An Online Controller Towards Self-Adaptive File System Availability and Performance

Xin Chen[1], Benjamin Eckart[1], Xubin He[1], Christian Engelmann[2], and Stephen Scott[2]

[1]Department of Electrical and Computer Engineering
Tennessee Technological University
Cookeville, TN 38505, USA
{xchen21, bdeckart21, hexb}@tntech.edu
[2]Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN 37831, USA
{engelmannc, scottsl}@ornl.gov

Abstract

At the present time, it can be a significant challenge to build a large-scale distributed file system that simultaneously maintains both high availability and high performance. Although many fault tolerance technologies have been proposed and used in both commercial and academic distributed file systems to achieve high availability, most of them typically sacrifice performance for higher system availability. Additionally, recent studies show that system availability and performance are related to the system workload. In this paper, we analyze the correlations among availability, performance, and workloads based on a replication strategy, and we discuss the trade off between availability and performance with different workloads. Our analysis leads to the design of an online controller that can dynamically achieve optimal performance and availability by tuning the system replication policy.

1 Introduction

Current large-scale distributed file servers typically consist of tens of thousands of nodes with petabytes of storage capacity, running on file systems such as the Lustre file system [1] and GFS [2]. One primary and common challenge of these large scale systems is how to provide high data availability and high performance services to users. Although many fault tolerance techniques have been proposed and used in both commercial and academic distributed file systems to achieve high availability, unfortunately, most of them, including replication [3] and RAID [4], have performance issues to some extent. For example, replication technology may increase the number of replicas for higher availability at the cost of occupying more I/O and network resources [3, 5], and RAID may sacrifice performance by increasing reconstruction speed to achieve higher availability [6]. In summary, high system availability typically sacrifices system performance, causing a trade-off to exist between the two. Thus, tuning systems for both optimal online system availability and performance is an outstanding problem for system designers and administrators.

With the increasing size and complexity of large-scale storage systems, manually tuning a system to achieve optimal online system availability and performance is impractical, difficult, and most likely error-prone

[7]. Furthermore, even if a fault tolerance approach claims to tolerate runtime failures, such as disk failures and unreliable network connectivity [8], it must take system workloads and application characteristics into account, or else the approach still may not achieve the expected objective. The reasons are that: (1) there is a high probability [7, 9, 10, 11] that software bugs and hardware transient failures occur due to higher system activities; (2) applications with different characteristics benefit from different fault tolerance policy [12].

In this paper, we present an online controller for distributed file systems based on a replication technique. The controller is able to dynamically tune the strategy employed for the replication process to achieve an online optimal availability and performance objective when runtime failures occur. There are two components in this design: system identification and controller implementation. For the task of system identification, two mathematical models are constructed to show the effects that the number of replicas have on system performance and availability, and we analyze the resulting model. For the task of controller implementation, the basic algorithm of the controller is presented and analyzed.

2 Replication strategy

Replication [13] is a widely used fault tolerance approach in distributed file systems. The replication strategy in this paper is a form of the primary/backup approach [14]: one storage node is designated as a head (primary) node; a given file or data object is replicated among several different storage nodes; and a master daemon exists, which is in charge of detecting the failures of storage nodes. As depicted in Figure 1, when an update to a file arrives, it is directed to the head node; after atomically updating in the head node, the updates are distributed to other nodes.

3 Mathematical Model

A key issue in the controller is how to dynamically tune the replication strategy to achieve an optimal system availability and performance objective. The focus of the model is on the number of replicas, which plays a key role in both system availability and performance [3, 12]. We assume there exists an optimal number of replicas for each file in the system r_{opt}, which means the

