Blizzard: Fast, Cloud-scale Block Storage for Cloud-oblivious Applications

Microsoft Research

James Mickens, Jeremy Elson, Edmund B. Nightingale, Darren Gehring, Krishna Nareddy

Asim Kadav
Vijay Chidambaram

Bin Fan

Osama Khan

The University of Wisconsin Madison
Carnegie Mellon University
Johns Hopkins
IT’S, UH, THE FUTURE

Cloud Computing
WHAT ARE WE GOING TO DO WITH ALL OF THIS CLOUD DATA?
My Goal

• Take unmodified POSIX/Win32 applications . . .
• Run those applications in the cloud . . .
• On the same hardware used to run big-data apps . . .
• . . . and give them cloud-scale IO performance!
My Goal

- Take unmodified POSIX/Win32 applications . . .
- Run those applications in the cloud . . .
- On the same hardware used to run big-data apps . . .
- . . . and give them cloud-scale IO performance!

- Throughput > 1000 MB/s
- Scale-out architecture using commodity parts
- Transparent failure recovery
Why Do I Want To Do This?

- Write POSIX/Win32 app once, automagically have fast cloud version
- Cloud operators don’t have to open up their proprietary or sensitive protocols
- Admin/hardware efforts that help big data apps help POSIX/Win32 apps (and vice versa)

BECAUSE THIS WOULD BE AMAZING
Our Solution: Blizzard

Blizzard virtual drive

PostgreSQL  eclipse

Remote disks
Our Solution: Blizzard

Remote disks

Blizzard virtual drive

RAID?

NAS?

EBS?
THE PROBLEM IS YOU
The naïve approach for implementing virtual disks does not **maximize spindle parallelism** for POSIX/Win32 applications which **frequently** issue `fsync()` operations to maintain consistency.
The naïve approach for implementing virtual disks does not maximize spindle parallelism for POSIX/Win32 applications which frequently issue `fsync()` operations to maintain consistency.
LISTEN

IOp dilation:
Nested striping

Rack locality:
Locality-oblivious storage

fsync() write barriers:
Delayed durability semantics

Excellence
Virtual disk

Remote disks
Client App
  fwrite(W_X)
  fwrite(W_Y)
Client OS
  fwrite(W_X)
  fwrite(W_Y)
Network
  fwrite(W_X)
  fwrite(W_Y)
Server OS
  fwrite(W_X)
  fwrite(W_Y)

IO
  queue

Time

(Op Convoy Dilation)
Fixing IOP Convoy Dilation

Virtual drive

Segment size = 4

Remote disks
Fixing IOp Convoy Dilation

Virtual drive

Segment size = 4

Remote disks

Random *and* sequential IOs hit multiple spindles!
TOTAL VICTORY
Rack Locality
Rack Locality

Segment size = 4

Blizzard client

Remote disks
AIN’T NOBODY GOT TIME FOR THAT
FDS To The Rescue (OSDI 2012)

- Hardware architecture
  - Full bisection bandwidth network (no oversubscription)
  - Allocate each disk enough network bandwidth to drive disk at full sequential speed (1 disk ≈ 128 MB/s ≈ 1Gbps)

- Result: locality-oblivious storage
  - Any client can access any disk as fast as local
  - Enables aggressive striping
Blizzard as FDS Client

Blizzard client handles:
- Nested striping
- Delayed durability semantics

FDS provides:
- Locality-oblivious storage hardware
- Server-side failure recovery
- RTS/CTS to avoid edge congestion
Blizzard as FDS Client

Blizzard client handles:
- Nested striping
- Delayed durability semantics

FDS provides:
- Locality-oblivious storage hardware
- Server-side failure recovery
- RTS/CTS to avoid edge congestion
POSIX/Win32 apps

Traditional big-data apps
ARE WE THERE YET?
The problem with fsync()

• Used by POSIX/Win32 file systems and applications to implement crash consistency
  – Disk only returns from fsync() when all prior writes have become durable
  – Ex: ensure data is written before metadata

Write data  fsync()  Write metadata
WRITE BARRIERS
RUIN
BIRTHDAYS

Stalled operations
limit parallelism!

Time
Delayed Durability

• Decouple durability from ordering
• Acknowledge `fsync()` immediately . . .
  – . . . but increment flush epoch
  – Tag writes with their epoch number, asynchronously retire writes in epoch order
Delayed Durability

• Decouple durability from ordering
• Acknowledge `fsync()` immediately . . .
  – . . . but increment flush epoch
  – Tag writes with their epoch number, asynchronously retire writes in epoch order

```
W_{Y0}  F_1  W_{X1}  W_{Y1}  F_2  W_{Y2}
```

App

Blizzard

Remote disk
• All writes are acknowledged . . .
• . . . but only \( W_{Y0} \) and \( W_{Y1} \) are durable!
• Satisfies prefix durability from ordering
  = Acknowledge \( \text{fsync()} \) immediately . . .
  = Some, all, or no writes from epoch \( N \) are durable
  = No writes from later epochs are durable
• Prefix consistency good enough for most apps, provides much better performance!

\[
W_{Y0} \quad F_1 \quad W_{X1} \quad W_{Y1} \quad F_2 \quad W_{Y2}
\]
Isn’t Blizzard buffering a lot of data?

Epoch 3
Epoch 2
Epoch 1
Epoch 0

Cannot issue!

In flight . . .
THERE ARE NO SPECULATIVE EXECUTION TECHNIQUES?
Log-based Writes

- Treat backing storage as a distributed log
  - Issue writes to log **immediately** and **in order**
  - On failure, roll forward from last checkpoint and stop when you find torn write, unallocated log block with old epoch number
Summary of Blizzard’s Design

- Problem: IOP Dilation
- Solution: Nested striping

- Problem: Rack locality constrains parallelism
- Solution: Full-bisection networks, match disk and network bandwidth

- Problem: Evil fsync()s
- Solution: Delayed durability
PROOF OF YOUR EXCELLENCE

I DO NOT SEE IT
Throughput Microbenchmark

Throughput (MB/s)

Block Size

32 KB  64 KB  128 KB  256 KB

Seq. Writes
Seq. Reads
Rand. Writes
Rand. Reads
Delayed Durability: Hiding Replication Penalties

![Bar chart showing exchange throughput (MB/s) for different replica configurations.](chart.png)
Blizzard vs EBS: Write IOPS and Read IOPS
Related Work

Virtual drives
- iSCSI, AoE
- Petal
- EBS, Azure Drive
- Salus

File systems
- BlueSky, pNFS
- OptFS
- BPFS
Conclusions

• Unmodified POSIX/Win32 apps can have cloud-scale IO!
  – Nested striping
  – FDS-style hardware substrate
  – Delayed durability semantics

• Raw perf: 1000+ MB/s

• 2x—10x app-level speedups
RTT Sensitivity

Throughput (MB/s)

Added Latency

- Seq. Write
- Seq. Read
- Rand. Write
- Rand. Read

0 ms
5 ms
10 ms
20 ms
Recovery

Write stream

Checkpoint Log
pos: 0
Write #: 0

Checkpoint Log
pos: 2
Write #: 2

Remote log

W0  W1  W2  W3

Not recovered!
Additional Details

• Blizzard maps each virtual block to backing physical block in the log
  – Allocation map included in checkpoints

• To avoid IOP dilation, use random permutation to determine next log position!

Prior example: 0, 1, 2, 3, 4, ...

What Blizzard really does:  $X_{n+1} = (aX_n + c) \mod m$

(Checkpoint a, c, m, $X_n$)