Proceedings of the 2nd international Petascale Data Storage Workshop (PDSW '07)
held during Supercomputing '07. November 11, 2007, Reno, NV.

Integrated System Models for Reliable Petascale Storage Systems

Brent Welch
Panasas, Inc.
6520 Kaiser Dr
Fremont CA 94555
1-510-608-7770

Welch@panasas.com

ABSTRACT

The big challenges facing petascale storage systems are not actually those of performance or traditional fault tolerance schemes, but instead the challenges stem from the inherent complexity of large distributed systems. The best way to address these problems is to build a system model into the system itself, and have the system continuously monitor its current state as compared with the desired state according to the model. Change is applied to the model first, and then the system strives to achieve the new model. The hard part is defining a good model, building good support for introspection, and exploiting these features in the system design. This paper describes this problem in more detail, explains some of the techniques we have used in building the Panasas distributed storage system, and concludes with a summary of the open problems with building petascale storage systems.

Categories and Subject Descriptors

D.4.3 [**File Systems Management**]: Distributed File Systems. D.4.5 [**Reliability**]: Fault-tolerance. D.4.7. [**Organization and Design**] Distributed Systems.

General Terms

Management, Measurement, Design, Reliability.

Keywords

Autonomics, Panasas, Cluster Management, Petascale

1. INTRODUCTION

Computer systems are complex, and distributed computer systems can be exceedingly complex. The primary consequence of this is that they can be fragile and therefore not as reliable as we want them to be[5]. We assume that standard techniques such as RAID and service failover are used to compensate for hardware and software faults. But even these standard approaches depend on simple issues like "is component X failed" that can be surprisingly deep. Component X might be operational, mostly, but fail sometimes. Consider a 500 GB disk that has 1 billion 512 byte sectors. There might be 600 of these sectors (.0006%) that are completely unreadable, yet the device is in service, and in fact is so healthy that it is not considered failed by the manufacturer. Or, component X might be servicing thousands of requests every second, but one request has been stuck or lost inside the system for minutes or even hours. Typical monitoring systems are not designed to make such refined judgments about failure – it is much easier to design to a simple black and white failure model. Admittedly, grown media defects are a particularly interesting problem within the storage industry, and vendors do create relatively sophisticated models for detecting and managing them. But what about the software system itself? What sort of models and monitoring should a large distributed system use to track its own health and increase its own reliability?

Making the jump from a single-server storage system to a distributed storage system requires a management component that is responsible for managing the rest of the system components. In mature industries like telecommunications, the management protocols are very precisely defined, so much as to be the subject of laws and regulations in some cases. The management protocols include fault detection as well as configuration change management. In contrast, in the field of distributed storage systems, each solution "rolls their own" management system. If they work well they remain out-of-sight and the overall system remains robust in the face of failure. If they don't work, then the system requires large amounts of human intervention to remain operational. Indeed, many storage systems need a team of full time administrators.

2. CLUSTER MANAGEMENT IN THE PANASAS SYSTEM

The Panasas system[4][8] includes several features that help the system manage itself automatically in the face of adversity. The elements of our solution include a distinct cluster management system that maintains a replicated system model that is updated via Lamport's PTP algorithm[3], a "manager controller" protocol that the cluster manager uses to control other services, a node monitor that manages the processes on each cluster node according to a local model, and a general monitoring system that provides performance metrics and module status.

The system model includes generic aspects of the distributed system such as node membership and service state, serviceability features that help the system detect and respond to failures (and to help administrators diagnose problems), and of course the system model includes features that are specific to our object-based distributed file system. By creating a robust distributed system foundation, the file system services that are layered on top can be optimized for efficient client-server interactions without getting

SuperComputing'07, November 10-16, 2007, Reno, NV.

bogged down in the hard problems posed by having a large distributed system.

Our system is composed of storage nodes that own a small amount of storage, and metadata manager nodes. We have a blade server hardware architecture with a flexible number of storage blades and manager blades. Our largest installations have around 500 storage nodes, each with 2 disk drives, and 50 manager nodes. Each node runs one or more services that take care of the node itself and work together to provide the over all distribute file system.

