Analysis of HDFS Under HBase
A Facebook Messages Case Study


University of Wisconsin-Madison *Facebook Inc.
Why Study Facebook Messages?

Represents an important type of application. Universal backend for:

- Cellphone texts
- Chats
- Emails
Why Study Facebook Messages?

Represents an important type of application. Universal backend for:

- Cellphone texts
- Chats
- Emails
Why Study Facebook Messages?

Represents an important type of application. Universal backend for:

- Cellphone texts
- Chats
- Emails
Why Study Facebook Messages?

Represents an important type of application. Universal backend for:

- Cellphone texts
- Chats
- Emails
Why Study Facebook Messages?

Represents an important type of **application**. Universal backend for:

- Cellphone texts
- Chats
- Emails

Represents **HBase over HDFS**

- Common backend at Facebook and other companies
- Similar stack used at Google (BigTable over GFS)
Why Study Facebook Messages?

Represents an important type of application. Universal backend for:

- Cellphone texts
- Chats
- Emails

Represents HBase over HDFS

- Common backend at Facebook and other companies
- Similar stack used at Google (BigTable over GFS)

Represents layered storage
Building a Distributed Application (Messages)

We have many machines with many disks. *How should we use them to store messages?*
Building a Distributed Application (Messages)

One option: use machines and disks directly.
Building a Distributed Application (Messages)

One option: use machines and disks directly. Very specialized, but very high development cost.
Building a Distributed Application (Messages)
Building a Distributed Application (Messages)

Use HBase for K/V logic
Building a Distributed Application (Messages)

Use **HBase** for K/V logic
Use **HDFS** for replication
Building a Distributed Application (Messages)

- Use **HBase** for K/V logic
- Use **HDFS** for replication
- Use **Local FS** for allocation

<table>
<thead>
<tr>
<th>Messages</th>
<th>HBase</th>
<th>Hadoop File System</th>
<th>Worker</th>
<th>Worker</th>
<th>Worker</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Machine 1</td>
<td>Machine 2</td>
<td>Machine 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FS</td>
<td>FS</td>
<td>FS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FS</td>
<td>FS</td>
<td>FS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FS</td>
<td>FS</td>
<td>FS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FS</td>
<td>FS</td>
<td>FS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FS</td>
<td>FS</td>
<td>FS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FS</td>
<td>FS</td>
<td>FS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FS</td>
<td>FS</td>
<td>FS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FS</td>
<td>FS</td>
<td>FS</td>
</tr>
</tbody>
</table>
Layered Storage Discussion

Layering Advantages
- Simplicity (thus fewer software bugs)
- Lower development costs
- Code sharing between systems

Layering Questions
- Is layering free performance-wise?
- Can layer integration be useful?
- Should there be multiple HW layers?

Messages

<table>
<thead>
<tr>
<th>HBase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop File System</td>
</tr>
<tr>
<td>Worker</td>
</tr>
<tr>
<td>Worker</td>
</tr>
<tr>
<td>Worker</td>
</tr>
<tr>
<td>Machine 1</td>
</tr>
<tr>
<td>FS</td>
</tr>
<tr>
<td>FS</td>
</tr>
<tr>
<td>FS</td>
</tr>
<tr>
<td>FS</td>
</tr>
<tr>
<td>FS</td>
</tr>
<tr>
<td>FS</td>
</tr>
</tbody>
</table>
Outline

Intro

▪ Messages stack overview
▪ Methodology: trace-driven analysis and simulation
▪ HBase background

Results

▪ Workload analysis
▪ Hardware simulation: adding a flash layer
▪ Software simulation: integrating layers

Conclusions
Methodology

Actual stack

Messages
HBase
HDFS
Local FS
Methodology

Hadoop Trace FS (HTFS)
- Collects request details
  - Reads/writes, offsets, lengths
- 9 shadow machines
- 8.3 days