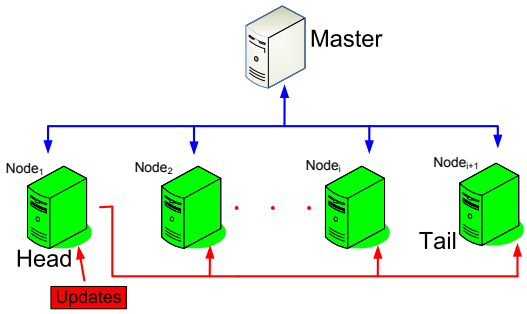

Figure 1. Replication strategy

expected system availability is guaranteed at the cost of the tolerated performance loss. Let r denote the number of replicas for each data object or file. When some failures occur and result in the degradation of system availability, r should be adjusted to adapt this change. There are two cases shown in Figure 2:

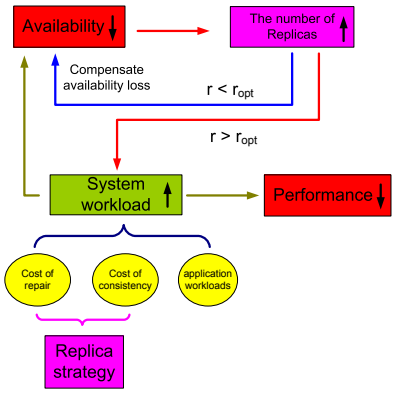

Figure 2. The analytical model

- $r < r_{opt}$: the degradation of system availability may not be compensated, or there is still some space to improve system availability with the acceptable performance loss.
- $r > r_{opt}$: the system may suffer more availability degradation and performance loss. Since the overhead of replication consists of the cost of repair and consistency, too many replicas implies higher replication overhead, which results in the degradation of performance. Meanwhile, studies in [9, 10] show that a higher system workload has a strong correlation to a higher failure rate, which has a negative impact on the system availability.

Thus, to make a decision of the optimal number r_{opt}, two questions need to be answered: (1) How much per-

formance will be sacrificed as a result of increasing the number of replicas? (2) What's the runtime system availability with a replica number r? Two models, performance and availability models, are constructed to explore these two questions.

3.1 Assumptions

To narrow the scope of the discussion, several basic assumptions will be applied to both performance and availability models:

1. The probability of a node to be unavailable is treated as an independent variable. The probability of correlated failures of nodes is not discussed here.

2. Since bursty access patterns and huge update operations for a single file are typical characteristics of a file system workload for large-scale scientific computing applications [15], it is highly possible that most of the nodes process large update intensive workloads in some time interval. Thus, we assume system workloads w only consist of update operations in this model because they will be the dominant factor for performance.

3. The system is fail-stop [14]. It implies that (1) a node halts when it incurs the first failure; (2) a failure node can be detected by the system.

4. The resource of the system in terms of number of nodes n is constant. New nodes join the system immediately when failure nodes are detected.

3.2 Performance Model

In this model, we concentrate on the effect of the number of replicas on the system performance with update intensive system workloads, and conclude that the system will suffer great performance degradation with more replicas in some cases given the resource constraint. Here are several definitions used in the discussion:

- $C_r(i)$**:** denotes the cost of repair overhead when a replica is unavailable during update operation i;

- $C_s(i)$**:** denotes the cost of synchronization overhead that is necessary to maintain the consistency among the replicas for an update operation i, in terms of time to finish all synchronizations;

- $C(i)$**:** denotes the cost of an update operation i without replication in a failure-free environment, in terms of time to complete;

- $C_{total}(i)$**:** denotes the total cost of an update operation i in the system, in terms of time to complete.

