


The word "panasas" is written in a lowercase, serif font with a slight shadow effect. To the right of the text is a stylized, glowing yellow logo that resembles a lowercase 'p' or a similar shape, with a white outline and a yellow fill.

Integrated System Models for Reliable Petascale Storage Systems

Brent Welch, PhD
Panasas, Inc.
welch@panasas.com

A horizontal collage of images at the bottom of the slide. From left to right: an offshore oil rig, a red Formula 1 race car, a clapperboard with the text "SC07, Nov 2007", a digital display showing "1000000", a globe of the Earth, and a modern skyscraper.

SC07, Nov 2007

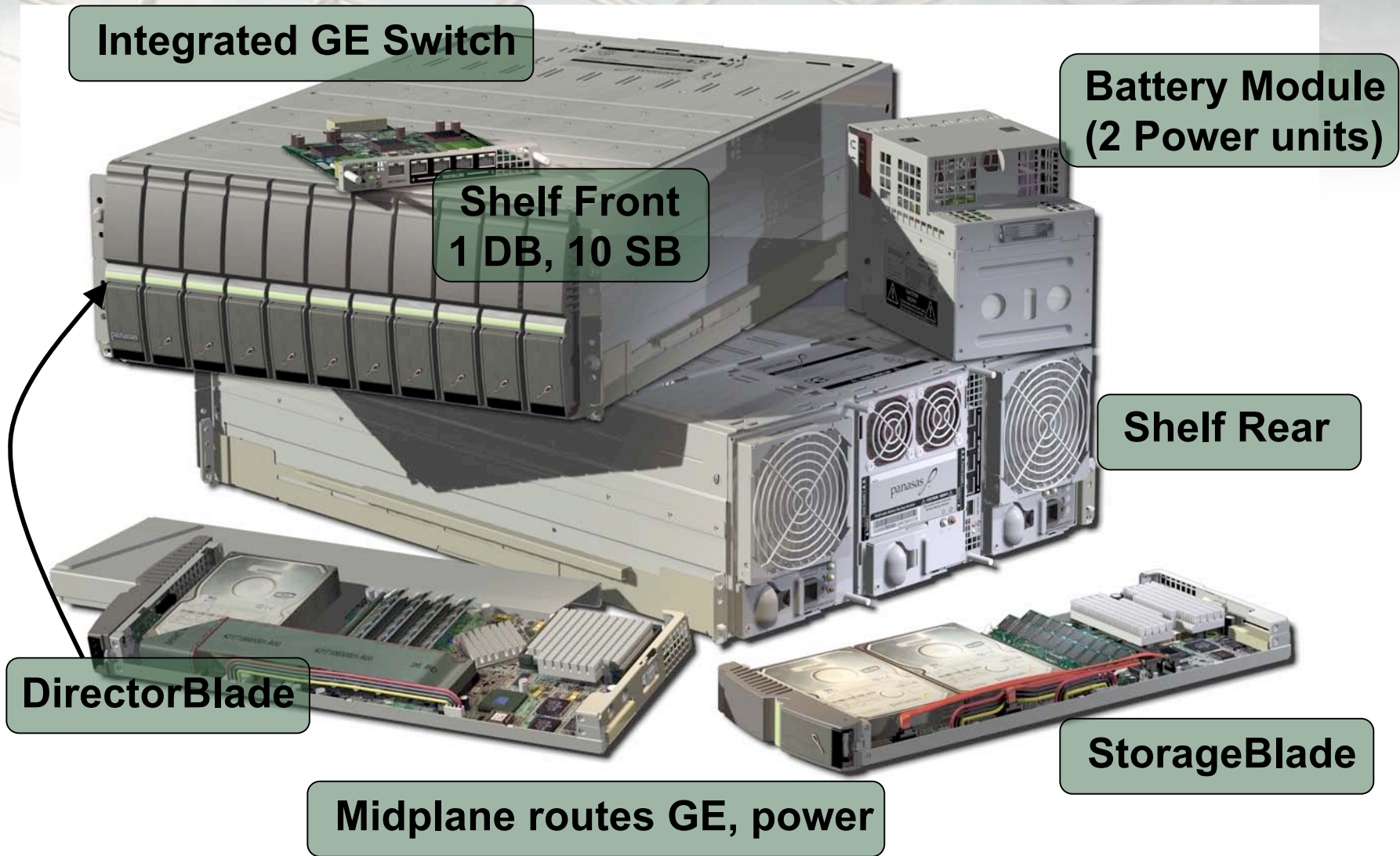
Abstract and Outline

- Petascale storage systems are built from very large distributed systems
 - Need a reliable distributed system platform
 - The system must manage itself (as much as possible)
- Integrated system models
 - Model-driven programming is a standard approach for clean design
 - Distributed systems have a lot of moving parts that must be in the model
- Model the desired state, monitor the current state
 - Subsystems strive to achieve the desired state
- Panasas storage system model
 - Examples from our system implementation

