



SnapMirror

Ohad Rodeh



Introduction

- This lecture is based on **SnapMirror: File-System-Based Asynchronous Mirroring for Disaster Recovery** R. Patterson, S. Manley, M. Federwisch, D. Hitz, S. Kleinman and S. Owara 2002
- SnapMirrorTM is used in Network-Appliance products



Main idea

- Perform asynchronous replication at the file-system level
- Run async-replication between two NetApp filers over the network
- Main results:
 1. When the target lags behind the source by 15 minutes data transfer is reduced by 30% to 80%
 2. Exploiting knowledge about deleted files is crucial to performance
 3. When the target lags behind the source by 30 minutes, response time on the source is reduced by 22%



Main challenges

- Reduce bandwidth
 1. Decide which blocks are dead
 2. Compression
- If source crashes
 1. Target must have consistent snapshot
 2. A half-completed transfer cannot leave the target in an unusable state
 3. Destination must have a recent consistent snapshot to fall back on
- Performance: disk reads on the source and disk writes on the target must be efficient



Requirements for disaster recovery

- Recover quickly
- Recover consistently
- Minimal impact on normal operations
- Up to date
- Unlimited distance
- Reasonable cost



SnapMirror design

- Periodically
 1. Reflect changes in source volume to destination volume
 2. Replicate source at block-level
 3. Use file-system knowledge to limit block-transfer
 4. Transfer only new or modified blocks
 5. Do not transfer de-allocated or deleted blocks



Algorithm

- Take a new snapshot of the source volume
- To determine which blocks to send: compare bitmaps between new snapshot and snapshot from previous update
- Destination jumps from one snapshot to the next when transfer is complete
- Effectively, the entire update is atomically applied to the destination



Notes

- Because
 1. source snapshots are always self-consistent
 2. Snapshots are applied atomically to target
- Therefore: target is always self-consistent
- The administrator sets the SnapMirror update frequency to balance the impact on performance against the lag time of the mirror



Initializing the mirror

- Create a base snapshot and copy it to target
 1. Can also copy all existing snapshots to target
 2. Make target equivalent to source
- The base snapshot serves two purposes
 1. Provide a reference point for the first update
 2. Provide a static self-consistent image which is unaffected by writes to the active file-system during the transfer
- The target is a block-by-block copy of the source

