Provenance Based Rebuild: Using Data Provenance to Improve Reliability

Technical Report UCSC-SSRC-11-04
May 2011

Brian A. Madden
madden@cs.ucsc.edu

Ian F. Adams
iadams@cs.ucsc.edu

Mark W. Storer
mwstorer@netapp.com

Ethan L. Miller
elm@cs.ucsc.edu

Darrell D.E. Long
darrell@ucsc.edu

Thomas M. Kroeger
tmk@soe.ucsc.edu

Storage Systems Research Center
Baskin School of Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064
http://www.ssrc.ucsc.edu/

Provenance Based Rebuild: Using Data Provenance to Improve Reliability

Brian A. Madden[†] Ian F. Adams[†] Mark W. Storer[‡] Ethan L. Miller[†]
Darrell D.E. Long[†] Thomas M. Kroeger[*]

[†] University of California, Santa Cruz [‡] NetApp [*] Sandia National Labs

Abstract

Traditionally, data preservation and reliability have used error correcting codes (ECCs) to ensure data safety. The development of general data provenance tracking systems provides a new opportunity for data reliability. We present a method that utilizes provenance to determine a datum's generating process and inputs, and then uses this information to recompute lost data. This method, called Provenance Based Rebuild (PBR) provides a new, complimentary reliability mechanism that integrates with traditional systems to offer a variety of benefits including fine grained prioritized rebuild and parallel rebuild. While PBR offers benefits that address weaknesses in current techniques, it also faces a number of challenges such as data placement, and infrastructure provisioning.

1 Introduction

The advent of general data provenance capturing systems, such as the Provenance Aware Storage System [9], offers the ability to track the processes, system state, and inputs used to generate a piece of data. By storing a datum's provenance along with the process and initial inputs that created it we have an effective set of tools for recomputing data in the event of corruption or loss. If such a failure were to occur these tools could be used to rerun a process with the correct initial inputs to recompute the lost data. This method of reconstruction, which we call *Provenance Based Rebuild* (PBR), solves many of the shortcomings of ECCs. PBR provides a number of novel benefits and improvements including: fine grained control over what files are rebuilt, the ability to prioritize the order in which those files are rebuilt, and potential for per-file parallelism during rebuild. While offering a number of benefits, utilizing PBR requires considerations such as provenance collection, data placement, and infrastructure provisioning.

Data protection typically comes in the form of error correcting codes (ECCs). Note that replication can be seen as a special case of an ECC where correcting errant data simply means copying from a replica. ECCs unfortunately suffer from drawbacks such as propagating block level errors silently. If an error occurs on write, or data is written to the wrong location, an ECC's parity data will also be incorrect as it is a computational result of the original data. Silent errors are dangerous as they are not explicitly caught by the storage system, and errant data may be used in future computations resulting in more erroneous data, with the potential for significant consequences.

ECC based reliability techniques also face challenges as the areal density of hard drives continues to grow. The number of input/output operations per second (IOPS) that a drive can achieve has not grown proportionally to the areal density, and as a result ECC rebuild times are growing [7]. Growing rebuild times are more than just a minor invonvience. The time between failure and recovery is a critical period where additional failures could cause complete data loss. Despite this danger, it is not uncommon to find rebuild times in the tens of hours.

ECCs also lack any sort of semantic knowledge of the data. If a particular file or group of files are more important than others there is no good way to prioritize those files should a rebuild occur. Additionally, the most common ECC configurations are incapable of rebuilding data at multiple granularity levels. Instead, the smallest unit that can be rebuilt is a fixed sized parity group, leaving individual file rebuilds out of the question. Despite these problems many trust their data to ECC systems. Many of these problems are detailed by Appuswamy [4], but whereas they see a need to fundamentally change the way that ECC techniques are implemented, we see an alternative approach that complements ECCs to create a system more robust than the sum of its parts.

Despite the shortcomings of ECCs, a solution that provides benefits such as fine grained rebuild control does not require fundamentally altering the way that current offerings function. By creating a new independent form of reliability, both systems can be utilized in tandem to capitalize on each system's relative strengths. PBR is a system well suited to address many of the problems left by traditional ECC based systems.

2 Background

We start by defining some basic terminology to eliminate any confusion in our discussion. A *process* is any action that transforms or outputs data. We refer to any data or parameter that is used by a process as an *input*. Similarly, the data resulting from the running of a process is an *output*. Outputs may be used as input to other processes; such outputs are referred to as *intermediate results*. A relationship between inputs, outputs and processes is stored in an output's *provenance* which maps the lineage and necessary inputs and processes to recreate an output.

