Proc. SRDS 2000 (IEEE CS 19th Symp. on Reliable Distributed Systems), Nuremberg, Germany, Oct. 2000, pp. 106-115.

Issues Insufficiently Resolved in Century 20 in the Fault-Tolerant Distributed Computing Field

(Invited Paper)

K. H. (Kane) Kim

DREAM Laboratory
University of California
Irvine, CA 92697, U.S.A.
khkim@uci.edu , http://dream.eng.uci.edu/

Abstract: As Century 21 just opened up, it is a fitting time to reflect on the evolution of the fault-tolerant distributed computing technology that occurred in the last century. The author's view of that evolution is sketched in this paper with emphasis on the major issues insufficiently resolved in Century 20. Such issues are naturally among what this author believes to be the prime subjects that need to be addressed in this decade by the research community. A substantial part of this paper deals with the issues that need to be resolved to advance the real-time fault-tolerant distributed computing branch into a mature practicing field.

Keywords: abort, availability, detection, distributed computing, fault, network surveillance, object, real time, recovery, reliability, replication, transaction, tolerance.

1. Introduction

From the beginning of the *distributed computing* (DC) era that occurred around mid-1970's, *fault tolerance* has been touted as one of the main advantages of shifting the computer system design from the centralized structure into a more distributed structure. As Century 21 just opened up, it is a fitting time to reflect on the evolution of the *fault-tolerant (FT) DC* technology that occurred in the last century, to be more precise, in the final quarter of Century 20. In this paper, the author sketches his view of that evolution.

In this short review, the author's emphasis is on the major issues insufficiently resolved in Century 20 rather than those well resolved. Such issues are naturally among what this author believes to be the prime subjects that need to be addressed in this decade by the research community. Some of those issues may not be sufficiently resolved even through this decade.

A substantial part of this paper deals with the issues that need to be resolved to advance the *real-time (RT) FT DC* branch into a mature practicing field.

In the next section, the liveliness aspect of the FT DC field as both a practicing field and a research and development (R&D) field is reviewed. The advances achieved in Century 20 are then briefly reviewed in Section 3. Major issues that remained insufficiently resolved through Century 20 are discussed in Section 4. The final section provides a summary and an argument for some cautious attitudes in tackling the research issues in this new century.

2. Liveliness of the FT DC field

2.1 Up, down, and up again

As far as industry specializing in fault tolerance is concerned, it is probably safe to say that such industry never flourished. To this author it appeared to be somewhat more lively in 1980's than in 1990's.

As major reasons, the following could be cited:

(1) The reliability of hardware components showed spectacular improvements in the last two decades. For example, by mid-1990's, the reliability of desk-top personal computer workstations and supporting file servers had shown steady improvements and it had caused the reliability concerns of common users of non-real-time computer-based application systems to have diminished considerably. On the other hand, general-purpose database management system (DBMS) vendors successfully incorporated into their system software reasonable abilities to keep the *integrity of data* in spite of component failures in non-real-time application environments.

(2) Besides the data backup and simple *transaction* [Gra94] mechanisms of DBMS software, the system software support needed for *higher-coverage FT computing* (e.g., automatic retry of a failed transaction) in environments where *specially hardened FT hardware modules* are not used, did not advance into mature forms in Century 20.

(3) RT computing applications remained a small segment of the computing applications and did not attract serious attentions of main-stream computing industry during 1980's and a large part of the 1990's.

Therefore, the concerns of main-stream computing industry were to merely maintain the integrity of the DBMS. It sufficed for them to facilitate *clean abort* of transactions whenever faults in intermediate computation results or uncommitted data were found. They did not feel that additional execution overhead and hardware and

software costs involved in facilitating higher-coverage FT computing, e.g., automatic retry of a failed transaction, were worthwhile. If they tried to do that, their products would not be competitive in the market of those times.

Now the FT DC field is bouncing back up again. There is still no sign of growth (or one can even say, resurrection) of specialized FT computing industry. However, the interests of main-stream computing industry in FT DC are now growing fast for at least two reasons:

(1) Rapid growth of the *Web server* market and customers' growing demands for *high-availability Web servers*, and

(2) Rapid growth of RT computing applications that started around mid-1990's, especially growing demands for computer-embedded communication-equipped devices / systems in this new decade.

The main-stream computing industry is trying to meet these demands by incorporating cost-effective FT DC mechanisms within the framework of general-purpose hardware facilities and general-purpose operating system (OS) architecture.

Now with the explosive growth of the electronic commerce activities under way, the Web server market appears to be becoming the most important market for major vendors of computing platforms. Some say that end users lose patience when Web sites handling competitive commercial activities take longer than 8 seconds to show results. This means that

(1) Such Web sites must be up "all the time", i.e., meet high-availability requirements; and

(2) Web sites must respond fast even when they are accessed by a large number of clients concurrently.

Therefore, the main-stream computing industry is now becoming better motivated to explore higher-coverage FT DC approaches than ever before.

In this new decade, the RT computing application market is no longer a negligible market even for major platform vendors. In RT applications, mere clean abort of transactions is rarely an acceptable approach because an abortion of a transaction leads more often than not to abandoning the application. Most sizable RT computing applications are now of DC type. Again, higher-coverage FT DC is becoming a lively field.

Overall, it is safe to say that the desire in any computing industry and any system design team for adopting fault tolerance approaches has fluctuated as changes in the following have occurred:

(1) Natural failure rates of hardware and software components,
(2) Environmental factors inducing disturbances into computing,
(3) Costs of computing failures to applications / missions,
(4) Costs of redundancy relative to the computer system budget.

