Analysis of the reliability/availability of distributed file systems in large-scale systems: A case study using simultaneous simulation

Shravan Gaonkar and William H. Sanders
Information Trust Institute, Coordinated Science Laboratory and
Department of Electrical and Computer Engineering,
University of Illinois at Urbana-Champaign
1308 W. Main St., Urbana, IL, U.S.A.

I. Introduction

Historically, scientific computing has driven large-scale, scientific computing to its limits. Towards the end of this decade, we are likely to achieve Petascale computing that would benefit many applications, such as climate and environmental modeling, 3D protein molecule reconstruction, design of spacecraft and aircraft, nanotechnology, and genetic engineering. While supercomputer performance has improved by over two orders of magnitude every decade [1], the performance gap between the individual nodes and the overall processing ability has widened drastically. That has led to a shift in the paradigm of designing supercomputers, from a centralized approach to a distributed one that supports heterogeneity. While most high-performance computing environments require parallel file systems, there have been several file systems, such as GPFS [2], PVFS2 [3], and GFS [4], that have been specifically proposed to support very large-scale scientific computing environments.

Our paper makes two contributions. We build a model of a large-scale computing system that uses a parallel file system, and we study the scalability and reliability of the file system over several hundred thousand processors. We evaluate our models with a new simulation methodology called *simultaneous simulation of alternate system configurations* (SSASC) [5], which enables us to evaluate simulations faster than the traditional approaches do. Using the example cluster file system, we show that SSASC can be used to analyze large scale models in an efficient manner compared to traditional simulation.

II. The Cluster File System

We now describe the abstract structure of a cluster file system described mentioned above, whose variant flavors are deployed in the HPC framework [2]–[4].

A. Architecture

Usually, a large-scale file system consists of a single *master* and multiple *file servers*. The master maintains the file system's metadata, which includes the access control information, mapping of files and directory names to their location, and mapping of allocated and free space. The master serves the metadata information only to the clients. Usually, files are divided into equal sized blocks. The blocks are stored on file servers. The file servers maintain the data and information about the file system and serve the clients with the required information using the file blocks. For reliability, the file blocks are replicated over multiple file servers. The *client* communicates first with the master (if necessary) and then with the appropriate file server to perform the read or write operation. The master is a central node that maintains the state of the entire file-system. The master node receives requests for the location of the file blocks. The master replies with the corresponding location of the file blocks on the file servers. The master does not maintain a persistent view of the file system; it updates that information from the file servers. The file servers are dynamic resources. New file servers can be added. Failed servers are purged automatically. Each file server can be as simple as a Linux file server running on off-the-self systems or a dedicated enterprise class storage server. The file server nodes can further augment reliability mechanisms, such as RAID, to protect the content on the individual server. The client nodes proxy the requests on behalf of the user or applications. The clients first access the master server to determine the location of the file/directory. Subsequent requests are made to the appropriate file server. Modern cluster file systems cache the requested files on the client side and only transmit or propagate write requests. GFS [4] explicitly avoids caching, since its file system is designed very specific to its requirement and workload characteristics.

The applications that run on the cluster are typical scientific computational workload. Each application runs a set of computational and I/O tasks. The application often uses multiple compute nodes for the task. All the nodes work more or less in synchronization. We obtain the various characteristics of a scientific workload, such as job arrival rate, job size, and job run length from [6]. The number of jobs are often distributed exponentially with respect to the job size.

B. Failure Model

We assume that the failure of the master server stops the progress of all the jobs waiting to be served by the master. The file system becomes unavailable until the master server is repaired and restarted for the applications. When a file server node fails, jobs running on it are postponed until the file server is repaired. Nodes can have multiple disks. The failure rate of the file server depends upon the failure rate of the disks. The disk failure MTTF is obtained from [7]. It is

possible that file server nodes use sophisticated redundancy and reliability techniques, such that advanced error-handling mechanisms result in very low failure rates.

Correlated Failure: Recent literature has discussed the importance of modeling correlated failures. We model two types of correlated failures: (a) correlated errors due to error propagation and (b) correlated errors due to hardware fatigue. It is possible that errors generated in the master server are propagated to the file server node(s). For every failure event in the master node, there is a small probability that the error will be propagated to the file server nodes. In the second kind of correlated failures, the authors of [7] have shown that disks that fail once have a greater likelihood of failing again. We incorporate the correlated model to represent the failure of the hardware components in the models.

