

Improving Network Performance through Task Duplication for Parallel Applications on Clusters

Xiao Qin

*Department of Computer Science,
New Mexico Institute of Mining and Technology,
801 Leroy Place, Socorro, New Mexico 87801-4796
xqin@cs.nmt.edu*

Abstract

While data replication is widely used in clusters to provide fault tolerance, it can heavily stress communication networks and degrade overall performance of parallel applications. The performance degradation is particularly unacceptable with disk-write-intensive applications. As a result, data duplication management for parallel applications running on clusters is a significant and urgent challenge. This paper presents the design, implementation, and evaluation of a network-aware task duplication management system, or TUFF, where redundant data can be regenerated by corresponding duplicate tasks rather than directly replicating through networks. In addition, TUFF is capable of improving availability performance of parallel applications, because TUFF allows two replicas of each I/O-intensive task to be executed on two different nodes. We have implemented and evaluated TUFF using extensive simulations under a diverse set of workload conditions. Experimental results show that TUFF improves the overall performance of parallel applications running on clusters by efficiently reducing network resource consumption.

1. Introduction

Significant cost advantages, combined with rapid advances in middleware and interconnect technologies, have made clusters a primary and fast growing platform for parallel computing. Typical examples of parallel applications include computational fluid dynamics, weather modeling, and seismic exploration, to name just a few. These applications share a common feature of having extremely high communication and storage requirements. Therefore, the completion times of parallel applications largely depend on the effective usage of network and storage resources, in addition to that of CPU and memory.

To fully utilize the resources of a cluster, it is essential to provide the cluster with the capability of

tolerating node failures. Our ultimate goal is to make high-performance clusters more dependable in the sense that in presence of failures in one or more nodes, parallel applications should continue execution and produce correct results. To maximize the reliability of file systems on clusters, fault-tolerance can be effectively incorporated by the disk mirroring technique [23]. The question is how to maintain duplicate data in backup nodes to achieve optimal performance. Therefore, this paper addresses the issue of improving the availability performance of clusters coupled with fault-tolerant file systems. The approach proposed in this study is motivated by two facts: (1) reducing communication overhead can result in significant performance improvements for parallel applications [12], (2) software-based active replication is a commonly used technique for providing fault-tolerance using space redundancy.

For a file system that incorporates fault tolerance in the form of disk mirroring, it requires high network bandwidth to transfer duplicate data of a newly created or modified file from primary nodes to backup nodes. If workload conditions are I/O-intensive in terms of disk-write accesses, a huge amount of data has to be frequently copied from the primary nodes to backup nodes. Consequently, the cluster will inevitably endure very high communication load, which in turn degrades the overall performance. To alleviate the performance degradation, we propose a task duplication management system, or TUFF, to minimize communication overheads resulting from fault-tolerant file systems. This work is partly motivated by the observation that in many cases network resources are overloaded while CPU resources are under utilized. Under such circumstances, there is a great incentive to trade CPU for network resources. More importantly, since our solution is based on the software redundancy of computations, our approach can tolerate several failures in processors and memory devices in addition to those in disks.

In our design, we schedule a parallel application by redundantly executing some of the write-intensive tasks, which intensively write a huge amount of data to disks.

Allocating the duplicate of a write-intensive task to a node where the task's backup data is stored can avoid directly copying the backup data from the primary node, thereby eventually improving the performance of the whole system. In particular, for clusters with fault-tolerant file systems under communication-intensive workload conditions, our proposed approach can significantly achieve performance improvements in both completion time and reliability.

This paper is organized as follows. In Section 2, related work in the literature is briefly reviewed. Section 3 introduces system and application models, and section 4 presents the task duplication management system for fault-tolerant file systems (*TUFF*). Section 5 measures the completion time performance of the TUFF management system by extensive simulations. Finally, Section 6 concludes the main contributions of this paper and comments on future directions for this work.

2. Related work

In a cluster system, scheduling [8] and load-balancing [16] techniques can be used to improve the performance of parallel applications by fully utilizing idle or under-utilized cluster nodes. A number of distributed load-balancing schemes for clusters have been developed, primarily considering a variety of resources, including CPU [9], memory [1][22], disk I/O [15], or networks [14]. These approaches have been proven effective in increasing the utilization of resources in a cluster, assuming that the network connecting the nodes in a cluster is not a potential bottleneck. However, the interconnection network in a cluster can become a bottleneck, especially if cluster resources are time-/space-shared among multiple parallel jobs and some of these jobs are communication-intensive. A job is communication-intensive if it contains many parallel tasks that exchange data or synchronized among themselves frequently. Furthermore, this network bottleneck becomes severe if some of its tasks access large amount of data remotely, e.g., from disks located on other nodes.

