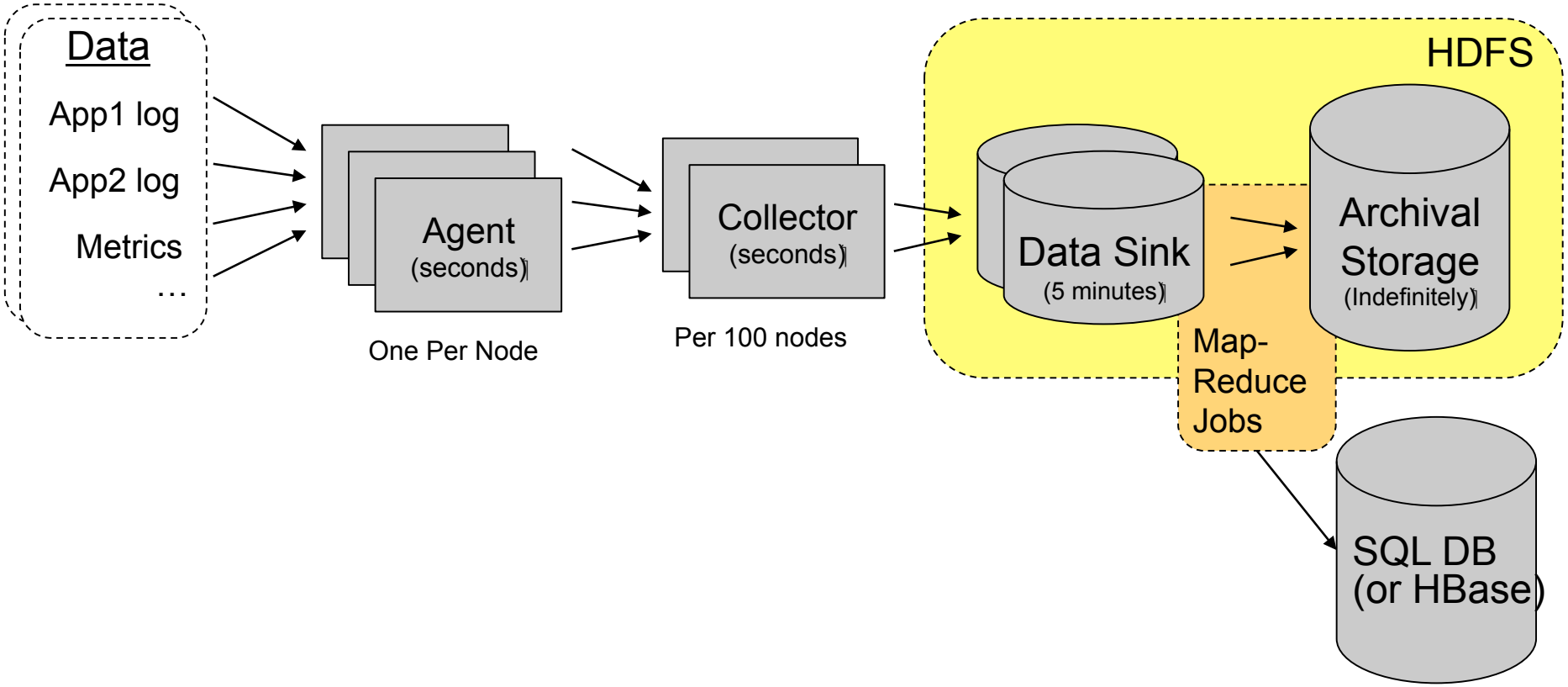
1. MapReduce processing is necessary at scale.
2. Reliability matters for log collection
3. Should use Hadoop, not re-write storage and processing layers



