Proceedings of the 2005 IEEE
International Conference on Robotics and Automation
Barcelona, Spain, April 2005

Application Cluster Service Scheme for Near-Zero-Downtime Services[1]

Fan-Tien Cheng,[2] Shang-Lun Wu, Ping-Yen Tsai, Yun-Ta Chung
Institute of Manufacturing Engineering
National Cheng Kung University
Tainan, Taiwan, R.O.C.

Haw-Ching Yang
Institute of Systems and Control Engineering
National Kaohsiung 1st U of Science and Technology
Kaohsiung, Taiwan, R.O.C.

***Abstract* - The required reliability in applications of a distributed computer system is continuous service for 24 hours a day, 7 days a week. However, computer failures due to exhaustion of operating system resources, data corruption, numerical error accumulation, and so on, may interrupt services and cause significant losses. Hence, this work proposes an application cluster service (APCS) scheme. The proposed APCS provides both a failover scheme and a state recovery scheme for failure management. The failover scheme is designed mainly to automatically activate the backup application for replacing the failed application whenever it is sick or down. Meanwhile, the state recovery scheme is intended primarily to provide an inheritable design pattern to support applications with state recovery requirements. An application simply needs to inherit and implement this design pattern, and then can accomplish the task of state backup and recovery. Furthermore, a performance evaluator (PEV) that can detect performance degradation and predict time to failure is developed in this study. By using these detection and prediction capabilities, the APCS can perform the failover process before node breakdown. Thus, applying APCS and PEV can enable a distributed computer system to provide services with near-zero-downtime.**

Index Terms – Application cluster service (APCS), failover scheme, state recovery scheme, performance evaluator (PEV), near-zero-downtime service.

I. INTRODUCTION

The international technology roadmap for semiconductors (ITRS) states that total allowable scheduled and non-scheduled down time of a factory information and control system (FICS) ranged from 240 min for 2003 to 120 min for 2007. This requirement means that the reliability of the FICS should exceed 0.9998. That is, the services provided by FICS should have near-zero-downtime performance. Generally, the services include application programs and computer systems that provide operating environments for applications. Consequently, if continuous services of application programs are required, their supporting computer systems must also be healthy.

To improve reliability, high availability cluster service schemes of computer systems are often proposed. High availability clustering technology [1] [2] was presented to automatically detect node failures during service processes. Upon detecting application program failure, the cluster service will launch a backup application program to continue providing application services from the point where the malfunctioning application program left off. Several commercial products, such as Microsoft Cluster Service (MSCS)[3], Matrix HA and Matrix Server of PloyServe[4], etc., can provide such cluster service schemes. However, the failure detection mechanisms (also denoted as the heartbeat mechanisms) of the above schemes are complicated, and private networks are required for implementing these mechanisms.

Shen et al. [5] designed and implemented the Neptune middleware system that provides clustering support and replication management for scalable network services. Neptune employs a loosely connected and functionally symmetric clustering architecture to achieve high scalability and robustness. This architecture shields the clustering complexities from application developers through simple programming interfaces. Additionally, Neptune provides replication management with flexible replication consistency support at the clustering middleware level. However, the replication management scheme of Nepture periodically replicates the state values to all of the nodes in the clustering system. Accordingly, the communication burden associated with the backup of state values will be rather high if the number of nodes exceeds three.

Kim et al. [6] presented a Load Cluster Management System (LCMS). LCMS follows the client-server management paradigm of the simple network monitoring protocol (SNMP), and consists of three managers with different roles, which distribute management functionality to all hosts in a cluster group. By using SNMP these managers can reduce the network bandwidth required in management operations. However, LCMS is only capable of fault detection and does not consider either application failover or state recovery.

To improve the availability of Open Source Cluster Application Resources (OSCAR), Leangsuksun et al. [7] proposed adopting component redundancy and developing a detailed failure-repair model for predicting the availability. However, the proposed scheme has only one standby server to backup the primary server; and instead of the entire application lever failover, this study only considers the failover of several key components.

Several investigations [8] [9] have developed methods and tools for increasing application program reliability and recoverability. Nevertheless, these methods/tools can only enhance the reliability of single application programs. These methods/tools cannot improve the reliability of computer systems that possess several object-oriented application programs, and nor are they capable of state recovery.

In the field of improving overall distributed system reliability, Osman and Bargiela [10] proposed a fault tolerant environment for open distributed computing (FADI). FADI can detect the occurrence of errors by monitoring user-process failures and node crashes. FADI also presented a non-blocking checkpoint mechanism, combined with a selective message logging technique, to perform backup and recovery operations for distributed processes. However, the proposed system does not prepare spare nodes to replace the faulty ones. Therefore, if faulty

[1] The authors would like to thank the National Science Council of the Republic of China for financially supporting this research under Contracts No. NSC-91-2622-E-006-019 and NSC-93-2212-E-006-022.
[2] The corresponding author. (e-mail: chengft@mail.ncku.edu.tw).

nodes cannot be repaired, their mission may not be fulfilled.

