A Fault-Tolerant Middleware Architecture for High-Availability Storage Services

Sangeetha Seshadri, Ling Liu, Brian F. Cooper
Georgia Institute of Technology
801 Atlantic Drive, Atlanta GA-30332
{sangeeta,lingliu,cooperb}@cc.gatech.edu

Lawrence Chiu, Karan Gupta, Paul Muench
IBM Almaden Research Center
650 Harry Road, San Jose CA-95062
{lchiu,guptaka,pmuench}@us.ibm.com

Abstract

Today organizations and business enterprises of all sizes need to deal with unprecedented amounts of digital information, creating challenging demands for mass storage and on-demand storage services. The current trend of clustered scale-out storage systems use symmetric active replication based clustering middleware to provide continuous availability and high throughput. Such architectures provides significant gains in terms of cost, scalability and performance of mass storage and storage services. However, a fundamental limitation of such an architecture is its vulnerability to application-induced massive dependent failures of the clustering middleware. In this paper, we propose hierarchical middleware architectures that improve availability and reliability in scale-out storage systems while continuing to deliver the cost and performance advantages and a single system image (SSI). Hierarchical middleware architectures organize critical cluster management services into an overlay network that provides application fault isolation and eliminates symmetric clustering middleware as a single-point-of-failure. We present an in-depth evaluation of hierarchical middlewares based on an industry-strength storage system. Our results show that hierarchical architectures can significantly improve availability and reliability of scale-out storage clusters.

1. Introduction

With rapid advances in network computing and communication technology and the continued decrease in digital storage cost, the amount of digital information continues to grow at an astonishing pace. Many organizations and enterprises today are facing the challenge of dealing with data storage ranging from terabytes and petabytes to exabytes, zettabytes, yottabytes and beyond. Such trends create continuous high demands for massive storage and storage services. High-availability scale-out storage clusters combine smaller units of storage to provide a scalable and cost-effective storage solution [1, 2, 10]. The scale-out clustered storage architectures allow enterprises to gain significantly in terms of cost, scalability and performance.

Current scale-out storage systems use active replication [16] based middleware to ensure consistent access to shared resources in the absence of centralized control and at the same time provide high throughput and a single system image (SSI). In order to guarantee high-availability to applications, the middleware typically maintains critical application state and check-point information persistently across nodes through active replication. Active replication based models are not only used in clustered storage systems, but are also common in distributed locking services [7] and high-performance computing [9].

However, though the use of symmetric active replication models removes hardware as both single-points-of-control and single-points-of-failure, it causes the storage middleware itself to now become a single-point-of-failure. In some instances, unavailability of a highly-available, active replication based service due to network outages and software errors can cause as much as 0.0326% unavailability [7, 8]. This is equivalent to three days of downtime over a period of 100 days and intolerable for many applications that demand above 99.99% uptime from storage services. Moreover, the current scale-out storage architecture is also vulnerable to application-induced failures of the middleware, in addition to other issues like application-level non-determinism [18] and middleware bugs themselves. By application-induced middleware failure, we mean that a single cluster application can cause an error which leads to unavailability of the cluster middleware for all other applications concurrently running over the cluster. For example, the highly available Chubby locking service reports that the failure of application developers to take availability into consideration "magnified the consequence of a single failure by a factor of a hundred, both in time and the number of machines affected" [7].

One obvious approach to improving availability and reliability in such systems is to provide application fault isolation and eliminate symmetric clustering middleware as a single-point-of-failure. While application fault isolation can be achieved through partitioning of a single storage cluster into smaller independent clusters [4, 3], without care, one may lose the SSI and the flexibility to access storage from anywhere within the system. The key challenge, therefore, is to provide fault-boundaries while continuing to deliver SSI and flexible accessibility. In this paper we introduce the notion of hierarchical middleware architectures. We organize critical cluster management services into a hierarchical overlay network, which separates persistent application state from global system control state. This clean separation, on one hand, allows the cluster to maintain SSI by communicating control state to all nodes in the network, and on the other hand, provides fault isolation by replicating application state within only a subset of nodes.

We present baseline availability analysis in flat and hierarchical clusters with varying storage nodes. Our analysis shows two important results. First, we show that simply increasing the number of hardware nodes in a symmetric system will not improve availability or reliability as long as the middleware is vulnerable to simultaneous failures. Second, we show that by trading some symmetry for better fault isolation, hierarchical overlay storage architectures can significantly improve system availability and reliability.