2.2 Clean abort, high-availability server, and RT recovery

As discussed in the preceding section, high-availability Web servers must be capable of doing more than clean abort. If a node crashes, the impacted on-going transactions must be cleanly aborted and the server function of the crashed node must be resurrected in another node within a reasonable amount of time. Therefore, this high-availability server approach is aimed for higher coverage in FT DC than the coverage goal of the conventional server relying on clean abort only. The former leads clients to experiencing disconnection from the server for shorter duration and incurs less costs to the applications / missions than the latter does.

When wireless network components are used in Web server applications, the fault rate and the needs for fault tolerance mechanisms tend to become more significant.

RT computing applications, e.g., video-conferencing, voice over IP (internet protocol), factory automation, defense applications, etc., require even higher coverage in FT DC. Attempts for automated retry of a failed transaction or concurrent redundant tries of a transaction are usually essential. That is, attempts to avoid any loss of a transaction are desirable. Therefore, forward or backward *RT recovery* from faults is a usual attempt.

This means that the interests of main-stream computing industry in FT DC technologies have been advancing in the past two decades as follows:

Clean abort technologies
→ high-availability server technologies
→ RT recovery technologies.

Of course, research communities dealt with all three types of FT DC approaches and various proven or promising solutions have been produced. However, the amount of research invested in the three areas has been largely proportional to the degree of interests of the main-stream industry shown in the areas.

3. FT DC advances in Century 20

3.1 Advances in hardened hardware component technologies

Spectacular advances in integration of numerous logic components which were earlier-generation building-blocks of computer systems into a smaller number of VLSI modules, have already melted considerably the reliability concerns of common users of non-real-time systems. In addition, computer industry achieved significant advances in late 1970's and 1980's in producing specially *hardened hardware modules* which were attractive building-blocks of high-reliability computing systems in safety-critical applications. Representative examples of such hardened modules are:

(1) Hardened processor modules: *comparing processor-pair*, *pair of self-checking processors*, and *voting-TMR*

(triple modular redundancy) processor module [Toy87].

(2) RAID (*redundant array of inexpensive disks*): This has become popular even in database-centric business computing applications.

(3) *Error-detection and error-correction coding* subunit: This has been extensively used in CPUs and communication processors and various peripheral devices.

Industry generally feels that these technologies are mature. Yet since standard general-purpose hardware modules have become so reliable and powerful in performance, much of the main-stream industry has growing interests in using software techniques instead of specially hardened hardware components to realize acceptable levels of system reliability.

3.2 Fault detection and network surveillance

Various basic approaches for fault detection were established in Century 20.

(1) *Timeout*, enforced by use of watch-dog timers;

(2) *Comparison* of the results of repeated or redundant executions;

(3) Error-detection and error-correction code;

(4) *Acceptance test* or reasonableness check, which is to check the reasonableness of intermediate computation results [Ran75, Ran95].

The first three techniques have generally matured and the last technique has also been used extensively although there is still large room for further research on the latter.

Another important category of fault detection techniques are those for:

(5) *Network surveillance* which is also called *membership maintenance* [Kop89]. Network surveillance is basically a (partially or fully) decentralized mode of detecting faulty and repaired status of DC components. It is aimed for minimizing the periods during which faulty components (e.g., processing nodes and communication links) are lurking in DC systems. This means to facilitate fast *learning* by each interested fault-free node of the *faults* or *repair completion* events occurring in other parts of the DC system.

The simplest version of the network surveillance technique is to have a master node make a periodic roll-call of other nodes in the system [Hec91]. This simple version has been practiced extensively but its limitation in scalability due to the high overhead is severe. Efforts made in Century 20 to find higher-performance versions exploiting further decentralization were extensive. However, the number of produced techniques which are practical and also yield to rigorous quantitative analyses of fault coverage, has been small. Examples of such techniques are:

The *periodic reception history broadcast* (PRHB) scheme [Kim93, Kop89] and the *time-triggered protocol* (TTP) scheme [Kop93] for RT network surveillance in bus-LAN based systems; and

The *supervisor-based network surveillance* (SNS) scheme [Kim99a] for use in point-to-point network based systems.

The important metrics in this area is the *detection latency bound* [Kim93].

3.3 Transaction

The scheme for transaction structuring was established by the database research community on the basis of the notion of atomicity and sphere of control formulated earlier [Dav73]. The basic transaction model is aimed for maintaining properties such as atomicity, consistency, isolation, and durability in spite of component failures [Gra94].

A transaction must end with either commitment of an update to the database in case of a successful execution or clean abortion in case of an execution failure. Once a transaction is aborted, the client which requested the transaction may repeat the request or take an alternative course of actions. This atomicity and clean abort rules must be observed regardless of whether the database is concentrated in one site or distributed over multiple sites.

Log-based schemes for efficient abort and commit and schemes for concurrent execution of multiple transactions have been well developed. Integrations of the transaction scheme with disk mirroring approaches and the use of RAIDs have also been used widely. The basic transaction technology has matured.

Various extensions of the basic transaction scheme have been studied [Gra94] but only the basic transaction scheme has been widely used in the practicing field.

3.4 Checkpointing and recovery line

Rollback-retry, also called *checkpointing-recovery*, was a technique developed in 1960's to increase the probability of successful completion of a sequential atomic real-time computation-segment [Cha72]. Checkpointing is an act of taking a snapshot of the state of a computation and saving the snapshot in a safe storage device. In the case of executing a database transaction, the same act of saving a snapshot is often called a save-point establishment.