The $C_{total}(i)$ is the sum of $C(i)$, $C_s(i)$, and $C_r(i)$; $C_s(i)$ and $C_r(i)$ are determined by the maximum values in $\{t_1, t_2, ..., t_n\}$ and $\{t'_1, t'_2, ..., t'_m\}$, respectively; t_n denotes the cost of nth synchronization for operation i; t'_m denotes the cost of mth repair for operation i if there are m replicas unavailable.

$$C_{total}(i) = C(i) + C_s(i) + C_r(i) \quad (1)$$
$$C_s(i) = t_{max} \in \{t_1, t_2, ..., t_n\}, (n \in \{1, 2, ..., r\})$$
$$C_r(i) = t_{max} \in \{t'_1, t'_2, ..., t'_m\}, (m \in \{1, 2, ..., r-1\})$$

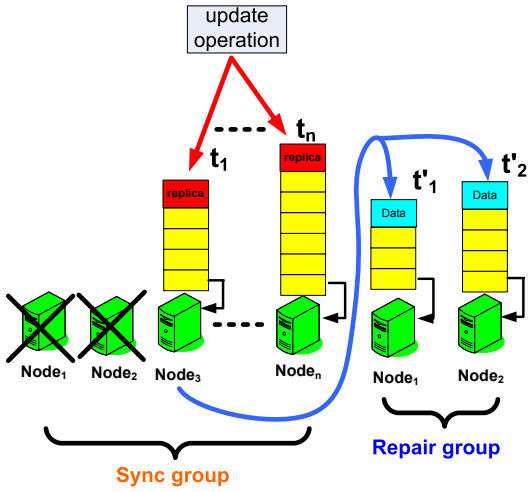

Figure 3. A process of an update operation

Figure 3 shows a case of an update operation. It shows that if the workloads are unbalanced among the nodes, $C_s(i)$ and $C_r(i)$ may be very large. Given by Formula 1, $C_{total}(i)$ is much higher than $C(i)$. And it also implies that an update operation may have a higher cost with more replicas, as the possibility of one replica node with a heavy workload increases.

Intuitively, it seems that if the workloads were balanced among all nodes, the overhead of an update operation with more replicas would be acceptable. We can try to verify this with a thinking experiment as depicted in Figure 4, given assumption 4 and that workloads are balanced among the nodes. Eight update operations are issued simultaneously; only eight nodes are in the system; there are two replicas associated with each operation; operations are processed one by one; no failure occurs during all updates in this experiment. To simplify the calculation, we assume each update operation i has same $C(i)$ denoted by α. When there is no replication, the total cost for completing all eight operations C, is $\beta + \alpha$ obtained by Formula 2, as all operations can be processed in parallel. Here, β denotes the processing cost of the previous operations in each node. However, if each update operation has two replicas, for the sake of simplicity, there is no head node; the two nodes are updated simultaneously. Thus, the total cost C', is $\beta + 2\alpha$, given by Formula 3.

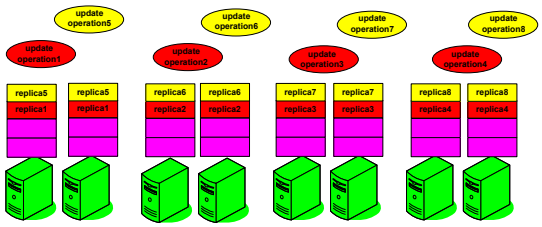

Figure 4. Eight nodes, two replicas and eight concurrent update operations

$$C = \beta + max(C(1), C(2), ..., C(8)) \quad (2)$$
$$= \beta + \alpha$$
$$C' = \beta + max(C(1), ..., C(4)) + max(C(5), ..., C(8)) \quad (3)$$
$$= \beta + \alpha + \alpha = \beta + 2\alpha$$

Let γ denote the relative execution time increase percentage in this experiment. In our example, $\gamma = 100\%$, using Formula 4 with $(\beta = 0)$.

$$\gamma = \frac{C' - C}{C} \times 100\% \quad (4)$$
$$= \frac{\alpha}{\beta + \alpha} \times 100\%$$

Let's generalize the result. Assume there are w concurrent update operations, n nodes, and r replicas, given that $m \leq n$ and $r \times m > n$. The formula of the relative execution time increase percentage $\Gamma(w, r)$ can be obtained:

$$C = \beta + \alpha$$
$$C' = \beta + \lceil \frac{r \times w}{n} \rceil \times \alpha$$
$$\Gamma(w, r) = \frac{C' - C}{C} \times 100\% = \frac{\lceil \frac{r \times w}{n} \rceil \times \alpha - (\beta + \alpha)}{(\beta + \alpha)} \times 100\%$$
$$= \lceil \frac{r \times w}{n} \rceil - 1, (\beta = 0) \quad (5)$$

