Adaptive Checkpointing with Storage Management for Mobile Environments

Kuo-Feng Ssu
University of Illinois, Urbana

Bin Yao
Purdue University, West Lafayette

W. Kent Fuchs, *Fellow IEEE*
Purdue University, West Lafayette

Nuno Ferreira Neves
University of Lisbon, Lisbon

Key Words — Mobile recovery, Storage management, Checkpointing, Recovery.

Summary & Conclusions — The limited stable storage available in mobile-computing environments can make traditional checkpointing and message logging unsuitable. Since storage on a mobile host is not considered stable, most protocols designed for these environments save the checkpoints on base stations. Previous approaches have assumed that the base station always has sufficient disk space for storing checkpoints. If there is not enough storage available, checkpoints might need to be aborted.

This paper describes an adaptive protocol that manages storage for base stations. The protocol integrates leasing storage management with a time-based coordinated checkpointing mechanism. The leasing enables storage managers to control disk-space effectively. Leasing prevents hanged processes from indefinitely retaining storage and, in addition, garbage collection is simple. Time-based checkpointing is integrated with leasing to reduce the number of messages for establishing consistent global states. The checkpointing mechanism uses a 3-level storage hierarchy to improve checkpointing performance.

Performance was evaluated by both implementation experiments and simulations. The results show that:

- the adaptive protocol reduces checkpointing overhead,
- the leasing mechanism maintains the desired storage assignment for base stations.

1. INTRODUCTION

Wireless networking [1] is an enabling technology for mobile computing. Wireless signals are subject to dispersion & interference, and thus wireless communication is inherently susceptible to data loss & disconnection. The challenges to dependable mobile computing include, but are not limited to [2 – 4]:

- varying communication bandwidths,
- high failure rates,
- frequent disconnections,
- heterogeneous networks,
- security risks,
- limited battery power,
- host mobility.

It is not appropriate to apply directly many of the checkpointing and recovery protocols [5 – 11] designed for fixed network distributed systems to mobile environments.

Several checkpointing protocols for wireless mobile environments have been proposed [12 – 18]. These protocols generally require the availability of extensive stable storage. Because storage on the mobile host is typically not considered stable, most of these protocols store checkpoints and message-logs on local base stations. Stable storage on the base station is also used to keep temporary information for better performance (*eg*, caching data), and hence the amount of storage in use changes dynamically. Previous checkpointing protocols assume that base stations have sufficient available storage to save checkpoints at all times. When stable storage on base stations is depleted, these previous protocols can fail.

This paper describes a leasing mechanism to manage storage for checkpoints. Before checkpointing, each process negotiates with a storage manager to determine the size & duration of the lease. Once the lease is agreed upon, a process can use the allocated storage for checkpoints. Storage space is returned to the manager when the lease expires. The process is allowed to request renewal of the lease before expiration. The storage manager can accept or decline the request, based on management protocols. The leasing mechanism not only manages stable storage effectively but also prevents storage retention by failed processes. A coordinated checkpointing protocol integrated with the leasing management is also described in this paper. The checkpointing protocol

- uses time for coordination to reduce communication overhead,

0018-9529/99/$10.00

· dynamically adjusts the locations used to store checkpoints in order to reduce transmission overhead.

Our experiments were implemented & evaluated in a specific wireless mobile network. The experimental results show that the adaptive checkpointing protocol achieved better performance through hierarchical checkpoint arrangements. Four negotiation protocols with the leasing mechanism were also evaluated. The results demonstrate that the adaptive protocol effectively managed the desired storage allocation for base stations using the 4 negotiation protocols.

2. RELATED WORK

2.1 Mobile Checkpointing

Ref [12] proposed a 2-phase (phase SEND and phase RECV) checkpointing protocol to store consistent global states for distributed mobile environments. The protocol creates a checkpoint whenever a mobile host receives a message in the SEND phase. All messages sent & received by the mobile host are logged. The mobile host's message-logs and checkpoints are saved on stable storage of the current base station. As the mobile host moves through cells, the checkpoints are scattered among base stations.

Ref [13] presented 2 independent checkpointing protocols for recoverable mobile environments. Protocol #1 establishes a checkpoint whenever a message is received. Protocol #2 performs checkpointing periodically, and logs all messages received. Both protocols suggest saving checkpoints in the stable storage on the base stations instead of on the mobile hosts. Ref [18] developed an approach to independent checkpointing with receiver-based logging for fast recovery and efficient garbage collection.

Ref [14] developed a non-blocking coordinated checkpointing protocol that requires a minimum number of mobile hosts to participate in checkpointing. Ref [16] showed that the protocol can result in inconsistent global states that cannot be used for recovery. In [17] the authors proposed an alternative non-blocking protocol that saves process state as mutable checkpoints on the local memory or stable storage. The mutable checkpoints are either discarded or transmitted to the base station, based on specific patterns of checkpointing and communication (z-dependencies) [19].

Ref [15] developed a time-based checkpointing & recovery protocol for wireless mobile systems. This protocol uses time to coordinate processes indirectly to establish consistent recovery points [15, 20]. The technique avoids many forced checkpoints and logs only unacknowledged messages. This protocol assumes that base stations are controlled by external organizations and mobile users cannot allocate any space on the base stations; thus all checkpoints are saved in the stable storage of the home network.

2.2 Storage Management & Leasing

IBM developed a data facility storage management subsystem (DFSMS) that used computer technology to reduce the human effort needed to manage storage data [21, 22]. Ref [23] introduced volume leases for providing server-driven cache consistency for large-scale distributed systems. The leasing approach reduced message traffic at servers for a trace-based workload of web accesses.