In a system of interacting processes, each of which performs checkpointing at various execution-points in a manner independent of checkpointing by other processes, a rollback of a process could cause an avalanche of rollbacks of interacting processes. This phenomenon was called the *domino effect* in [Ran75]. Since then, numerous researchers have studied how to make interacting processes to establish checkpoints in a coordinated manner so that a domino effect or a cyclic rollback propagation may be prevented [Ran95,Ssu99]. A *recovery line* for a process, say P1, is a set of checkpoints, each belonging to a different process, which will not be

crossed by a rollback of any process caused by the failure of P1. The techniques for managing recovery lines were extensively studied for the past 25 years but it appears they have not yet met much acceptance by practitioners.

3.4 Replication

Replication of databases and processes was a subject of extensive studies in the past three decades. Replicated databases where every transaction is executed on the replica designed as the primary and a subsequent update command is sent to other replicas, represent the simplest type of replicated databases [Bha87]. It has also been the most popular approach although the falling costs of RAIDs have also produced incentives for reducing the degree of replication. Other types of replicated databases have also been studied extensively but they have not met wide acceptance by practitioners yet.

For replication of processes, the types of needs and the types of replication structures are more diverse. About six basic types of replication structures were established in Century 20 [Kim94a, Kim98]. Numerous other replication structures can be viewed as minor variations of these six basic structures. Here a combination of replicated processes and executing node facilities is called a *FT computing station*. The six types of FT computing stations are briefly summarized below.

Structure 1: *Comparing pair* and rollback

Two replicas of a process run on two different nodes in parallel and their results of replicated execution of a task are compared [Toy87]. If a mismatch occurs, then the replicas roll back and make a retry.

Structure 2: *Pair of Self-checking Processing nodes* (PSP)

Structure 2a: A pair of single-processor nodes with an application-independent fault detection software component

Two replicas of a process run on two different nodes in parallel. Computational results of task execution on each node are validated by both the lack of signals from the fault-detection hardware and OS in the system and the execution of a *common (i.e., application-independent) fault-detection software component*. A typical fault-detection software component considered here is one that does a consistency check on the data structures in OS. The *primary-shadow cooperation* scheme is used in that the shadow node delivers its output (i.e., task results) to the rest of the DC system only when the primary node cannot produce an output [Kim95].

Structure 2b: *Pair of Comparing Pairs* (PCP) with result comparison

As depicted in Figure 1, replicas of a process run on four different nodes, organized as two pairs each of which is in turn a comparing pair, in parallel. The primary-shadow cooperation scheme is used in that the shadow pair delivers its output to the rest of the distributed computer system only when the primary pair cannot produce an output due to a mismatch or crash.

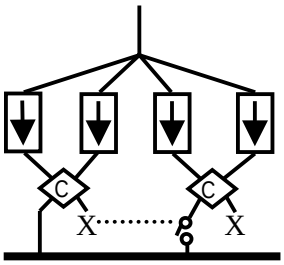

Figure 1. PCP station

Structure 3: *Distributed recovery block* (DRB) station

Figure 2 depicts this computing station. The DRB station [Kim94, Kim95] is essentially a PSP (Structure-2a or Structure-2b) station plus at least one of the following two optional software components: (1) an *acceptance test* which is an "application-dependent" fault-detection software component and (2) *alternate application algorithms*. As a part of the DRB development, a rigorous implementation model for the Structure-2a PSP station has been formulated [Kim95]. The acceptance and alternate algorithms, also called *try blocks*, can be incorporated by use of an elegant language construct, *recovery block* [Ran75, Ran95]. When two try blocks are used, the operating rule is that *the primary node tries to execute the primary try block whenever possible whereas the shadow node tries to execute the alternate try block.*

In Figure 2, primary node X uses try block *A* as the first try block initially, whereas shadow node Y uses try block *B* as the initial first try block. The two nodes pick up the same input data (by making the shadow to pick the data picked by the primary) and process the data in parallel by use of their current first try blocks, respectively. Both

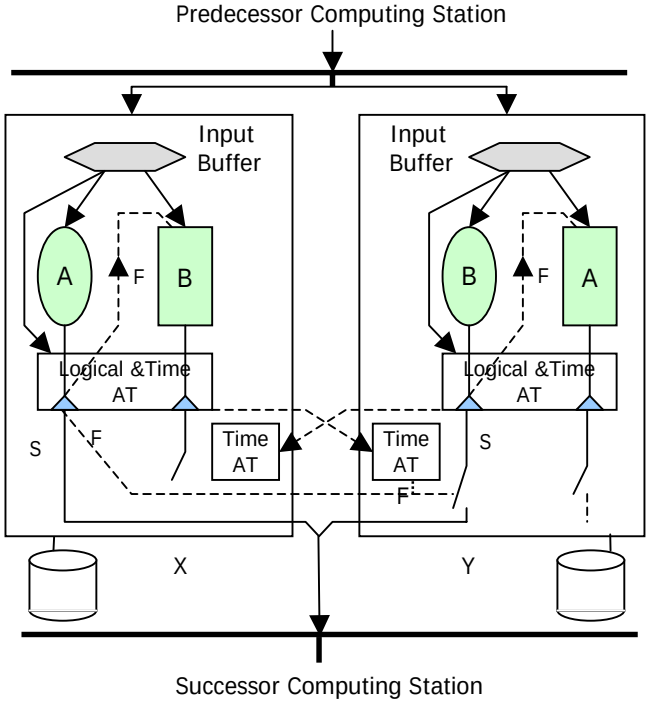

Figure 2. DRB station (adapted from [Kim96])

nodes then execute the same acceptance test to check if their computational results are reasonable. Timeout is a part of the acceptance test. If the primary node passes the acceptance test, it performs an output if its results without delay. This output action includes updating its local database and sending a data message to the successor computing station(s). If one of the two nodes fails before or at the acceptance test, the other node takes the role of the primary as soon as it discovers the failure of its partner and performs an output of its results. Meanwhile, the failed node attempts to become a healthy shadow node without disturbing the (new) primary node; it attempts to roll back and retry with its second try block to bring its application computation state including local database up-to-date.

