Fault Tolerant Wide-Area Parallel Computing

Jon B. Weissman
Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455
(jon@cs.umn.edu)

Abstract. Executing parallel applications across distributed networks introduces the problem of fault tolerance. A viable solution for fault tolerance must keep overhead manageable and not compromise the high performance objective of parallel processing. In this paper, we explore two options for achieving fault tolerance for a common class of parallel applications, single-program-multiple-data (SPMD). We quantitatively compare checkpoint-recovery and wide-area replication as a means of achieving fault tolerance. The experimental results obtained for a canonical SPMD application suggest that checkpoint-recovery may be preferable for small problems if local parallel disks are available, but wide-area replication outperforms checkpoint-recovery for larger-grain problems, precisely the problems most suited for the wide-area network environment. The results also show that it possible to accurately model and predict the overheads of the two methods[1]

1.0 Introduction

High-performance distributed computing across wide-area networks has become an active topic of research [1][3][4][11]. Metasystem and grid software infrastructure projects, most notably, Legion [4] and Globus [3], have emerged to support this new computational paradigm. Achieving large-scale distributed computing in a seamless manner introduces a number of difficult problems. This paper examines one of the most critical problems, fault tolerance. A large wide-area system that contains hundreds to thousands of machines and multiple networks has a small mean time to failure. The most common failure modes include machine faults in which hosts go down and get rebooted, and network faults where links go down. A single monolithic solution for fault tolerance that is acceptable to all user applications is unlikely. For example, some applications may require continuous availability, or may require protection from byzantine failures, or require light-weight, low overhead fault tolerance. The most appropriate method for fault tolerance clearly may be application-specific. This follows the current trend in distributed systems and operating systems in which generic functions once performed within the "system" are now being moved to user-space for increased flexibility and performance. Because general purpose systems often impose a high cost on applications that do not fit their assumptions, the maxim "pay for what you need" has been proposed as a guiding principle for application-centric policy decisions in metacomputing systems such as Legion.

---

1. This work was partially funded by grants NSF ACIR-9996418 and CDA-9633299, AFOSR-F49620-96-1-0472.

We believe that the relative performance of fault tolerance methods is a key piece of information needed to enable users to select the most appropriate method for their application. This is particularly true for high-performance applications. To this end, we have examined two fault tolerance two options, *checkpoint-recovery* and *wide-area replication*, for a common class of high-performance parallel applications, single-program-multiple-data (SPMD). To compare these approaches, performance models that characterize the overheads have been developed. These models enable quantitative comparisons of the two methods as applied to SPMD applications. We have not implemented complete fault tolerance solutions, but a sufficient subset to capture the associated overheads. In particular, we do not consider fault detection or fault recovery.

We compare wide-area replication across the Internet to application-level checkpointing for a canonical SPMD application. The wide-area replication is provided by the Gallop system, a wide-area scheduler, and has the advantage that no source code changes are required. The checkpointing version of the application required the insertion of checkpointing code. It is possible to perform application checkpoints with operating system support or binary code rewriting avoiding source changes, but we speculate that the checkpoint overheads would not differ dramatically. Two modes of checkpointing were compared with wide-area replication: a single NFS-mounted checkpoint disk and a parallel array of locally attached disks. The results indicate that the performance models are accurate and predictive, and that both methods appear to be appropriate under different conditions: checkpoint-recovery performs best for small problems providing fast local disks are available, while wide-area replication is generally more efficient for bigger problems.