Large distributed systems

- Petascale storage systems are built from very large distributed systems
 - Cluster technology aggregates lots of hardware resources
 - 1000's of drives
 - 100's of computers and controllers
 - 100's of services
 - 1000's of file system clients
 - Next year, multiply everything by 10
- Panasas blade server storage clusters, e.g.
 - 1000 drives, 2 each in 500 StorageBlades modules
 - 50 metadata manager DirectorBlade modules
 - 100 power supplies, 50 batteries (integrated UPS)
 - 100 switch modules
 - 5000 clients

Storage Cluster Hardware



- Model-driven cluster manager
 - Local database stores the system model
 - Replicas maintained with Lamport's PTP protocol
 - Recoverable control operations side-effect the rest of the system
- PTP
 - Two rounds of messages, vote and success
 - 7 msec non-replicated, 14 msec for 2, 3, 4, or 5 replicas, includes a local disk IO
 - Majority quorum needed for a vote to pass
 - Configure 1, 3, or 5 cluster managers

- Deciding to do something is different than getting it done
 - Distributed system components have an uncertain state
 - Control operations can fail
 - The cluster manager can fail
- We did *not* build a full transaction system
 - ACID includes “Atomic”; effects of transactions are not visible until commit
 - Each PTP round is atomic, but a cluster control operation requires more
- Decide – Control – Monitor
 - Commit tentative decision via a PTP transaction
 - Control distributed system elements
 - Conclude operation with a final PTP transaction
 - Monitor and re-evaluate as necessary

Modeling a Node

- Nodes sign onto the cluster via DHCP with extra info
 - Serial numbers and blade type
 - Chassis slot location
 - Software revision
 - System affinity
- Nodes have their own model
 - Boot, Configure, Self Test, Online, Offline, Upgrade, Fail, Power Down
 - pan_monitor nannies processes that should run in different states
 - Current node state exported to cluster manager
- Cluster manager can direct pan_monitor to change states
 - Cluster reboot or power down
 - Off-version node upgrade or downgrade

Lessons From the Node Model

- The cluster manager was primarily concerned with file system services, and the nuances of node management were tacked onto the first version
 - Started with a simple Online/Not Responding/Dead states
 - Now: Booting, Fsck, Off-Version, Low-Battery, Upgrading, Online, Offline, Software Failed, Hardware Failed, Factory Mode, Unavailable (and why)
- Uncontrolled bring-up seemed like a good idea, but what about
 - Network connectivity
 - Sometimes it isn't worth trying to bring up higher level services
 - System wide clock sync
 - Timer mechanisms get wonky when NTP is slewing the clock massively

- System states
 - Rebooting, Online, Paused, Offline, Upgrade, Powering Down
- Node States
 - Booting, Fsck, Off-Version, Low-Battery, Upgrading, Online, Offline, Software Failed, Hardware Failed, Factory Mode, Unavailable (and why)
- Service State
 - Inactive, Active with Backup, Active w/out Backup, Backup
- Bladeset (a collection of storage nodes) State
 - All OK, One MIA, One Down, Two Down
 - Too simple and limiting, it turned out
- Volume
 - Online, FSRC, Reconstruction, Paused, Offline, Down
 - Replacing Bladeset with per-volume storage node scoreboard

- Per state machine modules
 - Encapsulate data model for e.g. Volumes or Nodes
 - Sweeper thread to periodically look for work and verify state
- “Kick” model for complex control sequences
 - Instead of “Start Service” API, it is a “I would like the service to start”
 - API records desired state synchronously, then kicks the sweeper thread and the work happens in the background

- Modeling partial failures
 - Nodes that run most processes fine, but one keeps crashing out
 - Node manager can automatically restart them
 - Cluster manager only cares if it is a managed service
 - Services that process 1000's of requests a second, except for the one request that has been stuck for 20 minutes
 - Service periodically checks for these, and restarts itself if necessary
 - Flakey (customer) network environments cause nodes to flicker on and off the network
 - Might be fooled into a StorageBlade module reconstruction

Summary and Conclusion


- Data driven programming is not a new idea
 - PTP is a handy item in the distributed systems programmer's toolbox
- Start with a reliable distributed system platform
 - Before you do your never-fail, ever-scale, wire-speed storage system
- Recoverable control operations are a fact of life (and tricky)
 - Not practical to push transactional semantics through a 1000 node system
- Control systems are built from interlocking state machines
 - Controlling remote components is inherently uncertain
 - Decide – Control – Monitor
- Modeling partial failures is hard
 - Although yesterdays partial failures (e.g., bad sectors) are well understood