Approaches for using more than two try blocks and/or more than two processing nodes in a DRB station and those for using the same node-pair to form multiple virtual DRB stations have also been developed [Kim95]. The DRB scheme has the following major useful characteristics:

a) *Forward recovery* can be accomplished in the same manner regardless of whether a node fails due to hardware faults or software faults;

b) The increase in the normal task turnaround time is minimal because the primary node does not wait for any status message from the shadow node;

c) The cost-effectiveness and the flexibility are high because

c1) a DRB computing station can operate with just two try blocks and two processing nodes and

c2) the two try blocks are not required to produce identical results and the second try block need not be as sophisticated as the first try block.

If software fault tolerance is not a goal, the application task designer is not required to provide alternate algorithms and providing acceptance tests is also optional. In other words, the DRB structure becomes PSP Structure 2a. Considerable research has been performed on how to maintain consistency between the primary and the shadow nodes [Kim96].

Structure 4: *Voting triple modular redundancy* (TMR) station (more generally, Voting N-modular redundancy station)

Three replicas of a process run on three different nodes, respectively, and they take a vote with their execution results [Toy87]. When there is discrepancy, the result with a majority vote is used.

Structure 5: *N-Version programming* (NVP) station

This station is essentially a Structure-4 station plus 3 or more different application algorithms for each application task [Avi88]. While the voting logic is application-independent, the scheme requires designing multiple versions expected to generate *truly identical computation results*, which could be a restriction in cases where complexity of a task logic is high.

3.5 Software fault tolerance

Software faults are essentially *design faults*. Software fault tolerance has been studied by a relatively small segment of the research community over more than 25 years. It is fair to say that this technology has so far been practiced only by a tiny segment of the industry in a limited form.

4. Issues insufficiently resolved

4.1 Quantitative treatment

FT in DC systems is realized basically via allocating resources for redundant computation. In most systems, resource allocation cannot be done arbitrarily or carelessly. Needless to say, effective, let alone optimal, resource allocation is not possible in the absence of *quantitative characterizations of FT schemes*. Yet the research efforts made by the FT research community in Century 20 for such characterization are grossly inadequate.

In the application environments where clean abort is the recovery goal and high-availability servers are desirable, the most important metrics are:

(1) *Fault types and rates covered*,
(2) The extra hardware costs, and
(3) The extra time costs including the *overhead* for enabling fault detection, the *abortion time*, and the *server-down time*.

On the other hand, in the application environments where RT recovery is desirable, the most important metrics are:

(1) Fault types and rates covered, and
(2) *Recovery time bound* which is the maximum difference between a normal execution time for a task and the time for a task execution involving fault detection and recovery events.

In some applications such as space-borne applications, extra hardware costs are also an equally important metric.

One can say that FT approaches not yielding to easy quantitative analyses are *unsafe to use*. Using such approaches is a blind exercise of an art. In this author's opinion, the quantitative treatment needs to be emphasized the most by the FT DC research community in this new decade.

One can also see that the software reliability problem is fundamentally one of obtaining *analyzable software*. Ideally, analyzable node OS, analyzable middleware, and analyzable application software must be used to realize reliable DC systems. *Adding FT capabilities cannot be an excuse for violating this law*.

Fair and unfair modeling of fault sources

A component of a DC system exhibits faulty behavior as a function of its *internal organization*, the *reliability of its subcomponents*, and its *operating*

environment. Since a non-trivial component can exhibit faulty behavior in an unlimited number of different ways, a manageable model of faulty behavior of a component is an absolute requirement for the designer who decided or considers to use the component in construction of a FT DC system. Such a model is called here a *faulty behavior model* of a component. A combination or an abstraction of a combination of the faulty behavior models of components used in composing a DC system is called a *fault source model* of the system.

Numerous fault source models were used and proposed by the research community in Century 20. Unfortunately, more often than not,

> *subjective and non-scientific reasoning was used in adopting and assessing the reasonableness of fault source models.*

This often led to the lack of harmony, the lack of trust, and the lack of open-minded spirits among the researchers in the FT DC field.

A good fault source model must be a characterization of all "non-negligible" patterns of fault occurrences. Such a model must be based on good faulty behavior models of the components used in the system.

In the literature dealing with the analysis of FT DC algorithms, replaceable components have been most often modeled as units of one of the two extreme types.

(1) One extreme model, which is at the simplest end in the spectrum of conceivable faulty behavior models, is the *fail-silent unit* (FSU) model. An FSU can exhibit only absence of an explicit output upon occurrence of any internal fault. No erroneous values are explicitly sent out from such units.

(2) The model at the other extreme end is the *malicious unit* (MaU) model, also called the *Byzantine unit* model. A malicious unit is capable of not only sending out erroneous values but also sending out sequences of values as if they were carefully manufactured to cause troubles to monitoring and diagnosing units.