Our approach of adaptive checkpointing with leasing, manages storage on local mobile hosts, base stations, and home hosts, to reduce checkpointing overhead. This protocol dynamically determines the appropriate location to store checkpoints based on available resources.

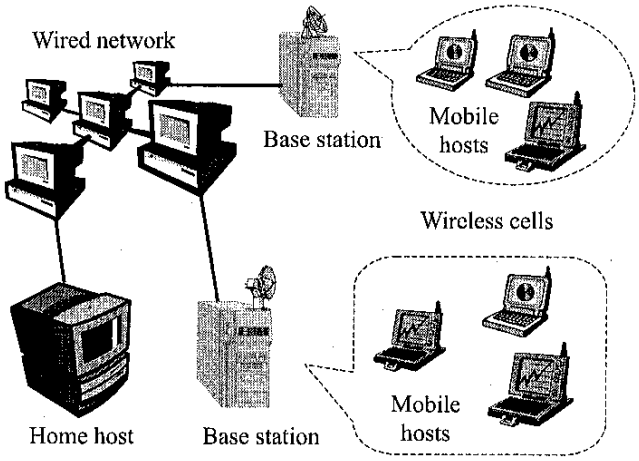

Figure 1: Example of the Wireless Mobile Environment

3. WIRELESS MOBILE ENVIRONMENTS

The system environment for wireless mobile computing in this paper is based on the mobile IP network architecture [24]. Mobile hosts are equipped with wireless interfaces to support mobility and connectivity. Fixed hosts with both wireless interfaces and wired network interfaces are called base stations. A mobile host communicates with base stations through a wireless channel and relies on base stations to maintain its network connection. Messages destined for the mobile host are first sent to its home host. The home host maintains location information of the mobile host, and forwards the messages to the mobile host through the base station. The geographical area covered by a wireless interface is called a cell. The mobile-hosts in the same cell have the same local base station. As the mobile-host moves to another cell, it disconnects the original wireless channel and requests the new base station to establish another communication channel. Figure 1 is an example of the wireless mobile environment.

There are two typical wireless environments for mobile computing —

· Local environment: Base stations belong to the individuals that use the mobile hosts. The users can freely access & store data in the base stations.

· Global environment: The mobile stations typically belong to a telecommunications company and the users can rent the stations for a period of time. Users might not be

able to control the storage in the base stations.

The wireless systems discussed in this paper include both environments.

4. STORAGE MANAGEMENT FOR BASE STATIONS

In mobile environments, users move from one cell to another at their own will. Because the number of users in a wireless cell is not fixed, managing storage based on a fixed number of users is inappropriate. A more flexible storage management mechanism is therefore needed for mobile environments.

4.1 Leasing

Leasing is a mechanism that can be applied to managing stable storage for base stations. It —

- provides flexibility when requested storage is less than the system capacity limit,
- can control usage when the storage exceeds that limit.

From information gathered at lease negotiation, the storage manager knows the exact storage amount at any specific time. So it can appropriately arrange for future space allocation. Leasing can also prevent storage resources from being held indefinitely by failed or hanged processes. Both the requesting process and the manager know the expiration time of a lease, thus garbage collection is simplified.

The leasing mechanism in this paper is described in this section 4.1. Every process that needs to use stable storage negotiates with the manager for the size & duration of the lease. When the lease expires, the process must either obtain a lease extension (new lease) or the space is returned to the manager. The size & duration of the new lease can vary from the original lease. The storage manager can either grant or decline the renewal, based on the management policy. The leasing mechanism has 4 features:

- Negotiation: The storage manager and the process negotiate the duration of the lease and the size of the storage. The lease is valid only when the manager & process agree on the lease.
- Cancellation: The process can cancel the lease and return the space to the storage manager at any time before the lease expires. The manager, however, does not have the right of cancellation.
- Renewal: The process has the right to request a new lease before the expiration time of the lease. The renewal request is either granted or declined.
- Expiration: Every lease has an expiration time. The process must return the storage to the manager if its lease is not successfully renewed.

4.2 Negotiation Protocols

The storage manager uses negotiation protocols to establish leases with processes and to control system behavior. Four alternative protocols are described in this section 4.2. The storage manager can switch between protocols, based on system states.

4.2.1 Greedy

With the `Greedy` protocol, a process simply requests the desired time-duration and necessary storage-size. The manager examines the lease schedules for available storage. The manager agrees to the lease if there is sufficient available space to satisfy the request, otherwise, the lease request is declined. There are no further negotiations between the process and the manager in this protocol. The `Greedy` protocol is easy to implement but it is not balanced in assigning storage space. For example, processes that issue multiple small requests have advantages over processes that make a single large request.

4.2.2 Greedy with delay

Instead of declining requests that cannot be immediately satisfied, the `Delay` protocol examines the schedules for possible leases. A process can accept the manager's proposal for the modified lease as long as the delay is within the process's allowable range. This flexibility provides an advantage over the simple `Greedy` protocol. The `Delay` protocol improves the average ratio of successful requests when requests are not uniformly distributed.

4.2.3 Reservation

Unlike the `Greedy` & `Greedy with delay` protocols, the `Reservation` protocol provides a mechanism for more balanced storage management by ensuring that the ratios of successful requests for all processes are roughly the same. With this protocol, the storage manager first calculates the ratio of successful requests in the current cell for the process asking for the lease. The manager then reserves the storage for the processes with lower ratios of successful requests. The lease is granted only if there is enough available space remaining after reservations are committed.