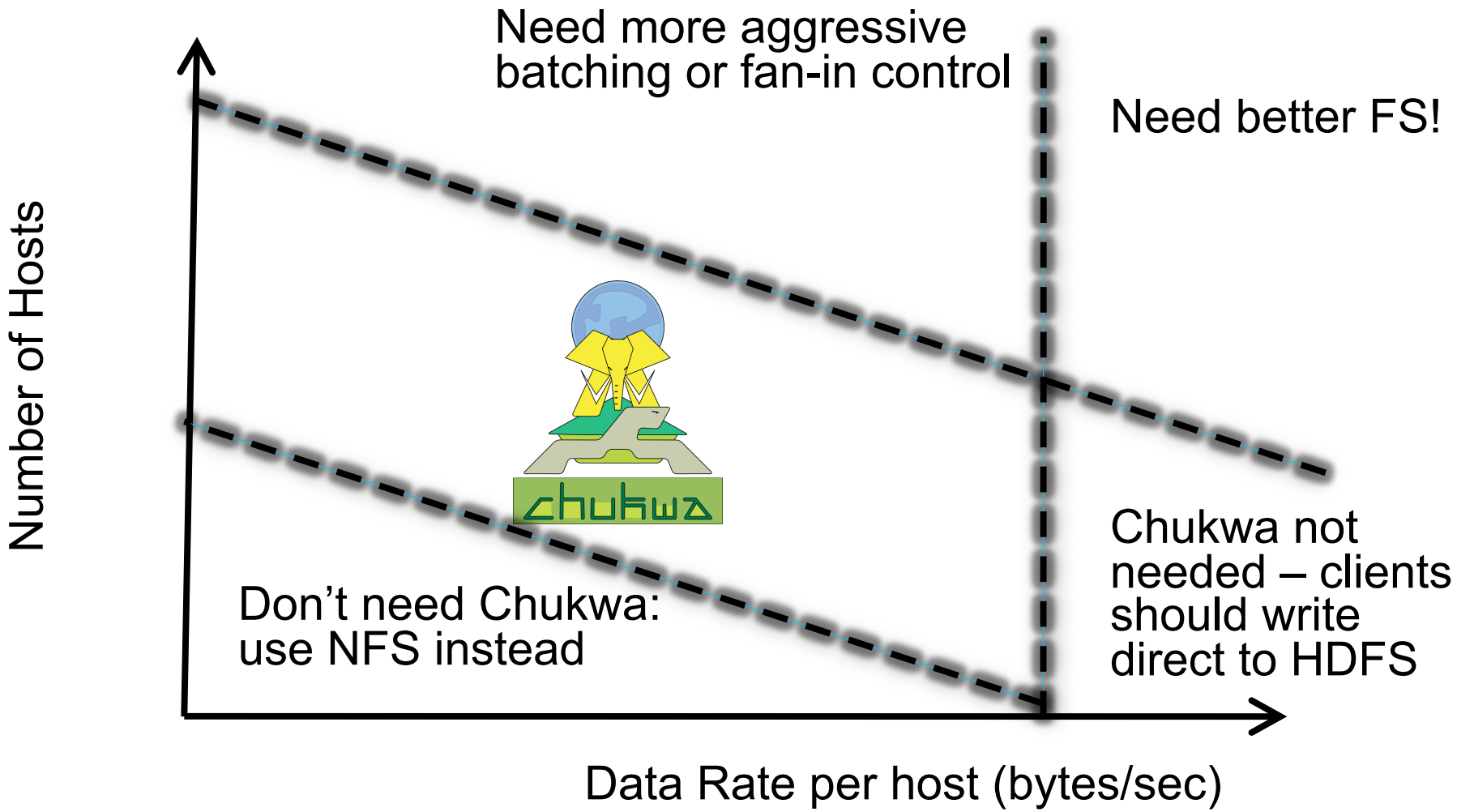
Leveraging Hadoop

- Really want to use HDFS for storage and MapReduce for processing.
  - +Highly scalable, highly robust
  - +Good integrity properties.
- HDFS has quirks
  - Files should be big
  - No concurrent appends
  - Weak synchronization semantics