There is a large body of excellent research focusing on the issue of improving communication performance over high-speed networks like FDDI, ATM, and Myrinet [9]. Research efforts have been made to build faster and cheaper communication mechanisms for clusters [7][9]. Alternatively, the communication performance can be improved by scheduling and load-balancing techniques [14]. For example, a communication-sensitive load balancer was proposed by Cruz and Park [5]. The balancer uses run-time communication pattern between processes to balance load. Orduña et al. proposed a criterion to measure the quality of network resource allocation to

parallel applications [13]. Based exclusively on this criterion, they developed a scheduling algorithm to fully utilize available network bandwidth. The above approaches can be considered complementary to our work, which is optimized to reduce communication load imposed by fault-tolerant file systems.

Parallel file systems (such as PVFS [3]) can deliver high-performance and scalable storage by connecting independent storage devices together. Fault-tolerance is one of the most important issues for parallel file systems on clusters. Recently, we designed and implemented a cost-effective, fault-tolerant parallel virtual file system (CEFT-PVFS), which provides parallel I/O services [23]. CEFT-PVFS incorporates fault tolerance in the form of disk mirroring by utilizing existing commodity disks on cluster nodes, thereby requiring no additional hardware. We have proposed four mirroring protocols to achieve different tradeoffs between reliability gain and performance degradation. The results show that CEFT-PVFS can improve the reliability of a cluster by a factor of 40. Regardless of the mirroring protocols employed in the CEFT-PVFS system, replicated data of a file has been copied from a primary node and transferred through the network to its corresponding backup node. Thus, the mirroring protocols are unable to reduce communication overhead induced from copying backup data.

Task duplication techniques for scheduling parallel applications have been addressed in [2][6][18]. These approaches can improve execution times of parallel applications by reducing the interprocessor communication overhead. However, these techniques are not applicable for a dynamic and resource-sharing cluster environment, since they are static in nature. Furthermore, none of these techniques considers I/O-intensive tasks that frequently access a file system, whereas the novelty of our approach is the seamless integration of scheduling to fault-tolerant file systems.

3. System and application models

3.1. System architecture

Figure 1 depicts a high-level structure of the system architecture, where each node maintains a load monitor, a load balancer, and a task duplicator. In this architecture, users can submit their parallel jobs (the terms job and application are used interchangeably throughout this paper) to the computing environment through client services. We assume that every job has a home node that is preferred for submission [17], based on the observation that input data of a job is usually stored in the home node or the job was previously created on its home node. Therefore, an initial distribution of jobs is already

performed by the client services that are aware of the home node of each job. In general, the load distributed in this way tends to be imbalanced because the client services simply distribute jobs without taking into account the load information. Thus, when the load monitor in a node detects that the system load is poorly distributed, the load balancer in this node will perform load balancing by dynamically assigning work from the overloaded node to other nodes with idle or under-utilized resources [16].

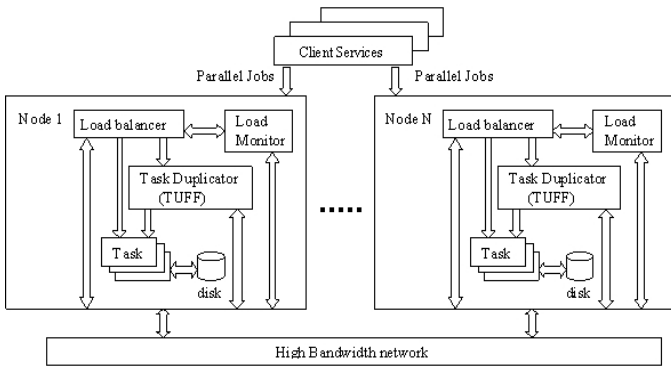


Figure 1. System architecture

If under loaded nodes are available in the system, tasks of a parallel job can be concurrently executed on these distinct nodes to take full advantage of distributed data sources and aggregate I/O bandwidth. If no under loaded nodes can be found in the system, the jobs will be executed locally on its home node.

