Exalt: Empowering Researchers to Evaluate Large-Scale Storage Systems

Yang Wang, Manos Kapritsos, Lara Schmidt, Lorenzo Alvisi, and Mike Dahlin
The University of Texas at Austin
We need to evaluate our prototypes
We need to evaluate our prototypes
Industrial deployment: tens of PBs
thousands of nodes

Researchers: hundreds of TBs
hundreds of nodes

• Salus (Wang et al. NSDI 13): 108 servers
• Eiger (Lloyd et al. NSDI 13): 256 servers
• Spanner (Corbett et al. OSDI 12): Hundreds of servers
Extrapolation?

- Measure with a small cluster
- Predict the behavior at full scale
- Assumption:
  - Resource consumption grows linearly with scale

Extrapolate: The system can scale to 1,000 nodes.
Extrapolation?

- Measure with a small cluster
- Predict the behavior at full scale
- Assumption: May not hold
  - Resource consumption grows linearly with scale
Can we run prototypes at full scale?
Can we run prototypes at full scale?

- Colocate multiple processes on one node
Can we run prototypes at full scale?

• Colocate multiple processes on one node

Problem:
Limited I/O resource

Processes
Network/Disk

Machines
Data content doesn’t affect system behavior

- Clients can write/read synthetic data
- Abstract away data on I/O devices
- Reduce resource requirement of each process
How to abstract away data?

• Discard data? (David, Agrawal et al. FAST 2011)
  – Doesn’t work with large-scale storage systems

• Our approach: Compress data
Requirements of compression

• CPU efficient
  – General-purpose algorithms (e.g. Gzip) are CPU heavy

• High compression ratio

• Lossless compression

• Be able to work with mixed data and metadata
Challenge: Data mixed with metadata

- System may add metadata
- System may split data (possibly nondeterministically)

Key: Locate metadata inside data
Solution: Tardis data pattern

- Make data distinguishable from metadata
  - **Flag**: sequence of bytes that does not appear in metadata
- Efficiently locate metadata: Follow sorted pattern
  - **Marker**: number of remaining bytes to the end

![Tardis data pattern diagram]

| Tardis | 1016 | 1008 | 1000 | ... | 0 |

1KB data chunk and 4-byte flags and markers
Tardis compression

33,000 times faster than gzip

Original data

Search for flag

Retrieve marker

Skip 504 bytes

Retrieve marker

Skip 1016 bytes:
Hit the end of chunk

Compressed data

Tardis 512:512

Starting point

Tardis 1024 : 16

Length
How to find an appropriate flag?

• Scan all metadata: Expensive

• Observation: Tardis is only used for testing

• A randomly chosen 8-byte flag works
  – HDFS
  – HBase
Testing with Tardis

• Run potential bottleneck nodes in real mode.
• Run most nodes in emulated mode.
Implementation

• Emulated devices: disk, network, and memory

• Disk and network: Transparent emulation
  – Byte code instrumentation (BCI)
  – Usage: java -Xbootclasspath exalt.jar <original app>

• Memory: Require code modification
  – None for HDFS; 71 LOC for HBase
Case studies

• Apply our emulator to HDFS and HBase
  – Measure their scalability
  – When we find a problem, analyze its root cause, and fix it

• Testbed:
  – Texas Advanced Computing Center (TACC)
Scalability of HDFS

- Increase number of RPC threads
- Put debug information in tmpfs
- Same as reported by HDFS developers
- Disable sync

Graph:
- Aggregate throughput (GB/s)
- Number of DataNodes (1.2K, 2.4K, 4.8K, 9.6K, 19.2K)
- Ideal throughput

Points:
- Increase number of RPC threads
- Put debug information in tmpfs
- Disable sync
- Put metadata in tmpfs
One problem of HDFS: Big files

HDFS performance degradation as file grows large.
Applying Exalt more broadly

• CPU intensive systems?
  – DieCast (Gupta et al. NSDI 2008)

• Data sensitive applications/benchmarks?
  – Record (on a large testbed) and replay (on a small one)

• The target system modifies data?
  – Ad-hoc solutions for de-duplication, encryption, etc
Conclusion

Industry

Researchers

https://code.google.com/p/exalt/