The architecture

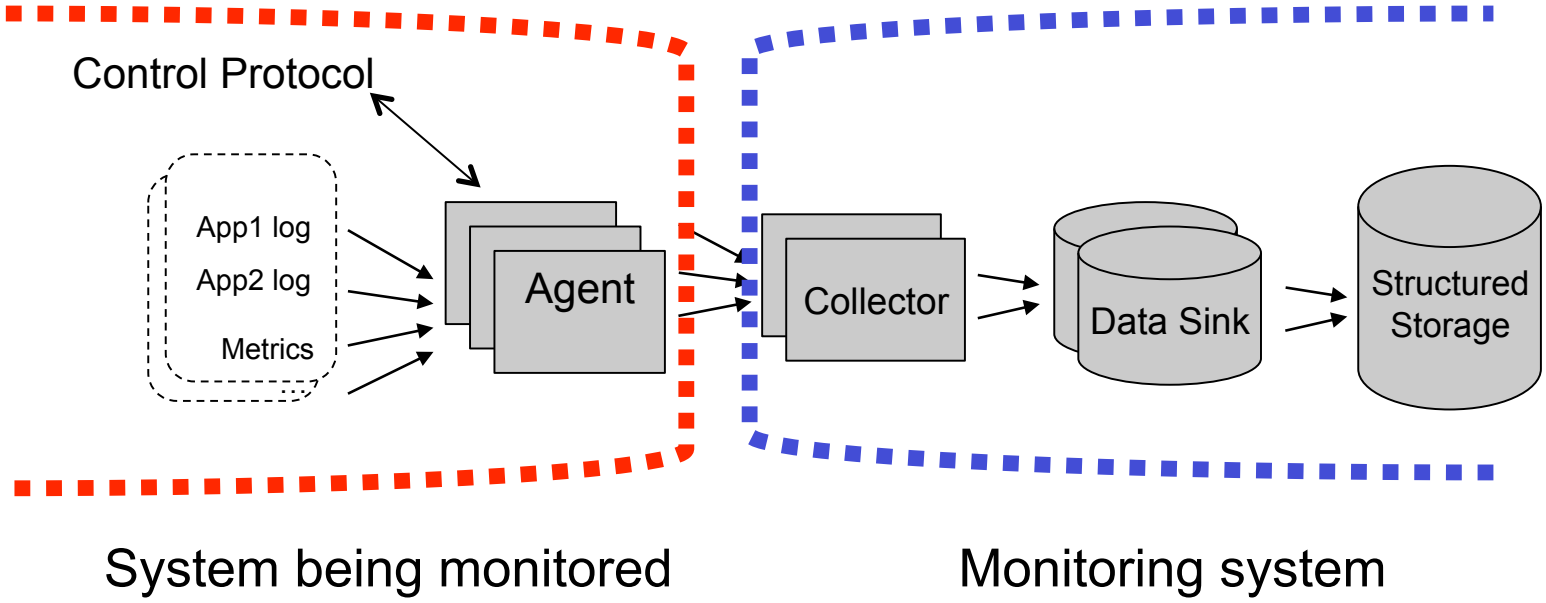


Design envelope



Respecting boundaries

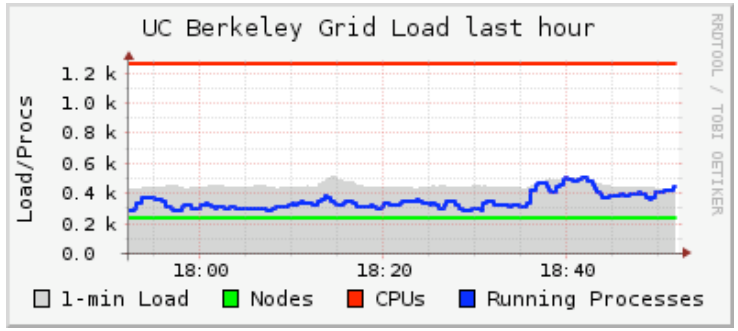
- Architecture captures the boundary between monitoring and production services
  - Important in practice!
  - Particularly nice in cloud context



Comparison

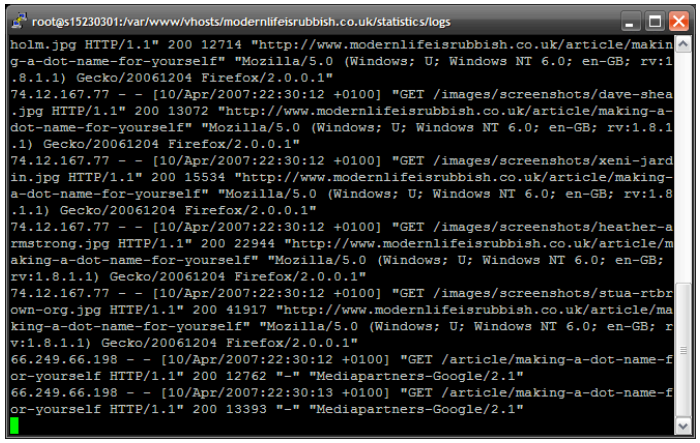
Ganglia

Amazon CloudWatch



Metrics

chukwa



Logs

Data sources

- We optimize for the case of logs on disk
  - Supports legacy systems
  - Writes to local disk almost always succeed
  - Kept in memory in practice – fs caching
- Can also handle other data sources – adaptors are pluggable
  - Support syslog, other UDP, JMS messages.