Block level differences

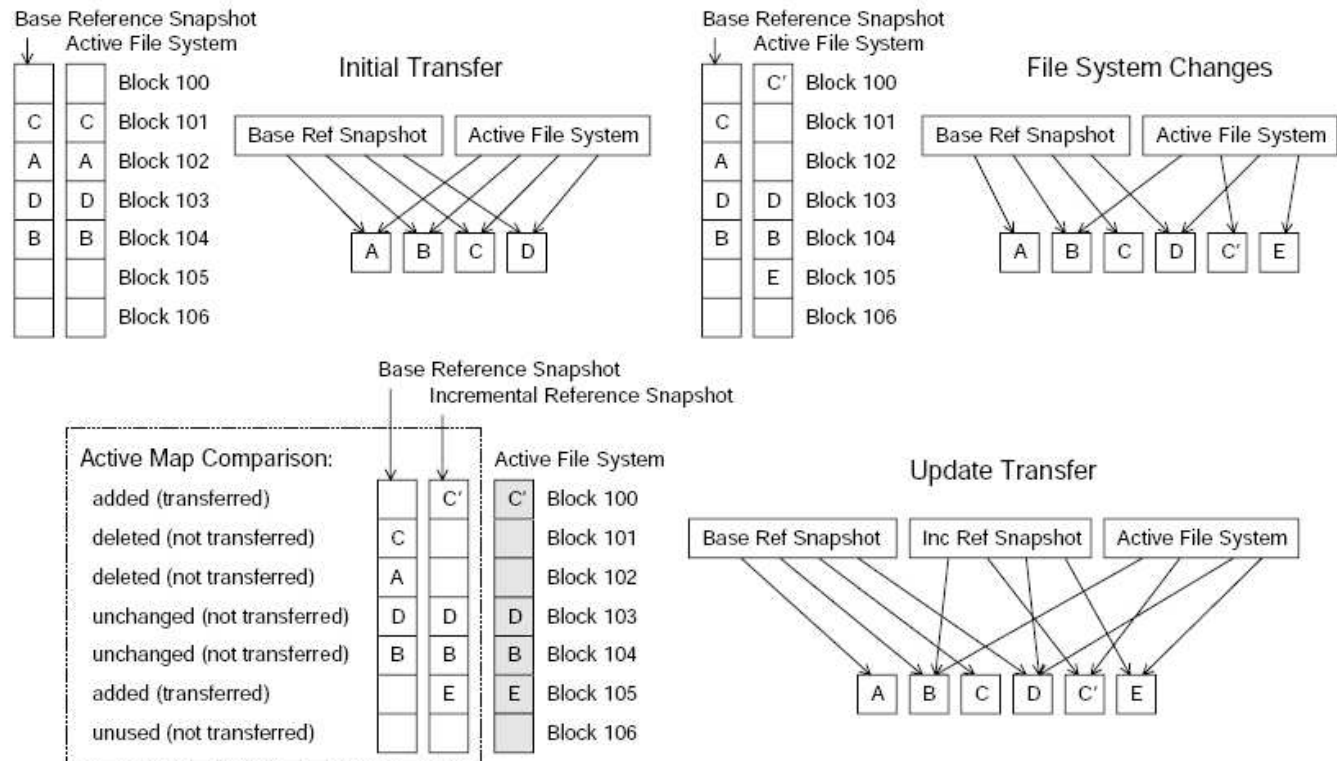


Figure 1. SnapMirror's use of snapshots to identify blocks for transfer. SnapMirror uses a base reference snapshot as point of comparison on the source and destination files. The first such snapshot is used for the Initial Transfer. File System Changes cause the base snapshot and the active file system to diverge (C is overwritten with C', A is deleted, E is added). Snapshots and the active file system share unchanged blocks. When it is time for an Update Transfer, SnapMirror takes a new incremental reference snapshot and then compares the snapshot active maps according to the rules in the text to determine which blocks need to be transferred to the destination. After a successful update, SnapMirror deletes the old base snapshot and the incremental becomes the new base.



Aborted transfer

- The new root block is not written until all blocks are transferred
- Therefore:
 1. A consistent file-system on the mirror is guaranteed
 2. The destination file-system is accessible in read-only state throughout the whole SnapMirror process
 3. Should a disaster occur, the destination can be brought immediately into a writeable state
- The destination can abandon any transfer in progress due to failure at source/network



Advantages

- Mirror is:
 1. Online
 2. Consistent, read-only. Can be made writeable if needed
- The schedule means that impact is as high as the administrator would like it to be
- Under most loads, SnapMirror can reasonably transmit to the mirror many times in one hour
- Works over standard TCP/IP connection



Limitations

- The time required for update is workload dependent
 1. If there are few updates, there is little work
 2. If the entire FS is overwritten, all the FS has to be transmitted
- Also limited by FS size: need to go over the entire free-space map
- The snapshots required for SnapMirror reduce the amount of free-space in the system
- By design, SnapMirror works only on entire volumes. Does not support finer granularity.



Data reduction through asynchrony

- An important premise of asynchronous mirroring is that periodic updates will transfer less data than synchronous updates
- Over time, many FS operations become moot. File data is overwritten or deleted.
- Essentially, periodic updates use the source volume as a giant write-cache