2.0 Related Work

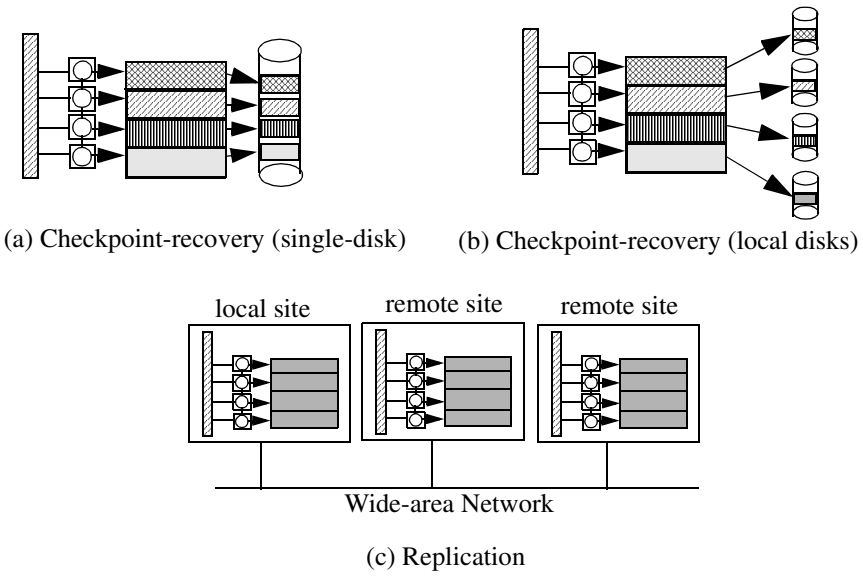
Numerous research groups are examining the problem of fault tolerance for parallel applications in distributed networks. Globus provides a heartbeat service to monitor running processes to detect faults [4]. The application is notified of the failure and expected to take appropriate recovery action. Legion provides mechanisms to support fault tolerance such as checkpointing, but the policy must be provided by the application. A component-based reflexive architecture allows fault tolerance methods to be encapsulated in reusable components that user applications may choose from [7]. Condor [6][13] provides system-level checkpoints of distributed applications but it not geared to high-performance parallel programs. A number of PVM-based checkpointing systems have also appeared over the past few years [2]. Our work is different from these and related projects as we are focused on the problem of deciding which fault tolerance method can be expected to perform best, not in a specific implementation of fault tolerance methods.

3.0 Fault Tolerance Options for SPMD Applications

In a canonical data parallel SPMD application, a set of identical tasks are created one per processor with each assigned a portion of a data domain such as a grid or a matrix. The tasks alternate between computing on their portion of the data domain and communications, typically in an iterative style. The state of each task is the updated

data domain that changes with each iteration. A transient network or single processor fault are the most common events that will cause such an application to fail. Executing SPMD applications in a fault tolerant manner can be achieved by checkpointing or replication (Figure 1). For the purposes of a direct quantitative comparison, a simple checkpoint model is adopted in which each SPMD task saves its portion of the data domain on disk at a set of pre-determined iterations. Recomputing lost iterations starting from the last checkpoint would be straightforward given the last iteration index. We studied two configurations for checkpointing: a single NFS-mounted disk from a file server (Figure 1a) and a parallel array of locally attached disks (Figure 1b). The dominant overhead for checkpoint-recovery is the cost of stopping the application and writing the checkpoints to disk. The local checkpoint model is useful only if the failed processor is expected to recover and pick up where it left off since it is the only processor that can access the checkpoint. The network checkpoint model would allow another processor to pick up the checkpoint since it is stored on a common server.

In option (c), each application replica runs in a different site to avoid overloading the available computation and communication resources of a particular site. The use of wide-area computing offers a solution to the resource demand of replication because some sites in a wide-area system will likely be underutilized. We have developed a wide-area scheduling system called Gallop [11] that remotely executes SPMD applications for improved performance. Gallop picks the best site to run an application based



(a) Checkpoint-recovery (single-disk)

(b) Checkpoint-recovery (local disks)

(c) Replication

Fig. 1: Fault tolerance options for SPMD applications. Four tasks (circles) on four processors (squares) operate on rectangular piece of data. In (a-b), the pieces are written to disk periodically. In (c), the local site is where the application request originates and is responsible for coordinating the remote replicas**.**

on an estimate of how well the site can run the job. Gallop performs this task by utilizing a local data parallel scheduling system called Prophet [12]. Prophet generates a performance prediction based on application and site resource information for each site in which it is running (refer to [12] for additional details).