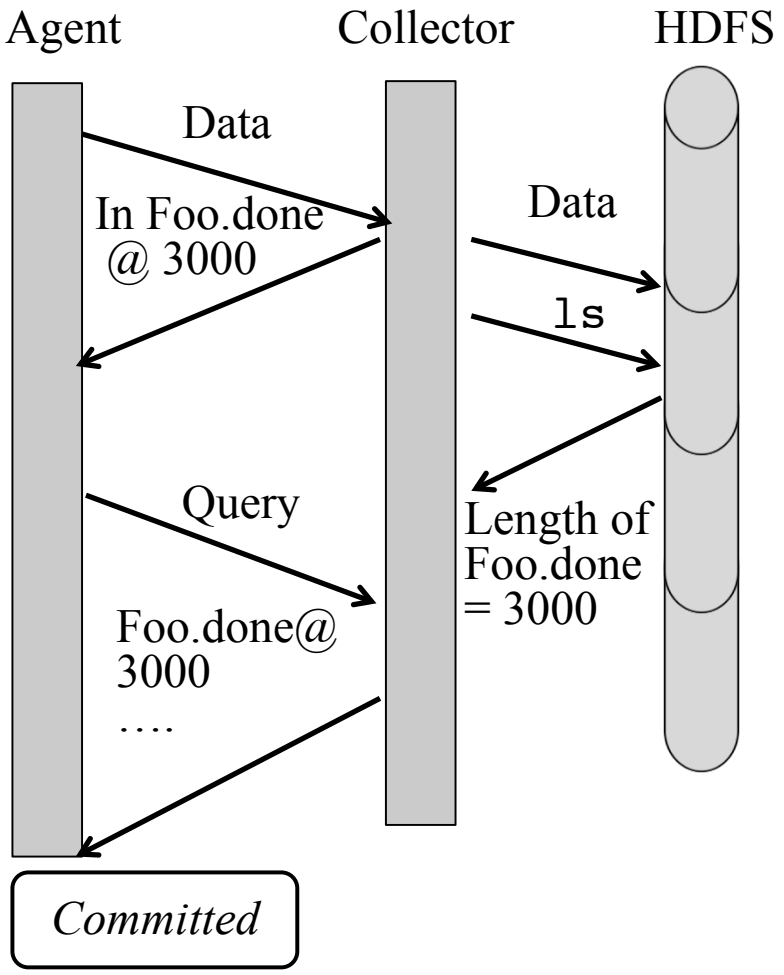
Reliability

- Agents can crash
- Record how much data from each source has been written successfully.
- Resume at that point after crash
- Fix duplicates in the storage layer