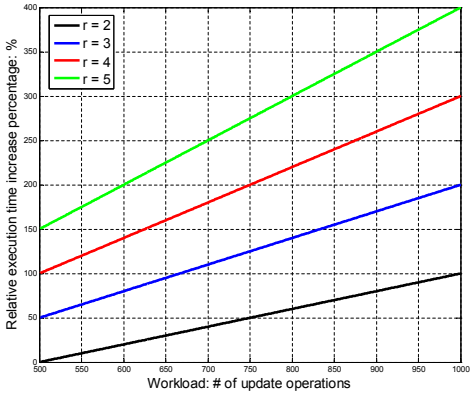

Figure 5. Relative execution time increase percentage

Formula 5 suggests two cases that increasing replicas would not incur too much performance loss (longer execution time) if: (1) the workload w is not too large; (2) there are sufficient nodes n. Figure 5 shows that with more concurrent update operations and more replicas, the performance drops quickly.

3.3 Availability Model

Recent studies [9, 10, 11] on failures in petascale file systems conclude that a higher system workload

has a strong correlation to a higher failure rate, which can imply lower system availability. In this model, the correlation between workloads and system availability is explored.

A widely used failure growth model, Goel-Okumoto model [16] assumes that the failure arrival process is a non-homogeneous Poisson process: the failures experienced by time t follow a Poisson distribution [17]. In our model, we let the failures experienced by workloads w follow a Poisson distribution instead of time t, and that the process has independent and stationary increments. The probability of failures $p(w)$, over workloads is given by the formula:

$$p(w) = 1 - e^{-\lambda w} \tag{6}$$

where λ denotes the average rate of failures in workloads w of the process. λ must to be estimated from the collected data.

In [8], the availability of a data object can be obtained by $1 - p(w)^r$, given r replicas. With Formula 6, we can obtain a formula of a predicted availability over workload w:

$$A(w) = 1 - (1 - e^{-\lambda w})^r \tag{7}$$

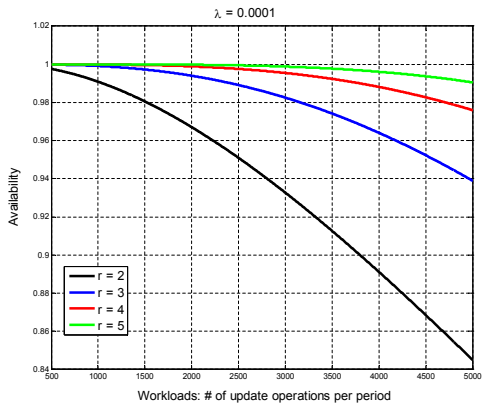

Figure 6. Availability $\lambda = 0.0001$

Figure 6 shows that with a smaller $\lambda = 0.0001$, we can get significant availability improvement by creating more replicas, especially when the workload is big, given the tolerance of high performance loss. However, Figure 7 shows that with a larger $\lambda = 0.001$, systems with a higher workload will not benefit greatly, in terms of availability, by increasing the amount of replicas.

