A Proactive Fault-detection Mechanism in Large-scale Cluster Systems

Wu Linping[1,2], Meng Dan[1], Gao Wen[1], and Zhan Jianfeng[1]

[1]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
{wlp, md, gw, jfzhan}@ncic.ac.cn

[2]Graduate School of the Chinese Academy of Sciences Beijing, China
wlp@ncic.ac.cn

Abstract

To improve the whole dependability of large-scale cluster systems, an online fault detection mechanism is proposed in this paper. This mechanism can detect the fault in time before node fails and enables the proactive fault management. The proposed mechanism is summarized as follows: First, the dynamic characteristics of cluster system running in normal activity are built using Time Series Analysis methods. Second, the fault detection process is implemented by comparing the current running state of cluster system with normal running model. The fault alarm decision is made immediately when the current running state deviates the normal running model. The experiment results show that this mechanism can detect the fault in cluster system in good time.

1. Introduction

The dependability and availability are the major challenges for the large-scale cluster systems. The researches related to cluster system dependability are divided into two categories approximately. One is High Availability (HA) technique based on redundant components and aims at providing continuous service. When one of the nodes fails, the fail-over procedures will begin immediately and the applications running on failed node are migrated to another redundant node[1]. The other is based on the prediction mechanism. The faults are detected in time and the correctly proactive maintenance program is implemented before node fails. This method avoids the costs of unplanned downtime of nodes. This method is named as Proactive Fault Management in several related researches[2]. The key question for proactive fault management is how to detect the fault in system quickly, that's efficient fault detection mechanism.

In fact, the fault detection mechanism has been applied to some mission-critical and safety-critical fields, such as the aircraft fault detection[4] and nuclear power plant fault detection[5]. Now two related works in computer systems are discussed. One is detection of software aging phenomena and the other is SMART technology for hard disk failure prediction. Software aging[6] is one kind of software fault, which is a phenomenon that the state of the software system degrades or crashes with time. The measurement-based rejuvenation approach deals with detection of the existence of software aging and predicting aging-related failures. In[7], the operating system resource usage and system activity data are collected at regular intervals from networked UNIX workstations. A statistical trend detection technique is applied to the collected data to detect the existence of aging and the estimated time to exhaustion is calculated using a non-parametric slope estimation technique. In[8], the dynamic characteristic of web server is built using time-series ARMA model and the ARMA model is used to detect aging and estimate resource exhaustion time. To improve the reliability of hard disk, the SMART[9] (Self-Monitoring, Analysis and Reporting Technology) failure prediction system is currently implemented in disk drives. The SMART can detect the fault in hard drives before disk fails. The researches related to SMART focus on the fault detection and failure prediction techniques. For example, in[9][10], the general framework is to detect anomalies, or variations from "normal" behavior, using a rank-sum null hypothesis test; in[11], the fault detection process is implemented using different statistical tests based on naive Bayesian classifiers.

Both software aging detection and SMART methods focus on part of system or part of fault models. In this paper, we give a proactive fault detection mechanism for large-scale cluster systems from system view. System monitoring is a necessary component for large-scale cluster systems. For example, the system monitoring in BlueGene/L is accomplished through a combination of I/O node and service node functionality[12]; for

This work is supported by the National '863' High-Tech Program of China (No. 2004AA616010) and the 15th key project of China (No. 2004BA811B09-1)).