We conducted a thorough experimental evaluation of the availability and reliability of our proposed solution. Utilizing the relationship between workload and software failure rate [19, 25, 12], we quantitatively show that hierarchical middleware architectures can provide significant improvements in reliability and availability of mass storage systems. For example, organizing a 32 node system into a hierarchical cluster can reduce downtime by nearly 94% and improve Mean-time-to-failure (MTTF) by approximately 16 times.

The rest of this paper is organized as follows. We first present the problem statement and describe our approach in Section 2. We develop probabilistic markov models for analyzing the availability and reliability of the proposed architectures in Section 3. In Section 4 we present an in-depth experimental evaluation of availability and reliability in both flat and hierarchical architectures. We compare and contrast the two models in Section 5, and conclude in Section 6 with related work and a summary.

2. Overview

Before presenting an overview of the proposed hierarchical overlay architecture, we first describe the conventional approach, its flat cluster architecture, and the potential problems with respect to application-induced failures.

2.1 Conventional Approach

High availability scale-out storage clusters are composed of nodes, a network that connects the nodes, and middleware that provides a highly available environment for applications that operate in the cluster. The back-end storage is shared by all nodes in the cluster. All nodes within the cluster are peers with no single-point-of-control. In this paper, we will refer to this ubiquitous form of clusters as a **flat** cluster (Figure 1). Examples of such systems exist in both research [4, 24, 9] and commercial [10, 1] settings.

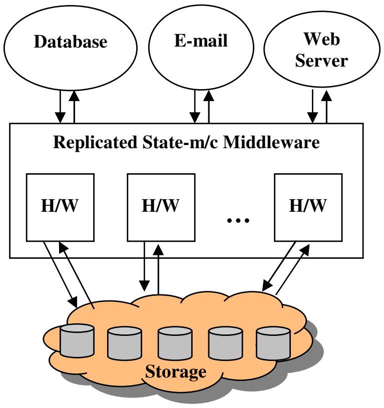

Figure 1. Traditional flat storage cluster.

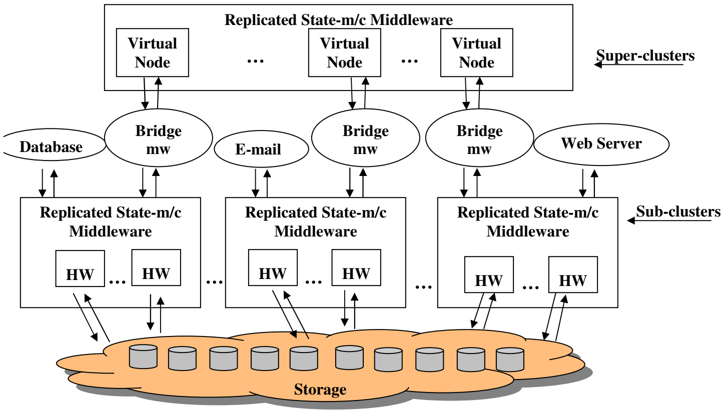

Figure 2. Hierarchical cluster.

In such systems, the middleware must ensure consistent access to the shared storage and provide a highly available environment to applications in the absence of centralized control. Typically this is achieved through active replication based on replicated state machines (RSMs) [16]. RSMs maintain consistent state on every node by applying an identical ordered set of requests, referred to as **events**, on each instance of the state machine. Since consistent state is maintained on all nodes, applications may failover to any node when the node currently serving the application fails.

Events belong to two categories - global or setup events used for liveness, node joins, departures, resource allocations etc., and application events that are used to maintain application state globally. Typically, middleware generated setup events belong to a (usually well-tested) pre-defined set. On the other hand, application events such as checkpoints and persistent state are generated by the applications and passed on to the middleware through interfaces provided by the middleware. This potentially opens up the middleware to faulty application generated events which when processed synchronously will cause all instances of the middleware to fail simultaneously, bringing down the whole cluster. The goal of this paper is to eliminate or reduce such application-induced dependent middleware failures in scale-out storage systems while maintaining the provision of SSI, scalability and high-availability of scale-out storage systems and services.

2.2 Hierarchical Middleware Architectures: Our Approach

We first present an outline of the design of hierarchical middleware architectures. The key idea for improving system availability and reliability is to define application-fault boundaries by separating global control state from persistent application state. Concretely, in order to prevent the symmetric middleware from becoming a single-point-of-failure, hierarchical architectures organize cluster management services into an overlay such that global control events are communicated to all nodes and application generated events are processed by only a subset of nodes. Note that, the hierarchy is only used to regulate control messages and does not interfere with the actual data path.