2.1 Maintaining the System Model with PTP

The cluster manager maintains a replicated database that stores the system model. It contains information about the physical nodes, networking configuration, storage configuration, and service state. All decisions are made via the PTP voting algorithm that ensures that results are committed successfully as long as a majority of the cluster managers are on-line. Partitioned members are brought up-to-date when they rejoin the quorum, and quorum membership can be adjusted dynamically by the administrator. The replicated, fault-tolerant system model is the base for the rest of the system.

The system model must record the desired state of the system because deciding to do something is distinct from accomplishing it. The cluster manager records the desired state and then sets up processes that attempt to achieve the new state. Once the desired state is verified, the system model is updated again to record a successful transition. The cluster manager has to be prepared to restart at any point, and re-verify the system model when it starts.

Given the PTP framework, which provides a robust decision making substrate, you still need a good system model to describe your system, and robust control protocols to control the elements of the system. Of course, this general idea is independent of storage and file systems *per se*, but the system model for a distributed storage system must reflect its unique requirements. For example, dual-ported disk drives can be managed by two different controllers. The controllers might be directly attached to a server, or they might be accessible over a SAN and potentially accessible from many different servers. The system model needs to reflect this and assign ownership of drives to the appropriate servers, accounting for the physical topology and compensating for failures. Without the notion of a general system model, assumptions about topology and failure modes get baked into the control protocols and become very difficult to change.

2.2 Manager Controller Protocol

The Panasas cluster manager uses a manager controller protocol to activate services and arrange backup relationships for failover configurations. The term "manager controller" (MC) refers to a process that can house several different service instances. The MC is a module within the process that manages the various service instances within it. The cluster manager communicates with the manager controller to activate and deactivate different service instances (i.e., "unit"), and the MC communicates heartbeat and other status information to the cluster manager. Table 1 lists the operations in the manager controller protocol (some operations are omitted for clarity).

Table 1. Manager Controller Operations

Sender	Message
MC	System Available
Cluster	Dataset Create, Dataset Delete
Cluster	Dataset Enumerate
Cluster	Dataset Send, Dataset Receive
Cluster	Unit Start, Unit Stop
Cluster	Unit Query
Cluster	Backup Enable, Backup Disable
Cluster	Configuration Reload
MC	Heartbeat
MC	Log State Change
MC	System Stop

The basic protocol is that the MC announces itself to the cluster with System Available, and the cluster manager responds by activating service instances with Unit Start. Services require a dataset for their persistent state, and this is replicated between a primary and a backup service instance. To make the decision about what to start, the cluster manager queries its system model for the expected state of other services, and the state of the MC with respect to its existing datasets. The cluster manager uses the dataset operations to ensure that the environment is set up correctly before it activates the primary and backup service instances.

For example, the file system metadata manager maintains a local transaction log that provides fault tolerance during online activity. The log is replicated in a fault-tolerant configuration, and the cluster manager ensures that the primary and backup are activated and know how to communicate. When failures occur, a new primary or backup needs to be selected, and transaction logs need to be synchronized between them. Note that the "Log State Change" message supports a kind of partial failure where a service can run, but finds that its logs are unusable for some reason. The protocol also includes queries (Unit Query, Dataset Enumerate) so the cluster manager can determine if the current state of the system matches the desired state recorded in the model.

A heartbeat protocol between the manager controllers and the cluster manager is used to detect failure. This is a hierarchical relationship where the cluster manager is responsible for making failure assessment decisions, rather than having the primary and backup service instances make local decisions. This allows the cluster manager to make broad decisions about the placement of backup services. Our current cluster manager is capable of managing thousands of manager controllers with tens of thousands of service instances. However, in a very large system with many thousand services, an additional level of hierarchy would be necessary to distribute the load of managing the services.

2.3 Storage System Model

The cluster manager maintains a set of state machines that model the distributed storage system. The state machines make persistent state changes via PTP updates. They may control remote processes and nodes, and they may trigger action by other state machines. There are state machines for the overall system, for nodes, for collections of nodes that comprise a fault domain, for software services, and for file system volumes.

