JOURNAL OF INFORMATION SCIENCE AND ENGINEERING XX, XXX-XXX (20XX)

Fault-Tolerance Implementation in Typical Distributed Stream Processing Systems *

WUHONG CHEN[1] AND JICHIANG TSAI[1]
[1]*Department of Electrical Engineering*
National Chung Hsing University
Taichung, Taiwan, ROC

Typical training simulation systems adopt distributed network architecture designs composed of personal computers because of cost, extensibility, and maintenance considerations. In this design, the functions of the entire system are easily affected by failures or errors from any computer during operation. Thus, adopting appropriate fault-tolerance processing mechanisms to ensure that the normal operation and functions of the entire system can be maintained when irregularities occur in a subsystem computer is an important consideration for typical training simulation system design. Since firearms training simulation system operations involve the transmission and processing of substantial amounts of streaming data, these can be considered typical distributed stream processing systems. In this paper, we examined typical distributed stream processing fault-tolerance mechanism designs and technique. We applied this technique to a typical firearms training simulation system to increase the operation reliability and availability. We used the transparent checkpoint method to implement the fault-tolerance mechanism processing program. The results of single-machine fault-tolerance mechanism tests and multi-machine synchronized fault-tolerance mechanism tests indicate that the performance of the checkpoint establishment and rollback recovery time can satisfy the system operation requirements.

Keywords: distributed stream processing, fault-tolerance, checkpoint, rollback recovery, high availability

1. INTRODUCTION

Following the evolution of training simulator technology, functional design for firearms training simulation systems [1] has gradually expanded from single-gun simulation to multiple weapons (including various types of guns and mortars) and multiple targets to cover all user training needs. Typical firearms training simulation systems often undergo simultaneous operation by numerous users within short periods. Thus, the system must be able to ensure normal operations even when processing and transmitting substantial amounts of audio, video, and input and output data to avoid influencing the training. When firearms training simulation is performed, the system must act in concert with the training operation to execute various parameter computations as well as audiovisual and sensor signal processing. Coordination and cooperation between each participating subsystem host is necessary to execute each function specified by the system. Thus, during the computing process, the status of each host is correlated with that of the other hosts. During execution, if a host fails after encountering an unexpected situation, such as a power outage or a temporary hardware or software error, the operational functions of the entire system are affected. To avoid this scenario, incorporating fault-tolerance processing mechanisms into firearms training simulation systems must be considered to en-

sure the systems' continued normal operation.

Typical training simulation systems adopt distributed network architecture designs composed of personal computers because of cost, extensibility, and maintenance considerations. In this design, the functions of the entire system are easily affected by failures or errors from any computer during operation. Thus, adopting appropriate fault-tolerance processing mechanisms to ensure that the normal operation and functions of the entire system can be maintained when irregularities occur in a subsystem computer is an important consideration for typical training simulation system design. In fault-tolerance processing technique for traditional distributed computing, the rollback-recovery mechanism is simple and has lower system overheads. Thus, this method is currently widely used [2, 3]. However, firearms training simulation system operations involve the transmission and processing of substantial amounts of streaming data. Therefore, checkpoint mechanisms for traditional distributed systems cannot be used to resolve related problems effectively. These should be considered as typical distributed stream processing systems (SPSs). In recent years, enhancing the reliability and availability of distributed stream systems has become a popular research topic, with numerous results published [4, 5, 6, 7]. These results have mainly been implemented by storing checkpoints and recording and passing messages to complement each other. Although distributed stream systems generally also adopt the rollback-recovery mechanisms of distributed systems to increase system stability, the behavior of the system in continuing to send substantial amounts of data must be further considered to make corrections. Additionally, the transmission of messages between computers must be recorded for correct system rollback-recovery.

In this paper, we examine technique for fault-tolerance mechanisms in typical distributed SPSs. We applied the technique to a firearms training simulation system to prevent system downtime caused by sudden crashes or failures in sub-computers during operation. We used the transparent checkpoint method to implement the fault-tolerance mechanism processing program without modifying the source program of the firearms training simulation system. This checkpoint method can also recover complete data and is more capable of satisfying the demands of practical application. The experimental results of single-machine fault-tolerance mechanism tests and multi-machine synchronized fault-tolerance mechanism tests indicate that the performance of the checkpoint establishment and rollback recovery time can satisfy the system operation requirements. The major contribution of this paper is that the design and implementation method is also applicable to other distributed stream processing systems without affecting their original functions.

The rest of this paper is organized as follows: Section 2 reviews the related literature on high availability for SPSs. Section 3 analyzes and compares fault-tolerance mechanisms and the establishment of checkpoints. Section 4 describes functional design and implementation for the fault-tolerance mechanism. Section 5 describes experimental results of the fault-tolerance mechanism tests. Finally, Section 6 concludes our work.

2. RELATED WORKS

High availability (HA) for SPSs has been actively researched. In the papers [5], [6],

[8], and [9], the authors had studied active standby (AS) or passive standby (PS) using the Borealis stream-processing engine. Balazinska *et al.* [5] achieved a flexible trade-off between availability and consistency by introducing the tentative data concept; Hwang *et al.* [8] produced the highest availability by paying for multiple upstream copies to send data to multiple downstream copies; Brito *et al.* [9] allowed replicas to execute without coordination and produce consistent results; and Hwang *et al.* [6] examined optimal checkpoint scheduling and backup machine assignment when multiple subjobs require checkpointing and many state storage machines are available.

Replication is typically used for general fault-tolerance methods to protect against failures. Sebepou *et al.* [7] described three major replication mechanisms—state machines, process pairs, and rollback recovery—that are applicable for HA in distributed systems. In the state machine approach, the state of a processing node is replicated on some independent nodes. All replicas process data in parallel and coordinate with each other by sending the same input to all replicas in the same order. The process-pairs model is a related approach that coordinates replicas using a primary–secondary relationship. A primary node acts as a leader to forward its input to a secondary node that maintains order and operates in lock-step with the primary node. In rollback recovery, processing nodes periodically send snapshots or checkpoints of their state to other backup nodes or stable storage devices. During recovery time, the state is reconstructed from the most recent checkpoint, and upstream nodes replay logged input tuples to reach a pre-failure state. These replication mechanisms have been adapted for use in most distributed SPSs.

An active replica [5] is a typical example of the state machine approach adapted for stream-processing applications. This type of system replicates producer and consumer operators symmetrically in a stream dataflow graph. Each consumer replica receives tuples from a producer replica and switches to another functioning producer replica when its producer fails. Strict coordination is unnecessary because the replicas simultaneously processing the same input and forwarding the same output eventually maintain consistency. All operators preserve their output queues and trim them based on acknowledgments periodically sent by consumers. In case of failure, all upstream replica nodes begin serving their downstream nodes once failure is detected to minimize recovery time.