The Object Management Group (OMG) submitted a request for proposal (RFP) for fault-tolerant CORBA (FT-CORBA) using entity redundancy [11] in CORBA 3 specification [12]. The FT-CORBA specification defines a standard set of interfaces, policies, and services that robustly support highly reliable applications. Based on FT-CORBA, Natarajan et al. [13] devised the DOORS framework, which is implemented as a CORBA service, to provide end-to-end application-level fault tolerance. In FT-CORBA, FaultDetectors are CORBA objects that detect faults via either pull-based or push-based mechanisms. The pull-based monitoring mechanism periodically polls applications to determine whether their objects are alive. The push-based monitoring mechanism (also known as a heartbeat monitor) requests that applications send periodic heartbeats to the FaultDetector. However, neither mechanism can detect application performance degradation.

Cheng et al. [14] developed a service management scheme (SMS), based on the Jini infrastructure and the technology of design by contract. A generic evaluator (GEV) is included in the SMS. The SMS can detect the following errors - service crashing, transmission of wrong messages, and service degradation. Furthermore, the GEV can periodically backup the execution status and parameters of a service to the database. The GEV informs the client if an error is detected in a service. The client then retrieves the backup information and sets it to the backup service, to continue the application process. However, the GEV simply uses a simple 3σ test to determine an abnormal state, an approach which is inadequate. The reason is that the parameters to be monitored for diagnosing a system tend to be multi-dimensional vectors rather than single variables. Therefore, multivariate analysis models, or more advanced diagnosis techniques are required.

This study proposes an application cluster service (APCS) scheme with the capabilities of failover and state recovery such that application programs under the control of the APCS can provide more reliable services. The APCS has an efficient and concise failure-detection mechanism (also known as a heartbeat mechanism). A design pattern for state recovery is also provided by the APCS. Furthermore, a performance evaluator (PEV) that can detect performance degradation and predict time to failure of a node is developed in this study. Combining APCS with PEV and using the detection and prediction capabilities of the PEV, the APCS can perform the failover process before an application program or a node breakdown. Consequently, a distributed computer system can provide services with near-zero-downtime by applying APCS+PEV.

The rest of this paper is organized as follows. Section 2 details the APCS scheme. Section 3 then introduces the PEV scheme. Next, Section 4 presents considerations for deploying the APCS and PEV. Subsequently, Section 5 shows comparisons of APCS+PEV with Microsoft cluster service (MSCS) and APCS. Section 6 then describes the reliability analysis for applying APCS+PEV. Next, Section 7 introduces an illustrative example of implementing the APCS and PEV in a manufacturing execution system. Finally, conclusions are made in Section 8.

II. APCS SCHEME

Figure 1 shows the cluster environment using the APCS. This cluster environment contains several nodes. A personal computer (PC) or any computer server may act as a node. Each node contains an APCS and several applications. The operating system application programming interface (OS API) is applied by the APCS to monitor and control each application within the node. Moreover, CORBA specification is adopted as the communication infrastructure among the APCSs. The communication protocols used among all of the applications depend on user preferences, and can include CORBA, DCOM, or JAVA RMI. Finally, the shared storage stores the statuses and records of all of the APCSs and applications. The APCS has a failover scheme and a state recovery scheme, implemented by the failover service manager (FSM) and the design pattern for state recovery (DPSR), respectively.

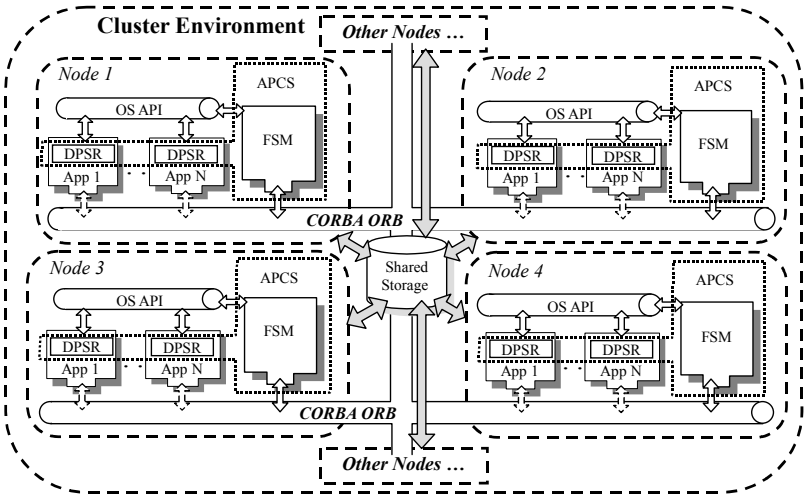


Figure 1: Cluster Environment Using APCS.

