

stdchk: A Checkpoint Storage System for HPC Applications

Sudharshan S. Vazhkudai

Computer Science and Mathematics Division
Oak Ridge National Laboratory
vazhkudaiss@ornl.gov

March 19th, 2009

Problem Space: Petascale Storage Crisis

- Scaling to a PF system creates unique challenges for checkpointing
 - Checkpointing is stymied by I/O bandwidth bottleneck issues
 - Current Lustre peak I/O bandwidth for Jaguar is ~ 284GB/s
 - 100,000 cores checkpointing their 2GB of memory takes around 12 minutes
 - This is best case!
- A survey of Tier 1 Jaguar apps suggest that they desire 1GB/s for every TF of peak computing
 - 1 PF computer will need about 1TB/s of I/O bandwidth
 - **Amdahl's balanced system law:** *A system needs a bit of IO per second per instruction per second* [Gray'06 IEEE Computer]
 - 1 TB/s is two orders of magnitude lower than what an ideal balanced PF machine should provide!

More problems...

- Sub-optimal center performance
 - Increased job turnaround time
 - Apps are unable to checkpoint at desired frequency
 - Wasted scratch storage space
 - Checkpoint images not properly cleaned up
 - > 100,000 files/time step; millions of files per run
 - Scratch space is a significant fraction of operations budget
- File systems unable to cope with checkpoint requirements
 - Bursty
 - Write once and hope to read never (in most cases)
 - Transient in nature
- Need ways to improve application checkpoint performance

Approach

- If you cannot afford a balanced system, develop management strategies and tools to compensate
- Exploit opportunities throughout the HEC I/O stack
 - Concerted use of resources can be brought to bear on the checkpointing problem
- Can we create an intermediate storage system to perform impedance matching between application and storage system performance?

Feasibility of a Checkpoint Storage System

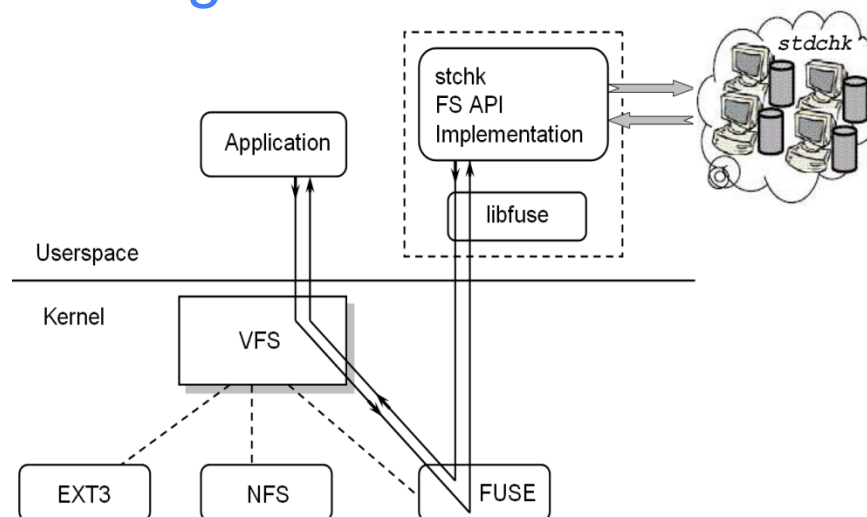
- Many applications oversubscribe for processors in an attempt to plan for failure
 - Use the oversubscribed processors and their memory resources
- Extreme-scale systems offer high aggregate memory and bisection bandwidth
 - Tens or even hundreds of TB/s; Jaguar: 568TB/s
- Machines can potentially be provisioned with SSDs
 - Inexpensive compared to memory, but slower (e.g., fusion-io ssd: 640GB with 1GB/s write bandwidth)
 - Dedicated “fat nodes” with more memory and SSDs as a “staging ground” for asynchronous I/O
- Survey of Tier 1 apps (GTC, S3D, POP, Chimera) and job logs on Jaguar indicate not all memory/core is used
 - Significant amount of residual memory per core
 - Checkpoint image sizes < 10% of the total memory usage

Aggregation at all levels of HEC Storage Hierarchy

- A dedicated storage system, geared towards checkpointing
- Aggregating memory resources from an application's own allocation in large machines
- Similar approach can be used for a checkpoint storage at all levels of the hierarchy:
 - Clusters
 - Abundant node-local storage in mid-size clusters
 - Desktop grids (e.g., a Condor-like system)
 - Unused desktop storage in workstations
 - For high throughput computing (HTC)