We have modified Gallop to support the scheduling of application replicas in multiple sites to compare with checkpointing. Gallop will create a fixed number of application replicas determined by a user-specified runtime parameter. A complete independent copy of each application will run in the chosen number of sites. Important issues relating to replica consistency, dynamic replica creation, etc., are not germane to the performance study, but are discussed elsewhere [10]. The dominant overhead for wide-area replication is the scheduling protocol used by Gallop which includes Prophet overhead, multiple messages exchanged between the local and remote sites, and (potentially) application file transmission to enable execution in remote sites.

4.0 Performance Models

We have developed performance models for checkpoint-recovery (CR) and wide-area replication (WR) to enable quantitative comparisons. We consider only checkpoint costs for CR as recovery costs are more difficult to model accurately due to the large variety of potential failure modes each with different characteristics. For example, the recovery time for a network failure vs. a machine failure may in fact be very different. However, some preliminary work with single machine failures and immediate restart indicated that the inclusion of recovery overhead did not change the relative performance of the two methods for the vast majority of problem instances.

In our CR model, checkpoints occur at a single place in the applications execution and checkpoints from all SPMD tasks are written atomically to disk. For SPMD applications each task performs a checkpoint of its data domain and the current iteration after it has computed on its data, and before any messages are sent during that iteration. The checkpoints occur every *k* iterations. The WR model assumes that the set of replicas are *static* and unchanging. When a replica fails, it is not restarted nor is a new replica scheduled elsewhere. A model that includes *dynamic* replica creation is the subject of future work. The performance of CR and WR may be defined in terms of the following parameters:

N: problem size

P: number of processors

I: number of iterations

k: checkpoint frequency

m: number of wide-area replicas (or sites)

λ: average rate of site failure (failures/minute)

β: desired reliability (desired probability of application success)

We make the assumption that the failure rate is exponentially distributed as is commonly done and that it is identical for all sites. Different failure rates for different sites could easily be accommodated, but are omitted for simplicity (disk-config is NFS or local).

The following cost functions can be defined (disk-config is NFS or local):

T_{CT} (N, P): completion time for NxN problem using P processors

T_{CP} (N, P, disk-config): checkpoint time for single checkpoint on disk-config

T_{CR} (N, P, disk-config): total time to perform the optimal number of checkpoints

T_{WR}(N, m): time to schedule m replicas remotely

P_f (N, P, λ): probability of application failure (single site failure)

The cost functions can be expressed in terms of these parameters and several system-dependent constants:

T_{CT} (N, P) = I * T_c(N, P)

- T_c is the average time per iteration and includes computation and communication

T_{CP} (N, P, NFS) = $T_{NFS_latency} + \frac{N^2}{P} T_{NFS_bw}$

- The latency and bandwidth terms include disk and network overhead; $\frac{N^2}{P}$ is the per processor data size

T_{CP} (N, P, local) = $T_{disk_latency} + \frac{N^2}{P} T_{disk_bw}$

- Local checkpointing only involves disk overhead; $\frac{N^2}{P}$ is the per processor data size

T_{CR} (N, P, disk-config) = k_{opt} * T_{CP} (N, P, disk-config)

- k_{opt} is the optimal number of checkpoints

T_{WR}(N, m) = $T_{WR_scheduling}$ (N, m) + T_{WR_upload} (m, bsize)

- WR consists of scheduling costs and file upload costs (which depends on binary and input file sizes)

$T_{WR_scheduling}$ (N, m) = $T_{WR_sched_latency}$ + m ($T_{WR_sched_overhead}$)

- Scheduling overhead consists of a base cost and a per site cost

T_{WR_upload} (m, bsize) = $T_{WR_upload_latency}$ + m ($T_{WR_upload_overhead}$)

- Upload consists of a base cost and a per site file transfer cost

P_f (N, P, λ) = $1 - e^{-\lambda T_{CT}(N, P)}$

- Exponential failure distribution - this gives the probability the application will NOT fail (in a site)

We have experimentally derived the constants for these cost functions in a local- and wide-area network testbed environment. In the next section, we show that these functions accurately predict the real overhead costs. We model two modes each for CR and WR. For CR, we experimented with local and NFS-mounted file systems. For WR, we experimented with remote sites in which application files (binaries and input files) were either pre-staged or required uploading at runtime[2]. For pre-staged files,

---

2. Another option is to transfer the much smaller source files and compile them on the remote site. This option may be examined in the future.