III. Simultaneous Simulation of Alternative System Configurations

Simulation is applied in numerous and diverse fields, such as manufacturing systems, communications and protocol design, financial and economic engineering, operations research, and design of transportation networks and systems, among many others. Unfortunately, simulations of large systems take a lot of computational resources and time. Even though clock speed of the sequential processors improves every year, the need to model and simulate complex systems also rises every year. Furthermore, the real utility of simulation lies in comparing alternatives before actual implementation [8]. That means that the system model has to be simulated multiple times for a large number of design configurations and parameter values to determine the best design configuration. To perform a thorough analysis of a large number of configurations with varying system design parameter values, it is important to develop efficient simulation methods that can evaluate a large number of alternative system configurations quickly and accurately. If the system under evaluation is scrutinized carefully, one will notice that changing the system configuration or parameter values does not alter the behavior completely. Rather, much of the system behavior is similar for most of the possible alternative configurations.

In our work on SSASC [5], we extended the ideas of Vakili [9] and Chen and Ho [10] with a methodology that exploits the structural similarity among the alternative configurations. Our technique exploits that structural similarity, consolidates the common events among the configurations into a single union of events, and generates a single *enabled event set*. Using the common enabled event set, it combines the advantages of the *single clock* technique [9] with adaptive uniformization to build an efficient SSASC algorithm. The contribution of the SSASC is an efficient simulation algorithm that evaluates all the alternative configurations of a system simultaneously, even when event rates vary greatly (by orders of magnitude), as is often the case in dependability evaluations.

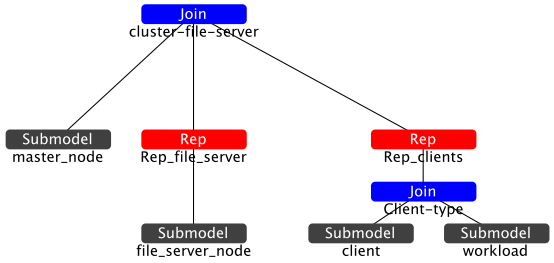

Fig. 1. Compositional SAN model of cluster file system.

While the algorithm is efficient compared to traditional simulations, there is room for improvement through elimination of non-optimal configurations using statistical methods. We discuss the possible approaches in Section VI.

IV. Stochastic Activity Network Model: Cluster File System

In this section, we describe the details of the modeling of a typical cluster file system using Möbius [11]. Figure 1 shows the *compositional model*. Each submodel in the composed model represents part of the execution process in the cluster file system. Each submodel is represented as a *stochastic activity network* (SAN) at the leaf node level. The model has one master node, m file server nodes replicated by "Rep-fserver," and n types of clients. Each client type simulates the characteristics of a particular parallel computing application. The index of the client type represents the magnitude and scale of the parallel application. A bigger client type index represents a larger application. For each client type, i replications of the client simulate the simultaneous users of that particular client (as represented by "Rep-1" to "Rep-n"). Each client request type is represented by a special token of type i.

When the cluster file server is started, all the nodes begin in a *working* state. Whenever a file request arrives at the master node, the master node determines the location of the file on the file server node and forwards the token to the respective file server(s).

The file server processes the requests and puts the tokens into a place labeled *processed_requests*. The file server only processes the requests if it is in the *working* state. The state of the file server is determined by the state of the disks. If the file server uses a simple Linux file server, its state is said to be *available* if the disks are available. If the file server uses a RAID file server, then its availability is determined by the availability of the RAID. Disk failure rates are modeled according to the observations documented in [7].

The client nodes are initially in the state *idle*. Each of them sends a request to the master node for the file location and goes into the *wait_for_file_server* state. Once the master node

returns its request, the client sends additional requests to the file server or goes back to the *idle* state. Along with each request to the file server nodes or the master node, the client increments a global counter labeled *global_request_count* and sets a timer to expire at the time by which it expects to be serviced. If the client is serviced before the timer expires, then it increments a global counter labeled *global_service_count*. With those two counters, we can determine the global lifetime count of total requests and successful services by the cluster file system.

Since there are n types of client nodes, each with specific workloads, tokens representing requests from the different types of clients are distinguished using the "extended places" feature from Möbius. The master node and file server nodes service the client nodes based on the priorities of their types. Client requests of type n_i get higher priority than client requests of type n_j if $i > j$.

The network latency is not explicitly modeled, but the transmission delay is controlled through addition of additional activities that mimic the latency. Depending upon the type of network support, the latency of the transmission can be determined and set appropriately using input variables. The latency in network transmission appears between the master and the file servers, the master and the clients, and the file servers and the clients.

Reward Metrics

The utility of the cluster file system cannot be gauged with a single reward measure. Therefore, we define different metrics to capture the different aspects of the model.

Availability: Availability of the cluster file system is defined as its ability to serve the client nodes. Here it is defined as the fraction of time when all the file server nodes along with the master node are in the *working* state.