Summary data for traced file-systems

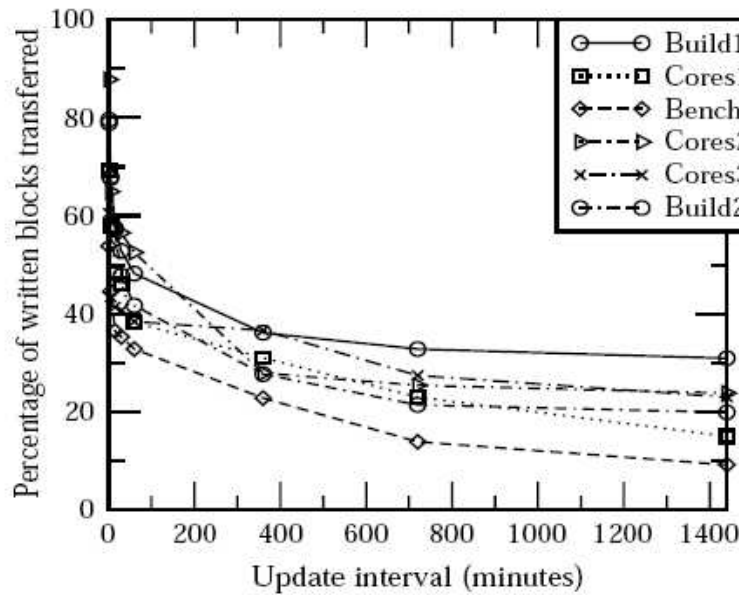
- Trace collection: 24 hours
- Percentage of blocks that were overwritten or deleted
52%-98%
- Average: 78%

Summary data for traced file-systems II

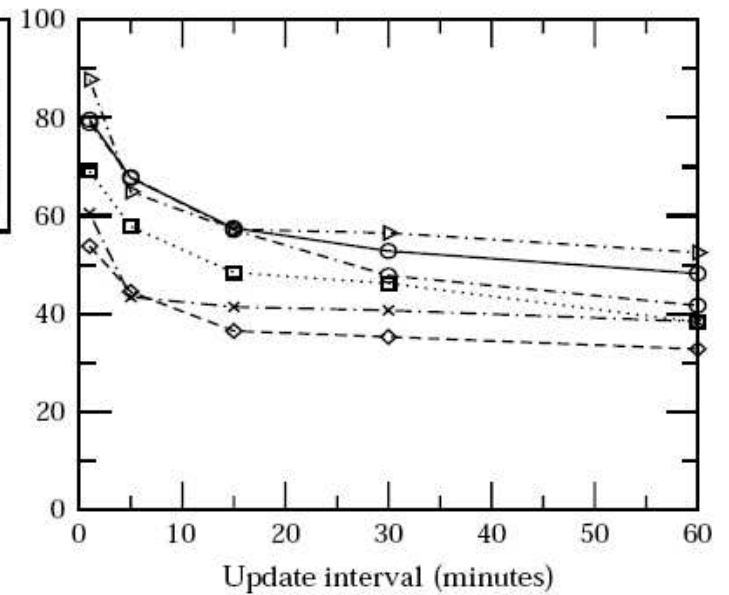
File System Name	Filer	Description	Size (GB)	Used (GB)	Blocks Written (1000's)	Written Deleted (%)
Build1	Ecco	Source tree build space	100	68	7757	69
Cores1		Core dump storage	100	72	319	85
Bench		Benchmark scratch space and results repository	87	56	512	91
Pubs		Technical Publications	32	16	262	59
Users1		Engineering home directories	350	292	10803	78
Bug	Maglite	Bug tracking database	16	11	1465	98
Cores2		Core dump storage	550	400	11956	76
Source		Source control repository	50	36	3288	70
Cores3	Makita	Core dump storage	255	151	1582	77
Users2		Engineering home directories and corporate intranet site	580	470	13752	53
Build2	Ronco	Source tree build space	320	271	34779	80
Users3		Engineering home directories	380	323	15103	85

Table 1. Summary data for the traced file systems. We collected 24 hours of traces of block allocations (which in WAFL are the equivalent of disk writes) and de-allocations in the 12 file systems listed in the table. The 'Blocks Written' is the total number of blocks written and indicates the number of blocks that a synchronous block-level mirror would have to transfer. The 'Written Deleted' column shows the percentage of the written blocks which were overwritten or deleted. This represents the potential reduction in blocks transferred to an asynchronous mirror which is updated only once at the end of the 24-hour period. The reduction ranges from 52% to 98% and averages about 78%.