Hwang et al. [8] extended the active-replica approach, allowing all upstream replicas to send their output to all downstream replicas. Because the downstream nodes receive data from many upstream nodes, the input stream of any downstream node may be unordered or contain duplicate tuples. Enhancing the operators with extra non-blocking filters to eliminate duplicates based on periodically exchanged timestamp messages overcomes these complications.

AS is a fault-tolerance method that combines the active-replica and process-pair approaches [4]. In AS, secondary nodes work in parallel with primary nodes and receive tuples directly from upstream operators. Secondary nodes also log result tuples in their output queues, but they do not forward tuples to secondary downstream neighbors as in the active-replica approach. Drawbacks to this approach include output preservation because of the non-deterministic nature of operators and log bounding each secondary.

Passive-replica approaches use a rollback recovery (also known as a checkpoint-rollback) mechanism consisting of PS and upstream-backup [4][5]. In PS, the primary replica periodically produces state checkpoints and copies them to the backup replica. The state includes data located inside operators and input and output queues. The

secondary node acknowledges the received state with the primary upstream node to drop tuples from the primary upstream node output queue. In case of failure, the backup node takes over by loading the most recent checkpoint to its current state. A variant of PS that allows independent checkpointing of fragments (sub-graphs) of the entire query graph [6] reduces the latency introduced by checkpointing. However, checkpoint granularity is at the entire operator level, and stream processing freezes when storing a checkpoint fragment to remote server memory.

The upstream-backup [4] model was proposed for operators whose internal state depends on low inputs. The upstream nodes act as backups for the downstream nodes by logging tuples in their output queues until all downstream nodes process their tuples completely. The upstream log is trimmed periodically using acknowledgments sent by downstream primaries. In case of failure, the upstream primaries replay their logs, and the secondary nodes rebuild the missing state before serving other downstream nodes. Compared to PS, upstream backup requires longer recovery, but has lower runtime overheads.

Repantis et al. [10] first identified the design principles for an HA replica placement algorithm by accounting for the particular characteristics of stream-processing applications. HA is based on the concept that by replicating components and distributing them across different nodes, replica failure does not interrupt application execution because other replicas continue to provide the service. Repantis et al. [10] incorporated these principles into a decentralized replica placement protocol that maximizes availability, respects resource constraints, and makes performance-aware placement decisions.

Shiokawa et al. [11] proposed a new HA scheme called adaptive semi-active standby (A-SAS) that adaptively adjusts costs between bandwidth use and recovery time. The scheme uses batch-based scheme concepts and cost models to estimate current costs. A-SAS behaves as an upstream backup until the size of the output queue reaches the previously set batch size; all data in the output queue are then sent to the downstream secondary. If A-SAS is set to a smaller batch, A-SAS allows larger bandwidth usage and shorter recovery time. Setting larger batches results in smaller bandwidth usage and longer recovery time. A-SAS periodically optimizes batch size using the bandwidth cost model and recovery time cost model.

Zhang et al. [19] proposed a hybrid HA method that combines the advantages of AS and PS approaches. The system behaves similarly to PS during normal conditions and uses fewer computing and bandwidth resources. When a transient failure is detected, it quickly switches to AS mode by activating a pre-deployed secondary copy in suspension. Once the primary copy becomes responsive again (e.g., after a load spike ends), the system returns to PS mode. Thus, the system mostly incurs small overheads, but provides fast recovery during failures and unavailability.

3. FAULT TOLERANCE MECHANISMS AND CHECKPOINTING METHODS

3.1 Comparison of Fault-Tolerance Mechanisms

"Fault tolerance" refers to the use of appropriate processing mechanisms when a system encounters errors to prevent the system from crashing. Operation is maintained at a lower performance. Thus, it can also be called "graceful degradation."

The errors caused by fault-tolerance events can be generally divided into the following categories [12]:

1. Performance: hardware or software components cannot satisfy the user's demands.
2. Omission: components cannot execute the actions of a number of special instructions or commands.
3. Timing: components cannot execute the actions of instructions or commands at the correct time.
4. Crash: certain components crash with no response and cannot be repaired.
5. Fail-stop: when the software detects errors, it terminates the process or action.

Additionally, three situations can be distinguished based on the timing of the error. The first are permanent errors. When these occur, they damage the software components, causing permanent harm and preventing the program from continuing to run. Programs typically must be restarted. Crashes are an example of this. The second are temporary errors. These cause only temporary damage to the software. After a period, the software can continue operating normally. The third are periodic errors. Components are occasionally damaged when these errors occur. Errors of this type are resolved within a short period. For example, two types of software conflict with each other and errors occur when they are opened simultaneously. This can be avoided by simply closing one of the programs.

Fault-tolerance mechanism design can develop more complete system architectures in anticipation of system errors, enhancing the fault tolerance. However, the costs and time that must be expended are considerable. Fault-tolerance mechanisms can be divided into three levels, that is, hardware, software, and system fault tolerance [13]. Hardware fault tolerance, as the name suggests, involves preparing additional backup hardware, such as central processing units, memory, hard disks, or power supply units. However, hardware fault tolerance can provide only the most basic hardware backups; it cannot prevent users from carelessly or accidentally tampering with programs or stop errors caused by the actual system design. Therefore, software fault tolerance must also be considered.

The main mechanisms of software fault tolerance are checkpoint storage and rollback recovery since they are simple and have lower system overheads. A checkpoint is a snapshot of the entire state of the process at the moment it was taken. It represents all the information that we would need to restart the process from that point. The implementation of software fault tolerance is to develop a utility program to store checkpoints of target system regularly. When errors occur, this utility program is used for rollback recovery. The system described in this study is a type of software fault tolerance.

The final fault-tolerance method is called system fault tolerance. A complete system architecture that can automatically store program checkpoints, memory blocks, and memory ranges is built. This system can detect errors occurring in the applications itself. When errors occur, the system can also provide corresponding processing, thereby correcting the errors. Table 1 shows a comparison of the three fault-tolerance mechanisms.