T_{WR_upload} is 0. WR overhead consists of scheduling overhead and file transmission overhead. Scheduling overhead includes Prophet and protocol overhead, the latter is dominated by wide-area message passing cost. Both components of WR, $T_{WR_scheduling}$ and T_{WR_upload} have a constant latency term and a per site term. Although the scheduling protocol and file transmission are largely parallel activities, we have empirically observed a dependence on the number of sites that is accurately modelled by a small linear constant. For $T_{WR_scheduling}$: as the number of sites increases, the number of protocol messages handled by the local site daemon increases in a linear fashion, and the probability that a message is delayed from a remote site back to the local site, also increases. For T_{WR_upload}: as the number of sites increases, the local ftp server will have to serve a proportionally larger number of sites. Note that WR depends only on the number of sites and not directly on the problem size N because the application creates the initial data domain internally, while CR depends strongly on N.

The function P_f gives the probability that a site (or application running in the site) will not remain "up" through a time t. A site is up if all constituent machines and connecting networks applied to the application are up. This function has the property that a longer running application (hence a larger T_{CT}) incurs a larger probability of failure. This probability failure model is needed to enable a meaningful and fair performance comparison between the two methods. Without a model for failure, it is unclear how many checkpoints or replicas are needed to obtain a desired level of reliability. We use the parameter P_f to construct the cost functions for both CR and WR.

First, we derive the cost equation for CR. For a given P_f, we determine the optimal number of checkpoints to perform. The optimal number of checkpoints balances the cost of checkpointing with the cost of re-executing old iterations in the event that the application fails and needs to be restarted from the last stored checkpoint. Finding this value requires minimization of the following expression for total overhead experienced using CR (we omit some of the function parameters for brevity):

$$(1 - P_f)\left[\frac{kT_{CT}}{2I}\right] + T_{CP} \cdot \frac{I}{k} \tag{1}$$

The first term in brackets is the average cost to re-execute iterations from the past checkpoint and the second term is the checkpoint cost. The factor of $\frac{k}{2}$ reflects a failure which occurs midway between checkpoints on average. Differentiating with respect to *k*, and solving for the minimum *k* yields:

$$k_{opt} = \left\lceil \sqrt{\frac{2I^2 \cdot T_{CP}}{T_{CT} \cdot (1 - P_f)}} \right\rceil \tag{2}$$

This agrees with similar results in the literature [5][9] and provides a mechanism to determine the minimum overhead, T_{CR}, for a given P_f.

For WR, P_f plays a different role. Given P_f, we can determine the number of replicas *m* that achieve a desired level of reliability β, where β is the probability that at least one replica finishes. The probability that at least one replica finishes is $1\text{-}P_f^m$

assuming independent site failures each with probability P_f. If sites have different failure probabilities then the equation becomes only slightly more complex. Solving for m yields the following:

$$m = \left\lceil \frac{\log(1-\beta)}{\log P_f} \right\rceil \quad (3)$$

Given a desired value of β, we can easily compute *m* and T_{WR}, and compare it with T_{CR} for a given problem instance. The dependent parameter β can be used to select between different fault tolerance methods (CR or WR) and provides a way to adjust the level of fault tolerance for WR. For example, if the number of available sites is less than the number of sites required to achieve the desired β, then CR becomes a more desirable option. Similarly, if β = 1, then CR is the logical choice since it is not possible to guarantee any replicas will finish if $P_f > 0$ (for static WR). However, if the user is willing to accept β < 1, then WR may offer some performance advantages as we will show. These models give us a way to predict the performance of CR and WR, and ultimately to enable the user to select the most suitable method given their preferences and constraints in the network environment.

