Secure Proactive Recovery – a Hardware Based Mission Assurance Scheme

Ruchika Mehresh[1], Shambhu J. Upadhyaya[1] and Kevin Kwiat[2]

[1]*Department of Computer Science and Engineering*
State University of New York at Buffalo, NY, USA
Email: *rmehresh@buffalo.edu;*
Email: *shambhu@buffalo.edu;*

[2]*Air Force Research Laboratory*
Rome, NY, USA
Email: *kwiatk@rl.af.mil;*

Abstract: Fault tolerance via redundancy or replication is contradictory to the notion of a limited trusted computing base. Thus, normal security techniques cannot be applied to fault-tolerant systems. As a result, a multi-phased approach is employed that includes fault/threat avoidance/prevention, detection and recovery. However, a determined adversary can still defeat system security by staging an attack on the recovery phase. This paper presents a hardware-based, proactive solution that can be built into any fault-tolerant, mission-critical system to secure the recovery phase. It also presents an evaluation that validates the feasibility and efficiency claims of this solution.

Keywords: Security, Fault tolerance, Mission assurance, Critical systems, Hardware

Introduction

Research in the past several decades has seen significant maturity in the field of fault tolerance. But, fault-tolerant systems still require multi-phased security due to the lack of a limited and strong trusted computing base. The first phase in this regard is avoidance/prevention, which consists of proactive measures to reduce the probability of any faults or attacks. This can be achieved via advanced design methodologies like encryption. The second phase, viz. detection that primarily consists of an intrusion detection system, attempts to detect faults and malicious attacks that occur despite any preventive measures. The final is the recovery phase that focuses on recuperating the system after the occurrence of an attack or fault. Generally, fault-tolerant systems rely on replication and redundancy for fault-masking and system recovery. These three layers of security provide a strong defense for fault-tolerant mission critical systems. Yet, if a determined adversary stages an attack on the recovery phase of an application, it is quite possible that the mission will fail due to lack of any further countermeasures. Therefore, these systems need the provisioning of another layer of defense to address attacks that may be brought on by malicious opponents during the recovery phase itself.

The *quiet invader* is another serious threat that this paper considers. Attacking the mission in its critical phase not only leaves the defender with less time to respond, but cancelling the mission at this late a stage is far more expensive than cancelling it at some earlier stage. In case where the defender is not left with enough time to respond to the attack, it can lead to major economic loss and even fatalities.

This paper proposes a framework for mission assured recovery using the concept of runtime node-to-node verification implementable at low-level hardware that is not accessible by the adversary. The rationale behind this approach is that if an adversary can compromise a node by gaining root privilege to user-space components, any solution developed in user space will not be effective (since such solutions may not remain secure and tamper-resistant). In the proposed scheme, entire verification process is carried out in a manner that is oblivious to the adversary, which gains the system an additional advantage. This paper also explores the potential of utilizing the test logic on processors (and hence the name 'hardware-based mission assurance scheme') for implementing secure proactive recovery paradigm. This choice makes the proposed solution extremely cost effective. In order to establish the proof-of-concept for this proposal, the prototype used is a simplified mission critical system that uses majority consensus for diagnosis and recovery. Finally, the security, usability and performance overhead for this scheme are analyzed using a multi-step simulation approach.

Related Work

Solutions proposed in the existing literature to address faults/attacks in fault-tolerant systems are designed to employ redundancy, replication and consensus protocols. They are able to tolerate the failure of up to *f* replicas. However, given enough time and resources, an attacker can compromise more than *f* replicas and subvert the system. A combination of reactive and proactive recovery approaches can be used to keep the number of compromised replicas under *f* at all times (Sousa et al. 2007). But, as the attacks become more complex, it becomes harder to detect any faulty or malicious behavior (Wagner and Soto 2002). Moreover, if one replica is compromised, the adversary holds the key to other replicas too. To counter this problem, researchers have proposed spatial diversity in software. Spatial diversity can slow down an adversary but eventually the compromise of all diverse replicas is possible. Therefore, it was further proposed to introduce time diversity along with the spatial diversity. Time diversity modifies the state of the recovered system (OS access passwords, open ports, authentication methods, etc.). This is to assure that an attacker is unable to exploit the same vulnerabilities that he had exploited before (Bessani et al. 2008).

Mission critical systems demand secure operation as well as mission survivability. This additional requirement calls for smart security solutions (Carvalho et al. 2011).

Threat model

An extensive threat model is presented here to analyze security in a wide range of scenarios. Assume that there are *n* replicas in a fault-tolerant, mission-critical application. It can tolerate the failure of up to *f* replicas during the entire mission.

Scenario 1: Attacks on Byzantine fault-tolerant protocols
Assume that no design diversity is introduced in the replicated system. During the mission lifetime, an adversary can easily compromise *f+1* identical replicas and bring the system down.

Scenario 2: Attacks on proactive recovery protocols
In proactive recovery, the whole system is rejuvenated periodically. However, the adversary becomes more and more knowledgeable and his attacks evolve with each succeeded/failed attempt. So, it is only a matter of time before he is able to compromise *f+1* replicas between

periodic rejuvenations. Furthermore, the compromised replicas can disrupt system's normal functioning in many ways like creating extra traffic. This can help delay the next round of recovery which gains adversary more time to compromise *f+1* replicas (Sousa et al. 2007). This is a classic case of attacking the recovery phase.