Provenance based rebuild (PBR) is the term we give to a system in which provenance is utilized in conjunction with original processes and inputs to reproduce data. By treating a datum's provenance as a road map to reconstruction, and reading the provenance chain backwards from our desired datum, we can determine what processes, inputs, shell variables, and dependencies were used to create that datum. Once the system state, inputs, and processes have been determined, the system can use this information to rerun the processes necessary to recompute the lost datum.

3 Architecture and Overview

The PBR architecture is composed of four major components: the underlying compute system, the provenance collection agent, the PBR store, and a rebuild system. Figure 1 shows the components of the PBR architecture.

The provenance collection agent is a software component, such as the Provenance Aware Storage System [9], that sits low in the software stack of each node in the compute system and collects data on how outputs were produced. Ideally this agent would be a privileged entity that could track fine granularity data such as shell variables, installed libraries, inputs including random seeds, processes, and any other data items useful in recreating the state of a compute system at the time a particular datum was generated.

During normal operation (non-recovery mode) the provenance collection agent would store the provenance, inputs, and processes on the PBR store. An input or process used in more than one computation would only be stored once to avoid unnecessary storage overhead. When recovery mode is activated a special marker is written to the provenance chain to indicate the start of a rebuild. This marker ensures that any new provenance written to the chain during a rebuild is not accidentally included for rebuild as well.

A PBR store is a physically distinct storage system separate from the compute system's main storage that is responsible for only the provenance, inputs, and processes used by PBR for data regeneration. Ideally this store would be protected by a combination of ECCs and replication such as a RAID system. By backing the crucial data needed to rebuild using PBR with a more traditional reliability system a single point of failure is eliminated.

Lastly, the rebuild system is any compute system capable of running the processes needed to rebuild data using PBR. This system can be as simple as reusing the original compute system. Ideally, the rebuild system would be something akin to the cloud [1,2], highly available and scalable so that the benefits of per-file rebuild, and parallelism can be exploited. The rebuild system will host a small software layer that is capable of reading a provenance chain and computing the set of file necessary for rebuild. It can do this by reading the chain forward from a given rebuild marker (or the start of the chain) to the most recent rebuild marker written by the collection agent. As the chain is read PBR determines the set of files resident on the system at the time of data loss.

As the chain is read, files created during that length of chain are added to the list, and any files deleted during that length of chain are removed from the list, yielding a complete set of files present on the system at the time of data loss.

Once the set of files to be recomputed has been established, the rebuild system will create a build script describing how each file was originally generated. The script will be responsible for setting shell variables, installing dependencies, and running the process with the correct inputs. Each build script will be bundled with the correct process, inputs, and necessary dependencies forming a self contained software bundle known as a rebuild unit RU. These RUs can be composed into larger shared dependency units (SDUs) to exploit locality and reuse common dependencies among RUs. For example, if a number of processes all rely on the same cryptography libraries they would be composed into a SDU so that the dependency would only need to be bundled, and in-

stalled once. Note that the issue of creating build scripts and replicating prior software state for re-computation is a difficult problem. Care must be given when generating these scripts to avoid software conflicts.

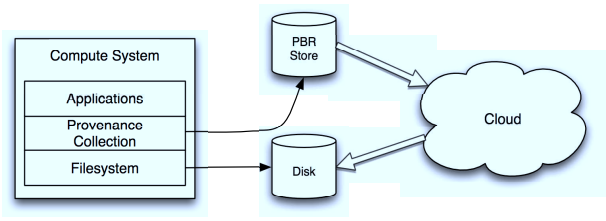

Figure 1: PBR Basic Overview: The four basic components: a compute system; a provenance collection agent; a PBR store for storing provenance, inputs, and processes; and a re-compute system, in this example the cloud, for use when data must be regenerated.

4 Benefits of PBR

The PBR system is capable of addressing the problem of growing rebuild times. This is immediately visible for files that have a small set of input data and a large output. In these cases PBR has to read minimal data for recomputation whereas ECC would need to read 6-8 times the size of the rebuilt data to recover. The second way PBR alleviates long rebuild times is by tracking data provenance on the file level. With provenance we have a way to rebuild each file independently of one another. Once a file's provenance, process, and input is read from disk they can be recomputed locally or distributed for processing elsewhere. By offloading rebuild elsewhere, bottlenecks such as lack of local resources can be overcome; live systems do not have to be interrupted by rebuilds; and per-file parallelism can be achieved. This parallelism is a boon to reliability, particularly as areal density continues to outpace IOPS, as PBR recovery time is only limited by the slowest process amongst all rebuild processes.