Each load monitor collects information pertaining to the utilizations of its local CPU, disk, and network load and periodically broadcasts the local load information to other nodes. Consequently, the load balancer in each node is aware of reasonably up-to-date global load information from its load monitor [11][20]. It is to be noted that when the number of communicating nodes in the system becomes large or when the system load is heavy, the communication overhead can be excessive. To reduce this undesirable overhead, Shin and Chang proposed the concept of buddy sets and preferred lists [19]. The focus of our paper is on designing an effective task duplication management system for clusters with fault-tolerant file systems, whereas a thorough discussion of the load balancer [15][16] and load monitor is beyond the scope of this paper and thus will not be included in this paper.

3.2. System model

The network in our model provides full connectivity in the sense that any two nodes are connected through either a physical link (e.g., Myrinet interconnects) or a

virtual link (e.g., Ethernet interconnects). We assume that CPU and memory burden placed by data communication is relatively low and, therefore, such CPU and memory load (due to data communication) is not considered. This assumption is reasonable since communication can be directly handled by network interface controllers without local CPU's intervention or buffer in the main memory [9].

We assume that multiple parallel applications running on a cluster have the same priority, and the tasks of the parallel applications are locally scheduled in a round-robin fashion. This assumption is based on the fact that most operating systems apply a priority-based scheduling policy that eventually reduces to a round-robin policy when applications being executed are CPU-intensive [8]. Furthermore, because most parallel applications are coarse-grained, it is reasonable to assume a round-robin local scheduler for a time-shared cluster. Each task is assumed to read or write data locally.

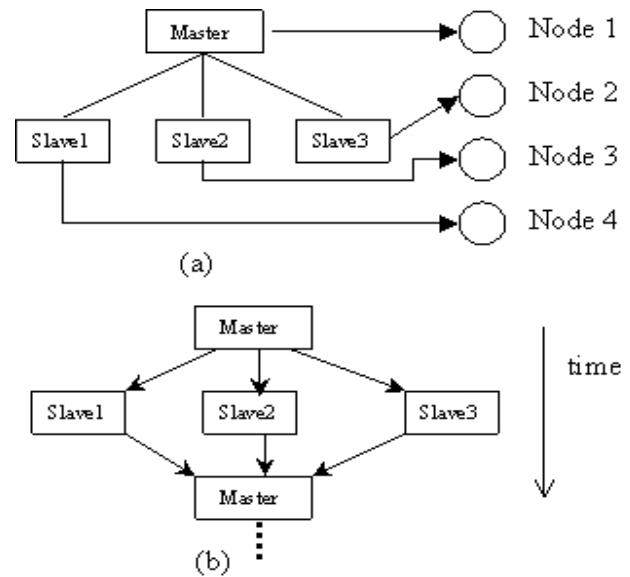


Figure 2. SPMD application model. (a) Four processes are allocated to four nodes; (b) The master communicates with the slaves.

3.3. Application model

In the SPMD application model, large data sets are decomposed across a group of nodes in a cluster. Processes (the terms process and task are used interchangeably throughout this paper) in a parallel job communicate with one another through a message-passing interface (Fig. 1). In this study, we focus on a bulk-synchronous style of communication, which has been used in the popular Bulk-Synchronous Parallel model [8][21].

This is the most common model for a variety of SPMD bulk-synchronous parallel applications, in which a master creates a group of slave processes across a set of nodes in a cluster (Fig. 2a). The slave processes concurrently compute during a computation phase before they are synchronized at a barrier where messages can be exchanged among these processes during the communication phase (Fig. 2b).

To design a load-balancing scheme that improves the effective usage of network resources in a cluster, we have to find a way to quantitatively measure communication requirements of parallel applications. Based on the communication requirements of the applications running on a cluster, we can approximately estimate the communication load of each node in the cluster.

In what follows, we introduce an application behavior model, which captures applications' requirements of CPU and network activities as well as that of disk I/O. This model is important and desirable because the TUFF system to be proposed shortly in Section 4 makes full use of this model to quickly calculate the load induced by parallel applications running on the cluster. Our model is reasonably general in the sense that it is applicable for both communication- and I/O-intensive parallel applications. Let N be the number of phases for a process (slave task) of a parallel job running on a node, and T_i be the execution time of the i th phase. T_i can be obtained by the following equation, where T_{CPU}^i , T_{COM}^i , and T_{Disk}^i are the times spent on CPU, communication, and disk I/O in the i th phase:

$$T^i = T_{CPU}^i + T_{COM}^i + T_{Disk}^i. \quad (1)$$

Therefore, the total execution time of the process is estimated at $T = \sum_{i=1}^N T^i$. Let R_{CPU} , R_{COM} , and R_{Disk} be the requirements of CPU, communication, and disk I/O, and these requirements can be quantitatively measured as:

$$R_{CPU} = \sum_{i=1}^N T_{CPU}^i, \quad R_{COM} = \sum_{i=1}^N T_{COM}^i, \quad \text{and} \\ R_{Disk} = \sum_{i=1}^N T_{Disk}^i.$$