1-4244-0054-6/06/$20.00 ©2006 IEEE

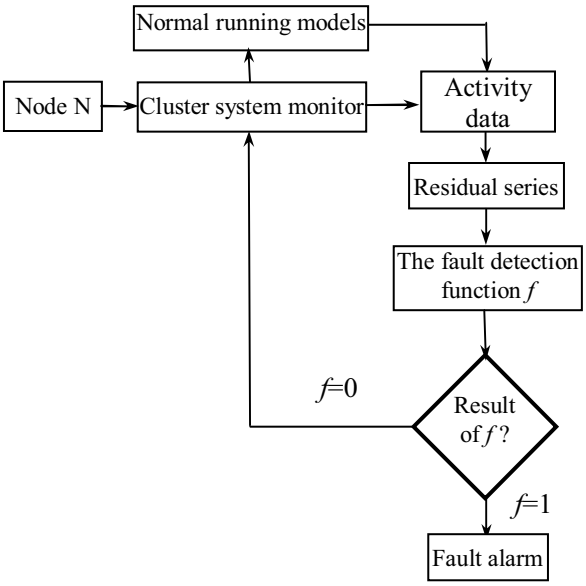
Dawning4000A[13], the system monitoring function is accomplished by the detector and data bulletin service[14] in the cluster operating system Phoenix[15]. The performance parameters related to system activity which is collected by system monitor at regular intervals are various, such as the usage of CPU and Memory or Swap, the voltage of nodes, the rotate speed of fan and the temperature of nodes. When faults occur in cluster system, the actual observation of those parameters by system monitor will deviate from their normal running model. So, online analysis the state information derived from cluster monitoring system can attain the symptom of fault. Currently, the analysis of state information is simple "threshold detection" method: if one of the performance parameters violates the acceptable threshold, the fault management system will consider that the fault occurred in cluster system and send out an alarm. For example, an alarm will be sent to administrator when the load of CPU exceeds 90%. The merit of threshold detection is simple to implement and appropriate when the signals are static or slowly varying. In cluster system, the performance parameters are varying with time, so threshold detection method is inappropriate.

This paper proposed a proactive online fault detection mechanism based on the activity data collected by cluster system monitor. The activity data describes the system running state. First, the normal running models are built using time series analysis on collected data when cluster runs normally. Second, the difference between a parameter's predicted value by normal running model and its directly observed value is termed a residual series. All parameters' residual series should be white noise series when the cluster system is behaving normally. So, the residual series gives the symptom of faults. In the end, the Likelihood Ratio Test (In the remainder, LRT for short) on the residual series is used to detect faults. The paper is organized as follows. Section 2 gives the fault detection procedure overview. The main algorithms including modeling the cluster normal activities and the LRT method are given in section 3. In section 4, we present the experiment results and section 5 concludes with future work.

2. Fault detection procedure

The fault detection procedure is illustrated in figure 1. Supposed that one application exclusive use of the nodes that it runs on, when one job marked *AppM* is running on one node marked *N* without faults (marked <*N, AppM*>), the performance parameter *S* is sampled at regular intervals and the time series $S(N,\triangle t)$ is formed. The normal running model of <*N, AppM, S*> is built using the time series model of $S(N,\triangle t)$. The procedure of building the time series model is discussed in section 3.1 in detail.

Figure 1. The online fault detection procedure



The time series model can predict the future value of parameter *S*.

After building the normal running models of every <*N, AppM, S*>, we propose an online fault detection method based on Slide Window. Define the window $W(<o, p, r>_{t-l+1}, <o, p, r>_t, l)$ for every <*N, AppM, S*> where the triple $<o, p, r>_{t-l+1}$ denotes the actual observed value *o*, the predicted value *p* by the normal running model and the residual *r=o-p* at *t-l*+1 time and $<o, p, r>_t$ denotes the actual observed value *o*, the predicted value *p* by the normal running model and the residual *r=o-p* at *t* time; *l* is the length of *W*. Now, the fault detection procedure is described as follows:

First step: normal running model selection. Select the normal running model marked *M* based on triple <*N, AppM, S*> where *N* is node name, *AppM* is job name and *S* is one of the performance parameters.

Second step: window *W* moves forward one pace. At *t* time, the window is $W(<o, p, r>_{t-l+1}, <o, p, r>_t, l)$. At $t+\triangle t$ time, the window moves forward one pace and is $W(<o, p, r>_{t+\triangle t-l+1}, <o, p, r>_{t+\triangle t}, l)$ where $p_{t+\triangle t}$ is the predicted value by model *M*, $o_{t+\triangle t}$ is the observed value collected from system monitor, the residual at $t+\triangle t$ time is $r_{t+\triangle t}= o_{t+\triangle t}-p_{t+\triangle t}$.

Third step: fault detection. $R(l,\triangle t)$ is one *l*-length residual series in $W(<o, p, r>_{t+\triangle t-l+1}, <o, p, r>_{t+\triangle t}, l)$. Using $R(l,\triangle t)$ as the input of fault detection function *f* and calculate the result of *f*. If *f*=1, one fault alarm decision is made. Otherwise, continue the fault detection process and return to the second step. The definition of fault detection function is given in 3.3.