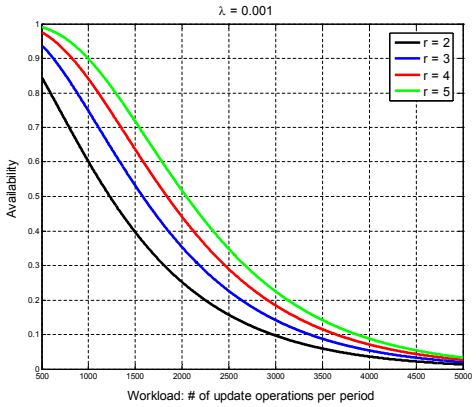

Figure 7. Availability $\lambda = 0.001$

4 Controller Design

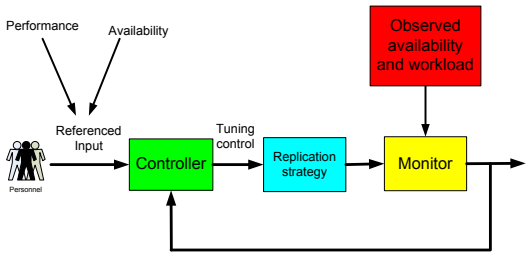

Figure 8. The work flow of the controller

Figure 8 shows the basic work flow of the controller: the desired availability and performance objective is set at the beginning, and then the controller takes the setting value and the feedback information from a monitor into account to tune the current replication strategy. The process will repeat until the difference between the pre-set values and measured values is tolerated.

The job of the monitor is to record the workloads n in a period, record the number of failures w_f in n, and calculate the availability via Formula 8 [16].

$$\begin{aligned} A_{monitor} &= \frac{w_{monitor} - w_{f-monitor}}{w_{monitor}} \\ &= 1 - \frac{w_{f-monitor}}{w_{monitor}} \end{aligned} \tag{8}$$

The basic job of the controller is to make a decision of a optimal number of replicas via the feedback information from the monitor. Since it is very possible that

the performance loss caused by more replicas cannot be tolerated, Jiaying et al. [12] suggests that the system can go back to the latest check point and restart to achieve both availability and performance objectives. The basic algorithm for our controller design is shown below:

Algorithm 1 : Controller Design

Input: $w_{monitor}, w_{f-monitor}, \gamma_{max-tolerance}$
Output: r_{opt}

$\lambda \leftarrow w_{f-monitor}/w_{monitor}$;
$A_{monitor} \leftarrow 1 - \lambda$;
if $A_{monitor} < A_{setting}$ **then**
  $r \leftarrow \ln(1 - A_{setting}) / \ln(1 - e^{-\lambda w_{monitor}})$;
  $\gamma \leftarrow \Gamma(w_{monitor}, r)$;
  if $\gamma > \gamma_{max-tolerance}$ **then**
    Go to the latest check point, and restart;
  else
    $r_{opt} \leftarrow r$;
    return r_{opt};
  end if
else
  return the previous r_{opt};
end if

5 Related Work

Douceur and Wattenhofer [3], Xin et al. [6], and Weatherspoon and Kubiatowicz [5] have previously shown the existence of a trade off among different fault tolerance strategies for system availability and performance. The importance of self reconfiguration has been demonstrated by the work of Ghandeharizadeh et al. [18]. The trade off between availability and performance and the importance of self reconfiguration form the basis and motivation behind this research. Ang and Tham [7] discuss the issues of load-dependent node availability, which provides a strong support for one of the basic assumptions of our model. Parekh et al. [19] present a software controller design used in performance management; we adopt their method here to design an online controller for maintaining an optimal system availability and performance.

6 Conclusion and Future Work

In this paper, two mathematical models are constructed to explore the correlation among availability, performance, and workload based on a replication strategy. Based on our analysis, we conclude that (1) system performance incurs significant degradation with more replicas in a period of high updating activities; (2) availability can have an expected improvement with more replicas when the failure rate is low. When failures are more frequent, however, replication may not be able to help the system to achieve the desired availability without sacrificing too much performance. In this case, it is better to go to the latest checkpoint and restart, instead of increasing replication. Lastly, we proposed an online controller which will help a system achieve an optimal runtime performance and an optimal availability via dynamically tuning the system replication policy. We are currently validating the proposed models via simulation and failure traces analysis, and in the future, we will implement such a controller in a parallel file system.

References

[1] "Lustre". Febrary 2008. `http://wiki.lustre.org/index.php?title=Main_Page`.