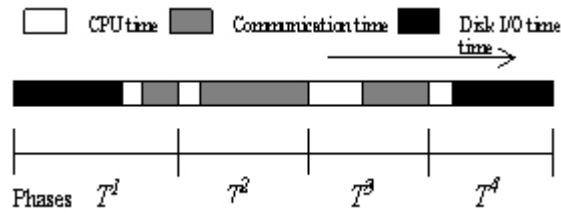


Figure 3. Example for the application model.

An example of the above notation is illustrated in Figure 3, where the process has four phases. In the first phase, the application spends a large fraction of time in reading data from a disk. The second phase spends a small amount of time on computation and the rest on communication. Thus, phase two is communication-intensive. The third phase is comprised of balanced CPU and communication requirements. The last phase is I/O-intensive in the sense that it spends a large fraction of time writing data to the disk.

We are now in a position to consider an execution time model for parallel jobs (the terms job, application, and program are used interchangeably) running on a dedicated (non-shared) cluster environment. Given a parallel job with p identical processes, the execution time of the job can be calculated as:

$$T_p = s_p + \sum_{i=1}^N \left\{ \text{MAX}_{j=1}^p \left(T_{j,CPU}^i + T_{j,COM}^i + T_{j,Disk}^i \right) \right\}, \quad (2)$$

where $T_{j,CPU}^i$, $T_{j,COM}^i$, and $T_{j,Disk}^i$ denote the execution times of process j in the i th phase on three resources, respectively. The first term on the right-hand side of Equation (2) represents the execution time of the sequential components including synchronization delay and computation in the master process; the second term corresponds to the duration of the parallel components. Let $s_{p,CPU}$, $s_{p,COM}$ and $s_{p,Disk}$ denote the sequential components' execution time on CPU, communication, and disk I/O accesses, we have:

$$s_p = s_{p,CPU} + s_{p,COM} + s_{p,Disk}. \quad (3)$$

The rest of the paper addresses the issue of reducing the communication cost of replicating backup data. As in [10], we ignore the specific issues related to clusters such as how to implement a mechanism to migrate tasks; rather, we concentrate on an abstract model that hides operation details but is applicable to a broad range of real-world cluster environments.

4. A task duplication management system

The approach presented in this section extends our previous work in load balancing strategies [15][16][17] and, thus, it is assumed that a load-balancing strategy is deployed to balance overall load across all the nodes of a cluster by dynamically migrating some load from overloaded nodes to other less- or under-loaded nodes. In what follows, a new network-aware task duplication management system (TUFF for short) for clusters with fault-tolerant file systems is presented.

Rather than considering remote I/O accesses, the focus of this study is on designing an effective way to

replicate data issued by local I/O accesses. The reason is that in practice, many cluster file systems are built on top of a native local disk file system, and each node has a local file system that is accessible only from that node. Importantly, the proposed approach can also be applied to clusters with parallel file systems since parallel I/O accesses can clearly take full advantage of the reduced communication overhead obtained by the proposed approach.

Since fault-tolerance is one of the most important issues for file systems on clusters, the design of our task duplication approach is premised on the fault-tolerance of the file system in a cluster. The file system fault-tolerance is assumed to be in the form of disk mirroring by utilizing existing commodity disks on cluster nodes. Direct copying is widely used in many fault tolerant file systems to maintain backup data, and this approach has been proven to be appropriate for CPU- or memory-intensive applications. In direct copying, newly modified or created data is required to be copied from its primary node to the corresponding backup node via network transfers.

For I/O intensive applications that need to intensively write large amount of data, replicating data by the direct copying approach will inevitably induce high communication overheads, and this disadvantage becomes more pronounced when workload conditions are communication intensive. In other words, the high communication overheads imposed by directly copying data from primary nodes to backup ones can potentially result in significant performance drop for parallel applications. To remedy this problem, we propose a network-aware task duplication scheme for fault-tolerant file systems, or TUFF, as an alternative to the direct copying approach. This approach obtains data replicas by regenerating such data locally in backup nodes, as opposed to direct copying. To do so, the TUFF management system concurrently executes the corresponding duplicate tasks of I/O-intensive applications on the backup nodes. Compared with the direct copying approach, the TUFF system is more beneficial for I/O- and communication-intensive applications.