4.2.4 Partial reservation

The `Reservation` protocol limits the use of available storage, to maintain balanced storage assignments. However, some reserved space might not be subsequently used because those processes that have lower success request ratios might terminate or leave the cell before their next checkpoints. Use of stable storage is reduced due to unnecessary reservations. Therefore, the `Partial reservation` protocol reserves only a portion of the requested space to improve storage utilization.

5. ADAPTIVE CHECKPOINTING WITH LEASING

Our approach uses time & leasing to coordinate checkpoint creation adaptively and indirectly. Ref [15] demonstrates that time can be used to implement coordinated checkpointing efficiently. Our storage manager uses the leasing mechanisms presented in section 4. A 3-level storage hierarchy is used to save checkpoints.

5.1 Checkpoint Creation

When the application begins, the protocol sets the checkpoint timers in all processes with a value equal to the checkpoint interval. Whenever a timer expires, a process

takes a checkpoint and resets the timer. The protocol uses a simple re-synchronization mechanism to roughly-synchronize the checkpoint intervals of the processes, even if drift rates of clocks are different. The content of the local timer is attached to each outgoing message. Whenever a process receives a message, the timer in the message is compared with the local timer. The process re-synchronizes the local timer if the value of the attached timer is larger.

To ensure that processes save consistent checkpoints, the protocol keeps a checkpoint number counter in each process. The counter is incremented whenever the process creates a checkpoint, and its current value, CN, is appended to each outgoing message. If a process receives a checkpoint number larger than the local one, the process creates a forced checkpoint before processing the message. For example, in figure 2, message m1 with checkpoint number CN is received by process P2 in checkpoint state (CN - 1) forcing a new checkpoint. The protocol logs all possible in-transit messages at the sender process to guarantee that they can be replayed during recovery. The sender process also logs both the send & receive sequence number counters. These counters are used for detecting lost & duplicate messages during retransmissions or failure recovery [25].

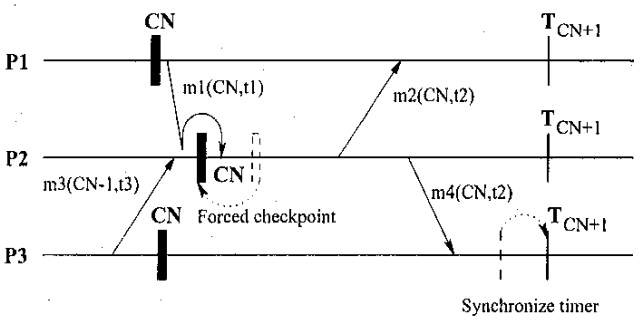

Figure 2: Time-Based Checkpoint Creation

5.2 Hierarchical Storage Management

The protocol uses a 3-level storage hierarchy to save checkpoints [26]. Checkpoints stored in level #1 are called soft checkpoints (SC); they are saved in the mobile host (*eg*, in a local disk or flash memory). Level #2 is the stable storage available in the base stations; level #3 corresponds to the home host. Levels #2 & #3 are both referred to as hard checkpoints (HC). Soft checkpoints are less reliable than hard checkpoints because they will be lost if the mobile host fails permanently. Hard checkpoints can survive mobile-host permanent failures but have higher overheads since they must be transmitted through the wireless channels. Based on the quality of service of the current network, this protocol can specify a ratio between soft & hard checkpoints for the best reliability & performance. For example, it can send a hard checkpoint to stable storage whenever a fixed number of soft checkpoints have been created on the local disk of the mobile host.

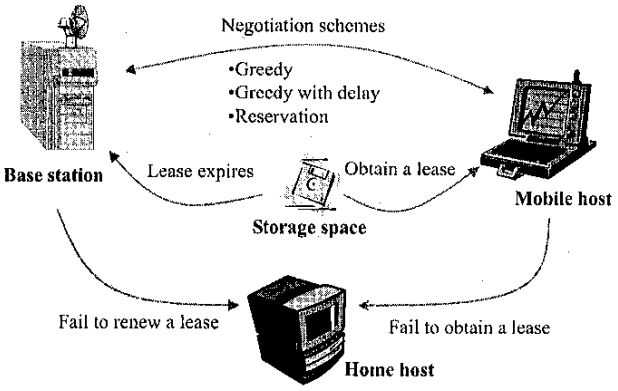

Figure 3: Leasing for the Base Station

There are distinct space requirements throughout the storage hierarchy. In the mobile host, it is only necessary to have space for 2 soft checkpoints. The stable storage on a base station must be shared among the mobile hosts currently in the cell. These mobile hosts can be executing different applications with distinct checkpoint intervals & sizes. Therefore, the base stations use the leasing mechanism to manage the stable storage. In the worst case, the home host must store 1 checkpoint for each process executing the application. It is assumed that there is enough space to store the checkpoints in the mobile & home hosts. This assumption is reasonable since these hosts likely belong to the same organization, which means that they can be configured to support the storage requirements of the applications.

The protocol first attempts to save the hard checkpoints in the base stations instead of the home host, due to performance advantages. The failure-free performance is better because 1 transmission-step is avoided. A checkpoint must pass through the base station first, before it is sent to the home host. Moreover, recovery is faster because checkpoints are closer to the mobile hosts. Requests for storage sometimes might not be immediately granted if there is sufficient space is not available in the base station. In this case, the protocol has to either postpone the hard checkpoint, or save it in another location.