To see the benefits of rebuilding in parallel consider a site that provides video on demand services to their customers. Videos are available in a number of formats and resolutions to provide good coverage to customers, but a disk has failed and a number of versions of a particular video have been lost. Using PBR the missing video formats can be recomputed in parallel allowing for lost formats to become available as their process finishes rather than waiting for the length of a full disk rebuild.

Another area that PBR can offer benefit over RAID is in its ability to ensure greater safety from block level errors. If a block level error does occur, total loss of that data is not assured. Provenance data describes how data was generated rather than being a computational derivative of it. If a block level error is found in an output, the process that created it can be re-run with the initial inputs to recompute the data. Additionally, if errors are found in data due to a bug in a process, that process can be swapped out with an equivalent or updated process and output recomputed using the updated process instead. Process swapping offers a few benefits beyond being able to re-create data from an updated, bug free process. It also allows users of PBR to decide to swap out old, possibly slow processes with newer, more efficient versions. Take for example a process that uses a serial algorithm to compute its output. If a newer, parallelized version of the algorithm exists, the process can be replaced and any future rebuilds will benefit from the new algorithm.

The descriptive nature of provenance also offers semantic insight into the data. By combining the fine granularity of PBR with the semantic insight that provenance offers we have the ability to prioritize file rebuilds, either via human interaction or heuristics. Prioritizing rebuilds are a particularly attractive option in situations where some data may be more valuable than others. For example, if a disk fails and data must be rebuilt, more recently accessed data may be rebuilt first with the assumption that it is more relevant, thus preventing complete data loss in the case of additional failures. The ability to prioritize rebuilds may make the difference between meeting a service level agreement (SLA) and being in breach of contract. As an additional benefit, in cases where a process is capable of streaming partial outputs as it operates, data can be streamed out and used in subsequent computation as the rebuild is happening, minimizing the amount of down time associated with disk failure.

In addition to parallel rebuild, prioritized rebuild, and safety from block level errors, PBR offers the benefit of diversity. As an orthogonal method of reliability PBR suffers from a different set of failure modes. Adding additional ECCs to a traditional setup, for example, may provide benefits, but it does not protect against failures such as an errant RAID controller. Using a combination of techniques reduces the likelihood that both mechanisms will fail simultaneously from a problem.

Lastly, a tradeoff between storage and computation exists; this tradeoff focuses on foregoing the storage of infrequently used intermediate or final outputs in favor of recomputing them later if needed. Utilizing a PBR system offers the added benefit of sharing much of the infrastructure that would be required to consider such a tradeoff. A more in depth discussion of this tradeoff can be found in

Maximizing Efficiency By Trading Storage for Computation [3].

5 Challenges and Tradeoffs

There are a number of challenges and tradeoffs that must be addressed to realize the benefits of PBR. These include provenance collection, data placement, and infrastructure provisioning.

There is a tradeoff in provenance collection, concerned with striking a balance between too much, and too little provenance information. With perfect provenance, or an exact history of a datum, we can make strong guarantees that rebuilding the datum with PBR will result in a bit for bit copy. We can make this guarantee as we have the necessary information to reconstruct the exact software state of the machine that was originally used to generate the data. As the detail of the provenance becomes more coarse, for example if random seeds or shell variables are not captured, the guarantees become weaker. The cost of capturing too much provenance means increased storage overhead and provenance graph pruning when a rebuild is necessary. The cost of capturing too little provenance results in weaker reliability guarantees, for example, not capturing random seeds may result in a rebuild that is not an exact bit for bit copy.

Another tradeoff in a PBR system is the placement and protection of the constituent data. The protection of the data provenance, inputs, and processes is vitally important – without these data the entire rebuild system is rendered useless. Placing the provenance, inputs, and processes together may yield better storage utilization but presents a danger in the face of correlated failure as the necessary data to recompute using PBR may be lost. Despite the difference between RAID and PBR systems we believe that many of the age old data placement strategies are still applicable. Initial inputs, provenance, and processes should be stored separately and redundantly to prevent a single point of failure. Furthermore, distinct geographic locations should be chosen for the replicas to prevent correlated failures. The choice of additional protection ultimately affects utilization of the PBR store, performance of the system should the PBR data become corrupt, and the overall reliability of the system.

Challenges such as the provisioning of resources must also be addressed. We believe that one of the greatest advantages of the PBR system is the parallel, and prioritized rebuild. For many, provisioning enough resources to rebuild a large number of files in parallel may be difficult or impossible using only local assets. We believe that the ideal medium for PBR rebuild is in the cloud. With the ability to easily grow and shrink resources as needed, the cloud provides an essential ability to provision large numbers of compute nodes as necessary. While there is still some challenge in determining the optimal number of nodes to acquire, and the best rebuild schedule to optimize the use of those resources, an ideal PBR system would include a scheduling utility for balancing the time to recovery and resource acquisition.