The advantage of the TUFF approach is that in case where workload is communication- and I/O-intensive, communication cost imposed by I/O-intensive applications that issue a large number of write operations can be significantly reduced. However, not all applications are able to enjoy the benefits of TUFF. In particular, most CPU-intensive applications are not eligible to make use of the TUFF management system, because it is unlikely for communication overhead reduced by TUFF to offset CPU cost spent running duplicate tasks.

A detailed pseudo code of the TUFF approach is outlined in Figure 4. There are three input parameters in

TUFF: task t , node i to which t is assigned, and node j in which t 's backup data resides. The objective of TUFF is to choose a way of updating backup data such that (1) the communication load is evenly distributed across network links of a cluster, thereby improving the cluster's network utilization, and (2) communication overhead of replicating backup data is minimized, relieving network bottlenecks.

TUFF: (task t , node i , node j)

1. Obtain $load_{COM}(i)$ and $load_{COM}(j)$;
2. **if** $load_{COM}(i)$ **or** $load_{COM}(j)$ is overloaded **then**
3. Calculate $load_{CPU}(j)$, the CPU load in load j ;
4. Calculate $load_{CPU}(t)$ and $load_{COM}(t)$, the CPU and communication overheads of the task duplication;
5. Compute $load_{COM}(t, i, j)$, the communication cost of replicating the backup data of t on node j ;
6. **if** ($load_{CPU}(j)$ is under loaded) **and**
 ($load_{CPU}(t) + load_{COM}(t) < load_{COM}(t, i, j)$) **then**
7. Concurrently execute a duplicate task of t
 on node j , generating the backup data locally;
8. **else** Execute task t without duplicating it on node j .
9. **else** Execute task t without duplicating it on node j .

Figure 4. Pseudo code of the Task Duplication scheme for Fault-tolerant File systems (TUFF)

The proposed TUFF consists of two main steps. The first step is to decide the necessity of applying the task duplication technique to reduce communication cost. The second step determines whether task duplication overhead can be offset by the benefits of minimizing the communication cost, thereby guaranteeing that task duplication can improve overall performance.

We consider a task t assigned to node i where t 's primary data is stored; and let node j be the node in which t 's backup data resides. To determine the necessity of running a duplicate of t on node j , TUFF detects whether node i or node j is overloaded with respect to network resources by obtaining the load information of each node in the system. Specifically, executing the duplicate task of t would be considered only if either node i or j is overloaded in terms of communication load.

Once the first step finalizes that it is desirable to reduce the communication load of node i and j , the second step in the TUFF system guarantees that every task duplication is capable of yielding performance gain. In practice, such a guarantee can be secured by using two criterions. The first criterion is that the processor in node j is under loaded. Thus, in case where the CPU of node j is not overloaded, TUFF might be able to use the CPU resource to its full potential by concurrently executing t 's

task duplication. If the CPU resource of node j is overloaded, there is no reason for TUFF to enforce the task duplication. The second criterion is that communication cost reduced by the task duplication is greater than the overhead posed by the duplicate running on node j . We can express this criterion as:

$$load_{CPU}(t) + load_{COM}(t) < load_{COM}(t, i, j) \quad (4)$$

where $load_{CPU}(t)$ and $load_{COM}(t)$ are the CPU and communication overheads induced by task t , and $load_{COM}(t, i, j)$ is the communication overhead imposed by the fault-tolerant file system to update the redundant data for task t (See Step 6 in Figure 4).

5. Performance evaluation

5.1. Simulation setup

To evaluate the completion time performance of the proposed TUFF management system, we conducted extensive trace-driven simulations based on a cluster with eight nodes. The trace files used in this study are extrapolated from those reported in [10][22]. Specifically, the traces in our experiments consist of the arrival time, arrival node, actual running time, I/O access rate, message arrival rate, and average message size. To simulate a multi-user time-sharing environment, the traces contain a collection of parallel jobs. The number of tasks in each parallel job is randomly generated according to a uniform distribution between 2 and N , where N is the number of nodes in the cluster. This configuration implies that the communication load implicitly increases as the cluster scales up. We have simulated a bulk-synchronous style of communication, where the time interval between two consecutive synchronization phases is determined by the message arrival rate.