Fourth step: if *AppM* is over and another new job is load, return to the first step.

3. The main algorithms

From above analysis, the key question is building the normal running models of every job on every node and using the residual series between observed value and predicted value of performance parameters as the symptom of fault detection. Based on the time series analysis technique, the residual series should be a white noise series and the distribution of the residual series follows the normal distribution $N(0, \sigma^2)$. If fault occurs in cluster system, the residual series will deviate from the white noise series and the distribution of the residual series doesn't follow the normal distribution $N(0, \sigma^2)$. The LRT method is used to construct the fault detection function *f*.

3.1 Modeling the normal activities

When job *AppM* is running on node *N* without faults, one of the performance parameters marked *S* (such as usage of CPU and Memory, network bandwidth, etc) is sampled at regular intervals by the cluster monitoring system. The sample cycle is $\triangle t$ and one time series marked $S(N,\triangle t)$ is formed where *N* is the length of series. The dynamic model is built using time series analysis on $S(N,\triangle t)$ and this model describes the normal running model of <*N, AppM, S*>. The following gives the detailed procedure of building the normal running model of <*N, AppM, S*> using time series analysis on $S(N,\triangle t)$.

First step: Data Preprocess.

For some unpredictable reasons, $S(N,\triangle t)$ is not a stationary time series. To improve the accuracy of fault detection, two data preprocess procedures are necessary for $S(N,\triangle t)$. One is moving average smoothing method and the other is computing the zero average of $S(N,\triangle t)$.

Moving average smoothing: the n-point moving average of $S(N,\triangle t)$ is $S^n(N-n+1,\triangle t)$ and $s_i^n = (\sum_{j=i}^{i+n-1} s_j) \Big/ n$.

Zero average: suppose μ is the average of time series $S^n(N-n+1, \triangle t)$, that's $\mu = \Sigma s_i^n \Big/ (N-n+1)$. The zero average of $S^n(N-n+1,\triangle t)$ is $Z(N-n+1,\triangle t)$ where every sample is $z_i = s_i^n - \mu$.

Second step: building the ARIMA model.

$Z(N-n+1,\triangle t)$ isn't a stationary time series generally. There is trend in it. Hence, we will first remove the trend by several differencing procedures. Define the one pace differencing operators $\triangledown$ is $\triangledown z_t = z_t - z_{t-1} = (1-B)z_t$ where *B* is the backshift operator, so the *d*-pace differencing is $\triangledown^d z_t = (1-B)^d z_t$.

The *d*-pace differencing of $Z(N-n+1,\triangle t)$ is $X(N-n+1-d, \triangle t)$ where the sample of new series is

$$x_t = \nabla^d z_t = \sum_{j=0}^{d} (-1)^d C_d^j z_{t-j}, t > d \qquad (1)$$

For each d=1,2,3…, calculate the Auto Correlation Function (ACF) and Partial Auto Correlation Function (PACF) of $X(N-n+1-d,\triangle t)$ respectively till the $X(N-n+1-d, \triangle t)$ can be regarded as a stationary time series (When the ACF of the time series is quickly decreasing, this is an indication that the time series is stationary).

Building the $ARMA(p, q)$ model for $X(N-n+1-d,\triangle t)$:

$$x_t - \sum_{i=1}^{p} \varphi_i x_{t-i} = a_t - \sum_{j=1}^{q} \theta_j a_{t-j}$$

where p is autoregressive order of the model, q is the moving average order of the model, a_t is a white noise series with mean zero and variance σ^2, and φ's and θ's are constants. If q=0, the model reduces to $AR(p)$. And if p=0, the model reduces to $MA(q)$. Suppose $\varphi(B) = 1 - \sum_{i=1}^{p} \varphi_i B^i$ and $\theta(B) = 1 - \sum_{i=1}^{q} \theta_i B^i$. So, the above model can be described as: $\varphi(B)x_t = \theta(B)a_t$ (2)

The key step is determining the order of the model, that's the value of (p, q). The ACF and PACF provide important information of (p, q). Table 1 gives the behavior of the ACF and PACF for $AR(p)$, $MA(q)$ and $ARMA(p, q)$ models.