Change over time in potential savings

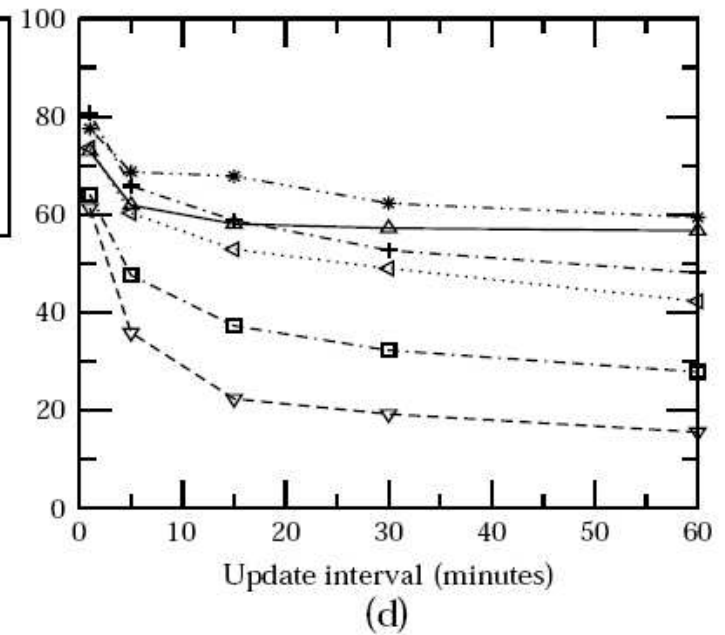
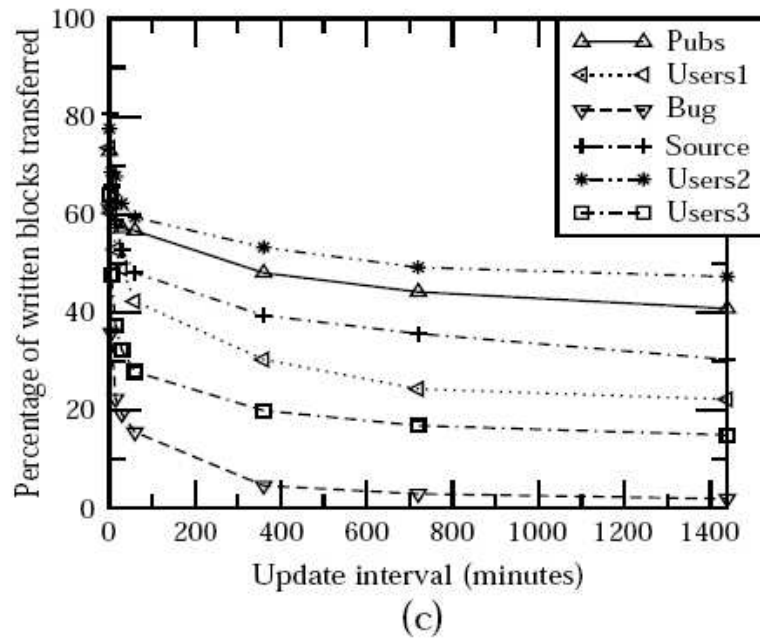


(a)



(b)

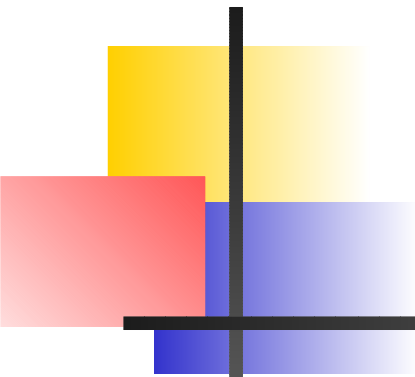
Change over time in potential savings II





Executive summary

Interval	Reduction	Average reduction
1 minute	10% - 20%	
15 minutes	30% - 80%	50%
24 hours	53% - 98%	75%



How important is it to avoid replicating deleted data?