To simulate parallel jobs with communication requirements, we assume that messages issued by each task are modeled as a Poisson Process with a mean message arrival rate. The size of each message is randomly generated according to a Gamma distribution with a mean size of 512Kbyte, which reflects typical data characteristics for many applications.

The arrival time and arrival node of parallel jobs are defined in the trace files, and we randomly generated the requirements for CPU time, the communication, and disk I/O of each job. Although this simplification deflates any correlations between communication and disk I/O requirements, we still are able to manipulate these two types of requirements as input parameters and examine the impact of communicational and disk I/O intensity on the system performance.

The amount of computation required in a task is

called the computation cost, and the amount of data transferred from a task to another is called the communication cost. The communication-to-computation-ratio, or CCR, of a parallel application is defined as its average communication cost divided by its average computation cost. In practice, the CCR of a parallel application ranges from 0.1 to 10. By default, the CCR in our experiments is chosen to be 3.

5.2. Overall completion time performance

In this section, we compare our TUFF with a conventional direct-copying approach. Since many fault tolerant file systems employ the direct-copying approach to generate backup data, we referred to this scheme as FF (an approach of Fault-tolerant File systems).

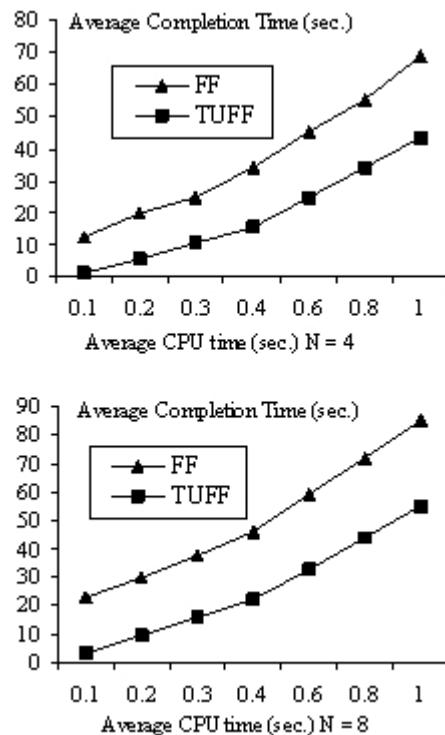


Figure 5. Average completion time under communication- and I/O-intensive workloads. N represents the number of nodes in the cluster. Communication-to-Computation Ratio is 3.

In this experiment, we intend to stress the communication- and I/O-intensive workload by setting the communication-to-computation ratio (CCR) to be 3. Since a realistic system is likely to have a mixed workload with I/O-, CPU-, and communication-intensive applications, we randomly choose 5% of jobs from the traces to be CPU-intensive by setting their message arrival rate to 0.

Figure 5 plots the average completion time of parallel applications as a function of the average CPU time on a cluster with 4 and 8 nodes, respectively. The first observation from Figure 5 is that when the average CPU time increases, the average completion time also increases. This is because Figure 5 reports the results for seven different traces (each with a different workload) with the same value of CCR. Thus, increasing the CPU time intensifies communication demands of parallel applications, which in turn leads to higher network utilizations. Therefore, all the parallel applications experience longer waiting time on sending and receiving data through the network.

Importantly, we observe from Figure 5 that the TUFF management system is significantly better than FF under the communication- and I/O-intensive workload conditions. For example, the improvement gained by TUFF over FF can be up to a factor of 14. We attribute this improvement to the fact that FF does not consider the utilization of network resources even when the network becomes a bottleneck. On the other hand, if the network is a critical resource in the cluster, TUFF attempts to reduce the communication cost by assigning task duplications to be executed on backup nodes.

A third observation drawn from Figure 5 is that adding more nodes into the cluster leads to a longer average completion time for applications with relatively high CCR values. This is because, as mentioned earlier in Section 5.1, the number of tasks in each parallel application increases as the cluster scales up. Hence, the communication demands of parallel applications are rapidly increasing. The implication of this result is that large-scale parallel applications with high communication demands can greatly enjoy the benefits of our new TUFF technique.

5.3. I/O-Intensive Workload Conditions

In this experiment, we further evaluate the performance of TUFF under I/O-intensive workload conditions. To stress the I/O workload, the average CPU time of I/O-intensive jobs is chosen in the range between 1.0 and 4.0 seconds with increments of 0.5 seconds, whereas CPU time of communication-intensive jobs is fixed at a value of 0.2 seconds. This workload reflects a scenario where disk I/O demands from I/O-intensive applications are higher than communication demands in the cluster. Figure 6 presents the average completion time of communication-intensive jobs as a function of the average CPU time of I/O-intensive jobs.