Actual stack
- HBase
- Messages
- HDFS
- Local FS

HDFS Traces
Methodology

Actual stack

- Messages
- HBase
- HDFS
- Local FS

→ HDFS Traces

→ MapReduce Analysis Pipeline

→ Workload Analysis
Methodology

Actual stack

Messages → HBase → HDFS → Local FS → HDFS Traces
Methodology

Actual stack

Messages ➔ HBase ➔ HDFS ➔ Local FS ➔ HDFS Traces

Background: how does HBase use HDFS?
Outline

Intro

- Messages stack overview
- Methodology: trace-driven analysis and simulation
- HBase background

Results

- Workload analysis
- Hardware simulation: adding a flash layer
- Software simulation: integrating layers

Conclusions
HBase’s HDFS Files

Four activities do HDFS I/O:

HBase memory:

HDFS files:

MemTable

LOG
HBase’s HDFS Files

Four activities do HDFS I/O:

- Logging

HBase memory:

- MemTable
- LOG

HDFS files:

HBase receives a put()
HBase’s HDFS Files

Four activities do HDFS I/O:

- Logging

After many puts, MemTable is full

HBase memory:

MemTable

HDFS files:

LOG
HBase’s HDFS Files

Four activities do HDFS I/O:

- Logging
- Flushing
HBase’s HDFS Files

Four activities do HDFS I/O:

- Logging
- Flushing

HBase memory:

HDFS files:
HBase’s HDFS Files

Four activities do HDFS I/O:

- Logging
- Flushing

HBase memory:

After many flushes, files accumulate

HDFS files:
HBase’s HDFS Files

Four activities do HDFS I/O:

- Logging
- Flushing

get() requests may check many of these
HBase’s HDFS Files

Four activities do HDFS I/O:

- Logging
- Flushing
- Foreground reads

get() requests may check many of these
HBase’s HDFS Files

Four activities do HDFS I/O:

- Logging
- Flushing
- Foreground reads
- Compaction

compaction merge sorts the files

HBase memory:

HDFS files:
HBase’s HDFS Files

Four activities do HDFS I/O:

- Logging
- Flushing
- Foreground reads
- Compaction

HDFS files:

HBase memory:

compaction merge sorts the files
HBase’s HDFS Files

Four activities do HDFS I/O:

- Logging
- Flushing
- Foreground reads
- Compaction

Baseline I/O:

- **Flushing** and **foreground reads** are always required
HBase’s HDFS Files

Four activities do HDFS I/O:

- Logging
- Flushing
- Foreground reads
- Compaction

Baseline I/O:

- Flushing and foreground reads are always required

HBase overheads:

- Logging: useful for crash recovery (but not normal operation)
- Compaction: improves performance (but not required for correctness)
Outline

Intro

▪ Messages stack overview
▪ Methodology: trace-driven analysis and simulation
▪ HBase background

Results

▪ Workload analysis
▪ Hardware simulation: adding a flash layer
▪ Software simulation: integrating layers

Conclusions
Workload Analysis Questions

At each layer, what activities read or write?

How large is the dataset?

How large are created files?

How sequential is I/O?
Workload Analysis Questions

At each layer, what activities read or write?

How large is the dataset?

How large are created files?

How sequential is I/O?
Cross-layer R/W Ratios

Baseline HDFS I/O:

1% writes

1% writes

reads

writes
Cross-layer R/W Ratios

Baseline HDFS I/O: 1% writes

All HDFS I/O: 21% writes
Cross-layer R/W Ratios

Baseline HDFS I/O: 1% writes

All HDFS I/O: 21% writes

Local FS: 45% writes

I/O (TB)
Baseline HDFS I/O:

All HDFS I/O:

Local FS:

Disk:

Cross-layer R/W Ratios

Baseline HDFS I/O: 1% writes

All HDFS I/O: compact LOG 21% writes

Local FS: R1 R2 R3 45% writes

Disk: cache misses 64% writes

I/O (TB)
Workload Analysis Conclusions

1. Layers amplify writes: 1% => 64%
   - Logging, compaction, and replication increase writes
   - Caching decreases reads
Workload Analysis Questions

At each layer, what activities read or write?

How large is the dataset?

How large are created files?