[2] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system". In *"SIGOPS Oper. Syst. Rev."*, volume 37, pages 29–43, 2003.

[3] John R. Douceur and Roger P. Wattenhofer. "Large-Scale Simulation of Replica Placement Algorithms for a Serverless Distributed File System". In *MASCOTS'01: Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01)*, page 311. IEEE Computer Society, 2001.

[4] David A. Patterson, Garth Gibson, and Randy H. Katz. "A case for redundant arrays of inexpensive disks (RAID)". In *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, 1988.

[5] H. Weatherspoon and J. Kubiatowicz. "Erasure Coding vs. Replication: A Quantitative Comparison". In *IPTPS 2002: Peer-To-Peer Systems: First International Workshop*.

[6] Qin Xin, Ethan L. Miller, Thomas Schwarz, Darrell D. E. Long, Scott A. Brandt, and Witold

Litwin. "Reliability Mechanisms for Very Large Storage Systems". In *MSST '03: Proceedings of the 20 th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST'03)*, page 146. IEEE Computer Society, 2003.

[7] Chee-Wei Ang and Chen-Khong Tham. "Analysis and Optimization of Service Availability in a HA Cluster with Load-Dependent Machine Availability". *Parallel and Distributed Systems, IEEE Transactions on*, 18(9):1307–1319, Sept. 2007.

[8] Kavitha Ranganathan, Adriana Iamnitchi, and Ian Foster. "Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities". In *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, page 376. IEEE Computer Society, 2002.

[9] Ramendra K. Sahoo, Anand Sivasubramaniam, Mark S. Squillante, and Yanyong Zhang. "Failure Data Analysis of a Large-Scale Heterogeneous Server Environment". In *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks*, page 772. IEEE Computer Society, 2004.

[10] Bianca Schroeder and Garth A. Gibson. "A large-scale study of failures in high-performance computing systems". In *DSN '06: Proceedings of the International Conference on Dependable Systems and Networks*, pages 249–258. IEEE Computer Society, 2006.

[11] Bianca Schroeder and Garth A. Gibson. "Understanding Failures in Petascale Computers". In *SciDAC 2007*, 2007.

[12] Jiaying Zhang and Peter Honeyman. "Performance and Availability Tradeoffs in Replicated File Systems". October 2007. `http://www.citi.umich.edu/techreports/`.

[13] Hui-I Hsiao and David J. DeWitt. "A Performance Study of Three High Availability Data Replication Strategies". In *PDIS '91: Proceedings of the first international conference on Parallel and distributed information systems*, pages 18–29, 1991.

[14] Robbert van Renesse and Fred B. Schneider. "Chain Replication for Supporting High Throughput and Availability". In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 7–7, 2004.

[15] Feng Wang, Qin Xin, Bo Hong, Scott A. Brandt, Ethan L. Miller, and Darrell D. E. Long. "File System Workload Analysis for Large Scale Scientific Computing Applications". In *MSST '04: Proceedings of the 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST'04)*, 2004.

[16] Jeff Tian, Sunita Rudraraju, and Zhao Li. "Evaluating Web Software Reliability Based on Workload and Failure Data Extracted from Server Logs". *IEEE Trans. Softw. Eng.*, 30(11):754–769, 2004.

[17] Michael Grottke. "Software Reliability Model Study". 2001. `http://www.statistik.wiso.uni-erlangen.de/lehrstuhl/migrottke/SRModelStudy.pdf`.

[18] Shahram Ghandeharizadeh, Shan Gao, Chris Gahagan, and Russ Krauss. "An On-Line Reorganization Framework for SAN File Systems". In *"Advances in Databases and Information Systems"*, pages 399–414. Springer Berlin, 2006.

[19] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. "Using Control Theory to Achieve Service Level Objectives In Performance Management". In *"Real-Time Syst."*, volume 23, pages 127–141. Kluwer Academic Publishers, 2002.