Figure 2 shows a two level hierarchical cluster that utilizes a hierarchical middleware architecture. Clusters at the upper level are called super-clusters and clusters at the lower level are called sub-clusters. Sub-clusters and super-clusters are flat clusters connected together by a bridging middleware. The bridging middleware is a cluster application that runs over a sub-cluster. It is essentially stateless and serves two main purposes: first, it creates a virtual node environment which emulates a hardware node by utilizing underlying resources; second, it provides a channel between sub-clusters and super-clusters for communicating global events. It also serves as a channel for communicating application-specific processing requests from super-clusters to sub-clusters.

Nodes in the super-cluster are called virtual nodes. Each virtual node represents a sub-cluster. Super-clusters provide SSI and manage the communication of global control state across sub-clusters. The construction of such a hierarchical model and the decision on critical hierarchical parameters such as sub-cluster size and number of hierarchical levels will rely on a number of system-supplied parameters, such as known or historical failure rates, lease timers (heartbeats required among the nodes for better response times), load skew per node and load skew per sub-cluster or per application category. Due to space constraints, we delay the discussion on issues regarding hierarchical cluster overlay construction to Section 5. Below, we focus more on the availability and reliability properties of our hierarchical organization.

Since the bridging middleware is a cluster application that runs over a sub-cluster, if the underlying node fails, the bridge application itself would instantaneously failover to another node within the sub-cluster. In the event of either a virtual node dropping out of the super-cluster or the corresponding node dropping out of the underlying sub-cluster, the bridge middleware will instantiate another virtual node over its underlying sub-cluster to participate in the super-cluster. The bridge middleware relies on the underlying sub-clusters to provide actual storage services.

In the hierarchical clustering paradigm, applications may be deployed on a cluster at any level in the hierarchy. Applications that are deployed at the higher levels distribute work to sub-clusters at the lower level according to the current system state. Typically, load balancing and resource management applications are placed at upper levels as they are shared across a number of sub-clusters. While global events are communicated to all nodes in the hierarchical cluster through the bridge middleware and super-clusters, application events are processed only by the nodes within the current sub-cluster in which the application is processed. As a result, the total application-generated middleware workload is partitioned amongst sub-clusters.

Storage cluster applications may have global or local access points for accepting external requests. Global access points will be owned by a virtual node at the top of the hierarchical cluster. External requests received at the global access point are decomposed and propagated along the cluster hierarchy to the appropriate sub-clusters at the lowest level. Local access points are provided for services specific to a part of the hierarchical cluster. This allows some cluster services to continue even if the hierarchical cluster is not fully connected momentarily. We next illustrate the partitioning of middleware workload in a hierarchical cluster.

2.3. Example Application: Data Migration

Moving data from one location to another to change its retention cost, availability, or performance is a well known strategy for managing data. With clustered storage it is possible for data movement to be managed internally to the cluster. The cluster middleware provides the persistent storage to track a data movement operation, which would at least include the source data location, the target data location, and a current copy location. The current copy location is kept to minimize the amount of duplicate data copied after recovery from a cluster transition.

For example, if the current copy location is updated for every 1GB of data moved, moving 300GB of data would result in 300 application-generated update events to the cluster middleware. In a flat cluster the middleware workload (number of events) per node for this data movement request would include the setup (control) events in addition

Availability	Fraction of time an application provides service to its users.
Reliability	Probability of failure-free operation over a specified period of time.
$MTTF$	Mean-time-to-failure
$MTTR$	Mean-time-to-restore
λ_{hw}	Hardware failure rate = $1/MTTF_{hw}$
μ_{hw}	Hardware repair rate = $1/MTTR_{hw}$
λ_{mw}	-Middleware failure rate = $1/MTTF_{mw}$
μ_{mw}	Middleware repair rate = $1/MTTR_{mw}$
λ_{vn}	Virtual node failure rate = $1/MTTF_{vn}$
μ_{vn}	Virtual node repair rate = $1/MTTR_{vn}$
λ_{vn_app}	Bridge middleware failure rate = $1/MTTF_{vn\_app}$
μ_{vn_app}	Bridge middleware repair rate = $1/MTTR_{vn\_app}$

Table 1. List of Terms

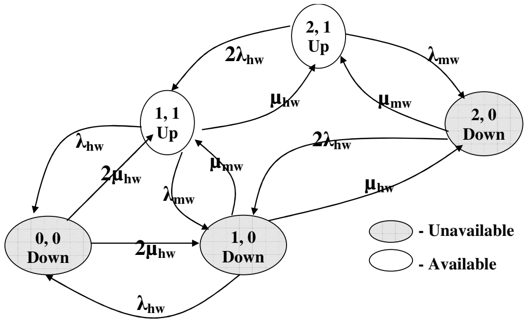

Figure 3. Availability of a 2 node cluster.