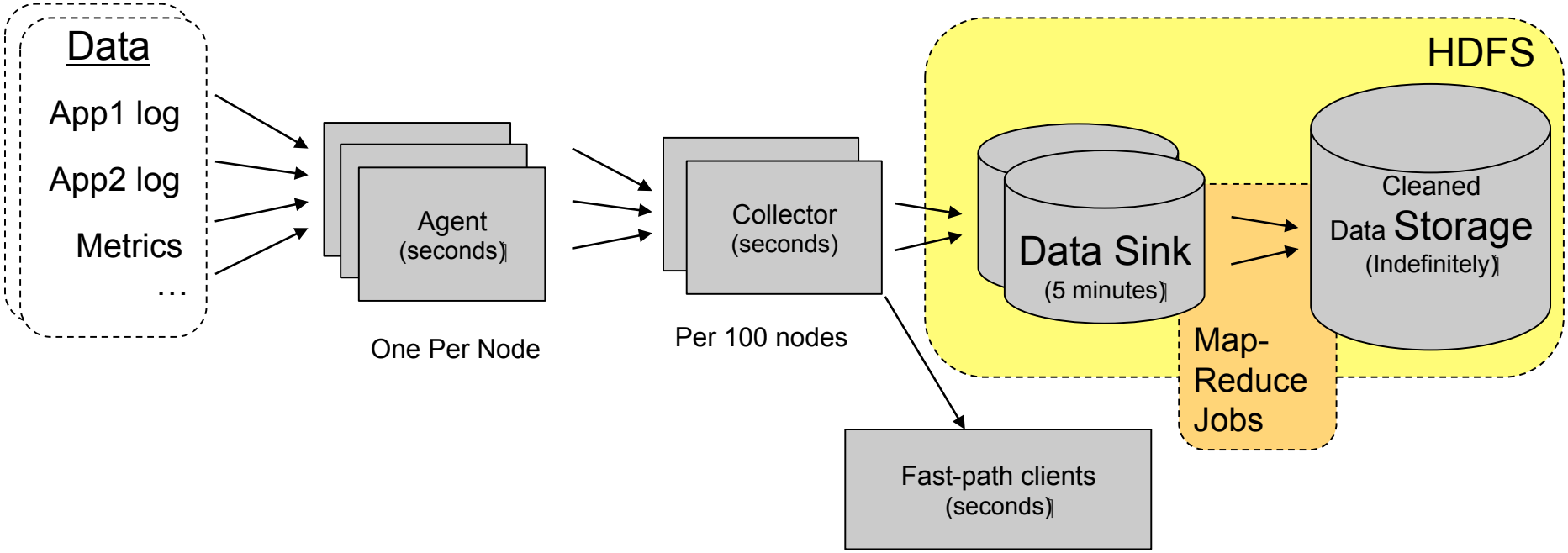
Data Sent and committed

Incorporating Asynchrony

- What about collector crashes?
- Want to tolerate asynchronous HDFS writes without blocking agent
- Solution: async. acks
- Tell agent where data will be written if write succeeds.
- Uses single-writer aspect of HDFS



Fast path


Data
App1 log
App2 log
Metrics
…
Agent
(seconds)
One Per Node
Collector
(seconds)
Per 100 nodes
HDFS
Data Sink
(5 minutes)
Cleaned
Data Storage
(Indefinitely)
Map-
Reduce
Jobs
Fast-path clients
(seconds)