2.1 Failover Scheme

The nodes in the cluster environment are assigned to one or other of two roles: *Master* and *Slave*. Only one node can act as *Master* and the others serve as *Slave*s. The *Master* sends heartbeats to each *Slave* such that each *Slave* is aware of the existence of the *Master*. The *Slave* should send an acknowledgement after receiving a heartbeat from the *Master*. If the *Master* does not receive this acknowledgement, it means that the *Slave* in question is down, and the failover process of the *Slave* is then launched. If a *Slave* fails to receive heartbeats from the *Master* on three consecutive occasions, the *Slave* will start investigating the existence of the *Master*. Following three successive inquiries without response, it is considered certain that the *Master* is down. The *Slave* then notifies all of the other *Slave*s to stop investigating. Subsequently, all of the *Slave*s will enter a reconfiguring state to select a new *Master*. Finally, the replacement process for the *Master* is launched.

The failover scheme of APCS is implemented by FSM that includes a node event listener, fault detector, application manger, and several Ap agents, as depicted in Fig. 2. The functions of the FSM modules are outlined as follows.

- The node event listener receives all of the messages sent from all of the other nodes, and distributes the messages to the fault detector or resource manager.
- The fault detector mainly handles the tasks of fault detection and node replacement, respectively. After performing node replacement, the fault detector forwards a request to the application manager to execute the fault recovery process of applications.
- The application manager is primarily concerned with the tasks of invoking applications and application replacement. The application manager is in charge of the entire Ap agents that monitor and control applications.

- Ap agents are governed by the application manger. An Ap agent can only monitor and control an application. An Ap agent can invoke, detect, and close an application by using OS API, and then reports the application status to the application manger.

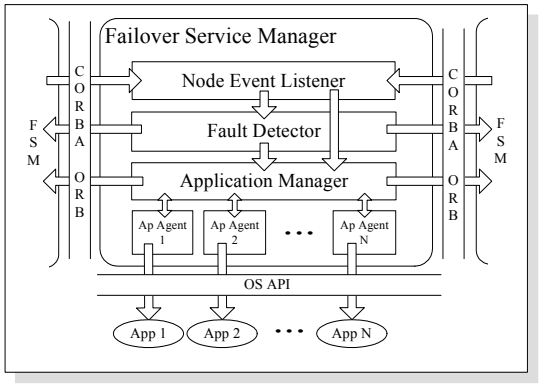


Figure 2: Block Diagram of Failover Service Manager.

2.2 State Recovery Scheme and Its Design Pattern

A design pattern for state recovery (DPSR) is developed to facilitate state recovery operations,. Figure 3 shows that DPSR consists of one "Memento" pattern [15]: *Memento*, and two "Command" patterns [15]: *SR_Object* and *StateMgr*. These patterns are presented below.

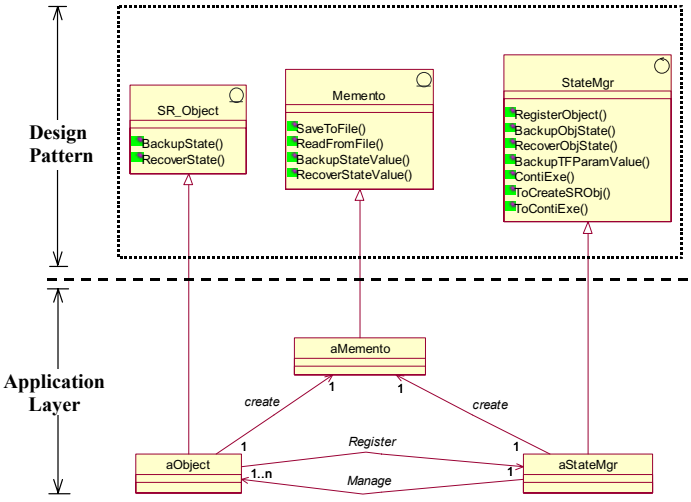


Figure 3: Design Pattern for State Recovery Scheme.