Scenario 3: Attacks on proactive-reactive recovery protocols
Proactive-reactive recovery solves several major problems, except that if the compromised node is recovered by restoring the same state that was previously attacked, the attacker already knows the vulnerabilities (Sousa et al. 2007). In this case, a persistent attacker may get faster with time, or may invoke many reactive recoveries exhausting system resources. Large number of recoveries also affects system's availability adversely. This is also an instance of attacking the recovery phase. Furthermore, arbitrary faults are very difficult to detect (Haeberlen et al. 2006).

Scenario 4: Attacks on proactive-reactive recovery with spatial diversity
Spatial diversity in replicas is proposed to be a relatively stronger security solution. It can be difficult and more time-consuming for the adversary to compromise *f+1* diverse replicas, but it is possible to compromise all of them eventually. This is especially true for long running applications. Also, most of the existing systems are not spatially diverse. Introducing spatial diversity into an existing system is expensive.

Time diversity has been suggested to complement spatial diversity so as to make it almost impossible to predict the new state of the system (Bessani et al. 2008). The complexity involved in implementing time diversity as a workable solution is very high because it has to deal with on-the-fly compatibility issues and much more. Besides, updating replicas and other communication protocols after each recovery will consume considerable time and resources. A decent workable solution employing space diversity still needs a lot of work (Banatre et al. 2007), so employing time diversity is a step planned too far into the future.

Scenario 5: The quiet invader
A quiet invader is an adversary who investigates a few selected nodes quietly and plays along with the system protocol to avoid getting caught. In this way he gains more time to understand the system. After gathering enough information, the adversary can design attacks for *f+1* replicas and launch all the attacks at once when he is ready or when the mission enters a critical stage. If these attacks are not detected or dealt with in time, the system fails. This is an evasive attack strategy for subverting the detection and recovery phases. Similar threat models have been discussed in literature previously (Todd et al. 2007, Del Carlo 2003).

Scenario 6: The physical access threat
Sometimes system nodes are deployed in an environment where physical access to them is a highly probable threat. For instance, in case of wireless sensor network deployment, sensor nodes are highly susceptible to physical capture. To prevent such attacks, any changes in the physical environment of a node must be captured. A reasonable solution may involve attaching motion sensors to each node. Readings from these motion sensors can be used to detect threats. In this case, the scheme proposed in this paper can be used to assure the mission.

System design

Assumptions

In this paper, a prototype (simplified and centralized fault-tolerant, mission-critical application) is used to describe and evaluate the proposed security scheme. No spatial or time diversity is assumed, though this scheme is expected to work with any kind of diversity.

Network can lose, duplicate or reorder messages but is immune to partitioning. The coordinator (central authority and trusted computing base) is responsible for periodic checkpointing in order to maintain a consistent global state. A stable storage at the coordinator holds recovery data through all the tolerated failures and their corresponding recoveries. Sequential and equidistant checkpointing is assumed (Elnozahy et al. 2002).

The replicas are assumed to be running on identical hardware platforms. Each node has advanced CPU (Central Processing Unit) and memory subsystems along with the test logic (in the form of design for testability (DFT) and built-in self-test (BIST)) that is generally used for manufacture test. Refer to Figure 1(a). All the chips comply with the IEEE 1149.1 JTAG (Joint Test Action Group) standard (Abramovici and Stroud 2001). Figure 1(b) elaborates the test logic and boundary scan cells corresponding to the assumed hardware.

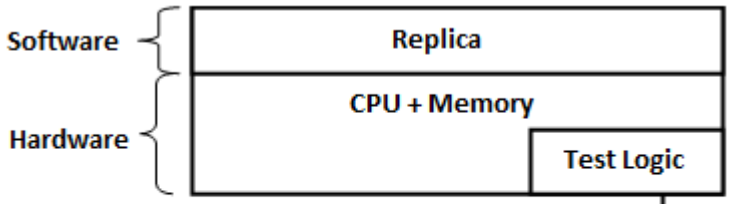

Figure 1(a): Replicated Hardware

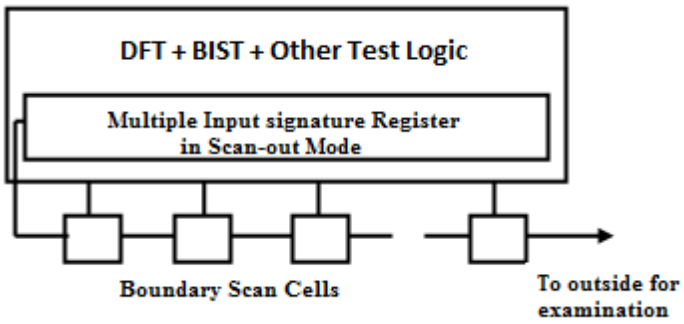

Figure 1(b): Capturing Signature

A software tripwire used to detect a variety of anomalies at the host is assumed to be running on each replica. By instrumenting the openly available tripwire source code (Hrivnak 2002), an ‘intrusion alert/alarm’ can be directed to a set of system registers (using low level coding).The triggered and latched hardware signature is read out by taking a snapshot of system registers using the ‘scan-out’ mode of the observation logic associated with DFT hardware. The bit pattern is brought out to the CPU ports using the IEEE 1149.1 JTAG instruction set in a tamper-resistant manner. Once it is brought out of the chip, it is securely

sent to the coordinator for verification and further action. This way, the system is able to surreptitiously diagnose all the adversary's actions.