to 300 application-generated events. In a hierarchical cluster a data movement operation can be initiated between any two storage nodes in the cluster at the expense of a setup cost. Also since super-clusters distribute work among its underlying sub-clusters, not all sub-clusters perform work per hierarchical cluster request. Thus for a two level hierarchy, while the setup messages would be propagated to and processed by all instances of the middleware, the actual application-generated requests will only be processed in the sub-cluster(s) handling the application.

3 Availability and Reliability Analysis

In this section we develop probabilistic markov models [20] for the analysis of availability and reliability of flat and hierarchical clustering architectures. Our models were generated using an automated tool developed by us. System states are represented as states in a Markov chain and failure and repair rates are represented by the transition rates between the states in a Markov chain. Failure and repair are treated as stochastic processes and Markov chains are the standard way to model stochastic processes in order to determine the average probability of being in a particular state. We assume that the time to failure and repair are exponentially distributed.

An application becomes unavailable when failure of the hardware or middleware occurs in a combination such that the application is no longer running and has to be restarted after restoration of the underlying components. We assume that there is sufficient redundancy in network connectivity and that it is not a limiting factor. Each node runs an identical instance of the middleware.

We assume that hardware failures are independent while middleware failures are correlated. Our aim is to concentrate on the effect of correlated middleware failures and study the effectiveness of our proposed hierarchical architectures in eliminating such failures. Therefore, without loss of generality, we do not consider independent middleware failures or correlated hardware failures in this study. Note that independent middleware failures typically will not cause a marked difference in our comparison.

Dependent middleware failures cause all instances of the middleware to crash simultaneously. Middleware repair is also assumed to be simultaneous. Middleware repair involves restarting of the failed components, synchronization of state and isolation of the failure inducing request (most often, the last event processed by the cluster). Since the middleware already makes critical application state available to all nodes within a single cluster, we assume that failover of an application, when it occurs, is instantaneous.

Table 1 gives a list of important parameters and terms used in the paper and their definitions, including Mean-time-to-failure (MTTF), mean-time-to-restore (MTTR), failure rate (λ) and repair rate (μ) of hardware, middleware, virtual node, and bridge middleware. Each state in the Markov diagram is given by a pair of values indicating the number of instances of hardware and middleware that are functional, and the status of the application. The transitions between states represent failure and repair rates. We illustrate our modeling methodology on two node flat clusters, sub-clusters and super-clusters. Extending this model to an N node cluster is straightforward.

3.1 Availability Modeling

Figure 3 shows the Markov model for availability of a flat two node cluster. Note that, since we consider only dependent failures of the middleware, only a single instance of the middleware is considered to be running on the entire cluster. The system is considered available if at least one instance of the hardware and the middleware are functional.

For the hierarchical system, the model in Figure 3 now represents the Markov model for applications deployed over the sub-clusters at the lowest level. At that level, the middleware instances are run over the actual hardware nodes. At higher levels, middleware instances are run over virtual nodes. In order to simplify our analysis, we consider the same middleware to be running at all levels in the hierarchy. We can represent the availability at a level running over virtual nodes by replacing λ_{hw} and μ_{hw} with λ_{vn} and μ_{vn} respectively in Figure 3. We next describe the reliability modeling and computation of virtual node failure rates.

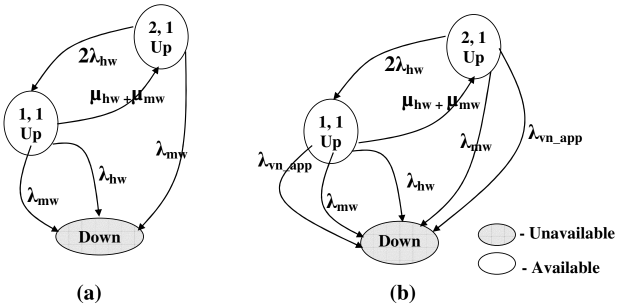

Figure 4. Reliability models: (a) Flat clusters (b) Virtual Node

3.2 Reliability Modeling

We analyze the reliability of clustering architectures using Markov models with absorbing states [20]. Figure 4(a) shows the Markov model for a two node system. The model for the flat architecture shown in Figure 4(a) also represents the reliability model for the lowest level sub-clusters in the hierarchical architecture. Again, the model can be extended to a sub-cluster of any size. As in the case of availability, we can obtain the Markov model for reliability of virtual node clusters (super-clusters), by replacing hardware failure and repair rates in Figure 4(a) by virtual node failure and repair rates.