A final challenge in PBR is the issue of the heterogeneous nature of the rebuild hardware. Systems and hardware are constantly changing, and these changes may present problems when trying to run old processes on new machines. In some cases architectures may have changed so much that it is impossible to run old software on the new hardware. Virtual machines offer some protection against this problem as old hardware can be virtualized, however this is not a perfect or guaranteed solution. This long term data problem is not solely a problem of PBR; all long term storage systems suffer from this problem as formats and software changes. It is simply all the more visible with PBR due to the reliance on using old processes to rebuild lost data, however PBR offers more semantic insight into the data through its provenance. As hardware evolves, software adapts, and using PBR's ability to swap out and upgrade processes a PBR system has the potential to adapt too. Organically growing and changing with new hardware and software is a much better alternative to blindly hoping bits can be decoded in the future.

6 Related Work

In the provenance aware storage system paper (PASS) [9], one of their proposed applications is automatic script generation to create—or recreate— a particular file based on its provenance. Our work here is effectively an extension of this idea to a grander scale. Similarly, the Chimera system [5] tracks data provenance, and allows users to both query and regenerate datasets on demand. Database transaction log replay [8] can also be thought of as a type of provenance based recontruction, as they replay tracked information to return the system to a usable state.

Earlier work by Adams *et al*. [3] proposed using data provenance and reconstruction as a method for improving storage system efficiency and recompute times. Our work here shares similarities, but it focused on reliability and reconstructions rather than overall storage system efficiency. Similarly, Yuan *et al*. [10] analyze scientific workflow to identify an optimum balance of intermediate results to reduce storage overhead and reduce computation costs. Ko *et al*. [6] examined the use of intermediate datasets in map-reduce operations. They show that by

selectively storing intermediate results can significantly speed up recomputation in the event of failures. Though Yuan and Ko apply provenance and workflow for the reconstruction of data, their goals are very different in their focus on improved efficiency and computation times.

Chapman *et al*. look at the problem of using provenance as a method for verifying the a conclusion or result. In their work they use causal networks as a method for quantifying belief in a result. This would approach would likely be of great use in our own proprosal for aiding in automatic verification of outputs recomputed by PBR.

7 Conclusions

We have introduced a novel method for improving data reliability complimentary to traditional ECC based approaches. Rather than rely solely on ECCs of the final data, we proposed storing a data's provenance, inputs, and processes to reconstruct the data when needed. With our technique comes with a number of challenges and trade-offs that must be addressed such as provenance collection, data placement, and infrastructure provisioning. Despite these challenges, PBR provides many novel benefits such as per-file prioritized rebuild, opportunity for trade-offs in rebuild time and storage overhead, and swapping out of obsolete or missing processes. With the PBR technique, organizations will have a numver of new opportunities and trade-offs with which to work with, all while providing and additional method to improve the long term reliability of their stored data.

References

[1] Amazon web services. `http://aws.amazon.com/`.

[2] Cloud power. `http://www.microsoft.com/en-us/cloud/`.

[3] I. Adams, D. D. E. Long, E. L. Miller, S. Pasupathy, and M. W. Storer. Maximizing efficiency by trading storage for computation. In *Proceedings of the Workshop on Hot Topics in Cloud Computing (HotCloud '09)*, 2009.

[4] R. Appuswamy, D. C. van Moolenbroek, and A. S. Tanenbaum. Block-level raid is dead. In *Proceedings of the 2nd USENIX conference on Hot topics in storage and file systems*, 2010.

[5] I. T. Foster, J.-S. Vöckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *Proceedings of the 14th International Conference on Scientific and Statistical Database Management (SSDBM '02)*, pages 37–46, 2002.

[6] S. Y. Ko, I. Hoque, B. Cho, and Indranil. On availability of intermediate data in cloud computation. In *HotOS 2009*, 2009.

[7] A. Leventhal. Triple-parity raid and beyond, December 2009.

[8] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems*, 17(1):94–162, 1992.

[9] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer. Provenance-aware storage systems. In *Proceedings of the 2006 USENIX Annual Technical Conference*, pages 43–56, 2006.

[10] D. Yuan, Y. Yang, X. Liu, and J. Chen. A cost-effective strategy for intermediate data storage in scienctific cloud workflow systems. In *24th IEEE International Parallel and Distributed Processing Symposium*, 2010.