Table 1. Comparison of Fault-Tolerance Mechanisms

Mechanism / Comparison	Hardware Fault-Tolerance	Software Fault-Tolerance	System Fault-Tolerance
Major technique	Hardware backup	Checkpoint storage Rollback recovery	Architecture with error detecting & correcting
Design complexity	Low	Medium	High
Time/cost expenditure	Low	Medium	High
Fault-tolerance Level	Low	Medium	High

Based on the functional features of firearms training simulation systems, we adopted *in-site passive standby* to implement the required fault-tolerance mechanism and substantially reduce hardware costs. This method is similar to the conventional passive standby method [4]. The primary host also stores checkpoints based on its working status (including input and output queue information and the internal state) on a regular basis. These checkpoints are stored directly on the local side. The use of additional secondary hosts is unnecessary. When the primary host fails, it restarts execution based on the checkpoint data most recently established. It then requests that the upstream host resend relevant information to recover the correct state before the error. This method is used considering the substantial increases in software and hardware stability currently exhibited by general computers. Crashes in the overall host rarely occur. Instead, problems are typically temporary shutdowns caused by excessive load from one or two processes within the system. Thus, preparing a secondary host to reach the fault tolerance goal is unnecessary. Specifically, when the number of system hosts increases, the costs for the necessary additional secondary hosts are considerable. Additionally, general computer operating performance has increased substantially compared to that of the past. Thus, regardless of whether checkpoint storage and rollback-recovery operations are required, this does not seriously affect the original tasks being executed.

3.2 The Establishment of Checkpoints

The application of checkpoints is primarily divided into two methods. The first is to establish a checkpoint library in advance to include checkpoint functions during program development. This is also called the nontransparent method. The second adopts a technique to intercept the application programming interface (API) to integrate or inject checkpoint computing capabilities directly into the program during execution. This is a transparent method [14, 15, 16]. Although the techniques adopted differ, the final goal of each is to be used for collecting and storing thread data in processes. This assists in the provision of correct messages when errors occur and facilitates additional recovery and correction.

Table 2 shows a comparison of the two checkpoint application methods. The table indicates that the library is a more efficient method. The only defect is that a source code is necessary for implementation. The injection or integration checkpoints do not require modification of the source program code. This method can also recover complete data and is more capable of satisfying the demands of practical application. However, because

injection adopts logical operations or computations for collecting data, a clear understanding of the program's operating state is required. If not, incomplete data is collected, substantially reducing the rollback recovery efficiency.

Table 2. Comparison of Transparent and Non-transparent methods

Method / App. conditions	Non-transparent (Checkpoint library)	Transparent (Checkpoint injection)
Necessity to modify source code	Yes	No
Implementation of the checkpoint operation time	Can be selected in the original process	Fixed time interval for computing
Collection of data reliability	According to the needs of source codes for control	Requiring adequate computing logic to achieve
Recoverable data	Part of the critical information	Complete information

Using injection checkpoints, or transparent checkpoints, does not intrude on users' programs. Users running program operations or computations or establishing checkpoint storage do not influence the program's source code. This method can be used for program error recovery, allowing the program to return to the position of the last stored checkpoint before continuing execution. Thus, a state with normal processes must be saved or preserved to enable recovery if an error occurs. The primary problem is effectively copying and returning the thread state without influencing the program operation status.

4. DESIGN AND IMPLEMENTATION

4.1 Functional Design and Allocation for the Fault-Tolerance Mechanism

The first step for the checkpoint computation program flow of the firearms training simulation system described in this study was to run the subsystem to be detected. Next, times were set for establishing checkpoints. Rollback recovery was performed when errors occurred in the system. The implementation method can be divided into multiple modules. Figure 1 shows a flowchart for each module. The basic process was as follows: First, each module was controlled by the central control module, such as checkpoint establishment time intervals and other information. Next, periodic checkpoint establishment was performed. The program can select establishment time periods or cycles itself while performing regular crash checking. If crashes do not occur, checkpoint establishment is performed again. The task execution repeats in this way. When the system crashes, the crash checking module detects and diagnoses any errors, which are passed to the rollback-recovery module for error recovery. Descriptions of the detailed functions of each module are provided below.

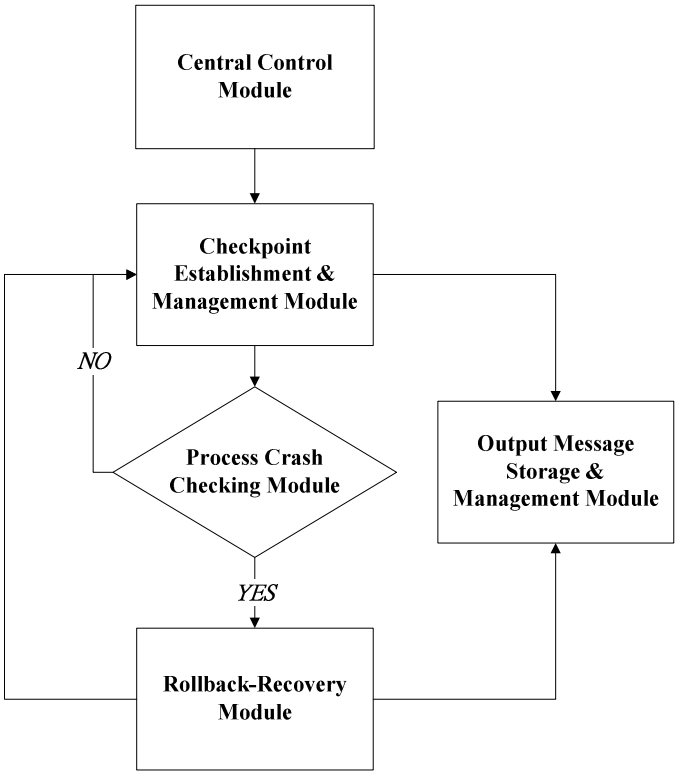

Fig.1. Flowchart for each module

4.1.1 Checkpoint Establishment and Management Module

Numerous types of transparent interception API techniques exist [16, 17]. The majority of these can be divided into user level and kernel level. Based on the feasibility consideration, we adopted one kind of the user level interception method to modify the external function import table of PE for this study. The main idea is to modify the program's flow or process, allowing it to jump to the interception function before returning to the original function. Thus, the original API function can still be implemented. If not, the interceptor loses its relevance. More specifically, the interceptor system API method adopts the interceptor dynamic linker library (DLL) method. That is, it uses the process external function import table (IMPORT) for function entry point redirection behavior. Therefore, in addition to considering the external DLL module for the process, the external function import table in the process itself must also be considered.

A number of functions related to `LoadLibrary()` are in the Win32 API. These can load or write in external DLLs during execution. As these are executed, new external modules are loaded. At this time, intercept actions must be performed on these modules, and redirection intercept repair actions must be performed on the external function import tables of the loaded modules. An additional problem encountered when satisfying the API interceptor is a special condition of the external function import table, that is, the

majority of general DLL import tables use function names as references for import functions. At this time, we must only compare string data from the import name table that corresponds to the function import table to determine whether it is the function that should be intercepted. Regarding certain DLLs, their import tables do not use function names for correspondence. Instead, they adopt the function ordinal of the import function in the function output table of the original DLL for correspondence. In this situation, the function to be intercepted has no name strings for comparison; instead, it has only the output function ordinal of the item in the original DLL. At this time, the header information of the DLL module must be found in the memory, and the output function table and output function name table for the module must be sought in this header information. Finally, the output function name must be found in the output function ordinal, and string comparison judgment must be performed with the name of the function to be intercepted.