Timeliness ratio: Response time is the time perceived by the end users as the interval between the time they send a request to the file server and the time the server gets back to them with results. In large parallel programs, it is quiescent time when the programs are performing I/O jobs or checkpointing. Here, we measure the *timeliness ratio*, which is defined as the ratio of total successful deliveries (delivery on time) of the data requests to the total number of client requests. In particular, it is the ratio of *global_service_count* to *global_request_count*.

Cost: Availability always comes at a price. Often, designers would like to know the trade-off. For instance, RAID file servers provide higher availability at a higher premium. We can measure and compute the cost of the components and their maintenance for different configurations of the file server. The relative costs can be compared against *availability* and *timeliness ratio* to determine the trade-offs.

V. Evaluation

We present here the results for preliminary evaluation of the SAN model for a sample single workload type. The number of client nodes accessing the file server is increased from 10 to 500. The number of file servers is increased from 4 to 10. We discuss the impact of these parameter values on the reward metric *useful response*.

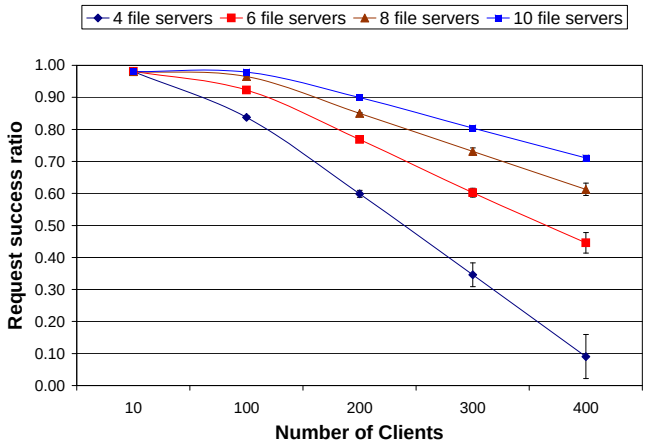

Fig. 2. Timeliness ratio as the load (number of client nodes) on the file server increases at 95% confidence interval.

Timeliness ratio

The reward measure on a cluster file system that most interests the user community (client nodes) is the response time. One could also measure the fractions of responses that were delivered before the expiry of the client's deadline for requests. Figure 2 gives the *timeliness ratio* as a function of the number of clients for four different cluster file server sizes.

The curves in the figure illustrate certain intuitive and non intuitive results. The timeliness ratio degrades as the number of client nodes is increased. That is expected, since the load on the server causes the master node and the file server nodes to miss some of their deadlines. The interesting aspect of the curves is that they have a knee point after which the ratio deteriorates rapidly. Furthermore, the cluster file server's average on-time response to requests only improves marginally with addition of each additional file server. In Figure 2, notice that the gap between the curve "4 file servers" and "6 file servers" is twice as large as the gap between "6 file servers" and "8 file servers". The implication is that there is a trade-off between the capability of the file server and the workload, and that it is not linear. The preliminary results presented serve two purposes. First, they allow us to test the correctness of the SAN model and its applicability to analysis of results that impact the deployment of cluster file servers on very large systems. Second, they show us that interesting observations can be made about the cluster file systems using an abstract modeling tool, and that there is an incentive to continue further research on this topic as discussed in the next section.

Efficiency

We also provide some preliminary insights on comparison of SSASC with traditional systems. We set the number of file servers to 4, but vary the number of client nodes from 10 to 256 in steps of 1. We run 1000 batches of each

TABLE I
COMPARISON OF SIMULATION TIME BETWEEN TRADITIONAL SIMULATOR AND SSASC

Number of client nodes	Total simulation time in seconds traditional simulation	SSASC	Speedup
10	74.08	11.26	5.78
64	5073.21	352.55	14.39
128	22669.58	997.84	22.72
256	89813.65	2999.78	29.94

configuration. In traditional simulation, each configuration is run independently, while in SSASC all of them are run simultaneously. Table I shows the speed-up obtained by running SSASC compared to the traditional simulation; it is of an order of magnitude. This improvement in efficiency would let us evaluate the system on a larger scale, enabling us to analyze the model critically and provide richer insights.

VI. FUTURE WORK

A. Improving Simulation Efficiency

While we want to develop models to improve understanding of the behavior of cluster file systems in very large-scale systems, we are also interested in developing techniques to reduce simulation execution time. Apart from improving the efficiency of the simulation algorithm itself, we propose to develop appropriate statistical methods to avoid fallacious conclusion and poor decisions while trying to identify the best optimal design choice for a model. Here, we focus on obtaining statistical efficiency by reducing the variance of the output random variables (reward metrics). Since we are comparing two or more alternative system configurations, *common random numbers* (CRV), are one approach to reducing variance. Since SSASC automatically uses CRV, it reduces simulation execution time by a magnitude of 1.5 to 1.7 times.