Conceptual basics

This paper presents a simple and practical alternative to spatial/time diversity solutions in order to increase the resilience of a fault-tolerant system against benign faults and malicious attacks. In particular, this is to address the threat of a quiet invader (Scenario 5 in Section 3). An adversary needs to compromise *f+1* replicas out of the *n* correctly working replicas in order to affect the result of a majority consensus protocol and disrupt the mission. The key idea is to detect a system compromise by an adversary, who has taken over some replicas (or has gained sufficient information about them) but is playing along in order to gain more time. From the defender's point of view, if the system knows which of the *n* replicas have become untrustworthy, the mission can still succeed with the help of surviving healthy replicas. Aggressive attackers can be clearly and easily detected and thus their attacks can be recovered from at an earlier stage. Smart attackers try to minimize the risk of getting caught by compromising only the minimum number of replicas required in order to subvert the entire system. Thus, a smart defender should be able to detect attacks surreptitiously so as not to make the smart attacker aggressive. This especially holds true for the cases when a smart attacker has been hiding for long and the mission is nearing completion. At this stage, the priority is not to identify the attacker but to complete the mission securely.

The proposed scheme offers a passive detection and recovery in order to assure the smart adversary of its apparent success and to prevent him from becoming aggressive. At some later stage, when the adversary launches an attack to fail *f+1* replicas at once, the attack fails because those replicas have already been identified and ousted from the voting process without the knowledge of the attacker. In this solution, it is required that there should at least be two correctly working replicas to provide a duplex system for the mission to succeed. The advantage of this approach is that in the worst case when all replicas are compromised, the system will not deliver a result, rather than delivering a wrong one. This is a necessary condition for many safety-critical applications.

If an adversary compromises a replica by gaining root privilege to user-space components, one should note that any solution developed in user space cannot be expected to remain secure and tamper-resistant. Therefore, this paradigm achieves detection of node compromise through a verification scheme implementable in low-level hardware. Software or hardware-driven tripwires are used to help detect any ongoing suspicious activity and trigger a hardware signature that indicates the integrity status of a replica. This signature is generated without affecting the application layer, and hence the attacker remains oblivious of this activity. Also, a smart attacker is not likely to monitor the system thoroughly as that may lead to his own detection. This signature is then securely collected and sent to the coordinator that performs the necessary action.

Checkpointing

In the prototype application used here, the checkpointing module that affiliates to the coordinator establishes a consistent global checkpoint. It also carries out voting procedures that lead to anomaly detection due to faults, attacks or both.

Coordinator starts the checkpointing/voting process by broadcasting a request message to all the replicas, asking them to take checkpoints. It also initiates a local timer that runs out if it does not receive expected number of replies within a specified time frame. On receiving the request message, all replicas pause their respective executions and take a checkpoint. These checkpoints are then sent to the coordinator (over the network) through a secured channel using encryption. On receiving the expected number of checkpoints, coordinator compares them for consistency. If all checkpoints are consistent, it broadcasts a commit message that completes the two-phase checkpoint protocol. After receiving the commit message, all the replicas resume their respective executions. This is how the replicas run in lockstep. In case the timer runs out before the expected number of checkpoints are received at the coordinator, it sends out another request message. All replicas send their last locally stored checkpoints as a reply to this repeated request message. In this application, the allowed number of repeated request messages caused by a non-replying replica is limited to a maximum of three. If a replica causes three consecutive repeat transmissions of the request message and still does not reply, it is considered dead. A commit message is then sent by the coordinator to the rest of the replicas if their checkpoints are consistent. In case that the checkpoints are not consistent, the coordinator replies with a rollback message to all the replicas. This rollback message includes the last consistent checkpoint that was stored in the coordinator's stable storage. All replicas then return to the previous state of execution as defined by the rollback message. If a certain replica fails to deliver consistent checkpoint and causes more than three (or a threshold count) consecutive rollbacks, the fault is considered permanent and the replica is excluded from the system.

A hardware signature is generated at each replica and piggybacked on the periodic checkpoint being sent to the coordinator. This signature quantifies the integrity status of a replica since its last successful checkpoint. For simplicity, the values used are – all-0s (for an uncompromised replica) and all-1s (for a compromised replica). All replicas are equipped with a host-based intrusion detection sensor that is responsible for generating these signatures. If the coordinator finds any hardware signature to be all-1s, then the corresponding replica is blacklisted and any of its future results/checkpoints are ignored at the coordinator. However, the coordinator continues normal communication with the blacklisted replica in order to keep the attacker unaware of this discovery.

Finally, all results from each of the non-blacklisted replicas are voted upon by the coordinator for generating the final result.

Using built-in test logic for hardware signature generation and propagation

As described under assumptions, the system uses a software-driven trip-wire that monitors it continuously for a specified range of anomalies. Tripwire raises an alarm on anomaly detection by setting the value of a designated system register to all-1s (all-0s by default). This value then becomes the integrity status of the replica and is read out using scan-out mode of the test logic. It is then securely sent to the coordinator for verification.