To complete the periodic checkpoints, a timer must be set for dynamic applications. The time intervals can also be set based on the user operations. In this study, we used the Windows system timer. This is provided by the 8254 chip and `INT 08H` hardware interrupt in the PC. The method used was `CWnd::SetTimer`. This allows one message to be sent at fixed intervals within the program. After receiving the message, corresponding processing must be performed. The processing is implemented within the `CWnd::OnTimer` function. When performing rollback recovery, the timer must be stopped and returned to the system. At this time, the `CWnd::KillTimer` function must be employed.

4.1.2 Output Message Storage and Management Module

Considering the demands of firearms training simulation systems, we adopted the precise recovery method in this study. Therefore, the messages sent and received by hosts must be recorded to assist in repeating these actions after rollback-recovery. The programs in the Windows system transmit messages to programs on other hosts using the Winsock mechanism. This is also implemented by calling on specific APIs. Thus, the construction method for this module is identical to that for establishing checkpoints and the management module discussed previously. The interceptor API method is also employed for storing relevant messages. Here, the Detour tool supplied by Microsoft is adopted for process completion [18].

The main objectives are as follows:

1. In addition to using transparent methods for injection, must be capable of withdrawing entirely.
2. Does not require special permissions during operations performed at the user level.
3. Can arbitrarily choose APIs for interception, including DLLs for Kernel, User, and Winsock.
4. Must be able to establish connections with the central control module to transmit and receive control messages.

The required three implementation techniques comprise the following:

1. Uses the Detour Library to inject the interceptor program code in the running application.
2. The interceptor execution program calls on the DLL function.

3. Determines what type of processing should be performed for the intercepted function.

In this section, interception of only the application output data is necessary. Obsolete messages are disposed based on responses from the downstream host.

4.1.3 Process Crash Checking Module

Each host requires a process crash checking module to enable error and crashing detection. This module usually uses ping/timeout to detect whether any crashes or network errors have occurred in processes. In detail, this module uses the two system functions of `OpenProcess()` and `WaitForMultipleObjects()` to list information from all processes currently operating in the host environment to investigate process problems. First, all messages from the processes currently running in the system are stored. A process identification code is then used to seek comparison methods and locate all relevant information for the process monitoring the application. Additionally, a timer must also be set. After a fixed period, these messages are updated to determine the status of the processes again. If the monitored applications are in a stopped process state and unable to respond, the central control module is notified immediately. Subsequently, the rollback recovery mechanism is initiated, and the execution of the processes is restarted.

4.1.4 Rollback-Recovery Module

The application principle for checkpoint technique is to establish a checkpoint when programs execute up to a certain point. The state of the program's execution (for example, memory location, memory block size, thread content, and data in the register) is saved. If errors occur in the system, this information can be used to recover from the error and allow the progress to continue. Thus, rollback recovery is extremely important to the checkpoint application. This process can be roughly divided into the following steps:

1. Detection: errors can be detected in this stage.
2. Diagnosis: error causes and the damaged created by the error can be determined in this stage.
3. Isolate and Block: the process in which the error occurred is isolated to prevent the error from spreading.
4. Recovery: in this stage, the correct data stored in an earlier checkpoint are used to replace the data where the error occurred.

When the program executes to a checkpoint position, key data is first stored to establish a checkpoint. The program then continues running. Before reaching the next checkpoint establishment position, the system performs rollback-recovery if an error occurs. The correct information stored in an earlier checkpoint is used to cover or overlay the part where the error occurred. The program can then return to the state when the error had not occurred and continue execution. In other words, after completing checkpoint storage, the application can continue running. These steps are repeated each time a checkpoint is established. Additionally, checkpoint rollback recovery must be performed when errors occur. The data in the checkpoint file must be separately written back to the virtual address space and execution content of the application. This confirms that the application can return to the state attained after the establishment of the checkpoint.

When performing rollback recovery, the thread processes must be stopped temporarily to prevent new information from covering the recovered thread content.

4.1.5 Central Control Module

The most important task of the central control module is to coordinate the checkpoint establishment times for each subsystem. In this study, we used an efficient checkpoint storage method called the sweeping checkpoint method [19]. This method stores checkpoints rapidly and does not incur high additional costs. Checkpoint establishment using this method is driven primarily by the downstream hosts rather than a counter. The central control module receives an acknowledgment from the downstream hosts indicating that all information from a checkpoint has been stored successfully. The hosts can then remove the messages from their output buffers to reduce the data they must store themselves when establishing checkpoints. In other words, because the relevant output messages must also be stored in the checkpoint and the output buffer size can be decreased substantially following reduction processes, this is an excellent time for checkpoint establishment. All hosts in the system must use this method to establish checkpoints for storage. Because the most downstream hosts have no driving acknowledgments further downstream, they must rely on timers to drive checkpoint storage. That is, the downstream hosts send regular acknowledgments to the upstream hosts to trigger checkpoint establishment. Using this method, triggering travels upstream layer by layer; thus, this is known as the sweeping checkpoint method.

The advantage of the sweeping checkpoint method is that it can substantially reduce additional costs. Hosts first reduce messages stored in their output buffers when implementing checkpoint storage, which increases the checkpoint storage speed. Additionally, because this method does not require all hosts to stop temporarily for checkpoint storage, it provides excellent operational efficiency and speed.

4.2 Fault-Tolerance Mechanism Implementation

When designing fault-tolerance mechanisms for firearms training simulation systems, how to store checkpoint information and application messages from each subsystem should be considered carefully. Usable, correct information is sought from the subsystems for rollback recovery. These important messages are then stored. The main storage objects are memory addresses and relevant application contexts used by each subsystem. In this study, we developed a checkpoint application that can store checkpoints from each subsystem a synchronized manner. When errors occur in a system, the fault-tolerance mechanism can be used for rollback recovery of the entire system. This allows the system to return to a state with no errors and continue running.

The checkpoint computing program of the firearms training simulation system implemented in this study inputs process names from each subsystem to capture process information and begin checkpoint storage on the target processes. Figure 2 shows the program flowchart.