While the CRV approach is mathematically sound, it is not scalable to a large number of alternative configurations. Earlier work by Nelson et al. [12] proposed a two-stage divide and conquer process, in which they screen out the worst candidates in the first stage and then select the best system out of the remaining alternatives with probability p^*. In the process of screening out the worst alternatives in the first stage, Nelson et al. divide the initial set into disjoint sub groups. Due to the disjoint groups, the screening process loses the cross-correlation that exists between the configuration across sub groups. While their approach augments the CRV approach by improving scalability, We propose to obtain stronger results (better p^*) when we retain the cross-correlation across sub groups. One approach is to divide the initial set into sub groups such that a subset of alternative configurations occurs in multiple sub groups. Intuitively, it seems that a configuration that gets screened out in multiple sub groups is more likely to be a bad design choice, and therefore needs to be purged. We are currently working on developing the required theory to compute the (better) p^* for choosing the best system out of N alternatives.

B. Improving the SAN Model

The SAN model described in Section IV is preliminary. Modern cluster file systems provide a large number of features that improve the reliability and availability of the file system. Currently, the read and write requests are not distinguished. Much as we can distinguish requests from different client types, we might also like to distinguish read and write requests, and process them. Some file systems, such as GPFS, cache the files at the client node. Only the write updates are propagated to the file servers. While such caching mechanisms improve efficiency, they add additional burden on the servers, to maintain state.

While each file server node might implement its own internal reliability mechanism, an interesting feature missing from the current SAN model is the impact of the replication mechanism implemented by the cluster file server. The cluster file server replicates the file blocks as described in Section II. A back-of-the-envelope calculation of the overall availability of all file server nodes for a single client request is about $1-q^R$, where R is the replication count and q is the probability that the file server has failed. That information can be incorporated into the simulation model where the client tries R replicas of the file server before it marks the request as failed. With the additional information, we can compare and contrast internal replications versus global replication in terms of cost and other reliability metrics.

VII. CONCLUSION

In this extended abstract, we described a case study of a very large-scale cluster file system. We discussed the structure of the model, and laid out the details on how to evaluate this real case study with the new SSASC algorithm, which would reduce the analysis and evaluation time using tools like Möbius.

While we provided preliminary results on the case study of the cluster file system, we plan to do an exhaustive analysis of the model using SSASC. We plan to incorporate the improvements discussed in Section VI-B in the SAN model. Some of the features require minor modification the Möbius tool itself. Furthermore, we are going to address the issues raised in Section VI-A to reduce the computation required to determine the optimal configuration of a cluster file system with respect to our reward measures.

VIII. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0406351. We would like to thank Jenny Applequist for her editorial comments.

REFERENCES

[1] E. Strohmaier, J. J. Dongarra, H. W. Meuer, and H. D. Simon, "The marketplace of high performance computing," *Parallel Computing*, vol. 25, no. 13–14, pp. 1517–1544, 1999. [Online]. Available: citeseer.ist.psu.edu/strohmaier99marketplace.html

[2] F. Schmuck and R. Haskin, "GPFS: A shared-disk file system for large computing clusters," in *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*, 2002, pp. 231–244.

[3] W. Yu, S. Liang, and D. K. Panda, "High performance support of parallel virtual file system (pvfs2) over quadrics," in *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, 2005, pp. 323–331.

[4] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, Bolton Landing, NY, USA, 2003, pp. 29–43.

[5] S. Gaonkar and W. H. Sanders, "Simultaneous simulation of alternative system configurations," in *Proceedings of the 11th Pacific Rim Dependable Computing*, Changsha, Hunan, China, 2005, pp. 41–48.

[6] H. Li, D. Groep, and L. Wolters, "Workload characteristics of a multi-cluster supercomputer," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Springer Verlag, 2004, pp. 176–193, *Lecture Notes Computer Science* vol. 3277.

[7] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?" in *Proceedings of the FAST '07 Conference on File and Storage Technologies (FAST)*, 2007, pp. 1–16.

[8] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*. McGraw Hill, 2000.

[9] P. Vakili, "Massively parallel and distributed simulation of a class of discrete event systems: A different perspective," *ACM Trans. Model. Comput. Simul.*, vol. 2, no. 3, pp. 214–238, 1992.

[10] C.-H. Chen and Y.-C. Ho, "An approximation approach of the standard-clock method for general discrete-event simulation," *Control Systems Technology*, vol. 3, no. 4, pp. 309–318, 1995.

[11] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. Webster., "The Möbius modeling tool." in *Proceedings of the 9th International Workshop on Petri Nets and Performance Models*, September 11–14 2001, pp. 241–250.

[12] B. L. Nelson, J. Swann, D. Goldsman, and W. Song, "Simple procedures for selecting the best simulated system when the number of alternatives is large," *Oper. Res.*, vol. 49, no. 6, pp. 950–963, 2001.