Performance analysis

Most of the mission critical military applications that employ checkpointing or proactive security tend to be long running ones. For instance, a rocket launch countdown running for hours/days. Therefore, this performance analysis will focus on long running applications.

This scheme employs built-in hardware for implementing security. Also, the security-related notifications piggyback the checkpointing messages. Thus, security comes nearly free for systems that already use checkpointing for fault tolerance. However, many legacy systems that do not use any checkpointing will need to employ checkpointing before they can benefit from this scheme. In such cases, cost of checkpointing is also included in the cost of employing the security scheme. To cover all these possibilities, the following three cases are considered.

Case 1: This case includes all the mission critical legacy systems that do not employ any checkpointing or security protocols.

Case 2: This case examines mission critical systems that already employ checkpointing. We assume the absence of any failures or attacks in this case. Note that this is the worst case scenario for a system that shifts from Case 1 to Case 2. Such a system will employ checkpointing in absence of any faults/attacks. Hence, they end up paying the price without any benefit. However, if a system is already a Case2, the proposed security scheme comes nearly free for it, if they choose to employ it.

Case 3: The systems considered under Case 3 employ checkpointing and the proposed security scheme (hardware signature verification), both. This case considers the occurrence of failures and security-related attacks.

These three cases allow us to study the cost of adopting this security scheme in all possible scenarios.

Since the proposed system is composed of hardware and software subsystems, one standard simulation engine could not be used to simulate the entire application accurately. Therefore, results obtained from individually simulating the software and hardware components are combined using the multi-step simulation approach (Mehresh et al. 2010).

Simplified system prototype development

Figure 2 presents a modular design of the mission-critical prototype application with *n* replicas. The coordinator is the core of this centralized, replicated system. It is responsible for performing voting operations on intermediate results, integrity signatures and checkpoints obtained from the replicas. Heartbeat manager broadcasts periodic ping messages to determine if the nodes are alive. Replicas are identical copies of the workload executing in parallel in lockstep.

Multi-step simulation approach

This paper uses multi-step simulation approach to evaluate system performance for the three cases defined above. This approach is required because there are currently no benchmarks for evaluating such complex systems. Multi-step simulation provides a combination of pilot system implementation and simulation to deliver more realistic and statistically accurate results.

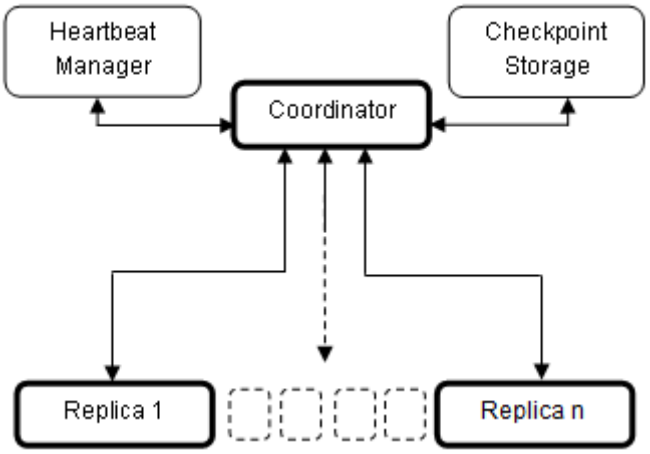

Figure 2: Overall system design

Researchers generally choose from a set of three methods to evaluate their systems. This choice depends on the system's current state of development. The first method, CMTC or (Continuous time Markov chains) is used when the system architecture has not been established yet (Geist and Trivedi 1990). When system design is available, the second method viz. simulation is generally used to model system's functional behavior. The third method is the most realistic one and involves conducting experimentation on a real-world system prototype. Relying solely on each of these methods has some associated advantages and disadvantages. Based on the system and its current state, choosing a wrong tool may increase the complexity and cost of evaluation with a possible decrease in result accuracy. For instance, if implementation is chosen as the evaluation tool, results will be more accurate and representative of the real world conditions (like hardware faults, network conditions, etc.) However, not only implementation is expensive to scale, but sometimes it may not even be possible or affordable to develop a prototype. In such cases, simulation is preferred because it is easier to develop and it scales rapidly at low cost. Simulation enables the study of feasibility, behavior and performance without the need of an actual system. It can also run at any speed compared to the real world and thus can be used to test a wide range of scenarios in lesser time than with a real system. However, accuracy is a major issue with simulation models. Designing a highly accurate and valid simulation model is not only difficult but sometimes costly in terms of time and resources.

Multi-step approach (Mehresh et al. 2010), involves a combination of theoretical analysis, pilot system implementation and simulation. This mix of analytical techniques can be optimized to obtain an evaluation procedure that minimizes its development effort and cost (resources and time), and maximizes its accuracy and efficiency.

This approach is a combination of three concepts: Modular decomposition, modular composability and parameterization (Meyer 1988). Modular decomposition consists of breaking down a problem into smaller elements. Modular composition involves production of elements that can be freely combined with each other to provide new functionality. Parameterization is the process of defining the necessary parameters for the problem.

The end result of employing this approach is the modular functional model of the system.