- In a file-system like FFS that reuses blocks, not very important
- When blocks are not reused, it is very important
- Out of 12 file-systems, only two reuse blocks
 1. Build2: 135GB written with 50GB of free-space
 2. Source: 13GB written with only 14GB of free-space

Performance analysis of deleted data

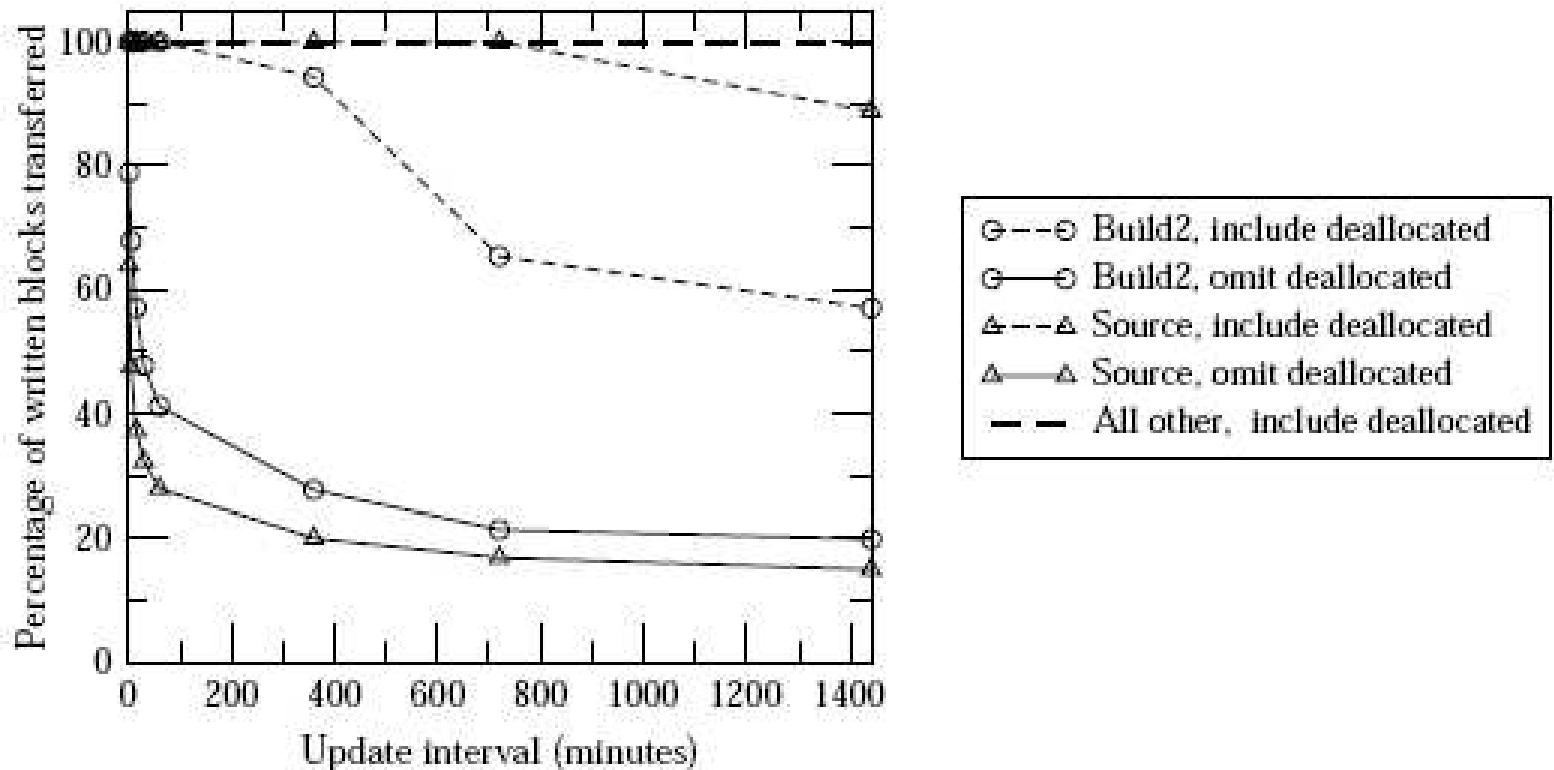


Figure 3. Percentage of written blocks transferred with and without use of the active map to filter out deallocated blocks. Successful asynchronous mirroring of a no-overwrite file system such as LFS or WAFL depends on the file system's active map to filter out deallocated blocks and achieve reductions in block transfers. Without the use of the active map, only 2 of the 12 measured systems, would see any transfer reductions.