As illustrated in figure 3, our protocol negotiates with base stations and the home host to determine the location to save the hard checkpoints. Whenever it is time to store a new hard checkpoint, the process contacts the local base station and tries to obtain a lease for the required space. Then, it transmits the checkpoint through the wireless link, and sends a completion notification to the home host. If it is unable to obtain a lease (within an allowable delay), the process stores the checkpoint directly in the home host. At this moment, the process has finished the checkpoint creation. On the home host, a monitoring process is initiated after arrival of the first completion notification. The monitoring process ensures that a new global state is saved in stable storage before the previous checkpoint is garbage collected by the storage manager.

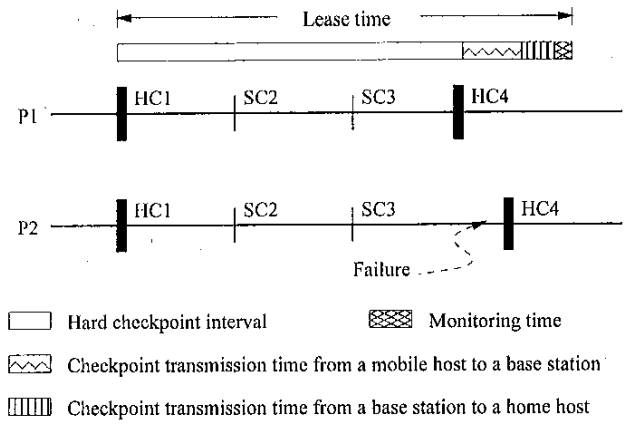

Figure 4: Soft & Hard Checkpoints

The monitor anticipates receiving a notification from all processes within a given monitoring time; otherwise it assumes that a failure could have occurred. In the latter case, the monitor requests from the base stations a copy of the previous checkpoint and saves them in the local stable storage.

The lease time must ensure that the current hard checkpoint of the process is safely stored in the base station until the next hard checkpoint is created. Moreover, it has to be sufficiently long to allow the home host to obtain the checkpoint copies in case of failures. Therefore, the lease time is set to be the sum of the

- hard checkpoint interval,
- monitoring time,
- time to transfer the checkpoint from the mobile host to the home host (see figure 4).

With this establishment of the lease time and the monitoring operation, at least 1 consistent global state can be preserved. Even if (see figure 4)

- timers are not well synchronized, and
- a permanent failure occurs during the time when some processes have completed their checkpoints while others are in progress,

the protocol still guarantees that there is a consistent state available for recovery. With failure-free execution, the global state will typically have been created before the leases expire. The monitoring process can send lease termination requests to the storage manager once all the notifications have been received.

5.3 Hand-Off Procedures

Before moving to another cell, the process notifies the storage manager at the current base station. The manager then forwards the hard checkpoint(s) of the process to the home host. After the checkpoint is saved safely by the home host, the checkpoint on the base station is removed. If the new cell provides storage service, and the process gets a lease, then the hard checkpoint can alternatively be sent to the new base station. This hand-off procedure simplifies garbage collection on base stations. When the mobile host leaves the current cell, the space occupied by its checkpoints becomes available for reallocation. This feature avoids having checkpoints scattered throughout the network while the mobile host moves around. The mobile host also does not have to maintain extra links to locate previous checkpoints.

5.4 Failure Detection & Recovery

The leasing mechanism provides enhanced fault detection for mobile applications. The storage manager anticipates receiving renewal or termination requests from the process before the lease expires. If there are no notifications concerning the lease, the manager assumes that the process that owns the lease has failed. The hard checkpoint is transmitted to its home host and the storage is returned to the manager. This scheme prevents losing 'necessary checkpoints' and 'wasted storage occupied by failed processes'.

Recovery is achieved by restarting the application process from a consistent global state. Depending on the type of failure, there can be 1 or 2 global states available. There can be a global state saved in the mobile hosts and another in the base stations or home host. The protocol determines the most recent checkpoint, using the checkpoint numbers. If the failure was permanent then at least one of the soft checkpoints is lost, which means that processes have to use the checkpoints saved in stable storage. The restarted processes replay the logged messages. Duplicated messages are detected using the received sequence numbers.

6. EVALUATION

6.1 Checkpointing Overhead

The overhead for saving checkpoints on a base station and on a home host was measured for a specific mobile environment. The mobile host was a Pentium II 300 MHz PC with 256 MB RAM and Red Hat Linux 5.0. The base station was a Sun Ultra Sparc 2 workstation with 512 MB RAM and Solaris 2.6. The connection between the mobile host and the base station was supported by the 2 Mbps Lucent WaveLAN and WavePOINT-II wireless interfaces. A Sun Ultra Sparc 1 workstation with Solaris 2.5 at another site 100 miles (160 km) away served as a home host.

The experiment was measured when the external loads on the machines & networks were very low (1:00 AM to 6:00 AM, during times of no backups). The mobile host started the timer and transmitted the checkpoints that ranged in size from 5 MB to 60 MB to the base station and the home host, respectively. The base station and the home host received the checkpoints, saved them to stable storage, and then sent an acknowledgment to the mobile host. The mobile host stopped the timer after the acknowledgment was received. Figure 5 shows the transfer-time for both the home host and the base station for the specified range of checkpoint sizes.

