

Designing Efficient Fault-Tolerant Systems on Wireless Networks

Guohong Cao
Department of Computer Science & Engineering
The Pennsylvania State University
University Park, PA 16802
E-mail: gcao@cse.psu.edu

1 Introduction

The falling cost of both communication and mobile computing devices (laptop computers, hand-held computers, etc.) is making mobile computing affordable to both business users and private consumers. People on the road want to use their laptops to read Email, login on remote machines, buy or sell stocks, and so on; rescue workers at disaster sites (fires, floods, earthquakes, etc.) would like to send messages, keep records, and communicate with each other by mobile devices; a general needs to gather information from his soldiers and send commands to them. In these applications, a system failure can cause loss of opportunity, financial loss, or even loss of human lives. It is clear that in many applications, highly dependable systems are needed.

Mobile wireless environments pose challenging problems in designing fault-tolerant systems because of the dynamics of mobility and limited bandwidth available on wireless links. Traditional fault-tolerance schemes cannot be directly applied to these systems. However, fault tolerance is much more important in mobile computing systems since mobile computing systems are more prone to failures. This is because (i) wireless networks have high error rates and more frequent disconnections and (ii) mobile devices are more prone to failure, physical damage, or loss. In this thesis, we address these problems at two levels: the network level and the operating system level. At the network level, we design fault-tolerant network protocols. In particular, we design efficient fault-tolerant channel allocation algorithms and fault-tolerant location management schemes. At the operating system level, we study checkpointing and recovery technique, which is an attractive approach for transparently adding fault-tolerance to distributed applications. Even though this technique can tolerate both mobile device and wireless network failures, we still study fault-tolerant network protocols, since wireless networks are not reliable and using operating system level approaches to tolerate network failures have high overhead compared to using network level approaches.

Channel allocation has received considerable attention, but none of the existing schemes takes into consideration site failures, communication link failures, or even network congestion. Although the checkpointing and recovery approach has been extensively studied in traditional distributed systems, the new constraints posed by mobile computing systems, such as low bandwidth of wireless channels, high search cost, and limited battery life, make traditional checkpointing algorithms unsuitable to them.

Mobile computing becomes an emerging trend. Due to the constraints of wireless networks, designing fault-tolerant systems on wireless networks is a challenge. The author has many years experience of designing

fault-tolerant systems on wireless networks. These experiences are valuable for people who are interested in enhancing the reliability and survivability of mobile computing systems. In the following, we present our preliminary results and ongoing work in these areas.

2 Fault-Tolerant Channel Allocation

On wireless networks, to establish a communication session (i.e., make a call), a *mobile host (MH)* sends a request to the *mobile support station (MSS)* in its geographical area (called a *cell*). The session is supported if a wireless channel can be allocated for the communication between the *MH* and the *MSS*. Many channel allocation algorithms have been proposed to avoid channel interference and efficiently utilize the limited frequencies. All these algorithms, which are referred to as *centralized* channel allocation algorithms, rely on a *mobile switch center (MSC)* to accomplish channel allocation. Recently, distributed channel allocation algorithms have received considerable attention due to their high reliability and scalability. In this approach, an *MSS* communicates with other *MSSs* directly to find the available channels and make sure that borrowing a channel does not interfere with other cells. However, existing distributed channel allocation algorithms require that the borrower waits for acknowledgments from its interference neighbors. Thus, a borrower cannot borrow a channel if it cannot communicate with any one of them. Since many communication sessions, such as hand-off, audio, and video, have time constraints, a long communication delay resulting from network congestion has the same effect as a communication link failure or an *MSS* failure, where a borrower may fail to borrow a channel even though there exists available channels. In a real network, under high traffic load, a cell has a large probability of experiencing intermittent network congestion, communication link failures, or node (*MH* or *MSS*) failures. Therefore, existing distributed approaches are not suitable in a real-life network.