This model is hierarchical in terms of the level of detail/granularity. The analysis starts with the most abstract form of the model and works downwards toward a more detailed level/finer granularity. Each module is recursively decomposed if the candidate set of sub-modules satisfy the composability property. When the maximum level of decomposition is reached and the complexity of any further decomposition is high, then the current level is replaced with a black-box. This black-boxed level of detail is complex to model for simulation purposes but it is simple when described using stochastic variables (fault rate, bandwidth, etc.) statistically derived from experimentation. It can also be defined via theoretical analysis, like the use of queuing theory in case of scheduling.

Since the upper (coarsely granular) levels of any functional model are logically simpler, they are less prone to design errors. Hence, simulation model starts its construction from the very top. It then develops downwards to finer levels of granularity, adding more complexity and detail to the design. However, for system prototype development or theoretical analysis, the lower level of detail is easier to handle. For instance, a prototype can deliver data about the network traffic without much work, but consider designing the various factors that affect the flow of traffic for a simulation model. Hence, the prototype development moves upwards towards a coarser level of granularity. Adding more functionality to a prototype (in moving upwards) increases the cost of evaluation. A designer can make these two progressions to meet in the middle where the accuracy, simplicity and efficiency of the evaluation can be maximized. Refer to Figure 3.

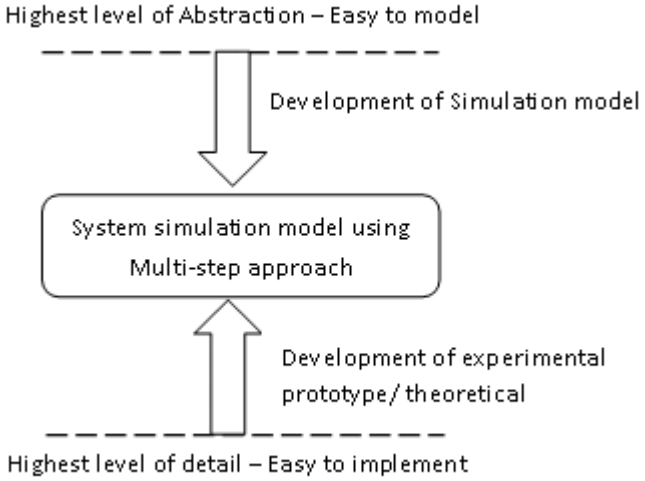

Figure 3: Development process of simulation model using multi-step approach

Three main tools are primarily used in the multi-step evaluation of this prototype: Java implementation based on Chameleon (Kalbarczyk et al. 1999), Arena simulation (Rockwell Automation 2000) and Cadence simulation. Arena simulation is discrete event and is used at the highest level of abstraction. This choice is also beneficial in working with long running mission critical applications. Conducting real-time experiments for long-running applications is not efficient and is extremely time consuming. The lower levels of abstraction that become too complex to model are black-boxed and parameterized using the Java implementation. Please note that Java implementation does not implement all the functionalities. It only implements those functionalities that generate data required to parameterize the black boxes. This java implementation consists of socket programming across a network of 100 Mbps bandwidth. The experiments are conducted on Windows platform with an Intel Core Duo 2 GHz processor and 2 GB RAM. Cadence simulation is primarily used for the feasibility

study of the proposed hardware scheme. To verify the precision of the employed simulators, test cases were developed and deployed for the known cases of operation.

This prototype (Java implementation) accepts workloads from a user and executes them in a fault-tolerant environment. Java SciMark 2.0 workloads used in these experiments are: Fast Fourier Transform (FFT), Jacobi Successive Over-relaxation (SOR), Sparse Matrix multiplication (Sparse) and Dense LU matrix Factorization (LU). The standard large data sets (Miller and Pozo 2004) are used.

Data generated by the experiments with Java implementation is collected and fitted probability distributions are obtained using Arena's input data analyzer. These distributions define the stochastic parameters for the black-boxes of the model. Once the model is complete, it is simulated in Arena.

Feasibility of the hardware component of this system (as described under assumptions) is examined as follows. The integrity signature of a replica is stored in the flip flops of the boundary scan chain around a processor. This part of our simulation is centered on a boundary scan inserted DLX processor (Patterson and Hennessy 1994). Verilog code for the boundary scan inserted DLX processor is elaborated in cadence RTL compiler. To load the signature into these scan cells a multiplexer is inserted before each cell, which has one of the inputs as test data input (TDI) and the other from the 32-bit signature vector. Depending on the select line either the test data or the signature is latched into the flip flops of the scan cells. To read the signature out, bits are serially shifted from the flip flops onto the output bus.

Rationale for using multi-step simulation

Simulation can only approximate the behavior of a real system. In a real system, the components (memory, processor speed, network bandwidth, etc.) have complex inputs, interconnections and dependencies that are not always easy to model. In addition to this, two similar components, such as two processors, can have different behaviors even if they are modeled the same for the purpose of simulation. These factors introduce disparity between the results obtained from simulation and the results from experimentation. To reduce this disparity, there exist many general-purpose simulation tools that allow the designing of stochastic system models. Stochastic parameters/variables can take into account a lot of unpredictable real-world factors. However, this approach presents the challenge of specifying the stochastic system parameters and variables, like probability distributions, seeds, etc. Mostly, values for defining system parameters and variables are taken from prior projects, sometimes without proper justification or verification, or are simply assumed.

Differences between results from simulation and experimentation can be ignored if only approximate comparisons are required (like observing a linear or exponential relationship between the input and output quantities). However, if the objective is to obtain values as close to the real-world experimentation as possible (as required in this case because there are no existing results to compare), then the need is to realistically parameterize the simulation model.

