RJ10427 (A0801-006) January 23, 2008
Computer Science

IBM Research Report

Fast Metadata Update by Using Two-Phase Commit on Volatile Memory and Disk

Tarun Thakur, Veera Deenadhayalan, Karan Gupta, Paul Muench
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

1 Abstract

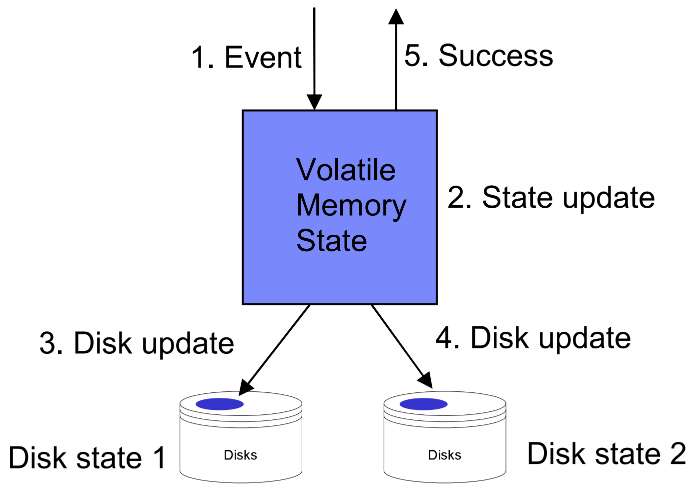
Efficient metadata management is a key aspect of a robust storage system. Traditional models of metadata management used in most modern storage systems are too restrictive and inefficient. The two key requirements on metadata management systems is a) consistency – integrity of the metadata is crucial at all times and under failures for correct functioning of a system, and b) reliability – system will reliably maintain the event and/or metadata update once a success is returned. There are instances when metadata update becomes a bottleneck in the system and in such cases, any improvement in metadata update performance will directly improve the system performance. By using the method described in this research report, a storage system can improve its metadata update performance with out incurring additional hardware costs and yet meet the client guarantees for metadata consistency and recoverability.

2 Related Work

2.1 Two-Phase commit on disk

Traditional storage systems cache the metadata in volatile memory and harden it on disk as and when the metadata changes. Even if a storage system uses nonvolatile memory to protect user data from power failure, it typically does not use the nonvolatile memory to store its metadata. In most cases, the reliability needs of the metadata are not met by the underlying nonvolatile memory that may have limited battery life. In these systems, as and when the metadata changes in volatile cache memory, it needs to be immediately hardened to disk, before posting completion to clients requesting the metadata update. To guarantee correctness, the metadata is updated in volatile memory using single phase commit and on disk using a two-phase commit protocol, which then permits rolling forwards or backwards if there is an interruption during the metadata update process. To meet the reliability needs, the metadata may be stored on multiple disks. The drawback of this method is the associated latency involved in writing to disk as and when the metadata changes; the clients have to wait till the metadata is updated on disk to guarantee that the state transition is persistent, thus directly impacting the overall system performance.

Overview of 2-PC on Disk

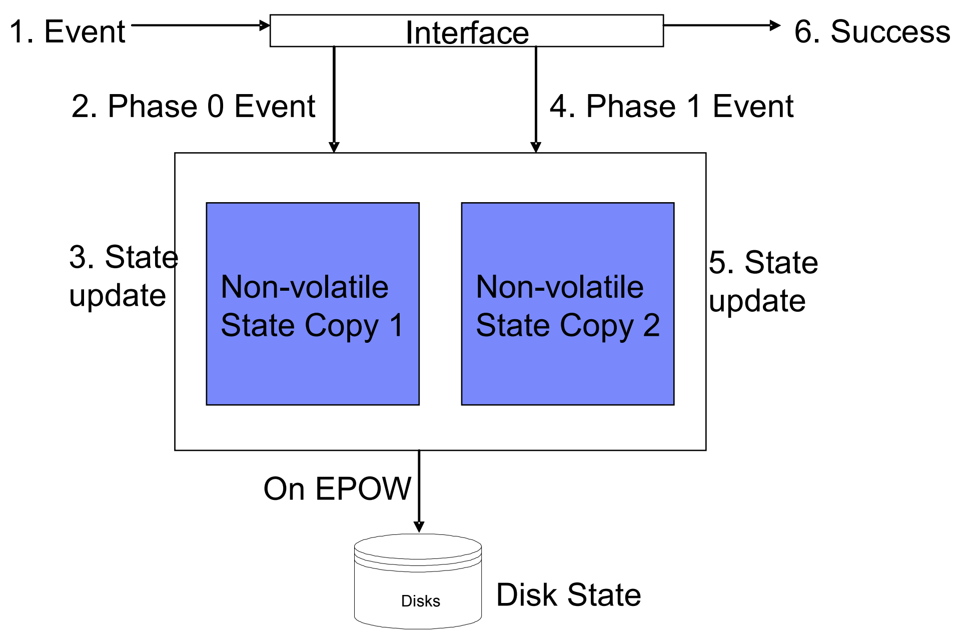


- For consistency requirements we need to update the 2 metadata states on disk in serial.
- Latency is **high** as the event has to wait for 2 commits on the disk