We [5, 7] proposed a distributed fault-tolerant channel allocation algorithm which tolerates communication link failures and *MSS* failures. In the algorithm, a borrower does not need to receive a response from every interference neighbor. By cleverly arranging channels and cells, a borrower only needs to receive responses from some of its interference neighbors, and then the failure of other neighbors does not affect this borrower. Thus, our algorithm tolerates network congestion, communication link failures, and node failures. Based on a typical cellular network model, in our algorithm, a cell can still borrow channels even though it cannot communicate with as many as 90% of its interference neighbors. Simulation results showed that our algorithm significantly reduces the failure rate under network congestion, communication link failures, and node crashes compared to existing channel allocation algorithms. Moreover, our fault-tolerant algorithm reduces the message overhead compared to known distributed channel allocation algorithms and outperforms them in terms of failure rate under uniform as well as non-uniform traffic distributions.

3 Coordinated Checkpointing for Mobile Systems

Coordinated checkpointing is a commonly used technique to prevent complete loss of computation upon a failure. Much of the previous work in coordinated checkpointing has focused on minimizing the number of synchronization messages and the number of checkpoints. However, these algorithms (called *blocking algorithms*) force all relevant processes in the system to block their computations during checkpointing. Checkpointing includes the time to trace the dependency tree and to save the states of processes on stable storage, which may be

long. Moreover, in mobile computing systems, due to the mobility of MH s, a message may be routed several times before reaching its destination. Therefore, blocking algorithms may dramatically reduce the performance of these systems.

Recently, nonblocking algorithms have received considerable attention. In these algorithms, processes need not block during checkpointing by using a checkpointing sequence number to identify inconsistent messages. However, these algorithms assume that a distinguished initiator decides when to take a checkpoint. Therefore, they suffer from the disadvantages of centralized algorithms, such as poor reliability, bottlenecks, etc. Moreover, these algorithms require all processes in the system to take checkpoints during checkpointing, even though many of the checkpoints may not be necessary.

The Prakash-Singhal algorithm was the first algorithm to combine these two approaches. More specifically, it forces only a minimum number of processes to take checkpoints and does not block the underlying computation during checkpointing. However, we [4, 3] found some problems in their algorithm and proved that there does not exist a non-blocking algorithm which forces only a minimum number of processes to take their checkpoints. This implies that there are three directions in designing efficient coordinated checkpointing algorithms. One extreme is to relax the non-blocking condition while keeping the min-process property. The other extreme is to relax the min-process condition while keeping the non-blocking property. Between these two extremes, we can also design blocking non-min-process algorithms that significantly reduce the blocking time as well as the number of checkpoints.

In [4], we proposed a min-process checkpointing algorithm for mobile computing systems. In [2, 1], we proposed a non-blocking algorithm. This algorithm is based on a new concept of “mutable checkpoint”, which is neither a tentative checkpoint nor a permanent checkpoint. Mutable checkpoints can be saved anywhere; e.g., the main memory or local disk of MH s. In this way, taking a mutable checkpoint avoids the overhead of transferring large amount of data over the wireless network to the stable storage at MSS s. We have developed techniques to minimize the number of mutable checkpoints. Simulation results showed that the overhead of taking mutable checkpoints is negligible. Based on mutable checkpoints, our non-blocking algorithm forces only a minimum number of processes to take their checkpoints on stable storage.

4 On-Going Work

The problems we are pursuing are as follows.

Efficient fault-tolerant channel allocation: Based on our fault-tolerant channel allocation algorithm and our simulation environment, we plan to do research along the following directions: i) designing an adaptive approach [6] which makes use of the temporal locality existing in most applications. In our approach, the borrower keeps the borrowed channels for a while before returning them to the lender. In this way, a load balancing is achieved and message overhead is significantly reduced. We plan to investigate various methods to determine how long the borrower should keep the borrowed channels; ii) providing QoS (Quality of Service) for mobile computing systems. To support multimedia traffic, it is necessary to provide QoS guarantees between end systems. In most existing schemes, bandwidth is reserved not only in the cell where the call is ongoing, but also in all the neighboring cells to handle hand-off. We plan to investigate the possibility of predicting

destination cells and will develop schemes for bandwidth reservation through channel borrowing.