Mostly researchers validate their simulation design by comparing their simulation results with the results obtained from the system implementation. In many cases, the actual system may not exist and hence it is not possible to validate a simulation model. Hence, the need is to

simplify the simulation model, so it can be easily verified for the logic. Adding excessive details to a model makes it more complex to understand and also makes it prone to design errors. For instance, in a network application, an attempt to design the various time-variant factors that affect its performance will not only be impossible to precisely model, but will also increase the probability of design errors. So, simulation model designers generally make simplifying assumptions like the availability of a 100Mbps network bandwidth at all times. However, any application rarely gets to use the entire bandwidth. Hence, the execution time obtained from a simulation of a network application is much more optimistic than in a real world implementation. Designers generally go to a specific level of granularity in the simulation design and then start making assumptions beyond that level. Multi-step approach tries to realistically estimate these 'assumed' values. This provides statistically accurate results along with a much simpler simulation model that is less prone to design errors.

Another reason for proposing multi-step approach is to avoid long or unbounded execution times of real-world experimentations. Sometimes there is a system prototype available but the runtime is directly proportional to some parameter/variable, for instance, the workload size. In this case, if large workloads are fed and one application run takes days, it becomes very inefficient to experiment with a large number of design alternatives. So, simulation is a better solution for this problem. However, as stated before, simulation has to be realistic.

Modeling system using the multi-step approach

Java is chosen for the prototype implementation because of its easy-to-use API for programming socket communication and the level of author's familiarity with it. For simulation purposes, discrete event simulation is chosen. Discrete event simulation is generally of three types: event-oriented, activity-oriented and process-oriented. The system is such that process-oriented approach turns out to be the most convenient and accurate one. Workload can be defined as an entity with attributes like size, arrival time, checkpoint rate, etc. The various stages of processing like network, replica execution, heartbeat management, etc. are modeled as separate processes. There are a variety of tools like CSIM, JavaSim, and ARENA that can be used to execute this simulation model. However, Arena is chosen for this demonstration since it has a user friendly drag-and-drop interface (Rockwell Automation 2000).

At highest level of abstraction, three main modules are considered- Network, Coordinator and Replica. Refer to Figure 4. Now each module is inspected to see if it can be further decomposed into sub-modules. To verify if a new level of hierarchy can be defined, the potential sub-modules are investigated for the following two properties:

i) *Composability:* The functionalities of the candidate set of sub-modules can be composed to provide the functionalities of its parent module.

ii) *Sufficiency:* The functionalities of the candidate set of sub-modules collectively describe the entire set of functionalities of its parent module.

The first module 'Coordinator' has three sub-modules by design. These three sub-modules collectively describe the entire set of functionalities provided by the coordinator. Therefore, these three sub-modules are composedly sufficient to describe the coordinator. Hence, one additional level of the hierarchy is added for the coordinator module. The second module represents network that is unpredictable and is complicated to decompose further.

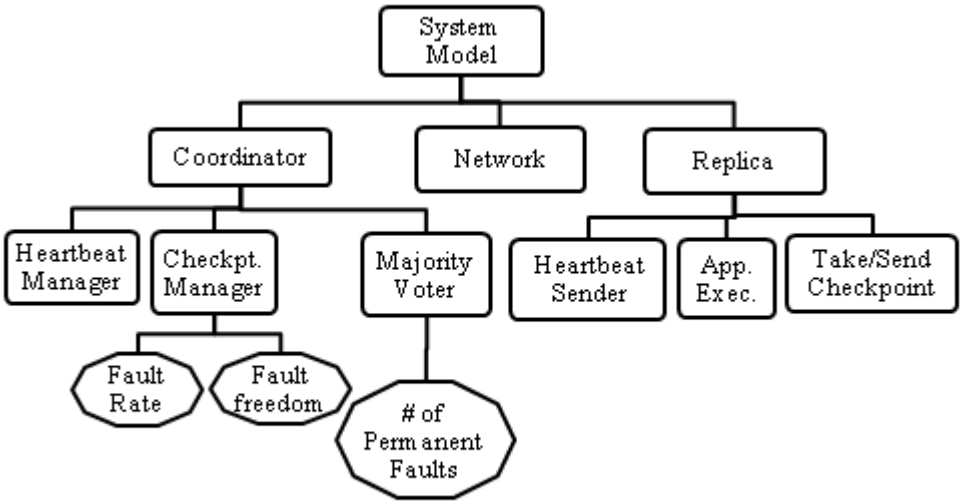

Figure 4: System model hierarchy of height 4 using the multi-step approach

Parameterization

Data recorded from several prototype runs is converted into probability distributions with the help of data analysis tools such as Minitab, Arena Input Analyzer, etc. Arena's input analyzer is used for data analysis and Arena student version for simulation of this prototype. Arena input analyzer fits the best possible probability distribution to the data. Various tests (like Chi-square test) can be conducted using these tools to find out how well the selected distribution fits the data.

Results

The prototype is analyzed for the three cases described earlier. In order to evaluate its performance in the worst case, checkpointing overhead should be maximum. Hence, sequential checkpointing is chosen (Elnozahy et al. 2002).