The reliability of the virtual node presented by a lowest level cluster that directly runs over the hardware is computed using the model shown in Figure 4(b). The model shows that a virtual node failure may be caused either due to failure of all instances of the underlying hardware, the middleware or the bridge middleware. Restoring a virtual node at a given level involves restoring the underlying hardware and virtual nodes up to that level (for clusters that may be running at a level > 2). We approximate virtual node repair rates to be the same as that of the hardware since hardware repair is orders of magnitude slower than that of software.

4 Evaluation of Clustering Architectures

In this section we evaluate the flat and hierarchical architectures based on the models presented in the previous section. We use failure and repair rates based on our experiences with real deployments of commercial storage systems. Since actual failure and recovery rates are sensitive information and held closely by vendors, we are unable to disclose the identity of the system and can only state that the numbers are representative of current deployments. Table 2 represents the failure and repair rates for the various components in the system. Note that the middleware failure rate is for the occurrence of correlated failures. Independent middleware failures are expected to occur more frequently [21]. We expect the bridge middleware to have failure and repair rates close to that of the clustering middleware. We validated our availability and reliability models against numbers reported for current deployments. Accordingly, the error in the availability model is less than 0.00004% while that for the reliability model is around 1.5%.

$MTTF_{hw}$	2 years
$MTTR_{hw}$	4 hours
$MTTF_{mw}$	3 years
$MTTR_{mw}$	15 minutes
$MTTF_{vn\_app}$	3 years (expected)
$MTTR_{vn\_app}$	15 minutes (expected)

Table 2. Component failure and repair rates

We have developed an automated tool to compose Markov models from specifications and compute availability and reliability. The tool, written in C++, uses the Mathematica package [23] for computations. The experiments reported in this paper are based on a system with a cluster size of up to 32 nodes. With the failure and repair rates presented in Table 2, when considering only independent failures of the components, a 2 node cluster was able to achieve 6 nines of availability. Based on this we choose a sub-cluster size of 2 for our experiments. Results with other sub-cluster sizes and other configurations have been briefly summarized in Section 4.3.3.

Our analysis reveals two interesting phenomena. First, although it is common to focus on hardware solutions to availability and reliability, our analysis (Section 4.1 and 4.2) show that middleware (software) failure rates are a far bigger contributor to unavailability. Second, we show that in the common case where increasing workload leads to more software failures, the hierarchical approach provides significantly higher robustness by isolating applications from middleware failures.

4.1 Baseline Availability Analysis

Figure 5 shows the baseline availability in flat and hierarchical clusters with 4-32 storage nodes. In order to present an apples-apples comparison, we compare only 2 level hierarchical organizations of the nodes with the sub-cluster size set to 2 nodes. As the graph shows, availability in the two levels of hierarchical clusters are comparable to that of the corresponding flat clusters. However, note that applications are now isolated from middleware failures in other sub-clusters. Also note that, just increasing cluster size has little effect on availability.

4.1.1 Sensitivity Analysis for Availability

Sensitivity analysis allows us to understand which parameters have the most pronounced effect on availability and reliability. We choose a 32 node system and compare flat and 2 level hierarchical configurations with sub-cluster size set to 2. Using this setup, we evaluate the sensitivity of availability and reliability to the parameters specified in Table 2.

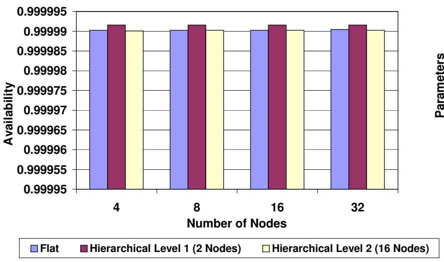

Figure 5. Baseline availability.

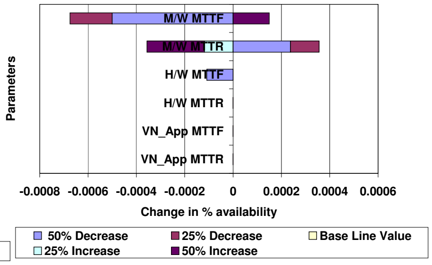

Figure 6. Sensitivity analysis of availability.