2.2 Two-Phase commit on memory

There are some modern storage systems that use nonvolatile memory to store metadata and achieve the same reliability guarantees, but improved metadata update performance. These systems typically don't just depend on battery protection, but use mechanisms such as Early Power-Off Warning (EPOW) to trigger the hardening process to disk. We may call such a nonvolatile memory system as highly resilient. In such a system, the trigger to harden the metadata is independent of when clients update the metadata. Thus, using only a single phase commit of metadata on memory could result in inconsistent metadata across power failures. To guarantee consistency of the metadata, which is required to guarantee correctness of the metadata update algorithm, these systems require the use of two-phase commit protocol in nonvolatile memory instead of disk. It should be noted that this is the primary difference when compared to work in section 2.1, where the two-phase commit protocol is used on disk and not in memory. However, there may be multiple copies stored on multiple disks to meet the reliability needs. The latency introduced by the additional phase, which does not involve any disk access, is much smaller compared to the disk latency required in "Two-Phase commit on disk" model. From the client's perspective, the nonvolatile cache memory offers much higher metadata performance primarily because it doesn't involve hardening the metadata to disk during normal operation. The drawback with this scheme is the requirement of a highly resilient nonvolatile cache memory to store the metadata, especially given that lower cost systems can't afford it.

Overview of 2-PC in Memory



• Latency is **low** as only non-volatile memory needs to be updated

• Failure on EPOW could result in **loss of state**

2.3 Issues with related work

	System Properties			System Benefits		
	Memory	**Hardening Algorithm**	**Commit to Disk**	**Warmstart Recovery**	**Cost**	**Update Latency**
2-PC on Disk	Volatile	▪ 1-PC in memory ▪ 2-PC on disk	Continuous	Read consistent copy from disk	Low	High
2-PC in Memory	Non- Volatile	▪ 2-PC in memory ▪ 1-PC on disk	EPOW	Consistent copy in memory	High	Low

2.4 Need for a New Solution

Now, consider a lower cost storage system that requires the improved metadata update performance, similar to that achieved in "Two-Phase commit in Memory" methodology, but cannot afford the highly resilient nonvolatile memory. The novel method described in this report here effectively combines the two methods described above to achieve improved metadata update performance using just volatile memory.

3 Summary

3.1 Brief Description

The goal of this research report is to propose a novel method that improves the metadata update latency of a storage system that can only afford volatile memory and disk to harden the metadata.

Section 2.1 describes a methodology that performs single-phase commit on volatile memory and two-phase commit on disk. Section 2.2 describes a methodology that performs two-phase commit in non-volatile memory. By combining the ideas described in section 2.1 and 2.2, viz., two-phase commit in volatile memory and two-phase commit on disk, and by batching unrelated metadata updates, we can bring the benefits of both the worlds together – fast metadata update on a system that does not have highly resilient nonvolatile memory.

3.2 Advantages

The core idea briefly described above has the following advantages:

1. Improved metadata update response time in traditional storage systems that cannot afford highly resilient nonvolatile memory, but can only afford volatile memory and disk to store its metadata
2. The method explained alongwith is a novel algorithm which deals with recovery of system control state post a system failure
3. The method explained alongwith proves that the algorithm can recover cluster state data with failure on failure scenario and will assist in making forward progress in the system

4 Description

4.1 Algorithm background

The **2-phase-commit (2-PC)** protocol in the distributed algorithm context:

1. A coordinator node generates **a transaction** (stimulus) for all the nodes in the distributed system
2. Each node in the **first phase** can either reply to the coordinator node with **an accept** or **an abort** response
3. If all the nodes reply with **an accept** message to the coordinator node, the coordinator node starts the **second phase** with **a commit** stimuli
4. If any of the nodes reply with **an abort** message to the coordinator node, the coordinator node starts the **second phase** with **a rollback** stimulus

The **2-phase-state** update protocol: This protocol can be used in a single system to always have one consistent state even with the application of new stimuli or a batch of new stimuli. This property when coupled with **roll-forward** or **roll-back** of consistent state provides fast recovery from failure scenarios. Traditionally this type of state update protocol is implemented in systems with non-volatile memory. Here we present an algorithm which extends the application of this 2-phase-state update protocol to the systems with volatile memory and persistent storage.