The following limitations of the two extreme modeling approaches were pointed out in [Kim94b]:

(1) The FSU model is an idealistic component model which should be taken as a "design goal" for each replaceable component. So far the main-stream business computing industry has generally relied on this faulty behavior model. However, it is also too simplistic a model of components for use in designing and evaluating some complex systems, even if absence of software faults can be assured before the modeling step. It is dangerous to rely completely on this model in constructing future complex safety-critical application systems;

(2) The MaU model (or at least all the versions appeared in literature) appears to have fundamental flaws. The main flaw is in that the probability of malicious behavior occurring in a real system is much smaller than the probability of one of other assumptions adopted in

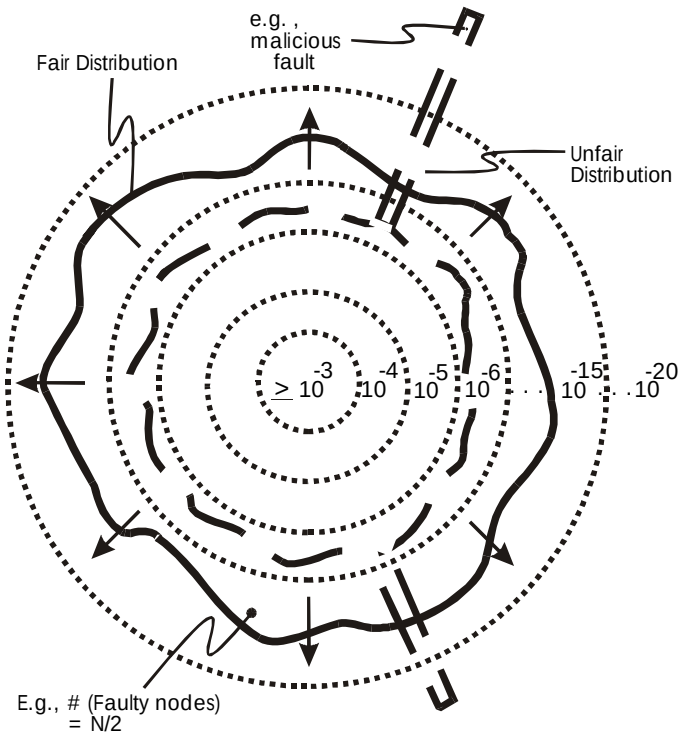

Figure 3. Fair distribution of concerns
(adapted from [Kim94b]

conjunction with the MaU model being violated. Typically an MaU model is accompanied by an assumption such as "the number nodes that can fail is limited to one third of the total number of nodes in the system". It is an "unfair" model in that it tends to draw attention to events of negligible occurrence probabilities while taking attention away from events of higher occurrence probabilities.

Therefore, an idealistic direction for advancing the state of the art in this important area is to develop a systematic method for *fair distribution of concerns over possible occurrences of anomalous events* during system design and validation. This author believes that the most promising direction is to lay out possible occurrences of all types of anomalous events in the space of occurrence probabilities and choosing "in a fair manner" the subset of the possible events to deal with. Figure 3 depicts such a view. The space of occurrence probabilities can be divided into concentric rings. Any event which has the occurrence probability of at least 10^{-3} is positioned within the innermost ring. Events which have occurrence probabilities between 10^{-4} and 10^{-3} must be positioned in the space between the innermost ring and the second innermost ring, etc.

Formulating a fault source model amounts to selecting a set of anomalous events to be concerned with. The way these selected events are spread in the space of occurrence probabilities determines the *fairness* or *unfairness* of the distribution of concerns. As illustrated

by a solid rugged circle in Figure 3, in a fairly distributed case, it should be possible to draw a circular boundary in the space of occurrence probabilities that meets the following requirements.

Fair distribution requirements:

(FDR1) All non-negligible events which the designer can envision are placed within the encircled space,

(FDR2) No events which are placed outside the circular boundary and thus to be ignored by designers and/or evaluators, have occurrence probabilities which are more than several magnitudes-of-order (e.g., 10 - 100 times) greater than the occurrence probability of any event placed within the encircled space.

Figure 3 also illustrates a case where the events to deal with are unfairly chosen. In this case, some events considered in conjunction with MaU models are selected but numerous events which have 10^{15} times greater occurrence probabilities than the probabilities of the selected malicious fault occurrences are ignored. Therefore, the FDR2 requirement is clearly not met.

Although fair distribution of concerns may sound like a natural thing to do, its effective practice requires considerable research in this new decade in areas which have been neglected up to now. Among other things, more statistical data on various types of component failures including correlation among different component failures and correlation between internal faults of components and observed failures, must be obtained.

Server-down time and recovery time bound

As mentioned in Section 2.1, the FT DC field is bouncing back up again. The main-stream computing industry is now becoming better motivated to explore high-availability server technologies and RT recovery technologies than just clean abort technologies. Therefore, the demands for quantitative treatment of server-down times and recovery time bounds are now growing. Since research conducted in Century 20 on such quantitative treatment was grossly inadequate, this represents a large important space for the FT DC research community to invade in this new decade.

4.2 RT FT DC (Real-Time Fault-Tolerant Distributed Computing)

In RT FT DC systems, fault detection and recovery actions should ideally be executed such that intended output actions of RT computations always take place on time in spite of fault occurrences. When it is not feasible, the fault tolerance actions which lead to the least damages to the application missions / users must be attempted.

In order to be practically useful in RT DC systems, a fault detection technique must at least yield a tightly bounded *detection latency* and a recovery technique must at least yield a tightly bounded *recovery time*. Only a tiny fraction of the research conducted in Century 20 dealt with rigorous analyses of detection latency bounds and recovery time bounds.

Even in non-RT application systems such as many Web server applications, detection latency and recovery time are becoming important performance metrics since they are major contributors to the server-down time. However, in such environments, average server-down times are more relevant than tight bounds on the server-down time are.

In fact, there are signs that even the major vendors of OS and communication infrastructure are gradually stepping up their efforts in making the timing behavior of their products more predictable. The justifications for keeping their products to continue to show wildly unpredictable timing behavior are losing strength. This will make the boundary between FT DC approaches used in non-RT application systems and those used in RT application systems to become blurred somewhat.