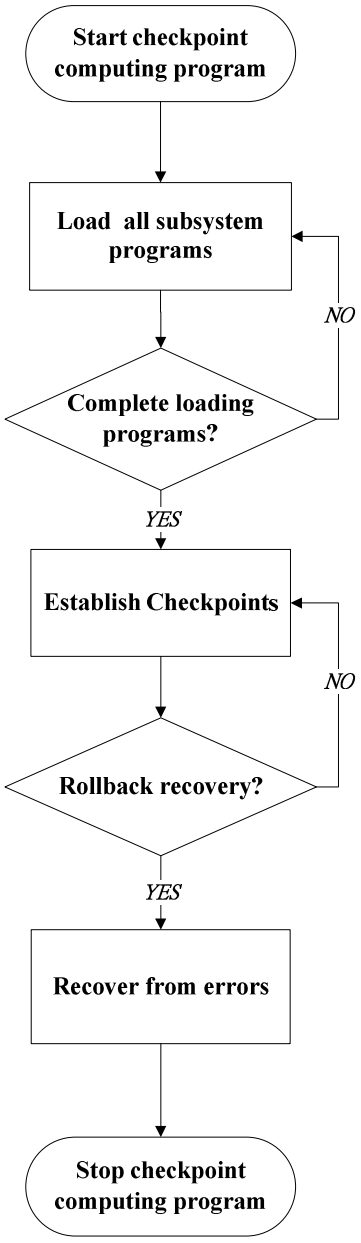

Fig. 2. Fault-tolerance program flowchart.

4.2.1 Checkpoint Establishment

Checkpoint establishment is first performed for each subsystem in the firearms training simulation system. This enables important information from each subsystem to be stored. When errors occur, rollback recovery can be performed. Figure 3 shows the checkpoint establishment flowchart; a detailed explanation is provided below.

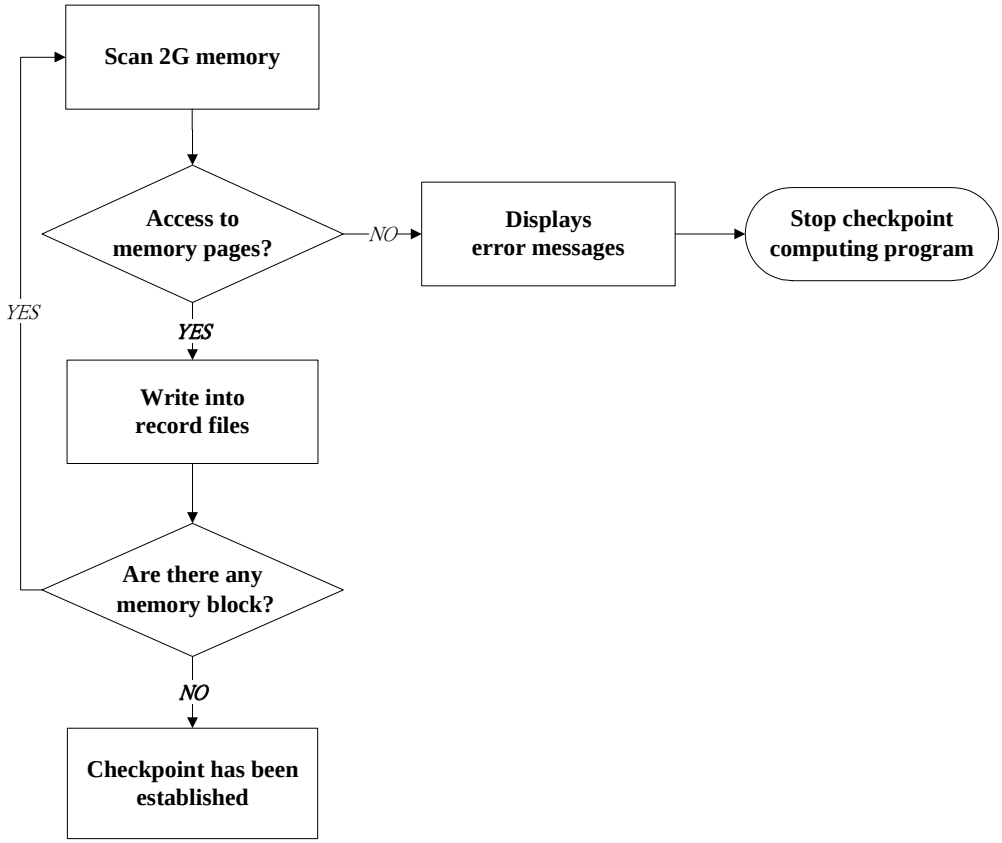

Fig. 3. Checkpoint establishment flowchart

Checkpoint establishment first stores thread content. Thread content is defined as the CPU register state when the program executes to a certain memory address. When rollback recovery is to be performed, execution must begin from the correct program code, not from the code where the error occurred. Thus, correct thread content should be stored in the *CONTEXT* structure announced previously. This structure is then written into the file. This can be considered the completion of one checkpoint storage, which ensures that execution can begin from correct thread content during rollback recovery.

Concerning memory states and content in the system, the virtual address space of the system must be investigated, because virtual address space contains all the execution information and dynamic configuration memory blocks of the system (such as thread stacks and heaps). Stacks in the memory are the parameters or local variables used by the application and are stored in the stack area. The majority of the heaps are used to store the small-scale memory configured by the application. Thus, if the memory ranges and content used in the virtual address space can be stored correctly, unexpected situations

that occur in the system can be recovered correctly.

Regarding how to store virtual memory information to the *CONTEXT* structure, before storing memory information, thread execution in the system must be stopped temporarily. Otherwise, the CPU continues scheduling, causing inconsistent storage results. At this time, `SuspendThread()` can be used to stop the application temporarily. Next, the `GetThreadContext()` function is used to complete the memory information storage. Before employing the `GetThreadContext()` function, the *CONTEXT* structure announcement must be completed, and the flags of the `ContextFlags` members in the *CONTEXT* structure must be set. Because the `ContextFlags` flags are primarily used to specify the register content to be obtained, and transmit the position or address of the *CONTEXT* structure to the `GetThreadContext()` function, this method can also be used to write the register content into this structure. Figure 4 shows a code snippet of the *CONTEXT* structure.

```
CONTEXT tcontext
SuspendThread(appthread);
Tcontext.ContextFlags = CONTEXT_FULL;
GetThreadContext(appthread, &tcontext);
Fwrite(&tcontext, sizeof (CONTEXT), 1, contextlog);
```

Fig. 4. *CONTEXT* structure code snippet

In the *CONTEXT* code snippet, the `ContextFlags` values are set to `CONTEXT_FULL`, indicating important registers to be obtained in the thread. `CONTEXT_FULL` is defined in *WinNT.h*. In this code snippet, it is defined before `GetThreadContext()`, indicating the CPU control register, integer register, and adjustment register to be obtained.

Storage of the virtual memory begins with scanning each block from `0x00010000` in the virtual address space of each subsystem. After this block scanning is finished, scanning automatically continues toward blocks with higher addresses. At this time, the states and content contained in the memory blocks used by each subsystem in the firearms training simulation system are stored. These scanning and storage steps are repeated until the memory address space `0x7FFEFFFF` is scanned.