Figure 3 shows that, in the application layer, *aStateMgr* should be implemented by inheriting *StateMgr*. *aObject* that needs the functions of state backup and recovery should inherit *SR_Object* and register at *aStateMgr*. When a state backup process is required, *aStateMgr* will ask *aObject* to create an *aMemento* object that inherits *Memento* to handle the packing functions of object states. *aStateMgr* then stores the object state values from *aMemento* to the shared storage. Meanwhile, when a state recovery process commences, *aStateMgr* first creates the necessary *aObjects* based on the status stored in the shared storage. Next, *aStateMgr* creates an *aMemento* to retrieve the object state values from the shared storage. Finally, the object state values are distributed to the corresponding *aObjects* by the *aMemento*. The detailed state backup and recovery processes are described in [20].

III. PEV SCHEME

The PEV is designed to detect whether a node is in a sick state or not by monitoring and evaluating the system resource parameters of the node. If a node is in a sick state, the PEV can also predict the time to failure of the node. Figure 4 illustrates the architecture of the PEV, which consists of several data collectors, the detection module, and prediction module. These three modules are detailed below.

- Data Collector

A data collect resides at each node for collecting system resource parameters via OS API. These parameters are then passed to the PEV by CORBA specification.

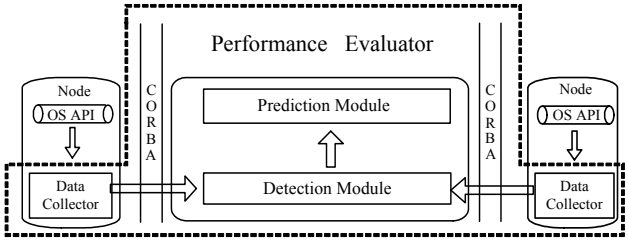


Figure 4: Architecture of Performance Evaluator (PEV).

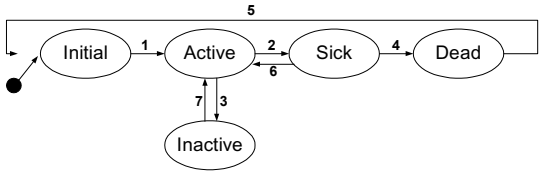


Figure 5: State Diagram of a Node

- Detection Module

Before introducing the detection module, the state diagram of a node is presented, as shown in Fig. 5. The state diagram includes five states: initial, active, inactive, sick, and dead. Table 1 lists the corresponding state transition table of the state diagram. Generally, a node is in the active state. However, when available resources are below the thresholds (due to software aging [16] for example), the node enters the sick state. The node returns to the active state if the available resources are recovered. On the contrary, if the available resources are exhausted, the node enters the dead state or, restated, the node is down. The above description demonstrates that the major purpose of the detection module is to detect whether or not a node is in a sick state. Therefore, as shown in Fig. 6, the fuzzy logic detection module is applied to infer a State_Index from the major resource performance parameters.

Table 1: State Transition Table of a Node.

Number	Current State	Trigger	New State
1	Initial	Initialization complete	Active
2	Active	Available resources being less than thresholds	Sick
3	Active	Service being paused	Inactive
4	Sick	Available resources being exhausted	Dead
5	Dead	Repair complete	Initial
6	Sick	Available resources back to normal	Active
7	Inactive	Service being resumed	Active

Microsoft Windows 2000 technical information [17] states that computer performance monitoring includes evaluating memory and cache usage, analyzing processor activity, examining and tuning disk performance, and monitoring network performance. To simplify the computer-performance-monitoring problem, this work only considers the factors related to software aging [16]. Software aging describes the accumulation of errors during software execution, eventually resulting in crash/hang failure. Gradual performance degradation may also accompany software aging. The major factors related to software aging are processor and memory. According to the technical report of Microsoft [17], the resource performance parameters related to processor and memory include processor time, privileged time, pool nonpaged bytes, available Mbytes, working set, pages/sec, pool nonpaged allocations, pool paged bytes, pages input/sec, pages output/sec, and so on. By applying multi-variate analysis and factor analysis, only four key parameters, namely, processor time, privileged time, pool nonpaged bytes, and available Mbytes are extracted as the inputs of the fuzzy logic detection module, as presented in Fig. 6. Among these inputs, processor time and privileged time are used to check processor bottlenecks, poll nonpaged bytes for memory leakage, and available Mbytes for memory shortage.

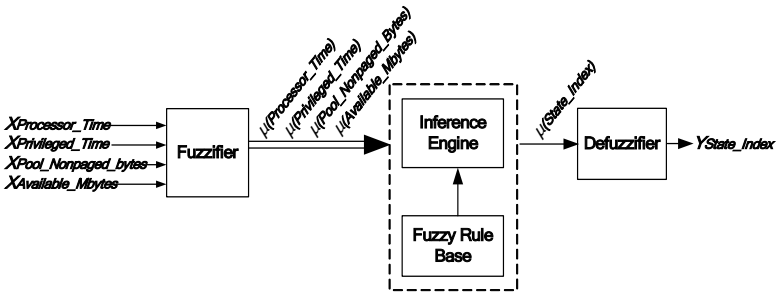


Figure 6: Fuzzy Logic Detection Module for Inferring the *State_Index*.