Currently the main challenge in development of the RT FT DC technology appears to be the *integration*. The state of the art has reached the point where promising component techniques which are effective in achieving specialized fault tolerance capabilities of subsystems when used in isolation, are available but their cost-effective integration remains insufficiently done. There is of course a large room for research to mature and further enhance these component techniques. However, a more urgent and important issue from the viewpoint of balanced technology development is cost-effective integration of such component techniques. There are two major types of integration that need to be developed.

RT FT computing stations + Network surveillance and reconfiguration (NSR):

The six fundamental types of replication structures that incorporate some fault detection capabilities were reviewed in Section 3.4. These or variations of these can be used to construct RT FT computing stations each dedicated to execution of one or a few types of RT tasks. Also, limited research conducted on network surveillance techniques was discussed in Section 3.2. In order to construct RT FT DC systems in a cost-effective manner, both categories of techniques need to be mobilized.

In addition, techniques for fast reconfiguration, which includes functional amputation of faulty components and redistribution of tasks to existing, newly incorporated, and repaired nodes, need to be used. Nevertheless, the three categories of techniques remain insufficiently integrated. This is considered a timely subject for research in this new decade.

Fault detection and replication principles + Object-oriented (OO) RT DC structuring techniques:

Nearly all RT fault tolerance techniques practiced in Century 20 were coupled with the approaches for process structuring of RT DC software. On the other hand, one of the several cutting-edge technology movements initiated in 1990's in software engineering is the *RT OO*

programming movement [IEE00, ISO, Kim97b,Kim00b, Sch00, WOR]. In this author's view, the most important goal of that movement has been to instigate a quantum productivity jump in software engineering for RT DC application systems. The movement is still in its youthful stage and its impact has just started surfacing up. However, its great potential is now much more clearly and widely recognized than it was in mid-1990's.

Therefore, adapting the existing RT fault tolerance techniques for integration into the powerful RT OO DC structure is an important integration issue. Fault detection and network surveillance techniques require easy adaptation. However, adaptation of replication and recovery techniques is a challenging research issue as witnessed in the case of applying the *cooperating primary-shadow replication principle* of the PSP/DBR scheme to the high-level RT OO programming and execution scheme called the *time-triggered message-triggered object* (TMO) scheme [Kim97a, Kim00a].

Scalability

Another major issue in RT FT DC is *scalability*. The complexity of RT FT DC application systems started growing faster as we entered this new century. Wide area network infrastructure is also increasingly used in new applications. To cope with this trend, the research community needs to enhance the scalability of network surveillance techniques and recovery techniques.

Also, one fundamental approach in RT DC which only a tiny fraction of the computer science community has explored is the *time-based coordination of distributed actions* [Kop97]. This approach is greatly effective in handling many FT related problems, e.g., network surveillance, keeping state consistency among replicas, and RT FT multicasts [Kim99b], etc.

4.3 Reliable multicast

Group communication in RT DC systems has been a subject of research for almost two decades but it is not yet a mature technological field [Cha84, Kem99, Kop97, Moc99, Tac98]. The main challenge in establishing group communication protocols is to deal with possible fault occurrences. Conversely, multicasts in absence of the possibility of component failures are simple programming problems [Kim99b]. In systems based on point-to-point networks, a multicast is merely a finite sequence of point-to-point single message communications. In systems based on physical broadcast facilities, a multicast may become a single broadcast with a group ID in the message header field.

There have been some proposals for using group communication protocols, in particular, hypothetical reliable multicast mechanisms, as basic building-blocks for FT DC systems. The validity of this thesis cannot be established until practical reliable multicast mechanisms get established. Until then, this author feels that it is sensible to cast group communication protocols as distributed application programs supported by execution engines using established RT fault tolerance techniques. Such engines should be capable of effectively handling failures of low-level components such as processors, paths in communication / interconnection networks, processor-network interfaces, and OS components.

Some challenging issues in implementing RT FT multicasts are:

(1) To ensure that the sender and all receivers reach the same correct conclusion without excessive delay that all the receivers correctly received the message;

(2) To ensure that the sender and all healthy receivers reach the same correct conclusion without excessive delay that at least one receiver failed to receive the message and thus the multicast was cancelled;

(3) To ensure that all healthy receivers reach the same correct conclusion without excessive delay that the sender became permanently disabled before confirming the successful receiving of the message by every receiver and thus the multicast was cancelled;

One fundamental approach that makes meeting the above requirements relatively easy is to make every receiver to process the message at a certain time called the *official release time*, e.g., 10am, by which the multicast can definitely be completed [Kim99b]. Then it is possible that an acknowledgment-message from a certain receiver arrives at the sender but the sender later notifies the receiver about the cancellation of the multicast. The cancellation notice must arrive at the receiver before the official release time.

Therefore, one problem with the above conservative approach is the large delay in completing a multicast. This is because the delay should include the time needed for the sender or some other authority to notify the receivers which received the message already, of the cancellation of the multicast. The delay and the official release time that can be adopted are functions of the fault source model adopted. RT FT multicasts require much more research.

4.4 OO FT DC

In recent years the OO DC movement, e.g., CORBA movement [Sch00], Java-based DC movement, DCOM and SOAP movement by Microsoft, etc., has become a major technological movement and most of the major industrial forces have now joined in the movement. As mentioned in Section 4.2, even the RT OO DC branch of the movement has become a significant movement. Justifications for process-structured DC have started losing strength.

FT OO DC technology is expected to become an active R&D field soon if it has not already [WOR, Nar99]. The challenging research issue is exploitation of intra-object concurrency while enabling high-coverage FT computing such as RT recovery.