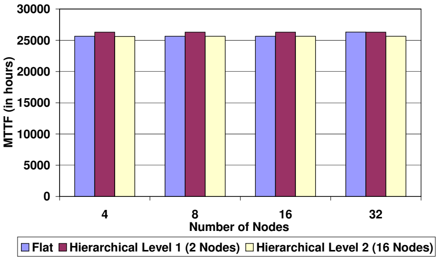

Figure 7. Baseline reliability.

We present the results of our sensitivity analysis using a tornado plot. Each bar represents the variation in availability with changes to a single parameter while all other parameters are fixed at the baseline configurations. The length of the bar is proportional to the sensitivity of the measure to the particular parameter. The parameters are specified along the y-axis and the x-axis shows the change in the measure (availability in this case).

Figure 6 depicts the sensitivity of system availability to component failure and repair rates in a hierarchical cluster. The plot for a flat cluster is similar except for the absence of the bridge middleware and has been omitted due to lack of space. The graphs show that, availability is most sensitive to middleware repair and failure rates as compared to hardware. Also, note that the bridge middleware's failure and repair rates have minimal impact on the availability of a super-cluster. From this analysis we conclude that dependent failures of the middleware continue to be the limiting factor in availability in both architectures. Also introduction of independently failing components (bridge middleware) impacts availability minimally.

4.2 Baseline Reliability Analysis

Figure 7 presents the variation in system reliability with increasing cluster size using the setup described in Section 4.1. The graph shows that reliability of both sub-clusters and super-clusters in hierarchical architectures are comparable to that of the corresponding flat clusters despite inclusion of additional components (bridge middleware). However, note that sub-clusters in hierarchical architectures are units with independent failure modes unlike the flat architecture. The graph also shows that increasing the number of redundant hardware nodes does not improve reliability as long as the middleware remains a single-point-of-failure.

4.2.1 Sensitivity Analysis for Reliability

Figure 8 depicts the sensitivity of reliability in super-clusters using the same set-up as Section 4.1.1. The sensitivity of reliability in sub-clusters and flat clusters (omitted due to lack of space) is similar, except for the absence of dependence on the bridge middleware component. The graphs shows that reliability is most sensitive to middleware failure rates. For example, a decrease of MTTF$_{mw}$ by 50% causes as much as a 50% drop in system MTTF. Again, note that bridge middleware failure rates hardly impact reliability when compared to middleware failure rates.

4.3 Modeling the Middleware Workload

Although the merits of defining application fault boundaries can be qualitatively understood, it is still desirable to quantify the availability and reliability of the resulting system. In this section, we use the middleware "workload-failure rate" relationship to characterize the benefits of defining fault-boundaries.

A general formulation of software reliability as a linearly increasing function of number of failures per encountered error (*s*), error density (*d*) and the workload (*w*) is provided in [19, 25]. Based on this formulation, we model the middleware failure rate as a linearly increasing function of middleware workload (number of events per unit time), assuming that the parameters *s* and *d* are constant for a single implementation. However, the relationship between software failure rates and the workload may vary depending upon the system, software and workload. Unfortunately, we do not have sufficient data to precisely establish this relationship.

We define the middleware workload under which the baseline failure and repair rates were observed as the standard workload (WL_{std}). We then introduce the notion of a normalized workload (NWL) which is the ratio of the actual observed workload to the standard workload. We consider the average case where workload is uniformly partitioned amongst the sub-clusters. The middleware workload in the super-cluster of virtual nodes consists primarily of global control and setup events. Based on observations in real systems, we expect this class of events to constitute around 5% of the total workload.

Note that, in general we can model middleware failure rate as 'some increasing function' of the middleware workload. In particular, we considered linear, sub-linear (square root) and super-linear (exponential) increasing functions. However, due to lack of space we present only the results for a linear variation in this paper [17].

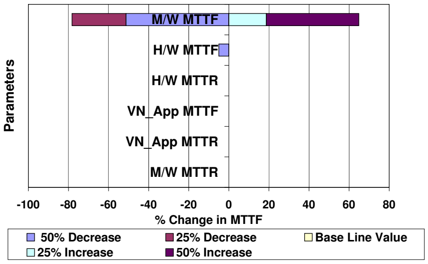

Figure 8. Sensitivity analysis of reliability.

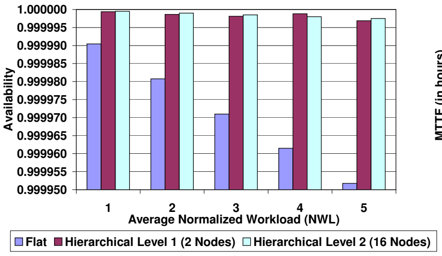

Figure 9. Availability with linear function.