The system states include Online, Offline, Paused, Rebooting, Powering Down, and Upgrading. These are global states that affect services and nodes, and what operations the administrator can perform on the system. Operations like Rebooting and Powering Down involve sub-states to sequence activities within other sub systems. For example, file system activity must be quiesced first before services are taken offline, and services must shut down cleanly before nodes are rebooted.

Services have states that include Inactive, Active With Backup, Active Without Backup, and Backup. The cluster manager maintains these states, and controls the services using the Manager Controller protocol described previously. Not every service requires a backup. Services like the management console and the DHCP server store their state with the cluster manager, and so they are simply activated on a new node if their node fails. Other services are node-local and run on every blade so it doesn't make sense to have a backup. The file system metadata manager is the only service that runs with an active backup service.

Node states include Booting, Fsck, Online, Unavailable, Failed, and Upgrading. When a node boots it signs on with the cluster manager via a modified DHCP protocol, and then cluster manager polls the node until it comes fully online. Once Online, nodes are not polled unless some service or a file system client notifies the cluster manager that a node is not responsive. The cluster manager ensures that a node is at the correct software revision (blades boot from a local copy of the software), and will automatically upgrade (or downgrade) a new node to a software version that matches the rest of the storage system.

Collections of storage nodes (which we call "BladeSets" because our nodes are blades in a chassis) form a physical storage pool and a failure domain. Files are striped across a subset of nodes using per-file RAID protection. When nodes are unavailable because they are rebooting (i.e., after a crash) or have failed permanently, then the files in that storage pool are degraded and may need to be reconstructed. Therefore, the per-node state machine signals a storage pool state machine that models states of All OK, One MIA, One Dead, and Two Down. These states reflect the overall availability of the storage pool.

Volumes are directory trees with optional quota that reside within a particular storage pool and are managed by a particular service. The volume states include Online, Paused, Fsck, Snapshot, Degraded, Reconstruction, and Down. The availability of a volume depends on both its storage pool and its metadata service. Therefore, the storage pool state machine and the service state machine will signal the volume state machine if events occur that may affect volumes. For example, if a node becomes unavailable then the volume is Degraded. If the node is unavailable for too long (6 minutes in our current system) then the volume enters the Reconstruction state. There are other actions such as administrative offline, volume recovery (i.e., fsck), and snapshots that affect volume state. These state changes require specific control messages to the metadata service for the volume. The APIs between the cluster manager and the metadata service convey the new volume state and the parameters, if any, associated with the state change.

Modeling the system as a set of interdependent state machines helps isolate the different subsystems and avoid ad-hoc logic within the implementation. Each subsystem typically employs a sweeper thread that periodically checks its desired state against the observed state of whatever resource it is managing. To effect change, other modules call APIs that set a new desired state and then kick the sweeper thread. Decoupled actions are important so that, for example, node management (e.g., node sign on, off version upgrade) can proceed independently from service activation and volume management.

2.4 Local Node Management

Each node in the distributed system is a computer system with its own operating system, configuration files, and a collection of processes that run as part of the overall distributed system. To eliminate the burden of managing each of these separately, nodes are configured and controlled by the cluster manager.

Every aspect of a node's configuration is pulled down from the cluster manager. Each node runs an "MCProxy" that is a manager controller that proxies for the blade itself, and for processes like Samba and the NFS server layer that cannot easily embed the MC module. Primarily it manages configuration files for the blade based on a master copy of the configuration stored with the cluster manager. When configuration changes occur, MCProxy is notified via the Configuration Reload message.

A process monitor maintains a local model of the node state, and runs different processes depending on the state. Processes are "nannied" so they are restarted if they crash. This is an elaboration on the "runlevel" and "inittab" concepts found in most Unix systems. The monitor exports its current state to the cluster manager, and the cluster manager is able to direct the monitor into different states to coordinate system startup, software upgrade, reboot, and power down.