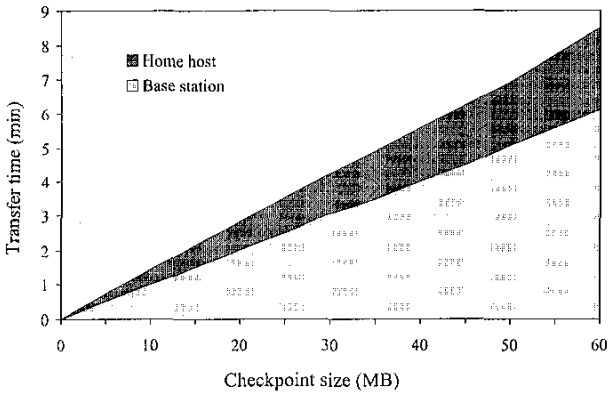

Figure 5: Comparison of Checkpoint Transfer-Time

6.2 Experimental Results

The checkpointing protocol with leasing was evaluated with the 4 negotiation schemes. Simulations were performed in a wireless cell containing mobile hosts and a base station. The mobile hosts communicated with a fixed network and obtained storage service from the base station. Processes on the mobile hosts periodically sent requests to the base station for storage space before taking hard checkpoints. The storage manager for the base station used leasing with the negotiation protocols to process the storage requests. Failure-free execution was assumed in the simulations.

Table 1 shows all parameters used for the simulations.

- The base station had 10 GB for storing checkpoints.
- The checkpoint size of a process ranged from 5 MB to 50 MB.
- The hard checkpoint interval was 30 minutes.
- A Gamma(3,1) distribution was used for the execution time of the process.
- A Poisson(3) distribution was used for the process arrival rate.

This arrival rate led to a slightly overloaded system (average storage requested: 13275 MB; standard deviation: 645 MB). The lease time of the process contained its hard checkpoint interval and the extra time required to transmit a checkpoint from the mobile host to its home host. The time used to transfer a checkpoint was based on the experimental results in section 6.1. The simulations were conducted for 110k simulation minutes. The boundary data collected during the first 5k minutes and the last 5k minutes were discarded.

The *request time* in our simulations is the time between 'when a process requests stable storage' to the time 'when it performs checkpointing'. The value of the request time is essential for the `Delay` protocol to re-synchronize the timers. A process obtaining a delayed lease propagates its timer to notify other processes. For the processes that communicate frequently, the values of the request time are typically smaller; for those processes that rarely exchange messages, the values are typically larger. If the request time is not long enough, the processes might not have sufficient time to synchronize the timers, and it could result in inconsistent global states. The value of the request time can affect success request ratios. The advantage in using the longer request time is that the process can request & obtain the required space earlier; the drawback is that the storage manager can only provide current storage information. The process might miss a chance to obtain storage released later. On the other hand, the process with shorter request time has the most recent information on available storage but loses the first opportunity to request a lease.

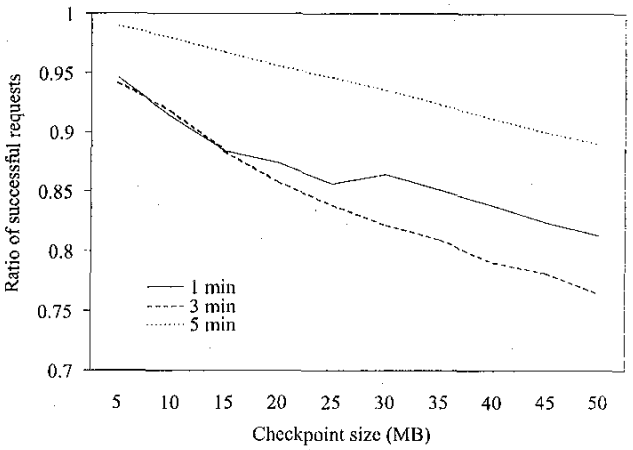

Figure 6: Comparison of Request Times

Figure 6 compares the performance of the `Greedy` protocol with various request times. The processes with 5-minute request time had higher success request ratios for all checkpoint sizes. However, this does not imply that the earlier request is always advantageous. Figure 6 shows that when the checkpoint size was larger than 15 MB, the processes with 1-minute request time had higher probability to obtain leases than the processes with 3-minute request time. Since mobile applications typically interact frequently, the simulations in the remainder of this paper used only the 1-minute request time.

The `Delay` protocol generally produced more successful requests than the `Greedy` protocol in the overloaded system (see figure 7). This result is due to two reasons:

1. Delay typically provided more opportunity to obtain leases. As demonstrated in figure 7, more delay time gives higher success request ratios.
2. Delay slightly decreased the number of total requests. Table 2 shows the average number of requests for processes with varying checkpoint sizes. Larger delay time did contribute to better success request ratios. However, this could be detrimental to the hard checkpoint interval.

Figure 8 compares the average ratios of successful requests for various negotiation protocols.

- `5-minute Delay` protocol achieved the highest average success ratios.

Table 1: Simulation Parameters

Parameter	Value or Range	Remark
Checkpoint size	5 – 50 MB	per 5 MB
Hard checkpoint interval	30 minutes	
Time to transfer a checkpoint	1 – 9 minutes	based on the results in section 6.1
Arrival rate	Poisson(3)	number of new processes in a minute
Execution time	Gamma(3,1) hours	battery limited
Delay time allowed	3, 5, 7 minutes	
Request time	1 minute	
Partial reservation ratio	0.005	
Maximum size of storage	10 GB	
Simulation time	110k minutes	

Table 2: Average Number of Requests

Delay Time (min)	Checkpoint Size (MB)									
	5	10	15	20	25	30	35	40	45	50
0	5.61	5.60	5.61	5.61	5.61	5.60	5.63	5.62	5.63	5.59
3	5.55	5.53	5.55	5.58	5.57	5.61	5.59	5.62	5.66	5.57
5	5.33	5.24	5.24	5.26	5.25	5.32	5.35	5.48	5.47	5.45
7	5.22	5.10	5.14	5.07	5.08	5.11	5.11	5.18	5.18	5.33