Table 1. ACF and PACF for AR, MA, ARMA

	AR(*p*)	*MA*(*q*)	*ARMA*(*p*,*q*)
ACF	Trails off	Cuts off after lag *q*	Trails off
PACF	Cuts off after lag *p*	Trails off	Trails off

For $AR(p)$ and $MA(q)$, the value of p or q can be determined by the cuts off of PACF or ACF. For $ARMA(p, q)$, two key points to determine (p, q) are:

- The ACF of $ARMA(p, q)$ begins decay at lag q.
- The PACF of $ARMA(p, q)$ decays starting at lag p.

After determine the order of $ARMA(p, q)$, the values of φ's and θ's are estimated by Least Squares method.

From (1) and (2), the *ARIMA* model of $Z(N-n+1,\triangle t)$ is $\varphi(B)\triangledown^d z_t = \theta(B)a_t, t \leqslant N-n+1$ (3)

Third step: the normal running model.

From the analysis of first and second steps, the normal running model of <*N, AppM, S*> is described as follows: $\varphi(B)\nabla^d (s_t^n - \mu) = \theta(B)a_t$, $t \leqslant N-n+1$ (4) where s_t is the sample of performance parameter *S* at time *t*.

For $\nabla^d (s_t^n - \mu) = \nabla^d s_t^n$, formula (4) can be described as: $\varphi(B)\nabla^d s_t^n = \theta(B)a_t$, $t \leqslant N-n+1$ (5)

The model (5) gives the normal activities for <*N, AppM, S*>. If *AppM* is running on node *N* without faults, the performance parameter *S* should follow model (5) and the residual series $\{a_t\}$ between actual observed series and predicted series should be white normal noise. Otherwise, *S* will deviate model (5) and the residual series $\{a_t\}$ will not be white normal noise anymore. So, the fault detection question in cluster system is transformed as testing whether the residual series is white normal noise or not. We use LRT method to verify whether the residual series is white normal noise or not.

3.2 Likelihood Ratio Test

For an *n*-length residual series $R(n, \triangle t)$, the null hypothesis and alternative hypothesis are defined as:

H_0: $R(n, \triangle t)$ is white normal noise with mean zero (cluster system is running normally);

H_1: The mean of $R(n, \triangle t)$ is not zero (there are faults in cluster system).

The level of significance $\alpha=P\{reject\ H_0|H_0\}$ is the probability of type I error (the probability of rejecting null hypothesis that is true) and is the false-alarm ratio.

Assume one time series whose samples come from normal distribution $N(\theta_1, \theta_2)$, where θ_1 is mean and θ_1 is variance, parameter space is $\Omega=\{(\theta_1, \theta_2); -\infty<\theta_1<\infty, 0<\theta_2<\infty\}$. So, the above hypothesis test can be described as follows:

H_0: $R(n, \triangle t) \sim N(\theta_1, \theta_2)$ and $\theta_1=0$, $\theta_2>0$

H_1: $R(n, \triangle t) \sim N(\theta_1, \theta_2)$ and $\theta_1\neq 0$, $\theta_2>0$

The subset $\omega=\{(\theta_1, \theta_2); \theta_1=0, 0<\theta_2<\infty\}$ of Ω is the parameter space for H_0. It is a composite versus composite test and we use LRT method to do this. The likelihood functions of $R(n, \triangle t)$ on ω and Ω are:

$$L(\omega)= L(0,\theta_2;r_1,r_2,\cdots r_n)=\left(\frac{1}{2\pi\theta_2}\right)^{n/2}\exp\left(-\frac{\sum_{i=1}^{n} r_i^2}{2\theta_2}\right)$$

$$L(\Omega)= L(\theta_1,\theta_2;r_1,r_2,\cdots r_n)=\left(\frac{1}{2\pi\theta_2}\right)^{n/2}\exp\left(-\frac{\sum_{i=1}^{n} (x_i-\theta_1)^2}{2\theta_2}\right)$$