5.0 Results

We performed an evaluation of CR and WR by simulating a wide range of failure rates (λ) to answer the following questions. Given a problem instance (N, P, I), network characteristics (local/NFS, m, P_f), and a desired degree of reliability (β) will CR or WR perform best? Which method might be expected to perform best as the problem size grows? What is the impact of uploading on the suitability of WR? We answer these questions for two specific experimental environments using a canonical SPMD application (STEN) that solves Poisson's equation on a NxN grid using an iterative method (Figure 1). STEN creates the initial data domain internally and does not use any input files, hence only application binary files need to be transferred under WR. We have two versions of STEN: (1) a CR version with user-level checkpoint code inserted and (2) a WR version that uses the Gallop wide-area scheduler as described earlier. Gallop was run using an experimental testbed containing Internet sites at the University of Texas at San Antonio (UTSA), Southwest Research Institute in San Antonio, University of Virginia, University of Kentucky, Argonne National Laboratories, and the University of California, San Diego. The local site is UTSA and contains 15 ethernet-connected Sparc 5s both configured with local disks (local), and with a NFS-mounted file system (NFS). The other sites contain Sparcs of similar capability. The local site is where the CR data was gathered. The experimental testbeds were used to gather data to construct the cost functions of Section 4.0. We first show that the experimentally derived cost functions are accurate. We then compare the performance of CR and WR using these cost functions for a wide range of parameter values.

5.1 Validating the Models

We performed a set of experiments with STEN using our local- and wide-area testbed to determine the cost functions for CR and WR. We ran STEN using a large

number of values for N and P and used linear regression to derive the constants from this experimental data:

T_{CP} (N, P, NFS) = 30 + (N^2/P)0.005 msec

T_{CP} (N, P, local) = 0.2 + (N^2/P)0.0005 msec

T_{WR_upload} (m, bsize) = 4821 + m (0.023) msec

$T_{WR_scheduling}$ (N, m) = 135 + m (28.4) msec

We then performed another separate set of runs for CR and WR using specific values of N (128, 256, 512, 1024) and P (1, 2, 4), and plotted these observed values (ten scatter points are shown for each x value plotted) against the derived cost functions (straight line). The results for WR (Figure 2) and for CR (Figure 3) are shown. For CR we show results for NFS (top row) and local file systems (bottom row), for P=1, 2, and 4 respectively.

In the majority of cases, the error between the overhead predicted by the cost function and the experimental overhead falls within 10% for CR with the exception of N=1024, P=1,2, local file system configuration. We speculate that disk buffer cache effects were exposed by the large write requests. The CR data indicates that the checkpoint cost functions for both local and NFS file systems provide a good fit as N and P vary. We observed that the cost of performing a single checkpoint depends on the total checkpoint data size and appears to be invariant to the number of processors. This was surprising for NFS as we expected more processors would create additional server load. It is more difficult to accurately model WR costs due to the high variance in wide-area communications. However, the cost functions give a reasonably good fit (within 10-30%) that is sufficient for quantitative comparisons. For a different network environment, the cost equations will have different constants which can be easily obtained by our test programs. The results indicate that it is possible to predict the overhead costs for a given network environment with sufficient fidelity to enable quantitative comparison of the two methods.

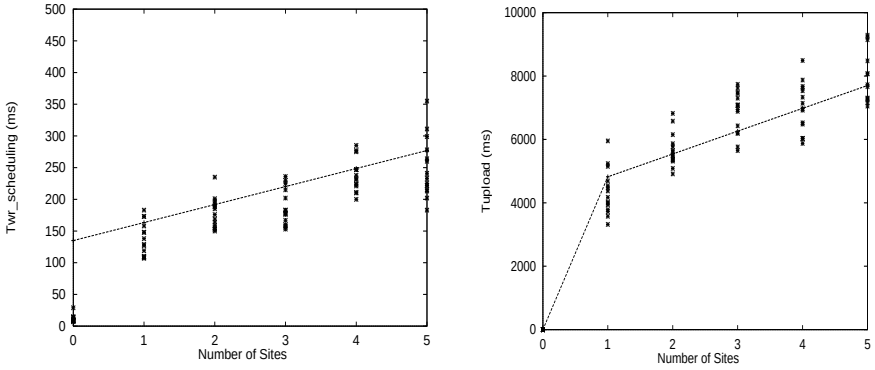


Fig. 2: Results for WR: scheduling and uploading. Upload results correspond to a 300 KB bina for STEN with ftp used to transfer the file. 0 sites corresponds to the use of the local site only.