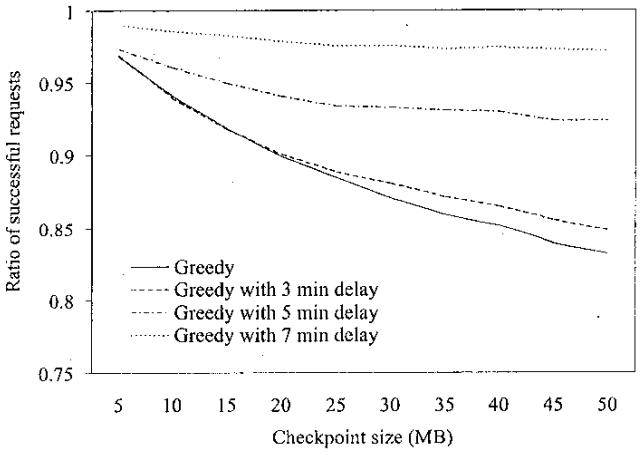

Figure 7: Performance *vs* Delay

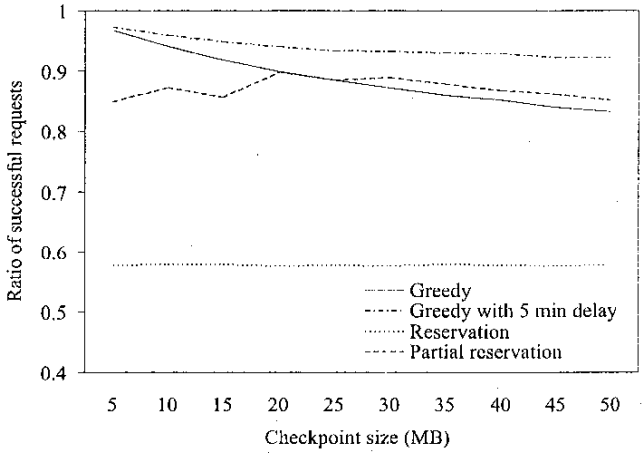

Figure 8: Average Success Ratios

· `Reservation` protocol achieved approximately equal success request ratios for various sizes of storage requests but had the lowest average ratios.

· `Partial Reservation` protocol with 0.005 reservation portion maintained balanced storage-assignment for various request sizes, and increased the average success request ratios by almost 30%.

The —

· `Greedy`,

· `5-minute Delay`,

· `Partial reservation`,

were not ideal protocols for storage management with checkpointing, although these protocols had good average ratios of successful requests. The standard deviations in successful requests produced by these 3 protocols were higher than the `Reservation` approach (see figure 9). The high standard deviation implied that some processes created hard checkpoints in the base station more frequently than other processes. The higher standard deviation also led to more 'consecutive aborted checkpoints' that affected the performance of the processes. The aborted checkpoint forced the processes to transfer hard checkpoints to the home host. These two behaviors resulted in widely varying checkpoint-overhead for the processes. Figure 10 shows that the `Non-leasing` and the `Greedy` protocols had more consecutive aborted checkpoints than other protocols in most cases. With the `Reservation` protocol, no process aborted checkpointing repetitively[1]. When a protocol has

[1]In figure 10, the average numbers of consecutive aborted checkpoints for the `Reservation` protocol are all zero.