Efficient fault-tolerant location management: In order to track the location of MHs, databases are used to store location information. We plan to design fault-tolerant location databases by using the quorum-based approach, where the location information of an MH is replicated at several places. Due to built-in redundancy, such an approach is fault-tolerant. To track down an MH , searching a nearby database improves performance. To implement a quorum-based system, mutual exclusion must be ensured. Since the first quorum-based mutual exclusion algorithm was proposed in 1985, it has been an open problem to reduce the delay in processing users' requests from $2T$ to T (T is the message delay). No one has been able to solve it until we [8] proposed an optimal scheme which has a delay of T . With our scheme, the waiting time of a request is nearly reduced to half and the system throughput is almost doubled. In future research, we plan to extend this work to replicated data management and then to fault-tolerant location management.

Checkpointing and recovery: Our existing work shows that no min-process non-blocking algorithm exists, and we have already proposed an efficient non-blocking algorithm [2] and an efficient min-process algorithm [4]. Our future work will be: i) to evaluate both approaches. It is difficult to say which approach is better since different approaches are suitable for different applications. We will identify the suitable applications for each approach. Also, we will try to find out if there exists an optimal scheme between these two approaches; ii) to combine the two-level recovery scheme and our checkpointing approach to design efficient recovery algorithms for mobile systems. In mobile systems, some failures are permanent, and then it is necessary to save checkpoints on stable storages in $MSSs$. However, transient failures can be recovered locally to avoid the overhead of transferring checkpoints from (to) the $MSSs$. We plan to design a two-level checkpointing and recovery algorithm for mobile computing systems to reduce performance overhead; that is, one level is used to recover permanent failures, and the second is used to recover transient failures. We also plan to conduct a comprehensive study of the developed algorithms.

References

- [1] G. Cao and M. Singhal. "Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing Systems". *IEEE Trans. Parallel and Distributed System*, To appear.
- [2] G. Cao and M. Singhal. "Low-Cost Checkpointing with Mutable Checkpoints in Mobile Computing Systems". *Proc. 18th Int'l Conf. on Distributed Computing Systems*, pages 464–471, May 1998.
- [3] G. Cao and M. Singhal. "On Coordinated Checkpointing in Distributed Systems". *IEEE Trans. Parallel and Distributed System*, pages 1213–1225, Dec. 1998.
- [4] G. Cao and M. Singhal. "On the Impossibility of Min-Process Non-Blocking Checkpointing and An Efficient Checkpointing Algorithm for Mobile Computing Systems". *Proc. 27th Int'l Conf. on Parallel Processing*, pages 37–44, Aug. 1998.
- [5] G. Cao and M. Singhal. "Distributed Fault-Tolerant Channel Allocation for Mobile Cellular Networks". *IEEE INFOCOM'99*, pages 584–591, Mar. 1999.
- [6] G. Cao and M. Singhal. "An Adaptive Distributed Channel Allocation Strategy for Mobile Cellular Networks". *Journal of Parallel and Distributed Computing, Special Issue on Mobile Computing*, 60(4):451–473, April 2000.
- [7] G. Cao and M. Singhal. "Distributed Fault-Tolerant Channel Allocation for Cellular Networks". *IEEE Journal of Selected Areas in Communications*, 18(7):1326–1337, July 2000.
- [8] G. Cao, M. Singhal, Y. Deng, N. Rishé and W. Sun. "A Delay-Optimal Quorum-Based Mutual Exclusion Scheme With Fault-tolerance Capability". *Proc. 18th IEEE Int'l Conf. on Distributed Computing Systems*, pages 444–451, May 1998.