2.5 Monitoring

Our monitoring system provides functionality similar to SNMP. Data elements with a structured schema are published from various software modules, including kernel modules, via a "provider" interface. Each provider implements a set of typed values. Types support rates, aggregation, and histograms. The interface lets other modules register interest in data elements, including threshold-based and value-based triggers, to reduce overhead from polling. Each node typically has several providers, and a single message agent process (i.e., magent) acts as their proxy for queries from other nodes. Local providers communicate only with the magent, and the magent handles queries from other nodes and relays notifications.

Values can also be set to cause side effects on the provider of the value. For example, the process monitor exposes a "runlevel" value, which reflects its current state. The cluster manager queries the runlevel to determine if a node is operating normally, or somewhere in the boot process, or in a failed state. To get the node to change state, the cluster manager assigns a new "runlevel" value to the node's monitor, which causes the process monitor to change the set of processes running on the node.

Performance monitoring is an obvious application of this facility. The provider library makes it easy to support rates and histograms. Our management console periodically polls blades

for performance metrics in order to display performance graphs. Our performance manager is a service that monitors load and capacity of storage blades, and it automatically migrates data among storage nodes to balance capacity.

3. LESSONS LEARNED

A positive lesson is that the split between the cluster management and the file system services was very successful. The file system metadata services are generally client-server applications that don't worry much about the larger distributed system. There are still the traditional issues associated with stateful services that manage cache consistency and callbacks for many clients. But the fact that there are tens or hundreds of meta data service instances, and hundreds or thousands of storage nodes is largely irrelevant to the metadata managers. The issues of scale are often about management, and management protocols can afford to be robust (e.g., use PTP), and benefit from a model-based approach.

However, our primary focus was on the file system, and so our initial system models were overly simplified. For example, the original node states were just Online, Unavailable, Dead, and Buried. This was sufficient for managing data availability and driving volume reconstruction. The cluster manager was content to wait a while, trigger construction if necessary, and move on to other issues.

In practice, nodes have a more complex local model that includes states like fsck, low-battery, and rebooting. The cluster manager could map those down to its simplified view, but that wasn't as helpful as it could be to administrators. "Unavailable" is especially interesting to an administrator – *Why* is it unavailable! Our current model includes an "unavailability reason" that indicates why the cluster manager thinks a node is unavailable. Reasons include Ping Failed, Fsck, Object Store Not Mounted, Wrong Software Version, Rebooting, Hardware Fault, and Offline.

Another area that had a limited model was software version compatibility. The cluster manager recorded the current version and would fence blades running the wrong revision. It would signal an upgrade manager that would load the right software and reboot the blade. However, system wide upgrades were done completely behind the back of the cluster manager, and then all nodes including those running the cluster manager would spontaneously restart. We have improved the API for the upgrade manager and we have enhanced the model so the cluster manager can ensure a clean restart that is transparent to file system clients.

An area that still needs work is modeling partial failures within a software service. Unlike hardware that fails because it is broken and you replace it, software fails because it has bugs and you need to live with it until a patch or upgrade cures the problem. The standard remedy for software is to restart it, and our node monitor will automatically restart processes that crash. A service can rely on this and panic if it determines if it is in a partially broken state. This is fairly crude and doesn't help find the root cause of the problem.

For example, a storage node that is busy, or buggy, might starve a particular request from a client or metadata manager, or an inconsistency in the object store might cause a file to be fenced from a client. These are partial failures within a service that is otherwise healthy, and they are triggered by outright failures elsewhere in the system.

4. RELATED WORK

There is a large body of recent work in autonomics[2], which is the general field of creating systems that can self manage. It is worth noting that any system that compensates for failures shows some degree of autonomics. Perhaps one of the most highly evolved technologies in this area is telecommunications, although they do not use the term. The telecommunications industry has standardized management protocols for the telephone switching infrastructure that can isolate faults within milliseconds of a failure. This is a very mature technology, and it represents a state of the art that we need to strive for in distributed systems.

Astrolab[6] is a system for scalable monitoring of distributed systems that is similar, but probably powerful, than our data monitoring facility.

The Ceph object-storage system[7] has similar structure to ours, with cluster managers that maintain a set of storage devices and hash function that determine object distribution. Like our system, PTP is used among the cluster managers. Its system model is oriented only toward the storage system, and it is not applied to the management of the nodes and services that make up the rest of the Ceph implementation.