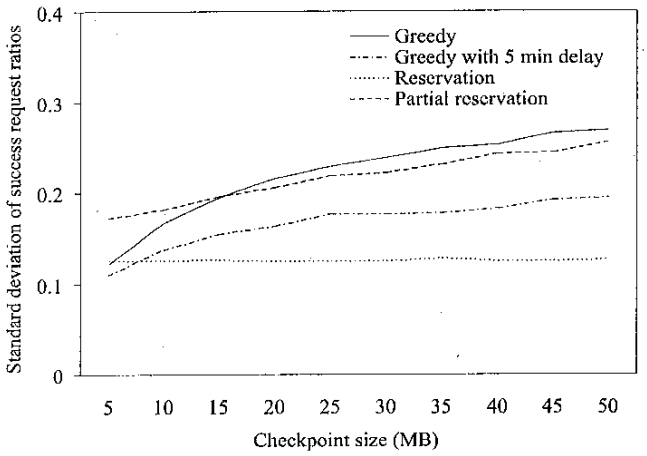

Figure 9: Standard Deviation of the Ratios

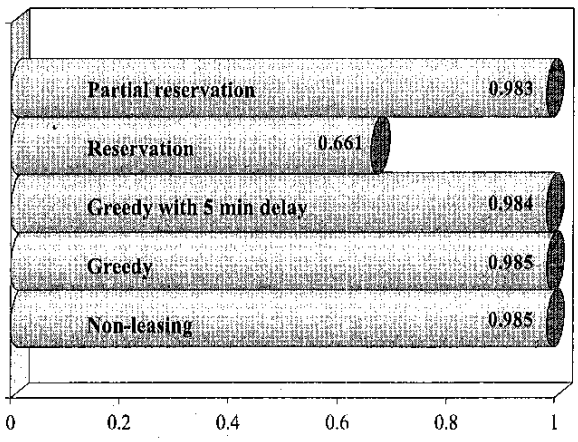

Figure 11: Storage Utilization

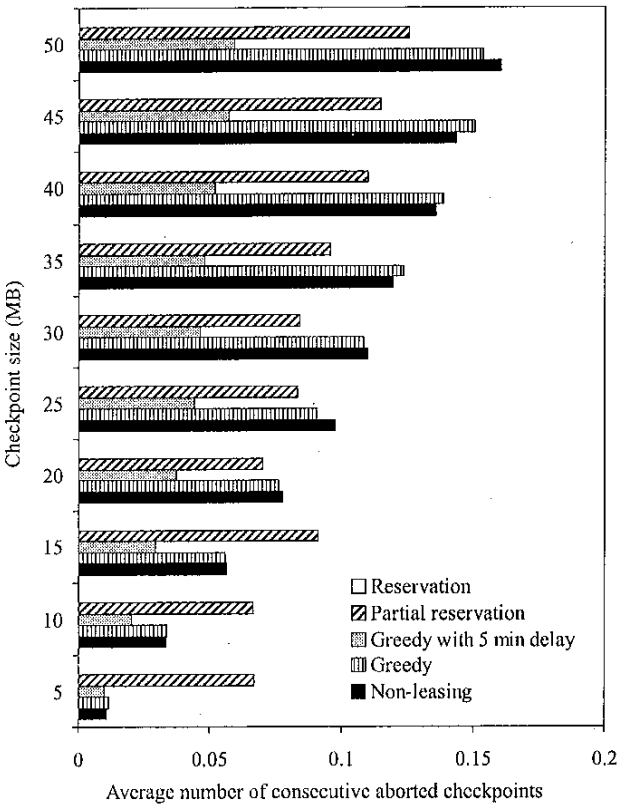

Figure 10: Number of Consecutive Aborted Checkpoints per Process

few consecutive aborted checkpoints, the processes can postpone the aborted hard checkpoint for enhanced execution performance since it is not as likely to miss its next hard checkpoint.

The `Reservation` protocol maintained the smallest worst-case hard checkpoint intervals on the base station but sacrificed storage utilization dramatically (see figure 11). The `Partial reservation` approach improved upon the `Reservation` protocol and achieved 0.98 storage utilization. Other negotiation protocols did not reserve storage for any processes so they had better storage utilization.

The experimental results showed that the adaptive checkpointing protocol successfully integrated with the various negotiation schemes. With the protocol, applications always have consistent global checkpoints and storage managers can use a variety of negotiation schemes to maintain desired space allocation.

ACKNOWLEDGMENT

We are pleased to thank Prof. Hewijin C. Jiau for her suggestions on drafts of this paper and the anonymous referees for their comments. This research was supported in part by the US Defense Advanced Research Projects Agency (DARPA) under contract DABT 63-96-C-0069, and in part by the US Office of Naval Research under contract N00014-97-1-1013.

REFERENCES

[1] E.A. Brewer, R.H. Katz, Y. Chawathe, *et al*, "A network architecture for heterogeneous mobile computing", *IEEE Personal Communications*, vol 5, num 5, 1998 Oct, pp 8 – 24.

[2] B.R. Badrinath, A. Acharya, T. Imielinski, "Impact of mobility on distributed computations", *SIGOPS Review*, vol 27, num 2, 1993 Apr, pp 15 – 20.

[3] G.H. Forman, J. Zahorjan, "The challenges of mobile computing", *Computer*, 1994 Apr, pp 38 – 47.

[4] M. Satyanarayanan, "Fundamental challenges in mobile computing", *Tech. Rep.* CMU-CS-96-111, 1996 Feb; Carnegie Mellon Univ.

[5] D.B. Johnson, W. Zwaenepoel, "Sender-based message logging", *Proc. IEEE Fault-Tolerant Computing Symp*, 1987, pp 14 – 19.

[6] E.N. Elnozahy, D.B. Johnson, W. Zwaenepoel, "The performance of consistent checkpointing", *Proc. IEEE Reliable Distributed Systems Symp*, 1992, pp 39 – 47.

[7] J.S. Plank, M. Beck, G. Kingsley, K. Li, "Libckpt: Transparent checkpointing under Unix", *Proc, Usenix Winter Technical Conf*, 1995, pp 213 – 223.

[8] Y.-M. Wang, Y. Huang, K.-P. Vo, *et al*, "Checkpointing and its applications", *Proc. IEEE Fault-Tolerant Computing Symp*, 1995, pp 22 – 31.

[9] P.E. Chung, Y. Huang, S. Yajnik, *et al*, "Checkpointing in CosMiC: A user-level process migration environment", *Proc. Pacific Rim Int'l Symp. Fault-Tolerant Systems*, 1997, pp 187 – 193.

[10] N. Neves, W.K. Fuchs, "RENEW: A tool for fast and efficient implementation of checkpoint protocols", *Proc. IEEE Fault-Tolerant Computing Symp*, 1998, pp 58 – 67.

[11] K.F. Ssu, W.K. Fuchs, "PREACHES: Portable recovery and checkpointing in heterogeneous systems", *Proc. IEEE Fault-Tolerant Computing Symp*, 1998, pp 38 – 47.

[12] A. Acharya, B.R. Badrinath, "Checkpointing distributed applications on mobile computers", *Proc.* 3rd *Int'l Conf. Parallel and Distributed Information Systems*, 1994, pp 73 – 80.

[13] D.K. Pradhan, P. Krishna, N.H. Vaidya, "Recoverable mobile environment: Design and trade-off analysis", *Proc. IEEE Fault-Tolerant Computing Symp*, 1996, pp 16 – 25.