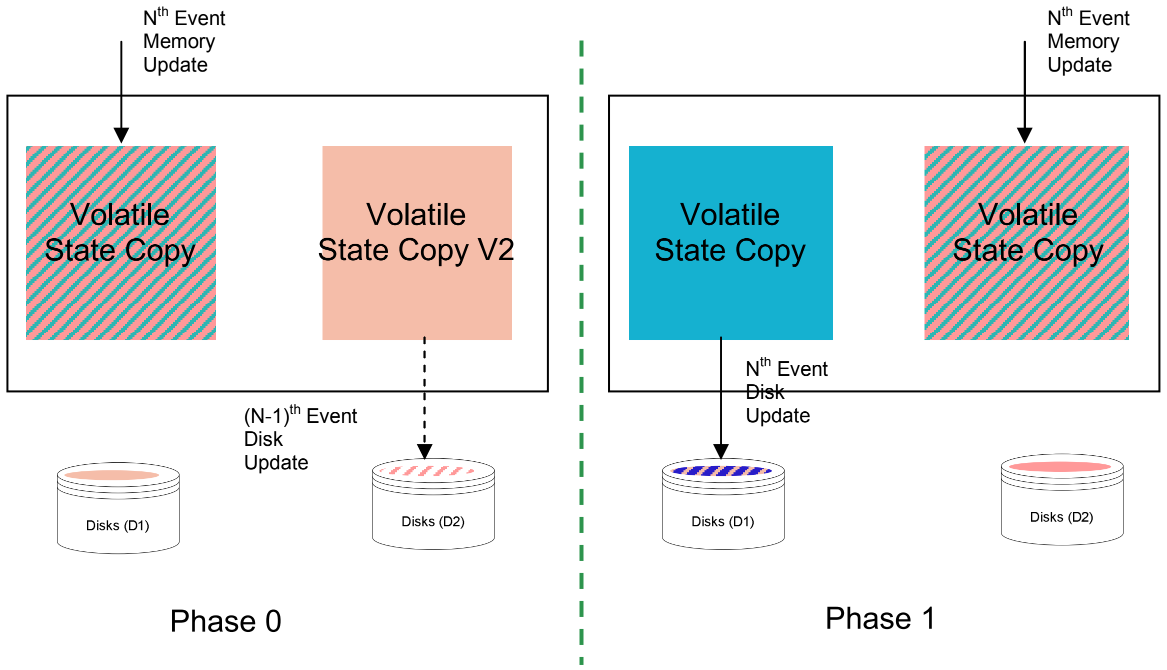
The benefit of this protocol for these systems is:

1. Consistent state aids in failure recovery
2. Performance benefit in terms of disk I/Os by virtue of batching of stimuli updates.
3. In systems with "warmstart" capabilities we can recover from memory without requiring read from the disks.

4.2 Algorithm Terminology

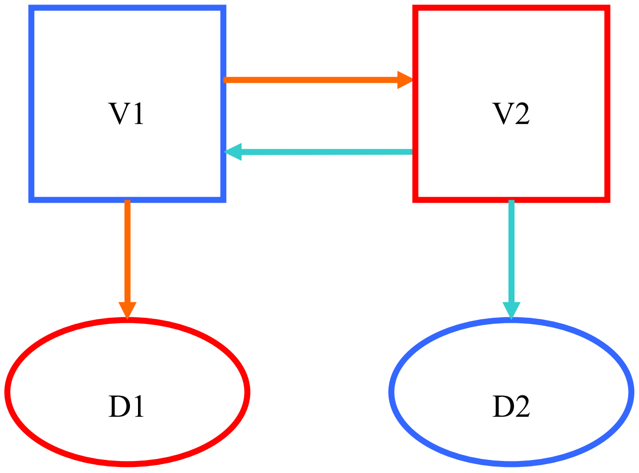
Terms Used	**Description**
V1, V2	2 copies in volatile memory.
D1, D2	D1 and D2 copies represent the copies (of cluster state data) stored on disk.
T	T implies *Toggle*. Toggle operation implies to mark the copy as inconsistent because we will update the contents of this copy of data in the following step.

4.3 Algorithm description



Parallel 2-PC: Phase Snapshot

The algorithm is based on parallel update of the contents in virtual memory and disk state. In the diagram below, the blocks and arrows with same colors represent the states which are tied together for update.



Based on this representation we can infer that,

1. Update of V1 (application of new stimuli) will be in parallel with update of D2 (V2 -> D2). We will refer to it as **Phase-0** of 2-phase-state update protocol.
2. Update of V2 (application of new stimuli) will be in parallel with update of D1 (V1 -> D1). We will refer to it as **Phase-1** of 2-phase-state update protocol.
3. The transition or **Toggle** between the two phases will have to be in parallel (simultaneous) as depicted by arrows in the above diagram. This also becomes the synchronization point as updates are only applied upon these phase boundaries.