The Kybos system is a research project[9] that has many parallels to the Panasas system. System elements try to achieve goals and adapt to changes in their configuration. Its management agent balances capacity across storage nodes and drives reconstruction. One interesting difference is its simplified brick model (i.e., "black boxes that are up or down") which is similar to our original approach.

The Self-* work at CMU[1] stresses self-managing systems. Their work describes supervisor processes that are analogous to the Panasas cluster manager. Like Ceph and Kybos, the system model is primarily concerned with placement of data, and less concerned with managing the nodes and systems that make up the implementation. Our work started from a similar position, and we came to appreciate the need for managing the distributed system itself as being at least as important as managing our file system data.

5. CONCLUSION

Model-based management protocols are a key element of large-scale, reliable systems. This paper has given some examples from the Panasas system design. We have been continually refining our system model as we gain experience. For example, the initial cluster manager was primarily focused on the storage services, and did not have a very rich model for underlying nodes. As another example, error conditions really need to be first class elements of the model so that administrators can manage errors, and so that errors are cleared automatically when appropriate.

The most difficult thing to model well is partial failures. Our current approach is to have services monitor themselves internally, and simply restart if they detect internal problems that are "too severe". However, the next step is to have richer models that help the system better pin-point the root cause of partial failures. For example, performance problems are often associated with faulty components that only partially operational. However, performance issues can have other causes such as network

bottlenecks or high load on the system. The ideal system would monitor itself for performance problems and partial failures and have a system model that allowed root cause diagnosis by the system itself. This is a very hard problem, but one that needs to be addressed for reliable petascale and exoscale storage systems.

6. ACKNOWLEDGMENTS

Jim Zelenka is a key designer of the manager controller protocol and the cluster manager, as well as the Panasas file manager. Marc Unangst is an architect at Panasas and has contributed substantially to the design of many aspects of the system. Richard Golding contributed to the design of the node monitor, the early system models, and the data monitoring system. I'd like to thank them and the many people at Panasas that have worked hard to make the Panasas system successful for its users.

7. REFERENCES

[1] Gregory R. Ganger, John D. Strunk, Andrew J. Klosterman. *Self-* Storage: Brick-based Storage with Automated Administration*. Published as Carnegie Mellon University Technical Report, CMU-CS-03-178, August 2003.

[2] Jeffrey O. Kephart, David M. Chess. "The Vision of Autonomic Computing." *Computer*, Vol. 36 No. 1, 41-50, 2003.

[3] L. Lamport. "The Part-Time Parliament." *ACM Transactions on Computer Systems*, Vol. 16 No. 2, 133-169, 1998

[4] D. Nagle, D. Serenyi, and A. Matthews. The Panasas ActiveScale storage cluster.Delivering scalable high bandwidth storage. In *Proc. of the 2004 ACM/IEEE Conf. on Supercomputing*, Nov. 2004.

[5] Oppenheimer, D., Archana Ganapathi, and David A. Patterson. Why do Internet services fail, and what can be done about it? 4*th USENIX Symposium on Internet Technologies and Systems (USITS '03),* March 2003.

[6] van Renesse, R., Birman, K. and Vogels, W. "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining." *ACM Transactions on Computer Systems*, 21(2):164-206, May 2003

[7] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. *Ceph: A scalable, high-performance distributed file system*. In Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI), Seattle, WA, Nov. 2006. USENIX.

[8] B.Welch, Z. Abbasi, G. Gibson, B. Mueller, M. Unangst, J. Zelenka, B. Zhou. *Scalable Performance of the Panasas Distributed File System*, Proceedings of FAST08, January 2008, San Jose, CA. (to appear)

[9] Theodore M. Wong, Richard A. Golding, Joseph S. Glider, Elizabeth Borowsky, Ralph A. Becker-Szendy, Claudio Fleiner, Deepak R. Kenchammana-Hosekote, Omer A. Zaki. *Kybos: self-management for distributed brick-base storage.* Research report RJ 10356, IBM Almaden Research Center, 26 August 2005