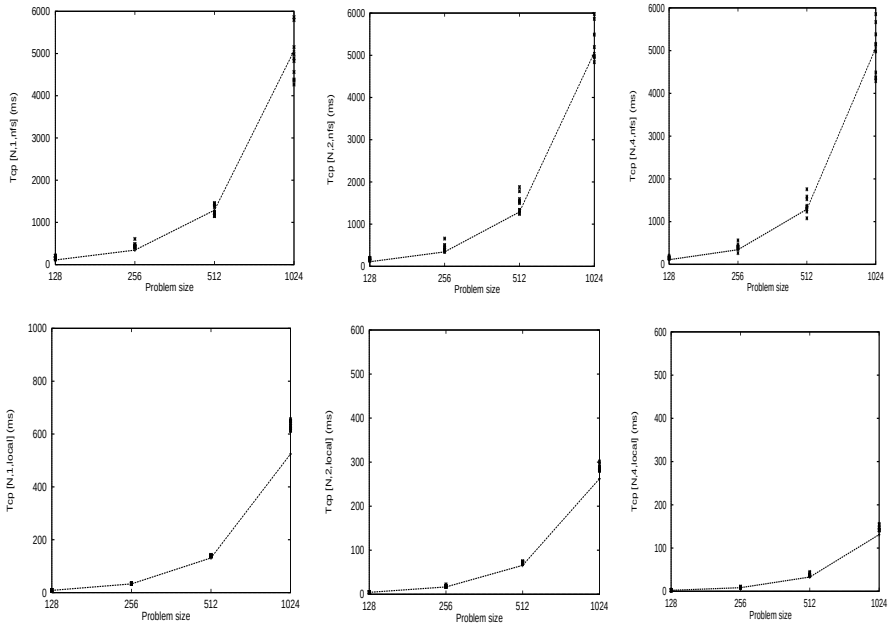


Fig. 3: Results for CR shown for NFS (top row) and local file system (bottom row), as a function of problem size. Each graph corresponds to a different P selected (1, 2, and 4). The scatter plots show 10 points per (N, P) pair: in some cases nearly identical points are plotted on the same space and appear darker.

5.2 Head-to-head Comparison

We now compare the *cost functions* T_{CR} and T_{WR} head-to-head to see how well the methods can be expected to perform under different failure and reliability parameters. For P_f, we vary λ to be 1 site failure per the following time intervals: 6 months, 1 month, 1 week, 1 day, half day, and 1 hour (corresponding to failure rates = l1, l2, ..., l6 respectively on the graphs). We vary β to be .999, .999999, .99999999 (corresponding to B = 1, 2, 3, respectively on the graphs). We also vary the problem size as before and since CR performance is invariant to P (for a fixed failure rate), we pick P=4 for the data plots (the other values of P yield similar graphs). We obtained the T_{CT} values for each (N, P) pair from the Prophet scheduler. Unless otherwise stated, we set I=1000 iterations. We show the total overhead under either method (T_{WR} vs. T_{CR}) on the y-axis as a function of varying failure rate for all graphs. The first set of results compares CR with WR without uploading (Figure 4). CR with a local disk configuration exhibits slightly better performance for small problems (N=128, 256, and 512; 128 is not shown), while WR is a clear winner for large problems (N=1024). WR performance does not vary much as β increases which suggests that a very high rate of reliability appears to be affordable. The results also indicate that for small problems, WR and CR offer similar performance. When a NFS-mounted disk configuration is used, WR is clearly superior for virtually all problem sizes (N=256, 512, and 1024; 1024 is off the