Let $L(\hat{\omega})=\max L(\omega)$ and $L(\hat{\Omega})=\max L(\Omega)$, that is

$$L(\hat{\omega})=\left(\frac{1}{2\pi\sum_{i=1}^{n} r_i^2/n}\right)^{n/2}\exp\left(-\frac{\sum_{i=1}^{n} r_i^2}{2\sum_{i=1}^{n} r_i^2/n}\right)=\left(\frac{ne^{-1}}{2\pi\sum_{i=1}^{n} r_i^2}\right)^{n/2}$$

$$L(\hat{\Omega})=\left(\frac{1}{2\pi\sum_{i=1}^{n} (r_i-\bar{r})^2/n}\right)^{n/2}\exp\left(-\frac{\sum_{i=1}^{n} (r_i-\bar{r})^2}{2\sum_{i=1}^{n} (r_i-\bar{r})^2/n}\right)$$

$$=\left(\frac{ne^{-1}}{2\pi\sum_{i=1}^{n} (r_i-\bar{r})^2}\right)^{n/2} \quad \text{where} \quad \bar{r}=\sum_{i=1}^{n} r_i \Big/ n .$$

So the likelihood ratio is

$$\lambda(r_1,r_2,\cdots,r_n)=\lambda=\frac{L(\hat{\omega})}{L(\hat{\Omega})}=\frac{1}{\left\{1+\left[n(\bar{r})^2\Big/\sum_{i=1}^{n}(r_i-\bar{r})^2\right]\right\}^{n/2}} \quad \text{and}$$

the test will reject H_0 if λ is small. The critical region of λ for H_0 is $0\leqslant\lambda\leqslant\lambda_0$. In another words, we will reject H_0 and accept that there are faults in cluster system when $\lambda\leqslant\lambda_0$. The value of λ_0 is calculated as:

$$\lambda=\frac{1}{\left\{1+\left[n(\bar{r})^2\Big/\sum_{i=1}^{n}(r_i-\bar{r})^2\right]\right\}^{n/2}}\leq\lambda_0 \text{, that's}$$

$$\frac{\sqrt{n}|\bar{r}|}{\sqrt{\sum_{i=1}^{n}(r_i-\bar{r})^2/(n-1)}}\geq\sqrt{(n-1)(\lambda_0^{-2/n}-1)}=c \qquad (6)$$

The left of formula (6) is a random variable marked $t(r_1, r_2, \ldots, r_n)$, and $t(r_1, r_2, \ldots, r_n)$ is a *t*-distribution with n-1 degrees of freedom. For the given *n* and the significance level α, the value of *c* can be found in the *t*-table based on $\alpha=P[|t(r_1, r_2, \ldots, r_n)|\geqslant c; H_0]$.

So $\lambda_0=\left(\frac{c^2}{n-1}+1\right)^{-\frac{n}{2}}$.

For the residual series $R(n, \triangle t)$, we reject H_0 if $\lambda\leqslant\lambda_0$. Otherwise, we accept H_0.

3.3 The fault detection function

For the residual series $R(l, \triangle t)$ in window $W(<o, p, r>_{t+\triangle t-l+1}, <o, p, r>_{t+\triangle t}, l)$, we use LRT method to test whether $R(l, \triangle t)$ is white normal noise with mean zero or not. For $R(l, \triangle t)$ in window $W(<o, p, r>_{t+\triangle t-l+1}, <o, p, r>_{t+\triangle t}, l)$, define the function $m(t+\Delta t)=\begin{cases}1, if(\lambda\leq\lambda_0)\\0, if(\lambda>\lambda_0)\end{cases}$ where λ and λ_0 are defined in section 3.2.

To get rid of the action of noise, we introduce the persistence checking parameter *w* into the Fault Detection Function (FDF). The FDF is defined as $f(s)=\bigcap_{i=s-w+1}^{s} m(i)$. If f=1, one fault alarm decision is made. It is obviously that one fault alarm is triggered iff the number of continuous $m(i)$=1 is *w*.

4. Experiment

In this section, we will prove the utility of our proactive fault detection mechanism by experimental method. In the experiments, the fault injection is used to simulate the fault environment.

4.1 Building the normal running models

The experimental setup in this paper consists of four nodes (*node1, node2, node3, node4*) whose configurations are dual P3 1GHz CPU, 1G Memory, 18GB hard disk and running Linux 2.4.20-8smp kernel. First, three MPI jobs (*App1, App2, App3*) are running on these nodes without faults in turn and the running time is 2400, 2700, 4600 seconds respectively. At the same time, the cluster monitoring system collects the performance parameters of every node at regular intervals and the sample cycle is 5 seconds. Based on the cross correlation analysis, we select 19 primary performance parameters from about 80 performance parameters of Linux operating system and these 19 performance parameters are shown in table 2. We build the normal running models of every one of 19 performance parameters and use these models as the basis of fault detection.