During scanning of 2G user memory space, when the memory blocks used by the subsystem are scanned, the *MEMORY_BASIC_INFORMATION* structure must be configured to store the memory information. First, whether the memory range to be stored is larger than the set buffer size is determined. If it is too large, a warning window appears to avoid direct access causing a memory overflow and thereby producing errors and crashes. Next, the `VirtualQueryEx()` function is used to examine the usage situation of the memory blocks in the virtual address space. If this is committed, the `VirtualQueryEx()` function is used to write the memory information into the *MEMORY_BASIC_INFORMATION* structure. The memory information stored in this structure is then used for data storage. This facilitates understanding of the range to be stored, from which memory address reading should begin, and the size of the range to be read. Next, the `ReadProcessMemory()` function is used to read the memory content of each block, which is first written to the buffer. If no errors occur, it is then written to the file. Thus, the steps for checkpoint storage in a firearms training simulation system are complete.

4.2.2 Rollback Recovery

If rollback recovery is required in the system because of poor human operation or errors in program execution, the checkpoint information stored previously should be used for rollback recovery. Figure 5 shows a checkpoint rollback recovery flowchart. To ensure that the system can be correctly recovered to the checkpoint state stored previously, threads must be recovered to their earlier states and the correct data in the checkpoint file must be written back to the virtual memory address space in the firearms training simulation system.

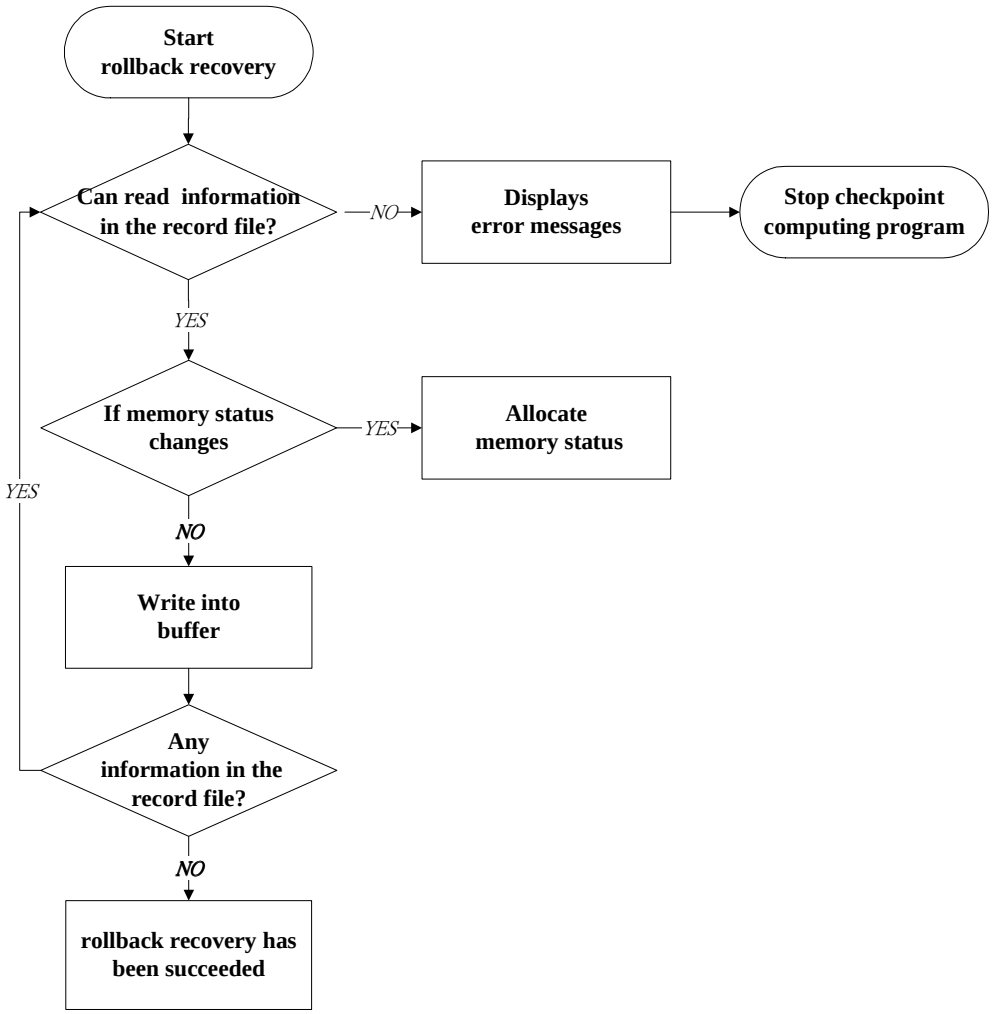

Fig. 5. Checkpoint rollback-recovery flowchart

During the rollback recovery process, the checkpoint file content must be copied to the register before being transmitted to the virtual memory location or address space. A

detailed explanation of this process follows.

First, the checkpoint data file stored previously is copied to the *MEMORY_BASIC_INFORMATION* structure. The `SetThreadContext()` function is then used to write the structure thread content back to the thread in the firearms training simulation system. The recovery process for the virtual memory region is as follows: at the beginning, the stored checkpoint memory information log file must be read to understand the application's memory usage and the initial position and block size of this memory region during checkpoint execution. After these data are obtained, where coverage should start and the range of coverage is known, facilitating the completion of rollback recovery. Additionally, current information regarding the usage conditions of the address in the address space of the application processes must be sought. This is compared with the state during checkpoint storage to determine whether the two states are identical. Differences between the two indicate that an interval exists from the most recently stored checkpoint. Memory state modification is necessary at this point. Next, the stored memory information must be read. At this stage, the memory protection status must first be judged to be readable. That is, Protect must be `PAGE_READWRITE`. If it is readable and writable, the memory content in the file is read, a base position or address is set, and the memory state is modified. Subsequently, the memory content in the file is written to the process memory. If it is unreadable, the `VirtualQueryEx()` function must be used to configure memory block categories with relative physical memory. That is, the memory category must be `MEM_COMMIT`. Ultimately, these two situations both use the `WriteProcessMemory()` function to write data to memory.