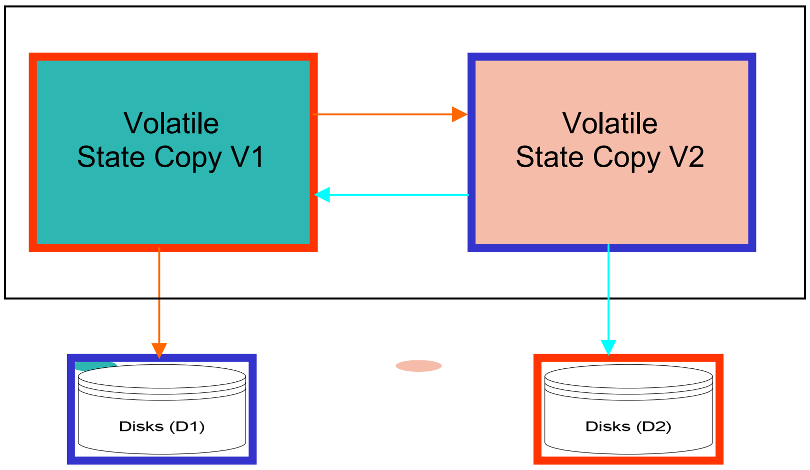
This described algorithm will guarantee that at any moment in the system:

1. There will be a consistent state both in memory and disk
2. The consistent state on disk will never be more than 1 stimuli batch behind the consistent state in memory.

Toggle algorithm for the state transition and proof of at least one consistent copy on the disk. Inconsistent state in the memory context is that a new batch of stimuli is being applied. Inconsistent state in the disk context is that a consistent state in memory is being made persistent on disk. In the table below,

- A value of 1 under either of 4 copies implies inconsistent state, and 0 implies consistent state
- Row in red is the transition or toggle point between phases or a phase boundary
- Rows in black are just intermediate states a system can be in without transitioning a phase
- There is no order in between the 2 intermediate states possible within a phase. Either of the 2 intermediate states can occur first. But both of them will occur and only once each
- These are the only states a system can be in given the above constraints

Parallel 2-PC: System States



V1	V2	D1	D2	
1	0	0	1	
1	0	1	0	Phase 0
0	1	0	1	
0	1	1	0	
1	0	1	0	Phase 1
0	1	0	1	
1	0	0	1	

Given the possible states a system can be in from above table, it is proved that at any given point we will have a valid consistent state both in disk and in memory.

Phase state variable:

1. This variable gets toggled at phase boundary and based on its value we know whether we are in Phase 0 or Phase 1
2. In the case of system recovery during warmstarts if we want to recover into a particular phase and want to avoid reading the disk, we will have to maintain a "**phase state**" variable
3. In normal operation also its toggle becomes the trigger point for state update

5 Analysis

5.1 Metadata State Update Time

Algorithm	**Metadata update Time**
2-PC on Disk	V + 2D
2-PC in Memory	2V
Parallel 2-PC	*Max (2V, 2D)*

- "V": Metadata update time in memory; "D": Disk update time
- In comparison to 2-PC on Disk if "V << D" then "Parallel 2-PC" performs no worse than 2-PC on Disk. For all other case where "V <= 2D", our methodology of "Parallel 2-PC" performs better than 2-PC on Disk
- In comparison to 2-PC on Memory if "V >> D" then "Parallel 2-PC" not only performs no worse than 2-PC on Memory but also provides consistent metadata state on disk and does not have to worry about failures on EPOW.

5.2 Comparison of all three arts

	System Properties			**System Benefits**		
	Memory	**Hardening Algorithm**	**Commit to Disk**	**Warmstart Recovery**	**Update Latency**	**Cost**
Parallel 2-PC	Volatile	▪ 2-PC in memory ▪ 2-PC on disk ▪ nth phase in memory and (n-1)th phase in disk are in parallel	Staged	Consistent copy in memory	Medium	Low+
2-PC on Disk	Volatile	▪ 1-PC in memory ▪ 2-PC on disk	Continuous	Read consistent copy from disk	High	Low
2-PC on Memory	Non-Volatile	▪ 2-PC in memory ▪ 1-PC on disk	EPOW	Consistent copy in memory	Low	High

The difference between our new model and the older one is that in case of power rest failures we will always do a **roll-back.** This might imply application of a stimulus more than once, but it will be from the same consistent state. So if the agent have case 4b they should be fine. In case of disk failures, we are no worse than any of the prior arts.