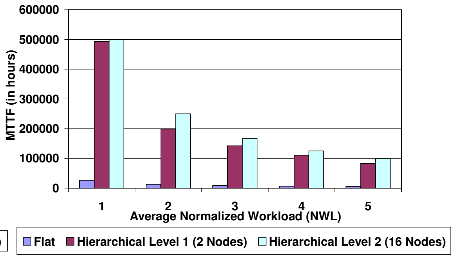

Figure 10. Reliability with linear function.

4.3.1 Effect of Workload on Availability

Figure 9 shows the variation of availability with increasing workload when the middleware failure rate is a linearly increasing function (with constant of proportionality NWL) of the middleware workload. The bars represent the availability in a flat cluster with 32 nodes and a 2 level hierarchical configuration of the same 32 nodes with sub-cluster size of 2. The figure shows that the hierarchical configuration provides better availability at both the sub-cluster level and the super-cluster level. At both levels, downtime is reduced by 94% on average, over the flat clusters. Expressed as downtime minutes, when the workload is 5 times the standard workload, the flat configuration translates to an additional 23 minutes downtime annually, compared to a sub-cluster in the hierarchical configuration.

4.3.2 Effect of Workload on Reliability

Figure 10 shows the variation of reliability with increasing workload using the same setup described in Section 4.3.1. As evident from the figure, all levels of the hierarchical configuration offer better MTTF than the flat architecture. Our results show that in a system with N nodes, compared to flat architectures, hierarchical architectures can improve the MTTF of the sub-cluster by nearly P times, where P is the size of the super-cluster. (In our 32 node system, the super-cluster size is 16, which means we get nearly 16 times improvement in MTTF).

4.3.3 Evaluation with other configurations

We repeated our experiments with other hierarchical setups, varying both system size and sub-cluster size. However, due to lack of space, we present only a summary of our results. In general, given a N node system, varying the sub-cluster size resulted in a corresponding variation in partitioning of the workload. As shown in Section 4.1 and 4.2, beyond a certain point the hardware is no longer the limiting factor, availability and reliability vary according to the partitioning of workload and are determined by the super-cluster size, P. Our results from the sensitivity analysis of availability (Section 4.1.1) and reliability (Section 4.2.1) show that even at upper levels of the hierarchy, the middleware failure and repair rates are the dominating factor in availability and reliability. Likewise, organizing the nodes into hierarchies with more than two levels indicate that the availability and reliability are primarily indicative of only the workload (and hence the middleware failure rates) at those levels. Even under a ‘workload-failure rate’ independence assumption, the availability and reliability at higher levels vary only marginally compared to sub-clusters.

A number of parameters can determine the selection of a particular hierarchical structure. Obviously, hardware and middleware failure and repair rates and specified availability and reliability targets are important factors for determining the appropriate sub-cluster sizes. For example, the availability (Figure 5) and reliability (Figure 7) of the system can help determine minimum sub-cluster sizes. In general the guideline for choosing sub-cluster sizes is that the size should represent the point in system growth where the hardware reliability ceases to be a limiting factor.

5 Comparison of Clustering Architectures

The hierarchical architecture trades-off system symmetry for application fault boundaries. As a consequence of this, while application-induced middleware faults are contained within sub-clusters, additional overhead in terms of global event communication and structural complexity are incurred. The disadvantage of decreased symmetry is that although applications can be deployed anywhere within the hierarchical cluster initially, failover is possible only within the sub-cluster resulting in possible load skews. In order to deal with this, techniques for dynamic workload based migrations of application across sub-clusters are required. We leave such directions to future work.

While the virtual node selection made by the bridge middleware is susceptible to dependent failures, the intercluster communication segments are expected to have independent failure modes. In our experience, these primitives tend to be well tested and unlike application-induced events, are not a major-source of failures.

Flat clusters that utilize symmetric middleware architectures have limited scalability. The middleware maintains tight lease timers for better response times in detecting failed nodes. The lease-time must increase with the number of nodes in the system and cannot be lesser than the worst round trip time between any two nodes in the cluster. Thus, in flat clusters, failure response times worsen with geographic spread. Hierarchical clusters on the other hand are a natural way to combine remote nodes. For example, the local clusters can form one level of hierarchy and remote clusters may maintain consistent state at another level. The hierarchy allows the flexibility of using different lease timers at different levels and thereby improves response times for applications running over sub-clusters.