If the virtual memory does not correspond to the physical memory, that is, the memory status may be `MEM_RESERVE` or `MEM_FREE`, the memory configuration must be changed. When the memory status is `MEM_RESERVE`, the configured memory block must be transmitted to `VirtualAllocEx()`. The memory status is then changed to `MEM_COMMIT`. If the memory status is `MEM_FREE`, the configured memory block must be transmitted to `VirtualAllocEx()`. The status is then set to `MEM_RESERVE`. This is then corresponded to the `MEM_COMMIT` status with actual storage. After modifying memory protection, the data stored in the earlier checkpoint can be written to memory successfully. After completing this process, thread execution is recovered. If memory properties have not changed, the memory properties do not have to be modified; instead, they can be directly covered. After all the memory properties have been checked, the memory content stored previously can be read from the stored checkpoint file. The memory content is then recovered based on each region. Thus, the application's virtual location and memory content can be recovered to the status stored in the earlier checkpoint. This indicates successful completion of checkpoint rollback recovery.

5. EXPERIMENTAL RESULTS

The firearms training simulation system described in this study comprises four subsystems. These are the host computer subsystem, the visual effects subsystem, the audio effects subsystem, and the image capture subsystem. When users execute each functional operation, the host computer subsystem is responsible for transmitting and processing messages and communicating with the other subsystems. Although in practice any subsystem can be chosen to perform checkpoint storage and rollback recovery, because in-

formation transmission and processing is primarily performed by the host computer subsystem, using this subsystem for checkpoint establishment and rollback recovery is more appropriate. Figure 6 shows the test environment and stream processing model of our firearms training simulation system.

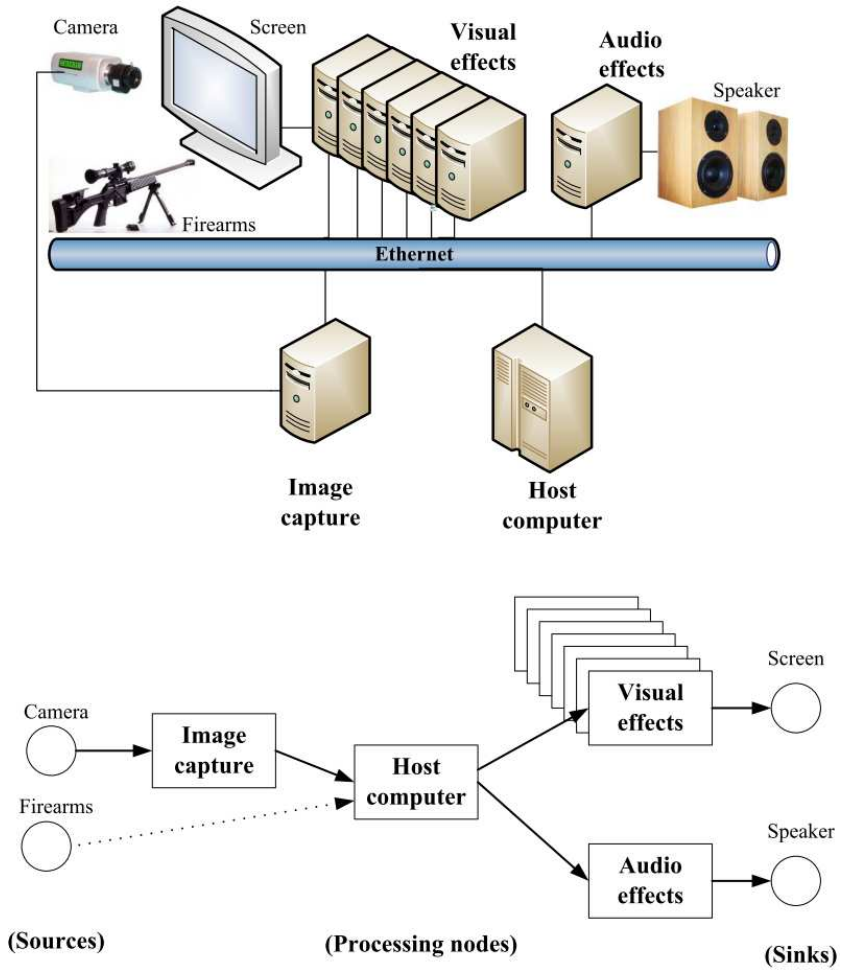

Fig.6. Test environment and stream processing model

5.1 Stand-Alone Fault-Tolerance Mechanism Implementation Test

Single-machine or stand-alone fault tolerance tests were performed first. They were conducted to examine the individual execution fault-tolerance programs in the host computer, visual effects, audio effects, and image capture subsystems and to assess their performance. The program record intervals were extended for the convenience of testing. Additionally, checkpoint establishment and rollback recovery was performed manually. The audio effects subsystem data was excessive, leading to a buffer overflow. After the

buffer capacity was modified, it was able to operate normally. Comparatively, the image capture subsystem is complex because it contains an infrared capture device and scans at a frequency of 150 Hz to identify the gun sources during aimed shooting. Because frequency changes are excessively rapid, access to the checkpoints of the six hosts in the image capture subsystem is often inconsistent, resulting in system crashes. Thus, during fault-tolerance mechanism computation of the image capture subsystem, the infrared capture device must be stopped manually to establish checkpoints and perform rollback recovery successfully. Table 3 shows a comparison of the computation time for each subsystem. The values in the table indicate that the host computer subsystem and the visual effects subsystem have the fastest checkpoint establishment and recovery times. The audio effects subsystem expends more time on checkpoint establishment because it includes numerous messages. The checkpoint information in the image capture subsystem is also substantial at only slightly less than that of the audio effects subsystem. Thus, it also requires additional time for rollback recovery. By calculating the computation times for each subsystem, we found that the longest checkpoint establishment time did not exceed 3 s, whereas the longest rollback recovery time did not reach 0.5 s. In comparison to the crashes caused by errors in the system, this performance is remarkable.

Table 3. Comparison of the computation time for each subsystem

Subsystem / Time	Host computer	Visual effects	Audio effects	Image capture
The longest checkpoint set-up time (ms)	20	19	2813	161
The shortest checkpoint set-up time (ms)	9	10	1346	153
The maximum error recovery time (ms)	107	114	340	316
The minimum error recovery time (ms)	105	110	316	114

5.2 Synchronized Fault-Tolerance Mechanism Implementation Test

Synchronized tests were implemented in the second part of testing. First, the checkpoint computation program of the firearms training simulation system was installed on each subsystem. These programs contained six visual effects subsystem hosts and six image capture subsystem hosts. Thus, any host can perform checkpoint establishment and rollback recovery. However, the host computer transmits messages to the visual effects and audio effects subsystems after performing rollback recovery. Thus, the host computer should perform rollback recovery. This avoids the occurrence of unanticipated errors, making program execution more stable.

Observing recovery information from each subsystem indicates that even with synchronized computation, the implemented computation results did not differ greatly from the single-machine or stand-alone results. The slight delays were determined to be caused by differences in recovery times for the subsystems and the influence of network cable

transmissions. Recovery of each subsystem can be completed within a maximum of 5 s, with the audio effects subsystem requiring the most time. The other, faster subsystems can be recovered within 1 s.