Two modes

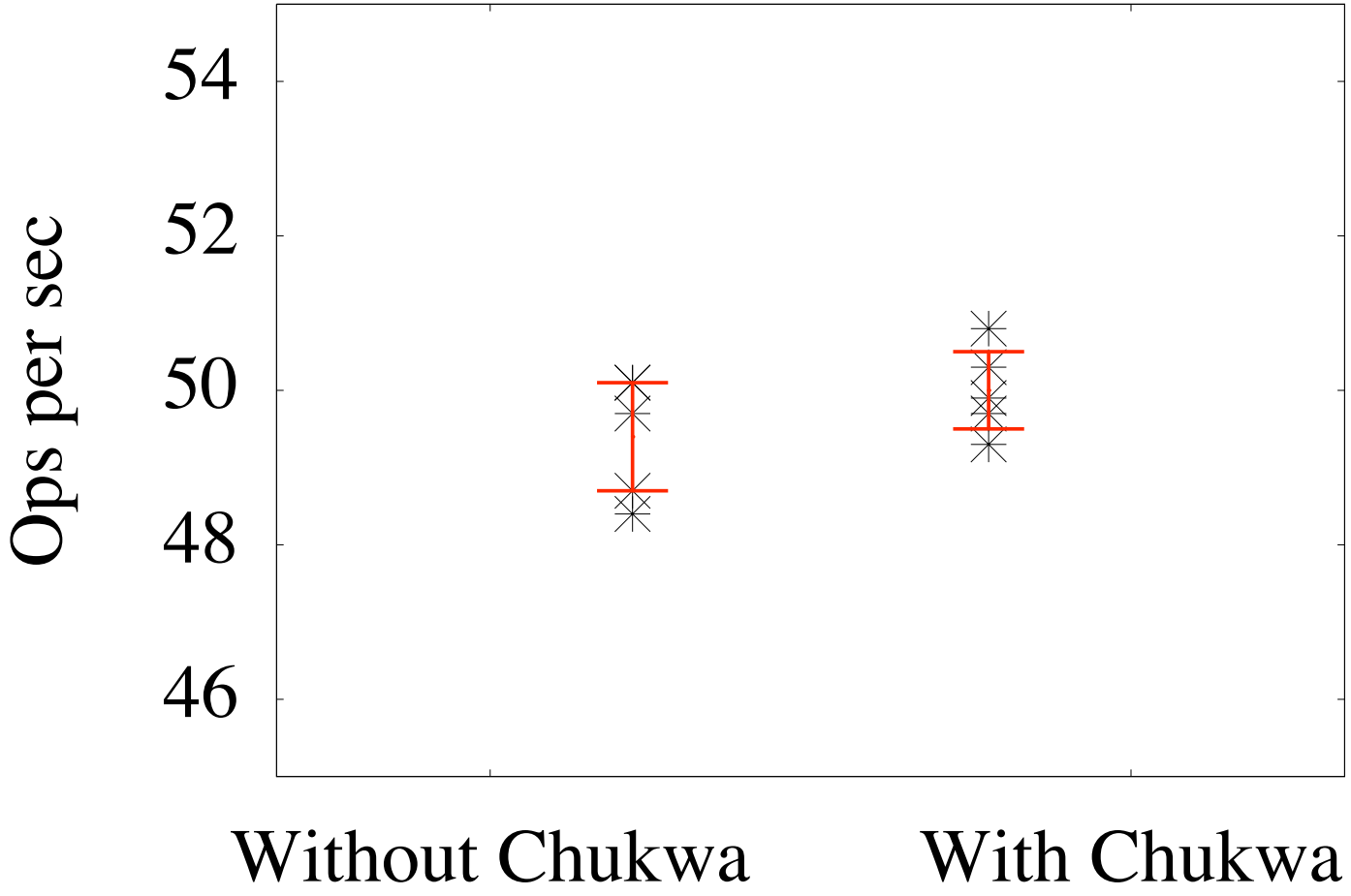
Robust delivery

- Data visible in minutes
- Collects everything
- Stores to HDFS
- Will resend after a crash
- Facilitates MapReduce
- Used for bulk analysis

Prompt delivery

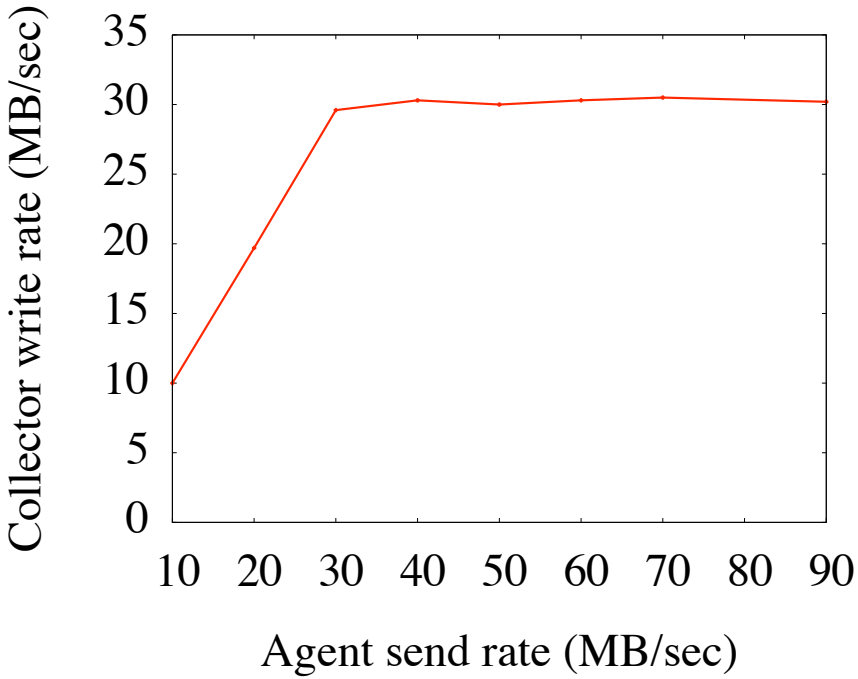
- Data visible in seconds
- User-specified filter
- Written over a socket
- Delivered at most once
- Facilitates near-real-time monitoring
- Used for real-time graphing

Overhead [with Cloudstone]