The weights of the four input parameters considered in the fuzzy rules of the inference engine are assigned to be equal. The semantic output variable is *State_Index*. Moreover, the value of the *Stat_Index* is normalized to be between 0 and 1. If it is between 0 to 0.3, then the node is in the sick state; otherwise the node is in the active state.

- Prediction Module

When the detection module detects that a node is in the sick state, the prediction module is launched to predict the time to failure. If a failure mode of software aging with monotonic-decay property is assessed, then the corresponding mean time to failure (MTTF) can be evaluated as follows.

Figure 7 shows three curves for predicting the MTTF of a monotonic-decay software-aging system, where t and y denote time and remaining resources, respectively. With enough samples: (t_i, y_i), $i=1, 2, \ldots, n$, the first curve that represents the simple linear-regression relationship between the estimation of y, $\hat{y}$, and t can be formulated as [18]:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 t_i \quad i=1, 2, 3, \ldots, n \tag{1}$$

where $\hat{\beta}_1 = \dfrac{\sum_{i=1}^{n}(y_i - \bar{y})(t_i - \bar{t})}{\sum_{i=1}^{n}(t_i - \bar{t})^2}$, $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \cdot \bar{t}$, $\bar{y} = \dfrac{\sum_{i=1}^{n} y_i}{n}$, and $\bar{t} = \dfrac{\sum_{i=1}^{n} t_i}{n}$.

By applying the concept of prediction interval [18], the second and third curves, $\hat{y}_{LB}$ and $\hat{y}_{UB}$, which denote the lower bound and the upper bound of the estimation of $\hat{y}$, are also illustrated in Fig. 7.

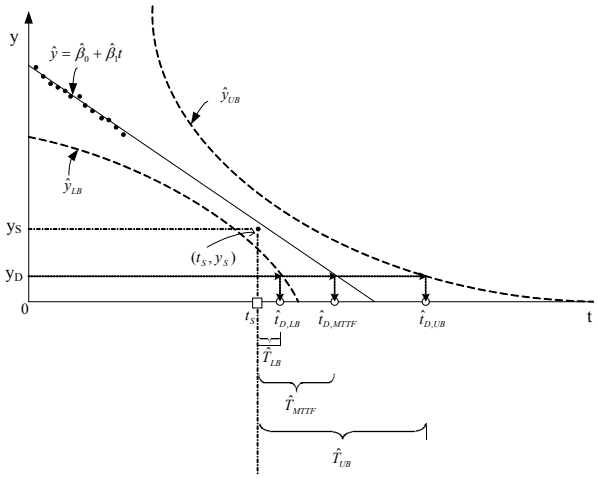


Figure 7 Estimation Curves for Predicting Mean Time to Failure

If the detection module detects that a node is in the sick state at time t_S, then the MTTF can be predicted below. Let y_D represents the remaining resources when entering the dead state. y_D, therefore, is given. Then, observing Fig. 7 and by applying Eq.(1) and the inverse regression methods of calibration [19], $\hat{t}_{D,MTTF}$, $\hat{t}_{D,LB}$, and $\hat{t}_{D,UB}$ can be devised by the intersections between y_D and $\hat{y}$, y_D and $\hat{y}_{LB}$, as well as y_D and $\hat{y}_{UB}$, respectively. As such, the most likely MTTF, $\hat{T}_{MTTF}$, the lower bound of MTTF, $\hat{T}_{LB}$, and the upper bound of MTTF, $\hat{T}_{UB}$, can be derived as:

$$\hat{T}_{MTTF} = \hat{t}_{D,MTTF} - t_S, \quad \hat{T}_{LB} = \hat{t}_{D,LB} - t_S, \quad \text{and} \quad \hat{T}_{UB} = \hat{t}_{D,UB} - t_S.$$

Notably, time-to-failure prediction modules should vary among different failure modes.