Dump/Restore

- Instead of SnapMirror, it is possible to use a generic algorithm that is independent of the file-system
- Dump
 1. Make a pass over the entire file-system
 2. Compare file time-stamps to previous dump
 3. Send over the wire all files that were changed



Dump/Restore II

- Restore
 1. Accept new files over the wire
 2. Create a new file-system that shares the unchanged files with the old snapshot
- Optimization: for every block received, check if the existing file already contains an exact same block

Comparison to Dump/Restore

- Since SnapMirror sends only modified blocks, it sends 39% less blocks over the wire

File System Name	Size (GB)	Used (GB)		Files		System	Data transferred (GB)	Time (sec.)	Rate (MB/s)
		Base	End	Base	End				
Users4	96	63	65	1001131	1054917	SnapMirror	2.1	140	15.4
						logical	4.0	493	8.3
Users5	192	135	150	5297016	6423984	SnapMirror	15.3	797	19.7
						logical	25.2	7200	3.6

Table 2. Logical replication vs. SnapMirror incremental update performance. We measured incremental performance of SnapMirror and logical replication on two separate data sets. Since SnapMirror sends only changed blocks, it transfers at least 39% less data than logical mirroring.

Comparison to Dump/Restore II

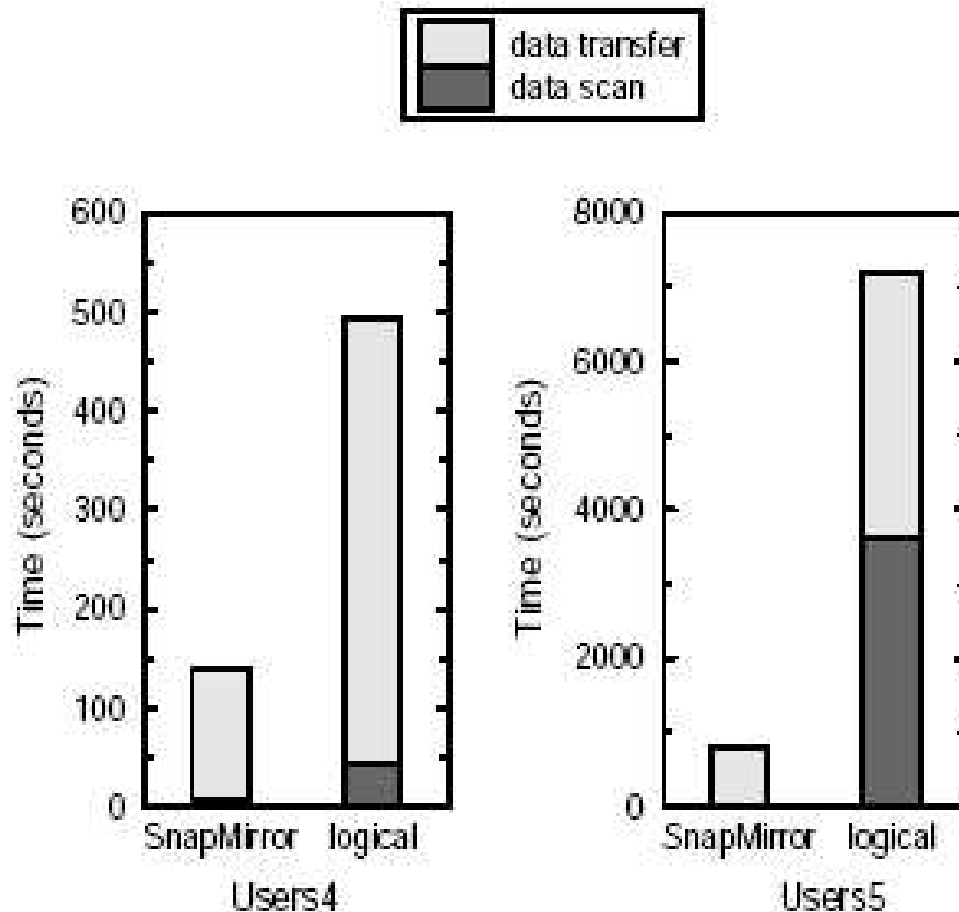


Figure 4. Logical replication vs. SnapMirror incremental update times. By avoiding directory and inode scans, SnapMirror's data scan scales much better than that of logical replication. Note: tests are not rendered on the same scale.)



Comparison to Dump/Restore III

- Logical mirroring is 3.5 to 9 times slower