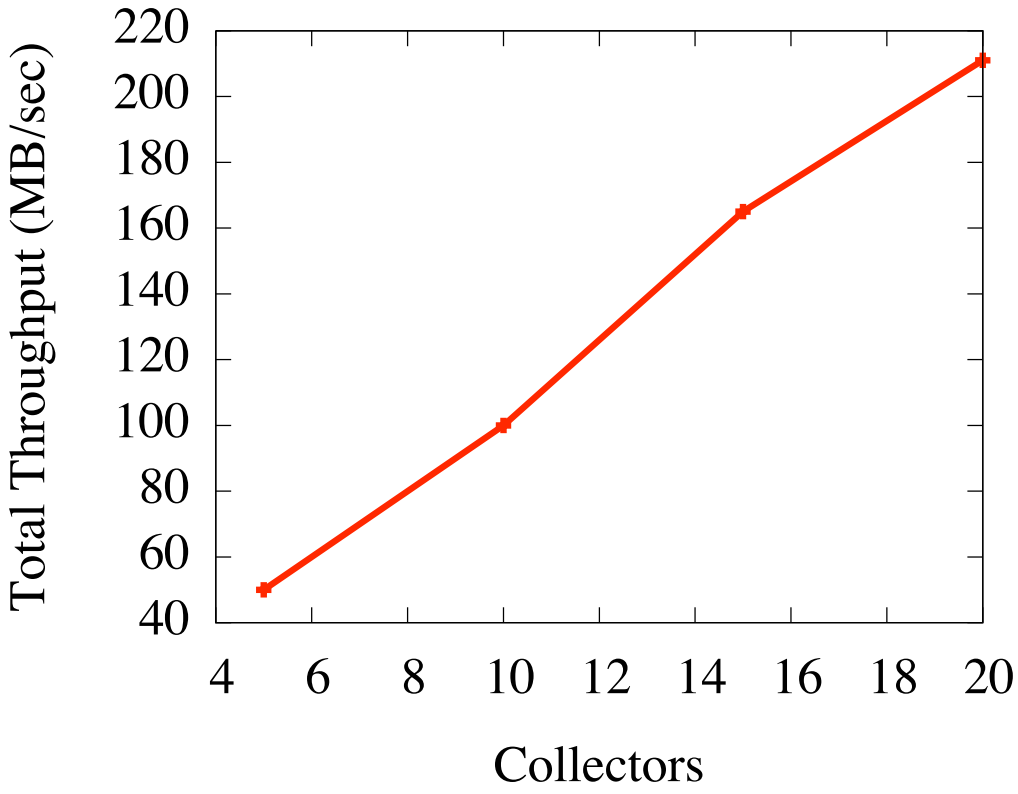
Collection rates

- Tested on EC2
- Able to write 30MB/sec/collector
- Note: data is about 12 months old



Collection rates

- Scales linearly
- Able to saturate underlying FS



Experiences

- Currently in use at:
- UC Berkeley's RAD Lab, to monitor Cloud experiments
- CBS Interactive, Selective Media, and Tynt for web log analysis
  - Dozens of machines
  - Gigabytes to Terabytes per day
- Other sites too…we don't have a census

Related Work

	Handles logs	Crash recovery?	Metadata	Interface	Agent-side control
Ganglia/ Nagios/ other NMS	No	No	No	UDP	No
Scribe	Yes	No	No	RPC	Yes
Flume	Yes	Yes	Yes	flexible	No
Chukwa	Yes	Yes	Yes	flexible	Yes

Next steps

- Tighten security, to make Chukwa suitable for world-facing deployments
- Adjustable durability
  - Should be able to buffer arbitrary non-file data for reliability
- HBase for near-real-time metrics display
- Built-in indexing
- ***Your idea here: Exploit open source!***

Conclusions

- Chukwa is a distributed log collection system that is
- *Practical*
  - In use at several sites
- *Scalable*
  - Builds on Hadoop for storage and processing
- *Reliable*
  - Able to tolerate multiple concurrent failures without losing or mangling data
- *Open Source*
  - Former Hadoop subproject, currently in Apache incubation, enroute to top level project.

Questions?

…vs Splunk

- Significant overlap with Splunk.
  - Splunk uses syslog for transport.
  - Recently shifted towards MapReduce for evaluation.
- Chukwa on its own doesn't [yet] do indexing or analysis.
- Chukwa helps extract data from systems
  - Reliably
  - Customizably

Assumptions about App

- Processing should happen off-node. (Production hosts are sacrosanct)
- Data should be available within minutes
  - Sub-minute delivery a non-goal.
- Data rates between 1 and 100KB/sec/node
  - Architecture tuned for these cases, but Chukwa could be adapted to handle lower/higher rates.
- No assumptions about data format
- Administrator or app needs to tell Chukwa where logs live.
  - Support for directly streaming logs as well.

On the back end

- Chukwa has a notion of parsed *records*, with complex schemas
  - Can put into structured storage
  - Display with HICC, a portal-style web interface.