Architecture

- Benefactors contribute memory or storage space
- Manager
 - Aggregates contributions, maintains metadata
 - Provides a stripe map (benefactor to chunk mapping) for the checkpointing client
- Writes
 - Checkpoint images are split into chunks and written to benefactors
 - Small scale testing: peak write throughput ~ 700MB/s, sustained ~ 560MB/s on a 10Gb/s cluster interconnect
- POSIX file system API using FUSE



Optimizations

- **Incremental checkpointing**
 - Detection similarity between successive checkpoint images
 - Lowers space and network cost
 - Apps can checkpoint at a finer granularity
 - Challenge is to detect similarity on-the-fly and without significantly slowing down the application
- **Compare chunk hashes from two successive intervals**
 - Chunk hashes maintained at the manager
 - Fixed-size chunks
 - Initial experiments with Fixed-size compare by hash suggest up to 24% reduction in size for 15 min intervals for BLAST checkpoints
 - Content-based
 - High similarity detection ratio but computationally intensive

More Optimizations (really work-in-progress)

- Draining of images from stdchk to stable storage
 - Writes as fast as the ability to mask this operation
- Pruning of checkpoint files
 - Purge images from previous interval once the current image is stored safely
 - File system is unable to perform such optimizations
- Specification in job script
 - Apps can potentially specify lifetime as hints
 - How many nodes for stdchk?
- On-the-fly instantiation
 - Setup stdchk before job startup; discard after job completion
 - Bring Your Own Storage...!

Summary

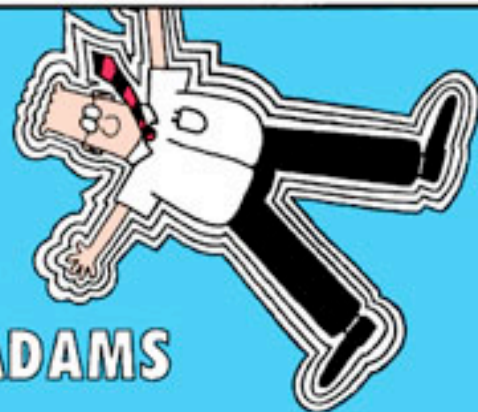
- What the checkpoint storage “is not”:
 - Not intended for long-term storage of checkpoint images
- What it “is”:
 - Low-cost storage for impedance matching between applications and stable storage
 - Intended to facilitate checkpoint-specific optimizations
 - e.g., Incremental checkpoints
 - To be used in conjunction with a parallel file system or other stable storage



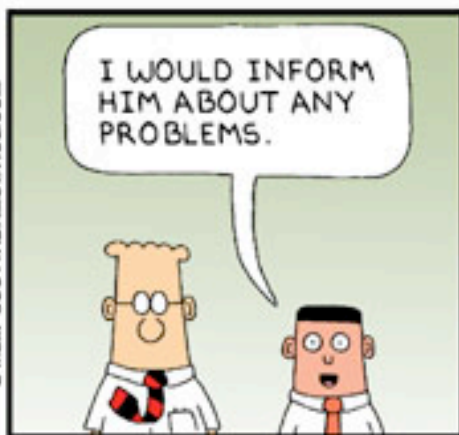
DILBERT®

BY

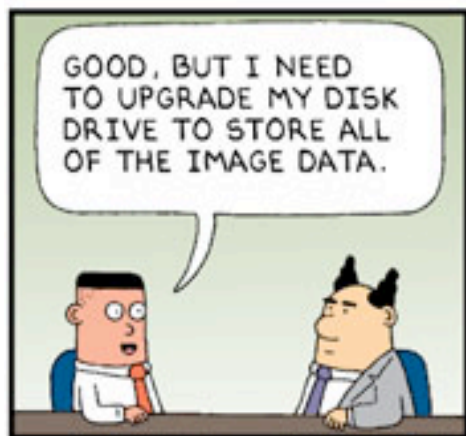
SCOTT ADAMS



E-mail: SCOTTADAMS@AOL.COM



© 2004 Scott Adams, Inc. Dist. by UFS, Inc.



© 1999 UFS



www.dilbert.com