How sequential is I/O?
Baseline HDFS I/O: 18% written
All HDFS I/O: compact LOG 77% written
Local (FS/disk): R1 R2 R3 91% written
Workload Analysis Conclusions

1. Layers amplify writes: 1% => 64%
2. Most touched data is only written
Cold Data

Baseline HDFS I/O:

All HDFS I/O:
compact, LOG

Local (FS/disk):
R1, R2, R3
Cold Data

Local (FS/disk):

Footprint (TB)
Cold Data

Local (FS/disk):

Footprint (TB): 0 20 40 60 80 100 120

R1  R2  R3  cold data
Workload Analysis Conclusions

1. Layers amplify writes: 1% => 64%
2. Most touched data is only written
3. The dataset is large and cold: 2/3 of 120TB never touched
Workload Analysis Questions

At each layer, what activities read or write?

How large is the dataset?

How large are created files?

How sequential is I/O?
Created Files: Size Distribution

Percent of Files

File Size

Created Files: Size Distribution

$\text{Percent of Files}$

$\text{File Size}$
50% of files are <750KB
90% of files are <6.3MB
Workload Analysis Conclusions

1. Layers amplify writes: 1% => 64%
2. Most touched data is only written
3. The dataset is large and cold: 2/3 of 120TB never touched
4. Files are very small: 90% smaller than 6.3MB
Workload Analysis Questions

At each layer, what activities read or write?

How large is the dataset?

How large are created files?

How sequential is I/O?
Reads: Run Size

50% of runs (weighted by I/O) <130KB
80% of runs (weighted by I/O) < 250KB
Workload Analysis Conclusions

1. Layers amplify writes: 1% => 64%
2. Data is read or written, but rarely both
3. The dataset is large and cold: 2/3 of 120TB never touched
4. Files are very small: 90% smaller than 6.3MB
5. Fairly random I/O: 130KB median read run
Outline

Intro

- Messages stack overview
- Methodology: trace-driven analysis and simulation
- HBase background

Results

- Workload analysis
- Hardware simulation: adding a flash layer
- Software simulation: integrating layers

Conclusions
Hardware Architecture: Workload Implications

Option 1: pure disk    Option 2: pure flash    Option 3: hybrid
Hardware Architecture: Workload Implications

Option 1: pure disk
- Very random reads
- Small files

Option 2: pure flash

Option 3: hybrid
Hardware Architecture: Workload Implications

Option 1: pure disk
- Very random reads
- Small files

Option 2: pure flash

Option 3: hybrid
Hardware Architecture: Workload Implications

Option 1: pure disk
- Very random reads
- Small files

Option 2: pure flash
- Large dataset
- Mostly very cold
- >$10K / machine

Option 3: hybrid
Hardware Architecture: Workload Implications

Option 1: **pure disk**
- Very random reads
- Small files

Option 2: **pure flash**
- Large dataset
- Mostly very cold
- >$10K / machine

Option 3: **hybrid**
Hardware Architecture: Workload Implications

Option 1: pure disk
- Very random reads
- Small files

Option 2: pure flash
- Large dataset
- Mostly very cold
- >$10K / machine

Option 3: hybrid
- Process of elimination
Hardware Architecture: Simulation Results

Evaluate cost and performance of 36 hardware combinations (3x3x4)

- Disks: 10, 15, or 20
- RAM (cache): 10, 30, or 100GB
- Flash (cache): 0, 60, 120, or 240GB

Assumptions:

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Cost</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDD</td>
<td>$100/disk</td>
<td>10ms seek, 100MB/s</td>
</tr>
<tr>
<td>RAM</td>
<td>$5/GB</td>
<td>zero latency</td>
</tr>
<tr>
<td>Flash</td>
<td>$0.8/GB</td>
<td>0.5ms</td>
</tr>
</tbody>
</table>
Cost/performance tradeoff for 36 hardware combinations
Upgrades decrease latency but increase cost.
Bad upgrade
10 disks

![Graph showing the relationship between Cost ($) and Foreground latency (ms)]
upgrade to 15 disks
upgrade to 20 disks
upgrade to 20 disks