Figure 6 shows that compared with FF, TUFF consistently delivers better performance in terms of the average completion time for communication-intensive

applications under I/O-intensive workload conditions. For example, if the average CPU time of I/O-intensive jobs is 4.0 seconds, the average completion time is 285% faster with TUFF than FF. More interestingly, Figure 6 indicates that the average completion time for the FF approach increases with the increase in I/O intensity. This is mainly because the increasing amount of data tends to be written into primary disks as the I/O intensity increases; hence, the communication overhead imposed by replicating data for backup disks becomes more significant. Consequently, communication-intensive applications inevitably experience longer waiting time on network processing.

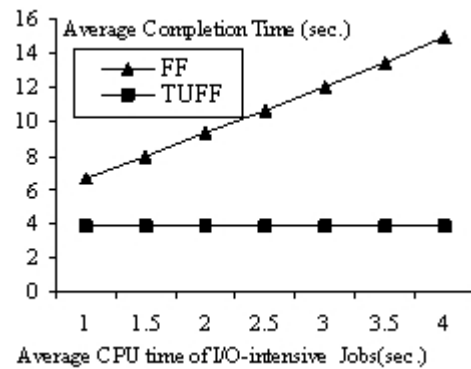


Figure 6. Average completion time under I/O-intensive workloads. The number of nodes is 8, CCR is 3, and average CPU time of communication-intensive jobs is 0.2 sec.

In contrast, the average completion time for TUFF is not sensitive to I/O intensity. The explanation is that TUFF greatly improves the utilization of network resources by reducing the communication overhead of replicating backup data, which dominates the performance of parallel applications when the I/O intensity is high. The conclusion of this analysis is that TUFF achieves significant reduction of communication cost, and the performance improvement of TUFF over FF largely depends on the amount of data created or updated in backup nodes.

6. Conclusion

For a cluster system, if its file system incorporates fault tolerance in the form of disk mirroring, copying data of newly created or modified files from primary nodes to their corresponding backup nodes will inevitably result in a high communication load. To solve this problem, we proposed a network-aware task duplication management system, or TUFF, which enhances utilization of network resources by effectively reducing network resource

consumption imposed by fault-tolerant file systems maintaining redundant data. The TUFF management system builds on the observation that there are many cases where network resources are overloaded while CPU resources are under utilized. Under such circumstances, it is appealing to trade CPU for network resources. In particular, rather than directly copying redundant data from primary nodes through network resources, the TUFF management system regenerates replicated data by concurrently executing the corresponding duplicate tasks of parallel applications that intensively write data to disks. The experimental results from simulations show that TUFF can significantly improve system performance with respect to completion time by up to a factor of 14. Furthermore, TUFF is capable of improving availability performance of parallel applications running on clusters under the condition that TUFF concurrently executes two replicas of each task on two different nodes.

Our future work will focus on a rigorous performance evaluation based on real communication- and I/O-intensive applications running on a cluster with TUFF. To do so, we are currently implementing the prototype of TUFF on a real-world cluster. Since we have implemented and installed a fault-tolerant file system (CEFT-PVFS [23]) on a cluster with 128 nodes, we will seamlessly integrate TUFF into CEFT-PVFS to make the cluster demonstratively more powerful and dependable for parallel applications.