For the following analysis (unless mentioned), checkpoint interval is assumed to be 1 hour. Table 1 presents the execution times for the four Scimark workloads. The values from Table 1 are plotted in Figure 5 on a logarithmic scale. It can be seen that the execution time overhead increases a little when the system shifts from Case 1 to Case 2 (that is, employing the proposed scheme as a preventive measure). However, the execution time overhead increases rapidly when the system moves from Case 2 and Case 3. The execution overhead will only increase substantially if there are too many faults/attacks present, in which case it would be worth the fault tolerance and security that comes along. As can be seen from the values in Table 1, an application that runs for 13.6562 hours for Case 1 will incur an execution time overhead of only 13.49 minutes in moving to Case 2.

Figure 6 shows the percentage increase in execution times of various workloads when the system upgrades from a lower case to a higher one. It is assumed that these executions do not have any interactions (inputs/outputs) with the external environment. The percentage increase in execution time is only around 1.6% for all the workloads when system upgrades from Case

1 to Case 2. The overhead for an upgrade from Case 1 to Case 3 (with mean time to fault, M =10) is around 9%. These percentages indicate acceptable overheads.

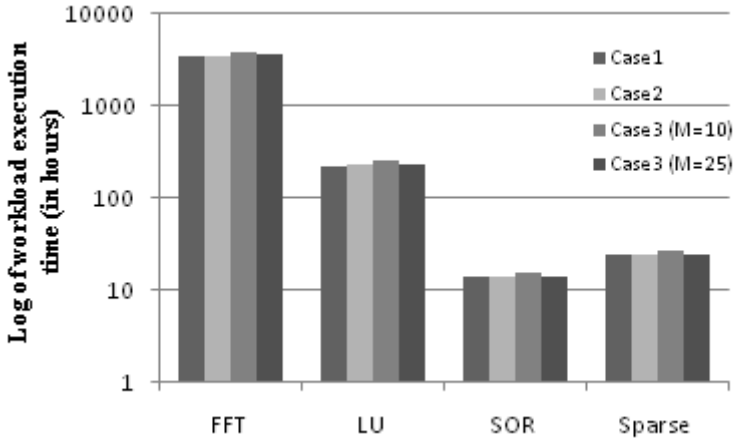

Figure 5: Execution times for Scimark workloads across three cases, on a logarithmic scale

Table 1: Execution Times (in hours) for the Scimark workloads across three cases

	FFT	LU	SOR	Sparse
Case 1	3421.09	222.69	13.6562	23.9479
Case 2	3477.46	226.36	13.8811	24.3426
Case 3 (M=10)	3824.63	249.08	15.2026	26.7313
Case 3 (M=25)	3593.39	233.83	13.8811	24.3426

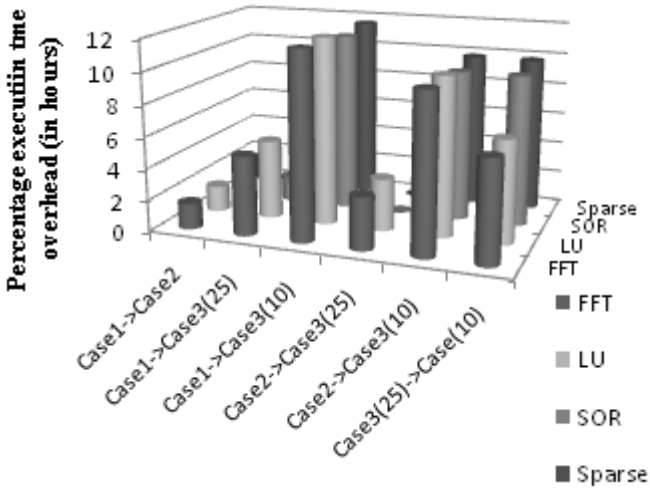

Figure 6: Percentage execution time overheads incurred by the Scimark workloads while shifting between cases

As Table 1 shows, for a checkpoint interval of 1 hour and M =10, the workload LU executes for approximately 10 days. Figure 7 shows the effect of increasing checkpoint interval for workload LU for different values of M ranging from 5 to 25. The optimal checkpoint interval values (and the corresponding execution times) for the graph plots in Figure 7 are provided in Table 2.

Please note that the multi-step simulation is used here and parameters for its model are derived from experimentation. Therefore, these results do not just represent the data trends but are also close to the statistically expected real-world values.

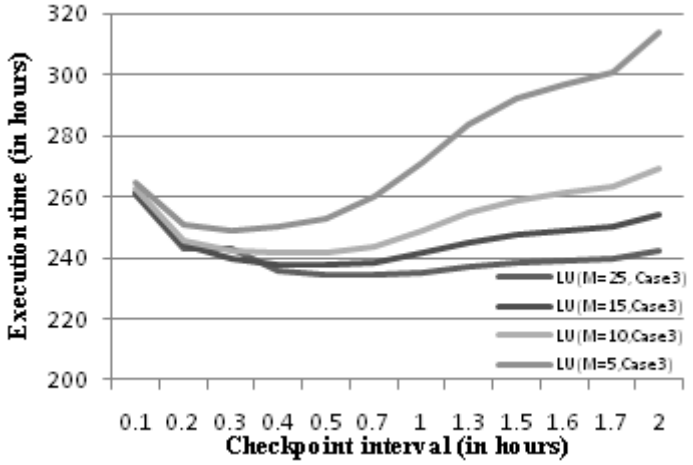

Figure 7: Effect of checkpoint interval on workload execution times at different values of M