4.5 Software fault tolerance

This author has not heard about any significant progress in this technology area in the past five years. An effective method for validating the schemes aimed at handling design faults is still lacking [Kim95]. The main difficulty is in creation of realistic fault conditions. In fact, the research community has not yet fulfilled its mission of demonstrating the software fault tolerance capability in convincing application contexts. Some successful demonstrations of the capabilities for detecting symptoms of design faults at run time have taken place but they did not include any successful RT recovery actions other than stopping the system operation. Successful accomplishment of this mission will require long-term persistent research effort since *use of artificially injected faults will not be a fully valid approach*. The DC application system used also needs to be of considerable complexity since otherwise the convincing cases of software fault occurrences are not likely to be encountered.

5. Summary and cautious attitudes

The liveliness of the FT DC field is in an upward move at this opening juncture of Century 21. A basic technical foundation for FT DC was laid out in the last quarter of Century 20. It contains among others basic techniques for fault detection and network surveillance, transaction structuring and execution, checkpointing and rollback, and replicated processing and recovery.

Major holes in the established foundation that need to be closed up in this new century include:

Quantitative characterizations of FT DC techniques,
Enhancement of RT FT DC technologies, especially,
integration of RT FT computing station construction techniques and network surveillance and reconfiguration techniques, and
application of established fault detection and replication principles to OO RT DC structuring techniques, and
OO FT DC, and
Software fault tolerance.

FT is essentially a redundancy management game. Fundamental types of redundancy, even if we restrict our concerns to the types formulated in the contexts of electronic computing, were identified in 1950's or earlier [von56]. FT is not a young field. Therefore, it pays for the research community active in this new decade to become familiar with the technical foundation established in Century 20. In fact, as early as 14 years ago, the following statement was made by Brian Randell, who made important pioneering contributions to FT computation models, and his co-author, J. Dobson:

> "*As a profession, we seem to specialise in re-inventing the wheel, and in inventing jargon that, by accident or design, obscures the fact of re-invention.*" [Ran86]

Also, FT problems for which hardware solutions of moderate costs exist but pure software solutions are sought merely because if found, they can be more cost-effective under the current hardware economy, should be viewed as short-term research problems. Who knows when RAIDs and a few more CPUs will become so cheap that nobody hesitate to add them to an existing configuration ?

The author believes that future emphasis by the FT DC research community on quantitative treatment of design techniques and protocols and scientific assessment of fault source models will lead to accelerated advances in FT DC technologies. It will also hopefully lead to more efficient open-minded unemotional reasoning atmosphere, which will have better effect of encouraging young researchers to enter and stay in the field.

Acknowledgements: The research work reported here was supported in part by the US Defense Advanced Research Project Agency under Contract N66001-97-C-8516 monitored by SPAWAR, and in part by the NSF Next-Generation Software (NGS) Program under Grant 99-75053.

References

[Avi88] Avizienis, A., Lyu, M.R., and Schutz, W., "In Search of Effective Diversity: A Six-Language Study of Fault-Tolerant Flight Control Software.", *Proc. IEEE CS 18th Int'l Symp. on Fault-Tolerant Computing (FTCS-18)*, pp.15-22.

[Bha87] Bhargava, B. ed., '*Concurrency Control and Reliability in Distributed Systems*', North-Holland Pub. Co., 1987

[Cha72] Chandy, K.M., and Ramamoorthy, C.V., "Rollback and recovery strategies for computer programs ", *IEEE Trans. on Comp.*, June 1972, pp.546-556.

[Cha84] Chang, J. M. and Maxemchunk, N., "Realiable Broadcast Protocols", ACM Transactions on Computer Systems, Vol. 2, No. 3, pp. 251-273, Aug. 1984.

[Dav73] Davies, C.T., "Recovery Semantics for a DB/DC System", Proc. 1973 ACM Annual Conf., Atlanta, Aug. 1973, pp.136-141.

[Gra94] Gray, J., and Reuter, A., *'Transaction Processing: Concepts and Techniques'*, Morgan Kaufman Publishers, 1994.

[Hec91] Hecht, M., et al., "A Distributed Fault Tolerant Architecture for Nuclear Reactor and Other Critical Process Control Applications.", *Proc. IEEE CS 21st Int'l Symp. on Fault-Tolerant Computing (FTCS-21)*, June 1991, Montreal, pp.462-469.

[IEE00] *'A special issue of Computer (a magazine of IEEE Computer Society) on Object-oriented Real-time distributed Computing*', June 2000.

[ISO] *ISORC '98 (IEEE CS Int'l Symp. on Object-*

oriented Real-time distributed Computing) Series; 1st held in April 1998, Kyoto, Japan; 2nd in May 1999, St. Malo, France; 3rd in March 2000, Newport Beach, CA. Proceedings are available from IEEE CS Press.

[Kem99] Kemme, B., Pedone, F., Alonso, G., and Schiper, A., "Processing transactions over optimistic atomic broadcast protocols", *Proc. 19th Int'l Conf. on Distributed Computing Systems (ICDCS '99)*, June 1999, pp.424-431.

[Kim93] Kim, K.H. and Shokri, E.H., "Minimal-Delay Decentralized Maintenance of Processor-Group Membership in TDMA-Bus LAN Systems", *Proc. IEEE CS 13th Int'l Conf. on Distributed Computing Systems (ICDCS '93)*, Pittsburg, May 1993, pp. 410-419.

[Kim94a] Kim, K.H., "Action-Level Fault Tolerance", Ch. 17 in Sang H. Son, '*Advances in Real-Time Systems*', Prentice Hall, 1994, pp.415-434.

[Kim94b] Kim, K.H., "Fair Modeling of Fault-Tolerant Distributed Systems", *Computer Communications*, Vol.17, No.10, Oct. 1994, pp.699-707. (An earlier version appeared in *Proc. IEEE CS 4th FTDCS*, Lisbon, Sept. 1993, pp. 173-180.)