References

- [1] A. Acharya and S. Setia, "Availability and Utility of Idle Memory in Workstation Clusters," *Proc. ACM SIGMETRICS Conf. on Measuring and Modeling of Computer Systems*, May 1999.
- [2] I. Ahmed and Y. K. Kwok, "A New Approach to Scheduling Parallel Programs Using Task Duplication," *Proc. Int'l Conf. Parallel Processing*, Vol.2, pp 47-51, Aug. 1994.
- [4] R. Billinton and R. N. Allan, *Reliability Evaluation of Engineering System: Concepts and Techniques*, Perseus Publishing, 1992.
- [3] P.H. Carns, W.B. Ligon, R.B. Ross, and R. Thakur, "PVFS: A Parallel File System for Linux Clusters," *Proc. the 4th Annual Linux Showcase and Conf.*, pp.317-327, 2002.
- [5] C. Chang, B. Moon, A. Acharya, C. Shock, A. Sussman, and J. Saltz. "Titan: a High-Performance Remote-Sensing Database," *Proc. the 13th Int'l Conf. Data Engineering*, Apr. 1997.
- [6] S. Darbha and D. P. Agrawal, "Optimal Scheduling Algorithm for Distributed Memory Machines," *IEEE Trans. Parallel and Distributed Systems*, Vol.9, No. 1, pp.87-95, January 1998.
- [7] C. Dubnicki, A. Bilas, K. Li, and J. Philbin, "Design and Implementation of Virtual Memory-Mapped Communication on Myrinet," *Proc. Int'l Parallel and Processing Symp.*, April 1997.
- [8] A.C. Dusseau, R.H.Arpaci, and D.E. Culler, "Effective Distributed Scheduling of Parallel Workload," *Proc. ACM SIGMETRICS Conf. on Measuring and Modeling of Computer Systems*, pp.25-36, May 1996.
- [9] P. Geoffray, "OPIOM: Off-Processor I/O with Myrinet", *Future Generation Computing System*, 2002(19).
- [10] M. Harchol-Balter and A. Downey, "Exploiting Process Lifetime Distributions for Load Balancing," *ACM Trans. Computer Systems*, vol. 3, no. 31, 1997.
- [11] C.-Y.H. Hsu and J.W.-S. Liu, "Dynamic Load Balancing Algorithms in Homogeneous Distributed Systems," *IEEE Proc. 6th Int'l Conf. Distributed Computing Systems*, pp.216-223, 1986.
- [12] R. P. Martin, A. M. Vahdat, D. E. Culler, and T. E. Anderson, "Effects of communication latency, overhead and bandwidth in a cluster architecture," *Proc. the 24th Annual Int'l Symp. on Computer Architecture*, Jun. 1997.
- [13] J.M. Orduña, V. Arnau, A. Ruiz, R. Valero, "On the Design of Communication-Aware Task Scheduling Strategies for Heterogeneous Systems," *Proc. Int'l Conf. on Parallel Processing (ICPP)*, pp.391-398, 2000.
- [14] X. Qin and H. Jiang, "Improving Effective Bandwidth of Networks on Clusters using Load Balancing for Communication-Intensive Applications," *Proc. the 24th IEEE Int'l Performance, Computing, and Communications Conf. (IPCCC 2005)*, Phoenix, Arizona, April 2005.
- [15] X. Qin, H. Jiang, Y. Zhu, and D. Swanson, "Dynamic Load Balancing for I/O-Intensive Tasks on Heterogeneous Clusters," *Proc. the 10th Int'l Conf. on High Performance Computing (HiPC 2003)*, pp. 300-309, Dec., 2003.
- [16] X. Qin, H. Jiang, Y. Zhu, and D. Swanson, "Towards Load Balancing Support for I/O-Intensive Parallel Jobs in a Cluster of Workstations," *Proc. the 5th IEEE Int'l Conf. Cluster Computing (Cluster 2003)*, pp.100-107, Dec. 2003.
- [17] X. Qin, H. Jiang, Y. Zhu, and D. R. Swanson, "Dynamic Load balancing for I/O- and Memory-Intensive workload in Clusters using a Feedback Control Mechanism," *Proc. the 9th Int'l Euro-Par Conf. Parallel Processing (Euro-Par 2003)*, pp. 224-229, Klagenfurt, Austria, Aug. 2003.
- [18] S. Ranaweera, and D. P. Agrawal, "A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems," *Proc. Int'l Symp. Parallel and Distributed Processing*, 2000.
- [19] K.G. Shin and Y.-C. Chang, "Load Sharing in Distributed Real-time Systems with State Change Broadcasts," *IEEE Trans. Computers*, Vol. 38, no.8, pp.1124-1142, Aug. 1989.
- [20] J. A. Stankovic, "Simulation of Three Adaptive, Decentralized Controlled, Job Scheduling Algorithms," *Computer Networks*, Vol.8, pp.199-217, 1984.
- [21] L. Valiant, "Bridging model for parallel computation," *Comm. of ACM*, 33(8):103-111, 1990.
- [22] X. Zhang, Y. Qu, and L. Xiao, "Improving Distributed Workload Performance by Sharing both CPU and Memory Resources," *Proc. the 20th Int'l Conf. Distributed Computing Systems*, Apr. 2000.
- [23] Y. Zhu, H. Jiang, X. Qin, and D. Swanson, "A Case Study of Parallel I/O for Biological Sequence Analysis on Linux Clusters", *Proc. the 5th IEEE Int'l Conf. Cluster Computing*, pp. 308-315, Dec. 2003.