 1. Time to scan for changes
 2. Efficiency is sending data over the network
- When scanning for changes: better to scan the active map then to traverse the file-system
- When sending data: more efficient to read/write blocks sequentially than to go through the file-system



SnapMirror on a loaded system

- Assess performance on a loaded system running SnapMirror
- Run SpecSFS against a NAS box which is also running SnapMirror



Testbed

- NAS box with a live file-system
- Number of clients running SpecSFS against the filer at a specific load
- 48 client processes
- 6 client machines
- Client machines: 167Mhz Ultra-1 Sun workstations running Solaris 2.5.1
- Switched Ethernet 100Mb/sec



Server

- F760 filer
- 21 disks, 18GB, 10K RPM
- 320GB volume
- Alpha 21164 processor, 600Mhz
- 1024MB RAM
- 32MB Non-volatile memory

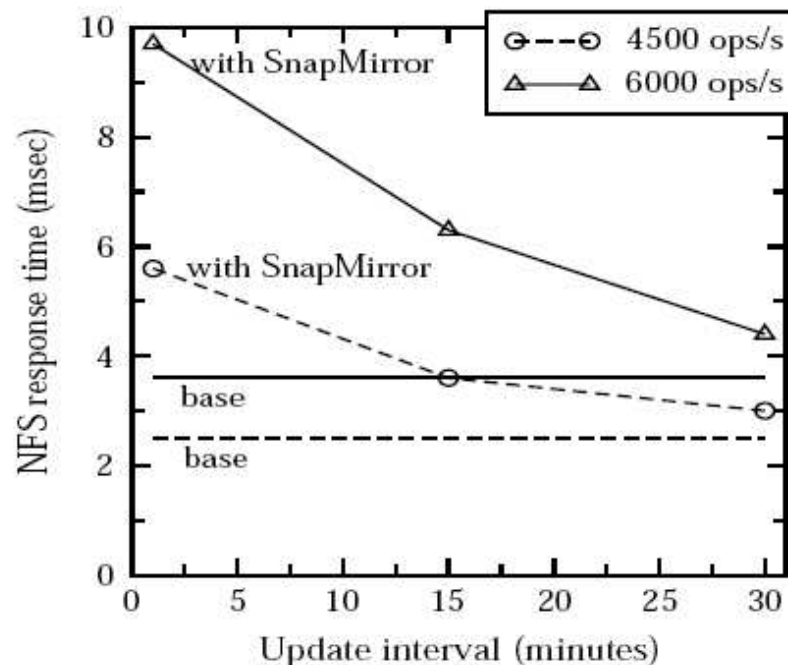
Performance

- Resource consumption decreases when update interval decreases
- When the update interval is 30 minutes, additional load is low

Load (ops/s)	Update Interval	CPU busy	Disk busy	SnapMirror data (MB)
4500	base	66%	34%	0
	1 min.	93%	50%	12817
	15 min.	74%	43%	6338
	30 min.	69%	40%	2505
6000	base	87%	54%	0
	1 min.	99%	67%	13965
	15 min.	94%	62%	8071
	30 min.	91%	60%	3266

Response time on a loaded system

- When the update interval is 30 minutes the delay in response time is only 22%





Summary

- SnapMirror shows a file-system level approach to asynchronous mirroring
- It has good performance and is deployed in the field
- It is important to compare performance against another implementation