[Kim95] Kim, K.H., "The Distributed Recovery Block Scheme", Ch. 8 in Michael R. Lyu, ed., *'Software Fault Tolerance'*, 1995, pp. 189-209.

[Kim96] Kim, K.H., Bacellar, L., and Subbaraman, C., "Primary-Shadow Consistency Issues in the DRB Scheme and the Recovery Time Bound", *Proc. IEEE CS 7th Int'l Symp. on Software Reliability Engineering*, White Plains, NY, Oct. 1996, pp.319-329.

[Kim97a] Kim, K.H., and Subbaraman, C., "Fault-Tolerant Real-Time Objects", *Communications of the ACM*, January 1997, pp. 75-82.

[Kim97b] Kim, K.H., "Object Structures for Real-Time Systems and Simulators", *IEEE Computer*, Vol. 30, No.8, August 1997, pp. 62-70.

[Kim98] Kim, K.H., "ROAFTS: A Middleware Architecture for Real-time Object-oriented Adaptive Fault Tolerance Support", *Proc. HASE '98 (IEEE CS 1998 High-Assurance Systems Engineering Symp.)*, Washington, D.C., Nov. 1998, pp.50-57.

[Kim99a] Kim, K.H. and Subbaraman, C., "Dynamic Configuration Management in Reliable Distributed Real-Time Information Systems", *IEEE Trans. on Knowledge and Data Engr.*, Vol.11, Jan./Feb. 1999, pp.239-254.

[Kim99b] Kim, K.H., "Group Communication in Real-Time Computing Systems: Issues and Directions", *Proc. FTDCS '99 (7th IEEE Workshop on Future Trends of Distributed Computing Systems)*, Cape Town, South Africa, Dec. 1999, pp.252-258.

[Kim00a] Kim, K.H. and Subbaraman, C., "The PSTR/SNS Scheme for Real-Time Fault Tolerance via Active Object Replication and Network Surveillance", *IEEE Trans. on Knowledge and Data Engr.*, Vol.12, No.2, Mar./April 2000, pp.145-159.

[Kim00b] Kim, K.H., "APIs Enabling High-Level Real-Time Distributed Object Programming", *IEEE Computer*, June 2000, pp.72-80.

[Kop89] Kopetz, H., et al., "Fault-Tolerant Membership Service in a Synchronous Distributed Real-Time System.", *Proc. IFIP WG 10.4 Conf. on Dependable Computing for Critical Appl.*, Santa Barbara, Aug. 1989, pp.167-174.

[Kop93] Kopetz, H., and Gruensteidl, G., "TTP - A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems", *Proc. 23rd IEEE Int'l Symp. on Fault-Tolerant Computing (FTCS-23)*, Toulouse, France, 1993, pp. 524-533. A revised version appeared in *IEEE Computer*, vol. 27 (1), 1994, pp. 14-23.

[Kop97] Kopetz, H., '*Real-Time Systems*', Kluwer Academic Pub., 1997.

[Moc99] Mock, M., Edgar, N., Schemmer, S., "Efficient Reliable Real-Time Group Communication for Wireless Local Area Networks", *Proc. EDCC 1999*, pp.380-400.

[Nar99] Narasimhan, P., Moser, L. E., and Melliar-Smith, P. M., "Using Interceptors to Enhance CORBA," *IEEE Computer*, July 1999, pp. 62-68.

[Ran75] Randell, B., "System Structure for Software Fault Tolerance.", *IEEE Trans. on Software Engineering*, June 1975, pp.220-232.

[Ran86] Randell, B., and Dobson, J. E., "Reliability and Security Issues in Distributed Computing Systems", *Proc. IEEE CS Symp. on Reliable Distributed Systems (SRDS '86)*, Jan. 1986, pp. 113-118.

[Ran95] Randell, B. and Xu, J., "The Evolution of the Recovery Block Concept", Ch. 1 in M. R. Lyu ed., '*Software Fault Tolerance*', John Wiley & Sons, Chichester, England, 1995, pp.1-21.

[Sch00] Schmidt, D.C., and Kuhns, F., "An Overview of the Real-Time CORBA Specification", *IEEE Computer*, June 2000, pp.56-63.

[Ssu99] Ssu, K., Yao, B., and Fuchs, K., "An Adaptive Checkpointing Protocol to Bound Recovery Time with Message Logging", *Proc. SRDS '99*, Lausanne, Oct. '99.

[Tac98] Tachikawa, T., Higaki, H., and Takizawa, M., "Group Communication Protocols for Realtime Applications", Proc. 18th IEEE ICDCS, 1998, pp.40-47.

[Toy87] Toy, W.N., "Fault-Tolerant Computing.", A chapter in *Advances in Computers*, Vol. 26, Academic Press, 1987, pp.201-279.

[von56] von Neumann, J., "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components", *Automata Studies*, Princeton, 1956, pp.43-97.

[WOR] *WORDS (IEEE CS's Workshop on Object-oriented Real-time Dependable Systems) Series;* 1st held in Oct. '94, Dana Point; 2nd in Feb. 1996, Laguna Beach; 3rd in Feb. 1997, Newport Beach; 4th in Jan. 1999, Santa Barbara; 5th in Nov. 1999, Monterey. Proceedings are available from IEEE CS Press.

Proceedings

The 19th IEEE Symposium on Reliable Distributed Systems

Nürnberg, Germany
October 16 – 18, 2000

Sponsored by
IEEE Computer Society Technical Committee on Distributed Computing
IEEE Computer Society Technical Committee on Fault Tolerant Computing
FAU, Informatik 3, Erlangen, Germany

In cooperation with
CNUCE CNR, Pisa, Italy

Los Alamitos, California

Washington • Brussels • Tokyo