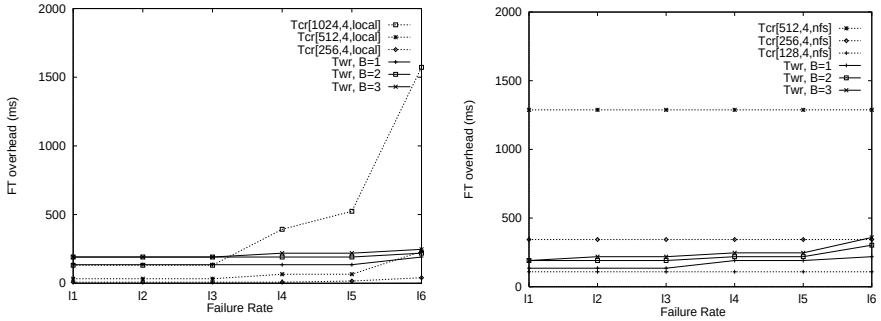


Fig. 4: Comparing WR w/o uploading to CR for local and NFS respectively.

graph). The flat-line indicates that the checkpoints are so expensive, that only a single checkpoint is affordable. When WR incorporates uploading, the results change fairly significantly (Figure 5). For both configurations, CR is clearly superior to WR due to high cost of uploading the STEN binaries. However as compared to the NFS configuration, WR is competitive for the largest problem (N=1024) for $\beta = .999$. Since our largest problem (N=1024 with I=1000) is small by some standards (T_c = 350 ms, which gives a total time of 350 sec), we wanted to know what would happen if the problem was scaled to the sizes one might expect in a metacomputing environment (and uploading was required). In particular, would WR become more attractive for larger problems that ran longer? To model a larger longer-running problem, we fixed T_c at 350 ms and simply increased I, keeping all other values such as T_c and T_{CP} constant. In reality, a larger problem also means that N and P both increase. If we assume that P grows in proportion to the increased computation then T_c could reasonably remain unchanged. Similarly, T_{CP} for the local configuration depends on the amount of data each processor writes to disk. If this is unchanged, then T_{CP} could also reasonably remain unchanged. However for the NFS configuration an increase in N will surely increase T_{CP} independent of P. Consequently, the T_{CR} results for NFS should be viewed as a lower-bound. When I is increased, WR begins to exhibit better behavior

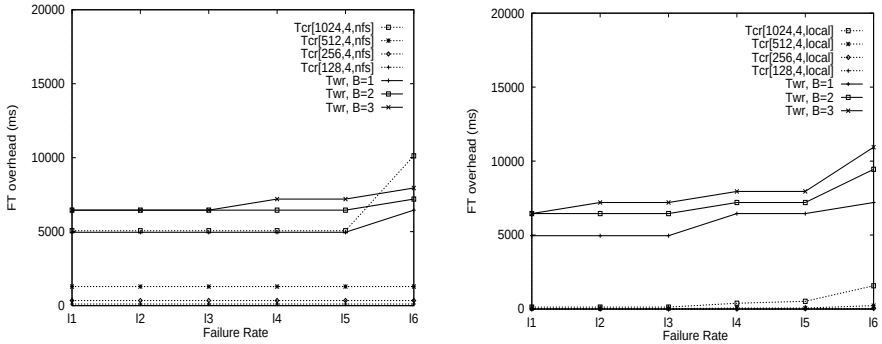


Fig. 5: Comparing WR with uploading to CR for local and NFS respectively.

than CR under the NFS and local configurations (Figure 6). For the local configuration (Figure 6b), the problem must be scaled significantly for WR to outperform CR (I=50000 or ~ 12 hours!).

We have shown that a meaningful quantitative comparison between WR and CR is possible for a typical SPMD application. For this application, our results suggest that CR is generally a less expensive method for small problems provided local disks are available, WR is cheaper for larger problems provided binaries are pre-staged, but when the problems become very long-running, CR may again be better. To apply our approach to a different application and network environment, some benchmarking to determine the cost function constants is a necessary precondition. This approach can be used to give the user some guidance in the selection of fault tolerance methods and determine whether affordable fault tolerance is possible given their application and network environment (provided estimates of P_f are known or can be approximated). But the precise benefit depends on the application and network environment at hand. Another interesting possibility is to provide both fault tolerance implementations for an application and allow the system to pick the best one automatically at run-time. Similarly, the cost models could be used by a metacomputing scheduler in deciding where to run an application. For example, if CR is the desired method for an application, then the scheduler should consider the predicted cost of CR in choosing the application's location.