Table 2: Approximate optimal checkpoint interval values and their corresponding workload execution times for LU (Case 3) at different values of M

	M=5	M=10	M=15	M=25
Optimal Checkpoint Interval (hours)	0.3	0.5	0.65	0.95
Execution Times(hours)	248.97	241.57	238.16	235.06

Conclusion

This paper proposes a hardware-based proactive solution to secure the recovery phase of fault-tolerant, mission-critical applications. A detailed threat model is developed to analyze the security provided by this solution. The most significant aspect of this research is its ability to deal with smart adversaries, give priority to mission assurance, and use redundant hardware for capturing integrity status of a replica outside the user space. Since this scheme is simple and has no visible application specific dependencies, its implementation has the potential to be application transparent.

For performance evaluation, a simplified mission critical application prototype is investigated using a multi-step simulation approach. The plan is to enhance the present centralized architecture into a distributed one for future research work.

The cost analysis defines various cases of application of the proposed security scheme (including its application to legacy systems with no fault tolerance). The performance evaluation showed promising results. Execution time overheads are observed to be small when faults are absent. As the fault rate increases, the overhead increases too. However, this additional overhead comes with strong fault tolerance and security. Overall, this solution is believed to provide strong security at low cost for mission critical applications.

Acknowledgment

This work was supported in part by ITT Grant No. 200821J. This paper has been approved for Public Release; Distribution Unlimited: 88ABW-2010-6094 dated 16 Nov 2010.

References

Abramovici, M., and Stroud, C.E. (2001) "BIST-based test and diagnosis of FPGA logic blocks," *IEEE Transactions on VLSI Systems*, volume 9, number 1, pages 159-172, February.

Banatre, M., Pataricza, A., Moorsel, A., Palanque, P., and Strigini, L. (2007) *From resilience-building to resilience-scaling technologies: Directions – ReSIST*, NoE Deliverable D13. DI/FCUL TR 07–28, Dep. Of Informatics, Univ. of Lisbon, November.

Bessani, A., Reiser, H.P., Sousa, P., Gashi, I., Stankovic, V., Distler, T., Kapitza, R., Daidone, A., and Obelheiro, R. (2008) "FOREVER: Fault/intrusiOn REmoVal through Evolution & Recovery," *Proceedings of the ACM Middleware'08 companion*, December.

Carvalho, Marco, Dasgupta, Dipankar, and Grimaila, Michael (2011) "Mission Resilience in Cloud Computing: A Biologically Inspired Approach," *6th International Conference on Information Warfare and Security*, pages 42-52, March.

Del Carlo, C. (2003) *Intrusion detection evasion*, SANS Institute InfoSec Reading Room, May.

Elnozahy, E.N., Alvisi, L., Wang, Y., and Johnson, D.B. (2002) "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys (CSUR)*, volume 34, number 3, pages 375-408, September.

Geist, R., and Trivedi, K. (1990) "Reliability Estimation of Fault-Tolerant Systems: Tools and Techniques," *Computer*, volume 23, number 7, pages 52-61, July.

Haeberlen, A., Kouznetsov, P., and Druschel, P. (2006) "The case for Byzantine fault detection," *Proceedings of the 2nd conference on Hot Topics in System Dependability*, volume 2, November.

Hrivnak, A. (2002) *Host Based Intrusion Detection: An Overview of Tripwire and Intruder Alert*, SANS Institute InfoSec Reading Room, January.

Kalbarczyk, Z., Iyer, R.K., Bagchi, S., and Whisnant, K. (1999) "Chameleon: a software infrastructure for adaptive fault tolerance," *IEEE Transactions on Parallel and Distributed Systems*, volume 10, number 6, pages 560-579, June.

Mehresh, R., Upadhyaya, S., and Kwiat, K. (2010) "A Multi-Step Simulation Approach Toward Fault Tolerant system Evaluation," *Third International Workshop on Dependable Network Computing and Mobile Systems*, October.

Meyer, B. *Object-Oriented Software Construction*, Prentice Hall, 1988.

Patterson, D., and Hennessy, J. (1994) *Computer Organization and Design: The Hardware/Software Interface*, Morgan Kaufmann.

Rockwell Automation (2000) Arena Software Simulation, URL: http://www.arenasimulation.com [Accessed: 17th July, 2011].

Miller, Bruce, and Pozo, Roldan (2004) Scimark 2.0 benchmark, URL: http://math.nist.gov/scimark2 [Accessed: 17th July, 2011].

Sousa, P., Bessani, A., Correia, M., Neves, N.F., and Verissimo, P. (2007) "Resilient intrusion tolerance through proactive and reactive recovery," *Proceedings of the 13th IEEE Pacific Rim Int. Symp. on Dependable Computing*, pages 373–380, December.

Todd, A.D., Raines, R.A., Baldwin, R.O., Mullins, B.E., and Rogers, S.K. (2007) "Alert Verification Evasion Through Server Response Forging," *Proceedings of the 10th International Symposium, RAID*, pages 256-275, September.

Wagner, D., and Soto, P. (2002) "Mimicry attacks on host-based intrusion detection systems," *Proceedings of the 9th ACM conference on Computer and communications security*, November.