6 Related Work and Conclusion

A large body of existing research studies availability and reliability from the view point of storage arrays [14, 22] and controllers [15, 11]. Our work emphasizes the fact that in large scale-out systems, middleware (software) reliability is often the key determinant in system availability and reliability. Our approach proposes to provide graceful isolation of application-induced middleware failures, while retaining SSI, by organizing cluster storage services into a hierarchical overlay. Note that, our approach is different from resilient overlay networks such as RON [5], which are designed to improve connectivity in Internet environments.

While fault isolation can be achieved through partitioning of a single cluster into smaller independent clusters [4, 3], without care, SSI and the flexibility to access storage from anywhere within the system will be lost in the process. Our goal is to provide fault-boundaries while continuing to provide these features. Although techniques like software rejuvenation [21] through rolling restart can reset software state, such techniques cannot deal with application-induced failures which would be encountered irrespective of the age of the software. Alternatively techniques like N-version programming [6] can be utilized to decouple middleware instance failures. However, this technique may be expensive and different implementations based on the same specification may still have common bugs [13].

This paper has presented our investigation on the availability and reliability of a fault-tolerant middleware architecture that uses a pre-determined hierarchical structure. Determining application placements over the hierarchical structure in order to minimize cross-cluster traffic and issues relating to dynamic creation and maintenance of the hierarchical structure are topics of ongoing research.

References

[1] Isilon systems. *http://www.isilon.com*.

[2] Veritas Foundation Suite. *http://www.veritas.com/us/products/foundation*.

[3] Ceph: A scalable, high-performance distributed file system. In *OSDI*, 2006.

[4] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. FARSITE: federated, available, and reliable storage for an incompletely trusted environment. In *OSDI*, 2002.

[5] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *SOSP*, 2001.

[6] A. Avizienis. The N-version approach to fault-tolerant software. *IEEE Trans. Software Eng.*, 11(12):1491–1501, 1985.

[7] M. Burrows. The chubby lock service for loosely-coupled distributed systems. *OSDI*, 2006.

[8] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *OSDI*, 2006.

[9] C. Engelmann, S. L. Scott, C. Leangsuksun, and X. He. Symmetric active/active high availability for high-performance computing system services. *Journal of Computers (JCP)*, 1(8):43–54, 2006.

[10] J. S. Glider, C. F. Fuente, and W. J. Scales. The software architecture of a SAN storage control system. *IBM System Journal*, 42(2):232–249, 2003.

[11] S. W. Hunter and W. E. Smith. Availability modeling and analysis of a two node cluster. In *5th Intl. Conference on Information Systems, Analysis and Synthesis*, 1999.

[12] R. K. Iyer, D. J. Rossetti, and M. C. Hsueh. Measurement and modeling of computer reliability as affected by system activity. *ACM Trans. Comput. Syst.*, 4(3), 1986.

[13] J. C. Knight and N. G. Leveson. An experimental evaluation of the assumption of independence in multiversion programming. *IEEE Trans. Softw. Eng.*, 12(1):96–109, 1986.

[14] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks RAID. *SIGMOD Rec.*, 17(3), 1988.

[15] K. K. Rao, J. L. Hafner, and R. A. Golding. Reliability for networked storage nodes. In *DSN '06*, 2006.

[16] F. B. Schneider. Replication management using the state-machine approach. *Distributed systems (2nd Ed.)*, 1993.

[17] S. Seshadri et al. A fault-tolerant middleware architecture for high availability storage services (extended version). *http://www.cc.gatech.edu/˜sangeeta/S3Reliable.pdf*.

[18] J. Slember and P. Narasimhan. Living with nondeterminism in replicated middleware applications. *Middleware*, 2006.

[19] M. Trachtenberg. A general theory of software-reliability modeling. *IEEE Trans. on Reliability*, 39(1):92–96, 1990.

[20] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1982.

[21] K. Vaidyanathan, R. E. Harper, S. W. Hunter, and K. S. Trivedi. Analysis and implementation of software rejuvenation in cluster systems. In *SIGMETRICS '01*, 2001.

[22] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. In *SOSP*, 1995.

[23] I. Wolfram Research. *Mathematica*. Version 5.0 edition.

[24] J. J. Wylie, M. W. Bigrigg, J. D. Strunk, G. R. Ganger, H. Kiliççöte, and P. K. Khosla. Survivable information storage systems. *Computer*, 33(8), 2000.

[25] D. Zeitler. Realistic assumptions for software reliability models. In *IEEE Int'l Symp. Software Reliability Eng.*, 1991.