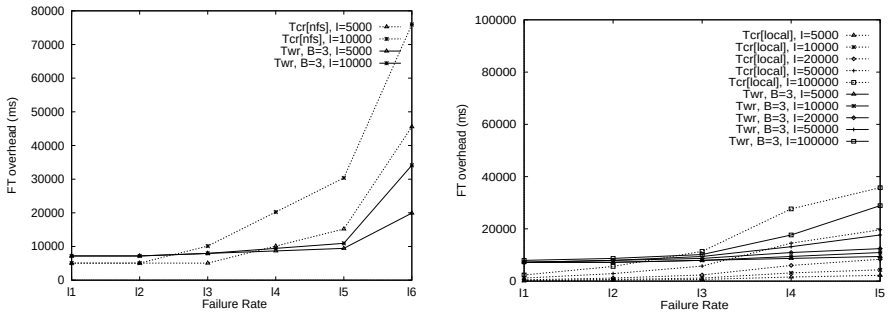


Fig. 6: Comparing WR with uploading to CR for NFS and local respectively as I increases

6.0 Summary

We have presented a technique that enabled quantitative comparisons between two fault tolerance methods: checkpoint-recovery (CR) and wide-area replication (WR) for SPMD applications. For high-performance applications in particular, the expected cost of fault tolerance may be an important factor in the method a user may choose to adopt. The results obtained for the stencil application indicate that both methods appear to be appropriate under different conditions: CR is generally cheaper for small problems providing fast local disks are available, while WR is generally cheaper for bigger problems provided binaries are pre-staged. We believe our technique can be used to support the "pay for what you need" policy currently advocated in

metacomputing systems. However, experimentation with additional SPMD applications is needed to confirm our assertion. Future work also includes an investigation into site reliability (the parameter P_f) and the development of tools to gather this information. Finally, we plan on incorporating the cost of recovery and dynamic replicas into our performance models.

7.0 References

[1] Bal, H. et al, "Optimizing Parallel Applications for Wide-Area Clusters," *Twelfth International Parallel Processing Symposium,*" March 1998.

[2] Casas, J. et al, "Adaptive Load Migration systems for PVM," *Supercomputing* 1994.

[3] Foster, I. and Kesselman, C., "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputing Applications*, 11(2), 1997.

[4] Grimshaw, A.S. and Wulf, W. A., "The Legion Vision of a Worldwide Virtual Computer," Communications of the ACM, Vol. 40(1), 1997.

[5] Jalote., P. , "Fault Tolerance in Distributed Systems," Prentice-Hall Publishers, Englewood Cliffs, New Jersey, 1994.

[6] Litzkow, M.J. et al., "Condor - a hunter of idle workstations," In *Proceedings of the 8th International Conference on Distributed Computing Systems*, June 1988.

[7] Nguyen-Tuong, A. and Grimshaw, A.S., "Using Reflection to Incorporate Fault-Tolerance Techniques in Distributed Applications," Computer Science Technical Report, University of Virginia, CS 98-34, 1998.

[8] Stelling, P. et al., "A Fault Detection Service for Wide Area Distributed Computations," *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, August 1998.

[9] Vaidya, N.H., "Impact of Checkpoint Latency on Overhead Ratio of a Checkpointing Scheme," *IEEE Transactions on Computers*, Vol. 46(8), August 1997.s

[10] Weissman, J.B. and Womack, D. "Fault Tolerant Scheduling in Distributed Networks," UTSA Technical Report, CS-96-10, October 1996.

[11] Weissman, J.B., "Gallop: The Benefits of Wide-Area Computing for Parallel Processing," *Journal of Parallel and Distributed Computing*, Vol. 54(2), November 1998.

[12] Weissman, J.B., "Prophet: Automated Scheduling of SPMD Programs in Workstation Networks," *Concurrency: Practice and Experience,* Vol. 11(6), May 1999.

[13] Zandy, V., Miller, B. and Livny, M., "Process Hijacking," *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing*, August 1999.