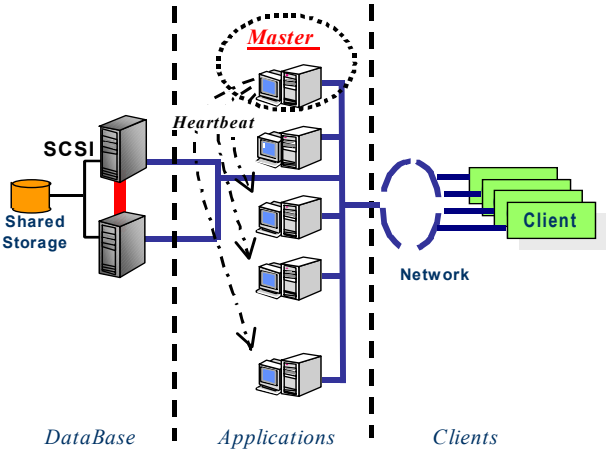
3.1 Integrating PEV with APCS

Figures 1, 2, and 4 illustrated the cluster environment using the APCS, the block diagram of FSM in the APCS, and the architecture of the PEV, respectively. To integrate the PEV with the APCS, the PEV is to substitute for *Master* and all the other nodes serve as *Slave*s. Each node implements a data collector of the PEV. Additionally, the functions of the heartbeat mechanism provided by the fault detector of FSM are no longer needed; while the other functions of FSM are still required. The deployment of the APCS and the PEV is demonstrated in Section 4.

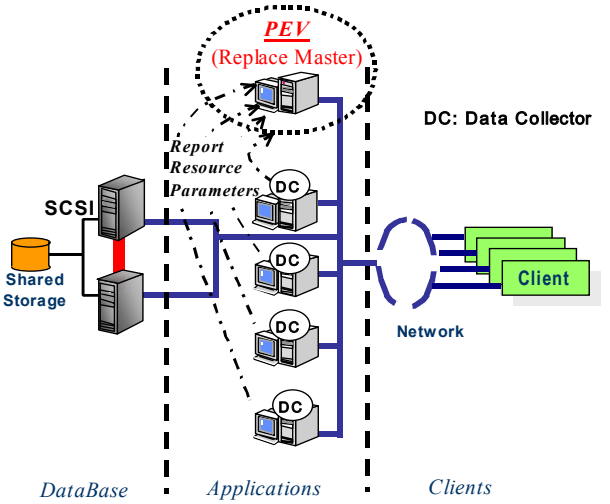
IV. DEPLOYMENT CONSIDERATIONS

The deployment diagrams of 3-tier high availability cluster services obtained by applying the APCS and the APCS+PEV are displayed in Fig 8. Figure 8(a) shows that the APCS is implemented in each node at the application tier, and one of the nodes acts as the *Master* that sends heartbeats through the public (cluster to client) network to all of the other nodes, which are *Slave* nodes. The shared storage that is required for the failover scheme of the APCS is combined with the system database.

Figure 8(b) illustrates that the *Master* node is replaced by the PEV and each of the other nodes still needs to implement the APCS due to failover and state recovery considerations. Besides, each node must own a data collector that obtains the required resource parameters and sends them to the PEV for health monitoring.



(a) With APCS



(b) With APCS and PEV

Figure 8: Deployment Considerations by Applying APCS and PEV.

V. Comparisons of APCS+PEV with APCS and MSCS

The traditional high availability clustering service was provided by MSCS. Table 2 lists the comparisons of APCS+PEV with APCS and MSCS. Basically the MSCS requires dedicated shared storage for each group of resources. This dedicated shared storage is different from the system database. On the other hand, the APCS and APCS+PEV do not require dedicated shared storage. The APCS and APCS+PEV can apply the system database to serve as the shared storage. Owing to this difference, MSCS is more costly to install. Each node using the MSCS broadcasts its heartbeat to all the other nodes in a group; while the heartbeat mechanism of the APCS only allows the *Master* node to send a heartbeat to the *Slave* nodes. Clustering services utilizing the APCS+PEV do not require a heartbeat mechanism. The MSCS requires a private (node to node) network to support the heartbeat mechanism, while the APCS applies a public (cluster to client) network to send heartbeats. The APCS+PEV does not require a private network either. The MSCS has no state recovery scheme; while the APCS and APCS+PEV do have state recovery schemes. Both the MSCS and the APCS do not have the capabilities of node sickness detection and time-to-failure (TTF) prediction. If the clustering service integrates the PEV with the APCS, then the health of individual nodes can be monitored. Furthermore, if the PEV detects that a node is in the sick state, the PEV can also predict the time to failure of this sick node and ask the APCS to perform failover operations before node breakdown. Accordingly, continuous services with near-zero downtime can be assured.

Table 2: Comparisons of APCS+PEV with APCS and MSCS.