Upgrading disk: 😞
upgrade to 30GB RAM
upgrade to 100GB RAM
upgrade to 100GB RAM

Upgrading RAM:
no flash!
upgrade to 60GB flash
upgrade to 120GB flash
upgrade to 240GB flash
Upgrade to 240GB flash

Upgrading flash: 🤚
Outline

Intro
- Messages stack overview
- Methodology: trace-driven analysis and simulation
- HBase background

Results
- Workload analysis
- Hardware simulation: adding a flash layer
- Software simulation: integrating layers

Conclusions
Software Architecture: Workload Implications

Writes are amplified

- 1% at HDFS (excluding overheads) to 64% at disk (given 30GB RAM)
- *We should optimize writes*
Software Architecture: Workload Implications

Writes are greatly amplified

- 1% at HDFS (excluding overheads) to 64% at disk
- We should optimize writes

61% of writes are for compaction

- We should optimize compaction
- Compaction interacts with replication inefficiently
Replication Overview

Machine 1
- HBase Worker
- HDFS Worker

Machine 2
- HBase Worker
- HDFS Worker

Machine 3
- HBase Worker
- HDFS Worker
Problem: Network I/O (red lines)
Solution: Ship Computation to Data

Machine 1
- HBase Worker
- HDFS Worker

Machine 2
- HBase Worker
- HDFS Worker

Machine 3
- HBase Worker
- HDFS Worker
Solution: do Local Compaction

Machine 1:  
- HBase Worker
- HDFS Worker

Machine 2:  
- HBase Worker
- HDFS Worker

Machine 3:  
- HBase Worker
- HDFS Worker

Connections:  
- Machine 1 to Machine 2: do compact
- Machine 2 to Machine 3: do compact
Solution: do **Local Compaction**

Machine 1
- HBase Worker
- HDFS Worker
- compaction

Machine 2
- HBase Worker
- HDFS Worker
- compaction

Machine 3
- HBase Worker
- HDFS Worker
- compaction
Local Compaction

Normally 3.5TB of network I/O
Local Compaction

Normally 3.5TB of network I/O
Local comp: 62% reduction
Local Compaction

Normally 3.5TB of network I/O
Local comp: 62% reduction
Local Compaction

- Normally 3.5TB of network I/O
- Local comp: 62% reduction
- Network I/O becomes disk I/O
- 9% overhead (30GB cache)
- Compaction reads: (a) usually misses, (b) pollute cache
Normally 3.5TB of network I/O

Local comp: 62% reduction

Network I/O becomes disk I/O
- 9% overhead (30GB cache)
- Compaction reads: (a) usually misses, (b) pollute cache

Still good!
- Disk I/O is cheaper than network
Outline

Intro
  - Messages stack overview
  - Methodology: trace-driven analysis and simulation
  - HBase background

Results
  - Workload analysis
  - Hardware simulation: adding a flash layer
  - Software simulation: integrating layers

Conclusions
Conclusion 1: Messages is a New HDFS Workload

Original GFS paper:

- “high sustained bandwidth is more important than low latency”
- “multi-GB files are the common case”

We find files are small and reads are random

- 50% of files <750KB
- >75% of reads are random
Conclusion 2: Layering is Not Free

Layering “proved to be vital for the verification and logical soundness” of
the THE operating system ~ Dijkstra

We find layering is not free

- Over half of network I/O for replication is unnecessary

Layers can amplify writes, multiplicatively

- E.g., logging overhead (10x) with replication (3x) => 30x write increase

Layer integration can help

- Local compaction reduces network I/O caused by layers
Conclusion 3: Flash Should not Replace Disk

Jim Gray predicted (for ~2012) that “tape is dead, disk is tape, flash is disk”

We find flash is a poor disk replacement for Messages

- Data is very large and mostly cold
- Pure flash would cost >$10K/machine

However, small flash tier is useful

- A 60GB SSD cache can double performance for a 5% cost increase
Thank you! Any questions?

University of Wisconsin-Madison

Facebook Inc.