[14] R. Prakash, M. Singhal, "Low-cost checkpointing and failure recovery in mobile computing systems", *IEEE Trans. Parallel and Distributed Systems*, vol 7, num 10, 1996 Oct, pp 1035 – 1048.

[15] N. Neves, W.K. Fuchs, "Adaptive recovery for mobile environments", *Communications of ACM*, vol 40, num 1, 1997 Jan, pp 68 – 74.

[16] G. Cao, M. Singhal, "On the impossibility of min-process non-blocking checkpointing and an efficient checkpointing algorithm for mobile computing systems", *Proc. Int'l Conf. Parallel Processing*, 1998, pp 37 – 44.

[17] G. Cao, M. Singhal, "Low-cost checkpointing with mutable checkpoints in mobile computing systems", *Proc.* 18th *Int'l Conf. Distributed Computing Systems*, 1998, pp 464 – 471.

[18] B. Yao, K.F. Ssu, W.K. Fuchs, "Message logging in mobile computing", *Proc. IEEE Fault-Tolerant Computing Symp.*, 1999, pp 294 – 301.

[19] R.H.B. Netzer, J. Xu, "Necessary and sufficient conditions for consistent global snapshots", *IEEE Trans. Parallel and Distributed Systems*, vol 6, num 2, 1995 Feb, pp 165 – 169.

[20] N. Neves, W.K. Fuchs, "Using time to improve the performance of coordinated checkpointing", *Proc. IEEE Int'l Computer Performance & Dependability Symp*, 1996, pp 282 – 291.

[21] J.P. Gelb, "System-managed storage", *IBM Systems J.*, vol 28, num 1, 1989, pp 77 – 103.

[22] S. Smith, "Data facility storage management subsystem", *Capacity Management Review*, vol 18, num 4, 1990 Apr, pp 3 – 4.

[23] J. Yin, L. Alvisi, M. Dahlin, C. Lin, "Volume leases for consistency in large-scale systems", *IEEE Trans. Knowledge and Data Engineering*, vol 11, num 4, 1999 Jul/Aug, pp 563 – 576.

[24] C.E. Perkins, *Mobile IP*, 1997; Addison Wesley.

[25] N. Neves, W.K. Fuchs, "Coordinated checkpointing without direct coordination", *Proc. IEEE Int'l Computer Performance & Dependability Symp*, 1998, pp 23 – 31.

[26] N.H. Vaidya, "A case for two-level distributed recovery schemes", *Proc. ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, 1995, pp 64 – 73.

AUTHORS

Kuo-Feng Ssu; Coordinated Science Laboratory; University of Illinois; Urbana, Illinois 61801 USA.
Internet (e-mail): ssu@ecn.purdue.edu

Kuo-Feng Ssu is a PhD student in Computer Science at the University of Illinois. He received the BS in Computer Science & Information Engineering from National Chiao Tung University, Taiwan. He is a member of the Phi Tau Phi honor scholastic society.

Bin Yao; School of Electrical & Computer Eng'g; Purdue Univ; West Lafayette, Indiana 47907 USA.
Internet (e-mail): yaob@ecn.purdue.edu

Bin Yao is a PhD student in the School of Electrical and Computer Engineering at Purdue University. He received the BS (1996) from the Department of Electronics at Beijing University. His research interests include availability and reliability in distributed systems, especially with wireless networks and mobile systems.

Dr. W. Kent Fuchs; School of Electrical & Computer Eng'g; Purdue Univ; West Lafayette, Indiana 47907 USA.
Internet (e-mail): fuchs@purdue.edu

W. Kent Fuchs received the BSE from Duke University, MDiv from Trinity Evangelical Divinity School, and PhD in Electrical Engineering from the University of Illinois. He is Head of the School of Electrical and Computer Engineering, Purdue University. He was a Professor in the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois. His research interests include dependable computing, testing, and failure diagnosis. Research awards include the Senior Xerox Faculty Award for Excellence in Research, selection as a University Scholar, appointment as Fellow in the Center for Advanced Studies, and the Xerox Faculty Award for Excellence in Research, all from the University of Illinois. He also received the Digital Equipment Corporation Incentives for Excellence Faculty Award, Best Paper Award IEEE/ACM Design Automation Conference, and the Best Paper Award VLSI Test Symposium. Dr. Fuchs has been a Guest Editor of special issues for *IEEE Trans. Computers* and the *IEEE Computer*. He has been a member of the editorial board for the *IEEE Trans. Computers*, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, and *J. Electronic Testing: Theory and Applications*. He is an IEEE Fellow.

Nuno Ferreira Neves; Dep'to de Informatica; Faculdade de Ciencias da Universidade de Lisboa; Bloco C5 - Piso 1, Campo Grande; 1749-016 Lisboa, PORTUGAL.
Internet (e-mail): nuno@di.fc.ul.pt

Nuno Ferreira Neves received a licenciatura (5 year degree, 1992) and a Masters (1995) in Electrical and Computer Engineering from the Technical University of Lisbon. His PhD (1998) is from the University of Illinois at Urbana-Champaign. He is an Assistant Professor in the Computer Science Department of the University of Lisbon. His research was recognized with several fellowships, and with the William C. Carter award at the 1998 IEEE International Fault-Tolerant Computing Symposium. His interests include fault-diagnosis and recovery techniques, intrusion detection mechanisms, and tools for the development of distributed applications.

Manuscript TR1999-401 received: 1999 May 15;
revised: 1999 September 9, October 21.
Responsible editor: M.R. Lyu
Publisher Item Identifier S 0018-9529(99)10314-2