Default view
Dashboard Builder
Options
Save Dashboard
Cluster Status
DFS Status
Time
Time Zone:
US/Pacific
Style:
Time Period
Refresh (min):
0
Save
Reset
Cluster Metrics by Percentage
cpu_idle_pcnt,cpu_system_pcnt,cpu_user_pcnt
100.00
50.00
0.00
23:40
23:50
00:00
00:10
00:20
cpu_idle_pcnt
cpu_system_pcnt
cpu_user_pcnt
Cluster Metrics by Percentage
eth0_busy_pcnt,eth1_busy_pcnt
100.00
50.00
Cluster Selector
Cluster
rcluster
Apply
Cluster Metrics
mem_total,mem_buffers,mem_cached,mem_used,mem_free
1.50x10^6
1.00x10^6
5.00x10^5
0.00
23:40
23:50
00:00
00:10
00:20
mem_total
mem_buffers
mem_cached
mem_used
mem_free
Cluster Metrics
load_15,load_5,load_1
3.00
2.50
2.00
1.50
23:40
23:50
00:00
00:10
00:20

Not storage, not processing

- Chukwa is a collection system.
  - Not responsible for storage:
    - Use HDFS.
    - Our model is store-everything, prune late
  - Not responsible for processing
    - Use MapReduce, or custom layer on HDFS
- Responsible for **facilitating** storage and processing
- Framework for processing collected data
- Includes Pig support

Goal: Low Footprint

- Wanted minimal footprint on system and minimal changes to user workflow.
  - Application logging need not change.
  - Local logs stay put, Chukwa just copies them.
  - Can either specify filenames in static config, or else do some dynamic discovery.
- Minimal human-produced metadata
  - We track what data source + host a chunk came from. Can store additional tags.
  - Chunks are numbered; can reconstruct order.
  - No schemas required to collect data

MapReduce and Hadoop

- Major motivation for Chukwa was storing and analyzing Hadoop logs.
  - At Yahoo!, common to dynamically allocate hundreds of nodes for a particular task.
  - This can generate MBs of logs per second.
  - Log analysis becomes difficult

Why Ganglia doesn't do this

- Many systems for metrics collection
  - Ganglia particularly well-known.
  - Many similar systems, including network management systems like OpenView
  - Focus on collecting and aggregating metrics in scalable low-cost way
- But logs aren't metrics. Want to archive everything, not summarize aggressively.
- Really want reliable delivery; missing key parts of logs might make rest useless

Clouds

- Log processing needs to be scalable, since apps can get big quickly
- This used to be a problem for the Microsofts and Googles of the world. Now it affects many more.
- Can't rely on local storage
  - Nodes are ephemeral
  - Need to move logs off-node
- Can't do analysis on single host
  - The data is too big

Questions about Goals

- How many nodes? How much data?
- What data sources and delivery semantics?
- Processing expressiveness?
- Storage?

Chukwa goals

- How many nodes? How much data?
  - Scale to thousands of nodes. Hundreds of KB/sec/node on average, bursts above that OK
- What data sources and delivery semantics?
  - Console Logs and Metrics. Reliable delivery (as much as possible.) Minutes of delay are OK.
- Processing expressiveness?
  - MapReduce
- Storage?
  - Should be able to store data indefinitely. Support petabytes of stored data.

In contrast

- Ganglia, Network Management systems, and Amazon's CloudWatch are all metrics-oriented.
  - Goal is collecting and disseminating numerical metrics data in a scalable way.
- Significantly different problem.
  - Metrics have well defined semantics
  - Can tolerate data loss
  - Easy to aggregate/compress for archiving
  - Often time-critical
- Chukwa can serve these purposes, but isn't optimized for it.

Real-time Chukwa

- Chukwa was originally designed to support batch processing of logs
  - Minutes of latency OK.
- But we can do [best effort] real-time "for free"
  - Watch data go past at the collector
  - Check chunks against a search pattern, forward matching ones to a listener via TCP.
  - **Don't** need long-term storage or reliable delivery (do those via the regular data path)
- Director uses this real-time path.

Related work summary

- Ganglia (and traditional NMS) don't do large data volumes or data rates
- Facebook's Scribe+Hive
  - Scribe is streaming, not batch
  - Hive is batch, and atop Hadoop
  - Doesn't do collection or visualization.
  - Doesn't have strong reliability properties
- Flume (from Cloudera)
  - Very similar to Chukwa
  - Emphasis on centralized management