6. CONCLUSION AND FUTURE WORK

In this study, we examined typical distributed stream processing fault-tolerance mechanism designs and technique. We applied this technique to a typical firearms training simulation system to increase the operation reliability and availability. We used injected (transparent) checkpoint methods to implement the fault-tolerance mechanism processing program. This program comprised a checkpoint establishment and management module, an output message storage and management module, process crash checking module, a rollback-recovery module, and a central control module. Each module was controlled by the central control module. Periodic checkpoint establishment was performed (the establishment time period can be self-selected). Regular crashing tests are also performed simultaneously. If crashes do not occur, checkpoint establishment is performed again. Task execution is repeated in this manner until a system crash occurs. When the system crashes, the crash test module detects and diagnoses the errors, which are then returned to the rollback-recovery module for error recovery. The results of stand-alone or single-machine fault-tolerance mechanism tests and multi-machine synchronized fault-tolerance mechanism tests indicate that the performance of the checkpoint establishment and rollback recovery time can satisfy the system operation requirements. The experiment results of this paper focus on the comparison of stand-alone and synchronized tests to evaluate the feasibility of fault-tolerance mechanism implementation. We suggest that further comparison against any existing solutions be explored in the future work.

The level of the fault-tolerance mechanism described in this study is “software fault tolerance”. In other words, we designed a fault-tolerance processing tool program to store checkpoints in a firearms training simulation system. When errors occur, this program is used for rollback recovery. Future studies can use designs at the “system fault tolerance” level in various training simulation systems by constructing a complete fault-tolerance architecture during the design phase of the training simulation system. This type of fault-tolerance mechanism can automatically store checkpoints, memory blocks, and memory ranges for the program, and even detect errors that occur within the applications itself. When errors occur, this mechanism can also provide corresponding processing, thereby correcting errors and substantially increasing the reliability and availability of the operations of training simulation systems or other typical distributed stream processing systems.

REFERENCES

1. E. Danielsen, “An introduction to firearms simulation technology,” http://www.aic.gov.au/media_library/publications/proceedings/18/danielsen.pdf
2. E. N. Elnozahy, L. Alvisi, Y. M. Wang, and D. B. Johnson, “A survey of rollback-recovery protocols in message-passing systems,” *ACM Computing Surveys*, vol.

34, issue 3, Sep. 2002, pp. 375-408.
3. R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed system," *IEEE Trans. Software Eng.*, vol. 13, no. 1, Jan. 1987, pp. 23-31.
4. J. Hwang, M. Balazinska, A. Rasin, U. Cetintemel, M. Stonebraker, and S. Zdonik, "High-availability algorithms for distributed stream processing," in *Proceedings of the 21st International Conference on Data Engineering*, 2005, pp.779-790.
5. M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker. "Fault-tolerance in the Borealis distributed stream processing system," *ACM Transactions on Database Systems*, Vol. 33, No 1, Article 3, 2008, pp.1-44.
6. J. Hwang, Y. Xing, U. Cetintemel, and S. Zdonik, "A cooperative, self-configuring high-availability solution for Stream Processing," *IEEE 23rd International Conference on Data Engineering*, 2007, pp. 176-185.
7. Z. Sebepou, and K. Magoutis, "CEC: Continuous eventual checkpointing for data stream processing operators," *IEEE/IFIP 41st International Conference on Dependable Systems & Networks*, 2011, pp. 145-156.
8. J. Hwang, U. Cetintemel, and S. Zdonik, "Fast and high-available stream processing over wide area networks," in *Proceedings of the 23rd International Conference on Data Engineering*, 2008, pp. 604-613.
9. A. Brito, C. Fetzer, and P. Felber, "Minimizing latency in fault-tolerant distributed stream processing systems," *IEEE 29th International Conference on Distributed Computing Systems*, 2009, pp. 173-182.
10. T. Repantis, and V. Kalogeraki, "Replica placement for high availability in distributed stream processing systems," in *Proceedings of the 2nd ACM international conference on Distributed event-based systems*, 2008, pp. 181-192.
11. H. Shiokawa, H. Kitagawa, and H. Kawashima, "A-SAS: An adaptive high-availability scheme for distributed stream processing systems," *IEEE 11th International Conference on Mobile Data Management* , 2010, pp. 413-418.
12. Ali Ebnenasir, "Software fault-tolerance," Computer Science and Engineering Department Michigan State University U.S.A http://www.cse.msu.edu/~cse870/Lectures/SS2005/ft1.pdf.
13. Israel Koren and C. Mani Krishna. "Fault-Tolerance Systems," Elsevier Inc. 2007.
14. J. Plank, M. Beck and G. Kingsley, "Libckpt: transparent checkpointing under UNIX," *Usenix Conference*, 1995.
15. Johny Srouji, Paul Schuster, Maury Bach, and Yulik Kuzmin. "A transparent checkpoint facility on NT," *2nd USENIX Windows NT Symposium*, August 1998.
16. P. Emerald Chung, Woei-Jyh Lee, Yennun Huang, Deron Liang, and Chung-Yih Wang. "Winckp: a transparent checkpointing and rollback recovery tool for Windows NT applications," Fault-Tolerant Computing, 1999. Digest of Papers. *Twenty-Ninth Annual International Symposium*, June 1999, pp. 220-223.
17. P. Emerald Chung, Yennun Huang, Chandra Kintala, Woei-Jyh Lee, Deron Liang, Timothy K. Tsai, and Chung-Yih Wang. "NT-SwiFT: software implemented fault tolerance on Windows NT," *2nd USENIX Windows NT Symposium*, August 1998.
18. Galen Hunt and Doug Brubacher, "Detours: binary interception of Win32 functions," *3rd USENIX Windows NT Symposium*, July 1999.
19. Z. Zhang, Y. Gu, F. Ye, H. Yang, M. Kim, H. Lei, and Z. Liu. "A hybrid approach to high availability in stream processing systems," *IEEE 30th International Conference*

on Distributed Computing Systems, 2010, pp. 138-14.

Wu-Hong Chen (陳戊鋐) received the master degree in Information Engineering and Computer Science from Feng Chia University in 2001. He is now a Ph.D. candidate in Electrical Engineering, National Chung Hsing University, Taichung, Taiwan R.O.C. His research interest includes distributed system, fault tolerance and stream processing.

Jichiang Tsai (蔡智強) received the Doctor degree in Electrical Engineering from National Taiwan University in 2000. He is now a Professor in Electrical Engineering, National Chung Hsing University, Taichung, Taiwan R.O.C. He has published more than 30 papers in international conferences and journals. His research interest includes dependable computing, parallel and distributed system, embedded system, stream processing, clouding computing.