	MSCS	APCS	APCS+PEV
Dedicated Shared Storage	Needed	No Need	No Need
Heartbeat Mechanism	Broadcasting by Each Node	Only Send by *Master*	No Need
Private Network	Needed	No Need	No Need
State Recovery	No	Yes	Yes
Sickness Detection & TTF Prediction	No	No	Yes

VI. Reliability Analysis for Applying APCS+PEV

The reliability of the APCS+PEV, R_{A+P}, is derived to evaluate the improvement of overall system reliability. Suppose that a system has N similar nodes with the same reliability, R, and the clustering system needs M nodes for the system to function properly. In this case, N-M backups are involved. Assume that the reliability of PEV is R_{PEV}. Then, R_{A+P} can be expressed as [14]:

$$R_{A+P} = R_{PEV} \left(\sum_{i=0}^{N-M} C_i^N (1-R)^i R^{N-i} \right) = R_{PEV} \cdot R_{APCS} \qquad (2)$$

The reliability of the original system without any backup is $R_0=R^M$. Therefore, the improvement of the reliability is $R_{A+P}-R_0$.

Table 3 tabulates the reliability improvement with M=1 and R_{PEV}=1. Table 3 shows that N=2 is required for R=0.99, N=3 for R=0.98, and N=4 for R=0.95 to guarantee that the R_{APCS} exceeds or equals 0.9999. If a mirror-type backup is prepared for the PEV and the reliability of each node of this mirror-type PEV is 0.99, then Table 3 shows that the R_{PEV} is 0.9999 (N=2). Consequently, we have:

$$R_{A+P}=R_{PEV} \cdot R_{APCS}=0.9999 \cdot 0.9999=0.9998.$$

Table 3: Reliability Improvement with M=1 and R_{PEV}=1.

	0.9	0.95	0.98	0.99
N=2	0.09	0.0475	0.0196	0.0099
N=3	0.099	0.049875	0.019992	0.009999
N=4	0.0999	0.04999375	0.01999984	0.00999999

The ITRS states that the scheduled and non-scheduled down time of a FICS ranges from 240 min for 2003 to 120 min for 2007. The requirement mentioned above means that the reliability of the FICS should exceed 0.9998. From the above analysis, if only one healthy node is required for proper FICS functioning, then for R=0.99 a total of four nodes (two for APCS, and two for PEV) are required to ensure that the FICS has 0.9998 reliability.

VII. Illustrative Example

Figure 9 presents an illustrative example for a IC-packaging factory involving two nodes, several pieces of

equipment, and one shared storage. Two PEVs with a mirror-type backup configuration are also shown in a dotted-line box.

The first version clustering system (with APCS only) operates as follows. Under the normal condition, EM1 is running at Node 1 for managing several pieces of equipment. Besides, WT2 is a backup and remains at Node 1. Moreover, WT1 is running at Node 2 for handling the WIP tracking task. Additionally, EM2 is a backup and resides at Node 2. Provided that APCS2 detects a failure of WT1, then APCS2 notifies APCS1 to invoke WT2 for recovering the service provided by WT1. In another case, if APCS2 detects that Node 1 is down via the heartbeat mechanism, then APCS2 launches EM2 to resume the management of those pieces of equipment originally managed by EM1. However, in this case, the services provided by EM may be paused.

The second version clustering system (with the APCS+PEV) performs as follows. Under the normal condition, EM1 and WT1 are running at Node 1 and Node 2, respectively. Provided that APCS2 detects a failure of WT1, then APCS2 notifies APCS1 to invoke WT2 to recover the service provided by WT1. In another case, if the PEV detects that Node 1 is sick and its time to failure is also predicted, then the PEV will inform APCS2 to have EM2 take over the service originally provided by Node 1 before Node 1 breaks down. In this case, continuous service by EM is assured. Notably, if the reliability is 0.99 for Node 1, Node 2, PEV1, and PEV2, then the second version clustering system (by applying the APCS+PEV) shown in Fig.9 will have reliability of 0.9998, as analyzed in Section 6.

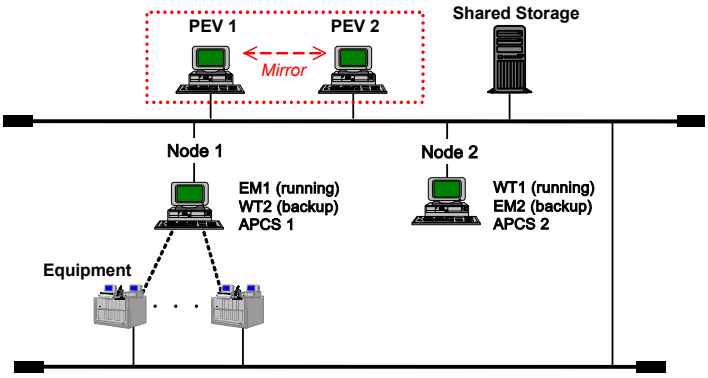


Figure 9: Illustrative Examples with APCS and APCS+PEV.