panasas



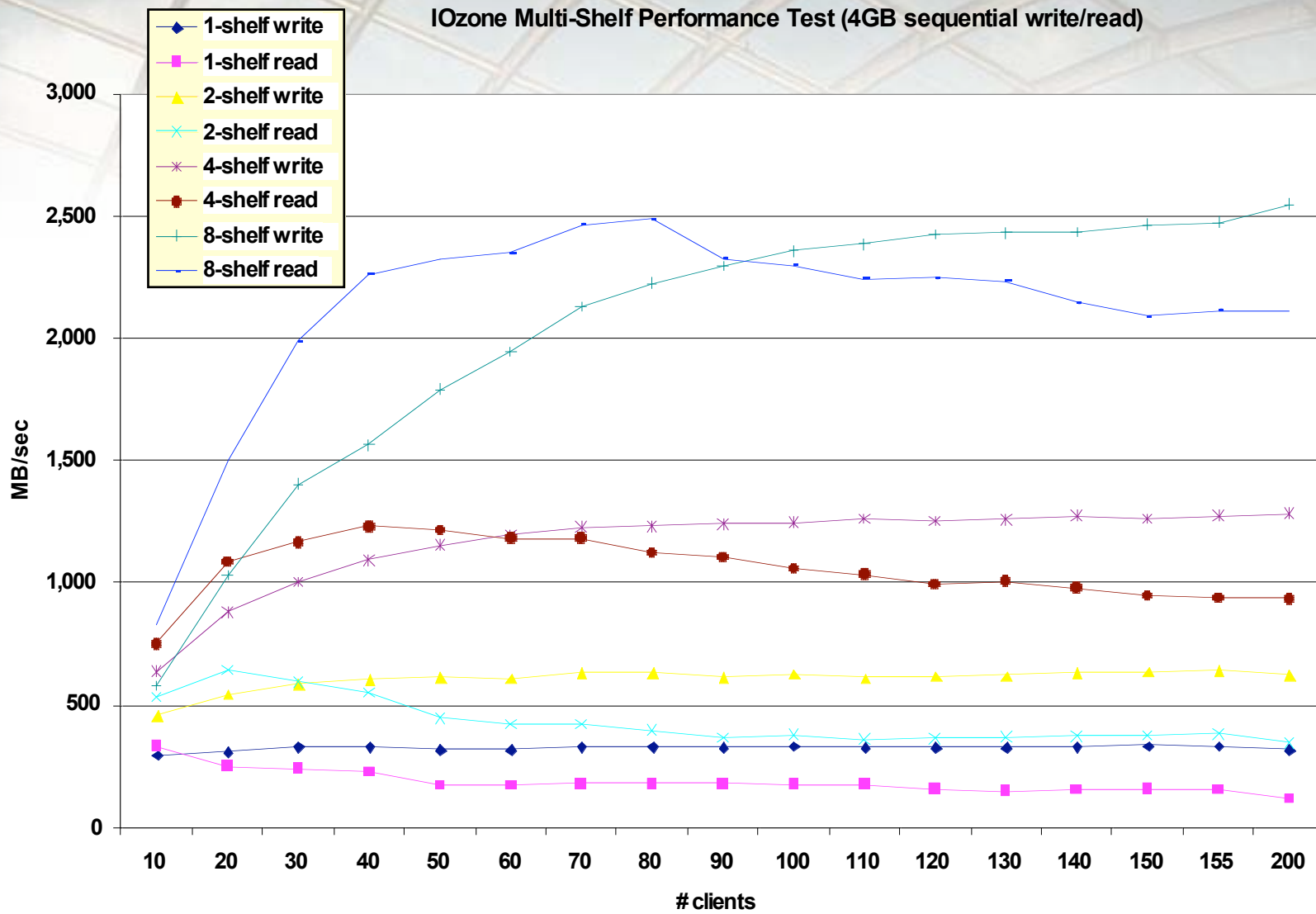
Thank You!



SC07, Nov 2007

- Reliability designed into Panasas Hardware:
 - Redundant power supplies and fans
 - Redundant network connections to each blade
 - Built in UPS for power fail protection
 - ECC memory
 - Backup network built into shelf
- Reliability built into Panasas Software:
 - RAID 1 & 5 data redundancy with scalably fast reconstruction
 - Background, file-aware media, parity & attributes scrubbing and recovery
 - Proactive monitoring including disk SMART, heat, fans, battery
 - Scalable, high performance Backup and Restore
 - Proven FreeBSD base operating system
 - Mirrored Blade OS – protection against errors & repair in the OS partition
 - Systems services failover; file service metadata manager failover
 - Media + Disk failure => rebuild succeeds w/ loss of one file (fenced), not millions of files

Scaling Clients



- DirectFLOW client
 - Standard installable file system
 - Supports all common Linux flavors
- DirectorBlade cluster
 - Divides namespace into virtual volumes
 - Allows metadata to scale (no bottleneck)
- StorageBlade cluster
 - Allows wide striping for large files
 - Read ahead/write behind for small files
- Acts as a highly scalable IP-SAN
 - Parallel, direct client to disk over GE
 - Shared files and scalable metadata
- Demonstrated Scalable Performance
 - 350 MB/sec/shelf scaling linearly to 10 GB/sec with 30 shelves
 - >600 MB/sec/shelf with new 10GE switch

Panasas DirectFLOW™ data path