Discussion

This work assumes that the reliability of the shared storage is 1 and the network reliability is 1. The shared storage is actually the system database whose reliability must be 1 according to practical considerations; while this study assumes that network reliability of 1 can be achieved by a network resource failover scheme. Future research will examine how to incorporate a network resource failover scheme into the clustering system using the APCS+PEV.

VIII. Conclusion

Two novel schemes for high availability clustering services are proposed in this work. The first scheme utilizes the APCS only. The APCS has a concise heartbeat mechanism that is executed via the public network. Besides, the APCS possesses both failover and state recovery schemes. The second scheme applies both the APCS and the PEV. The PEV can detect whether a node is sick or not. Moreover, if a node is sick, the PEV can also forecast its time to failure. Consequently, the heartbeat mechanism is not required for the scheme applying the APCS+PEV. Because the PEV can predict the time until failure of a sick node and notify the APCS of the backup node to perform failover process before the node's breakdown, near-zero-downtime services can be guaranteed.

References

[1] M. Evan and S. Hal, *Blueprints for High Availability: Designing Resilient Distributed Systems*, NY: John Wiley & Sons, February 2000.

[2] P. Folyed and H. Michael, High Availability: Design, Techniques And Processes, NJ: Prentice Hall. 2001.

[3] R. Gamache, R. Short, and M. Massa, "Windows NT Clustering Service," *Computer*, vol. 31, no. 10, pp.55-62, Oct. 1998.

[4] PolyServe, Matrix HA/Server, PolyServe Corporation. [Online]. Available: http://www.polyserve.com/

[5] K. Shen, T. Yang, and L. Chu, "Clustering Support and Replication Management for Scalable Network Services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 11, pp.1168-1179, Nov. 2003.

[6] M.-S. Kim, M.-J. Choi, and J.-W. Hong, "Highly Available and Efficient Load Cluster Management System Using SNMP and Web," in *Proc. of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2002)*, Florence, Italy, pp. 619-632, April 2002.

[7] C. Leangsuksun, L. Shen, T. Liu, S. Hertong, and S. L. Scott, "Availability Prediction and Modeling of High Availability OSCAR Cluster," in *Proc. IEEE International Conference on Cluster Computing*, Hong Kong, pp. 380-386, 1-4 Dec. 2003.

[8] Y. Huang and C. Kintala, "Software Implemented Fault Tolerance: Technologies and Experience," in the 23rd *International Symposium on Fault-tolerance Computing* (FTCS), Toulouse, France, pp.2-10, June 1993.

[9] Veritas FirstWatch ,Veritas Corporation. [Online]. Available: http://www.veritas.com/us/products/firstwatch

[10] T. Osman and A. Bargiela, "FADI: A Fault Tolerant Environment for Open Distributed Computing," *IEE Proceedings of Software*, vol. 147, no. 3, pp. 91-99, June 2000.

[11] Object Management Group, *Fault Tolerant CORBA Specificatoin*, OMG Document orbos/99-12-08 edition, December 1999.

[12] J. Siegel, *CORBA 3 Fundamentals and Programming*, OMG Press, 1999.

[13] B. Natarajan, A. Gokhale, and S. Yajnik, "DOORS: Towards High-performance Fault Tolerant CORBA," *IEEE International Symposium on Distributed Objects and Applications*, Antwerp, Belgium, pp. 39-48, 2000.

[14] F.-T. Cheng, H.-C. Yang, and C.-Y. Tsai, "Developing a Service Management Scheme for Semiconductor Factory Management Systems," *IEEE Robotics and Automation Magazine*, vol. 11, no. 1, pp. 26-40, March 2004.

[15] G. Erich, H. Richard, J. Ralph, and V. John, *Design Patterns: Elements of Reusable Object-Oriented Software* (1st ed.). USA: Addison Wesley Professional, 1994.

[16] S. Garg, A. van Moorsel, K. Vaidyanathan, K. Trivedi, "A Methodology for Detection and Estimation of Software Aging," in *Proc. of 9th International Symposium on Software Reliability Engineering*, pp. 282-292, Paderborn, Germany, November 1998.

[17] Performance Monitoring, Microsoft. [Online]. Available: http://www.microsoft.com/windows2000/techinfo/reskit/en-us/default.asp?url=/windows2000/techinfo/reskit/en-us/core/fnca_pt3_intr.asp

[18] N. Draper and H. *Smith Applied Regression Analysis* (2nd ed.). USA: Addison Wesley Professional, 1981.

[19] R. G. Krutchkoff, "Classical and Inverse Regression Methods of Calibration," *Technometrics*, vol. 9, pp. 425-439, 1967.

[20] Shang-Lun Wu, "Application Cluster Service with Failover Capability," master's thesis of Institute of Manufacturing Engineering, National Cheng Kung University, Tainan, Taiwan, R.O.C., August 2004.