For each $<N, AppM, S>$, where $N \in \{node1, node2, node3, node4\}$, and $AppM \in \{App1, App2, App3\}$, and S is one of performance parameters in table 2, we build the normal running model using the collected data. The follow gives an example: when *App1* is running on *node1* without faults, the normal running model of *activepg* is built using the collected data. The process of building the normal running model of $<node1, App1, activepg>$ is given in Figure 2: (a) is the original sample data, the number is 480; (b) is the data after 31-point moving average and zero average, the average $\mu=1.9832\times10^5$, the front 30 sample data are ignored and the number of remainder data is 450; (c) is the ACF ($k\leqslant50$) for (b) data, the ACF of (b) is not quickly decreasing, so (b) is not one stationary time series; (d) is the data after one pace differencing of (b); (e) and (f) are the ACF ($k\leqslant200$) and PACF ($k\leqslant200$) of (d) respectively, according to table 1, the (d) can be modeled

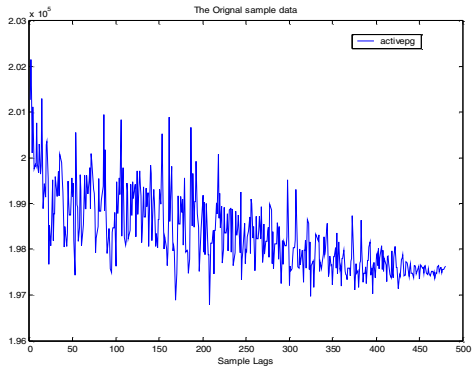
Table 2. Performance parameters

Num	Name	Description
1	File_sz	Number of used file handles.
2	Inode_sz	Number of used inode handlers.
3	Super_sz	Number of super block handlers allocated by the kernel.
4	Dquot_sz	Number of allocated disk quota entries.
5	Rtsig_sz	Number of queued RT signals.
6	activepg	Number of active (recently touched) pages in memory
7	Txpck_ps	Total number of packets transmitted per second
8	kbmemused	Amount of used memory in kilobytes. This does not take into account memory used by the kernel itself.
9	kbswpused	Amount of used swap space in kilobytes.
10	Cswch_ps	Total number of context switches per second.
11	Runq_sz	Run queue length (number of processes waiting for run time)
12	Ldavg_one	System load average for the last minute
13	Percentage_user	Percentage of CPU utilization that occurred while excuting at the user level.
14	Percentage_system	Percentage of CPU utilization that occurred while excuting at the system level.
15	Frmpg_ps	Number of memory pages freed by the system per second. A negative value represents a number of pages allocated by the system (a page has a size of 4 kB or 8 kB according to the machine architecture).
16	Interrupt_ps	Total number of interrupts received per second.
17	Tps	Total number of transfers per second that were issued to the physical disk. A transfer is an I/O request to the physical disk. Multiple logical requests can be combined into a single I/O request to the disk.
18	Txerr_ps	Total number of errors that happened per second while transmitting packets.
19	Txdrop_ps	Number of transmitted packets dropped per second because of a lack of space in linux buffers.

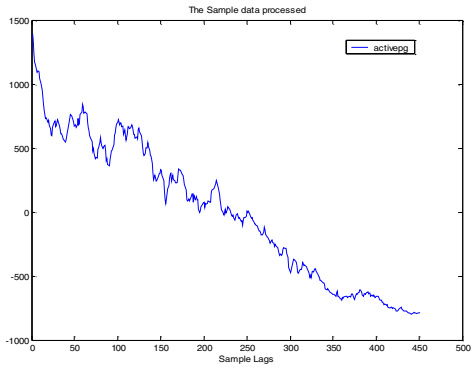
as $ARMA(6,4)$ and $\varphi(B)=1-0.2491B^{-1}-0.004738B^{-2}+0.01776B^{-3}+0.8369B^{-4}-0.3423B^{-5}+0.03303B^{-6}$; $\theta(B)=1+0.07579B^{-1}+0.1311B^{-2}+0.05879B^{-3}+0.9948B^{-4}$; (g) is the residual series whose mean is zero and variance is 580; The QQ plot in (h) shows that the residuals in (g) is white normal noise. In the end, the normal running model of <*node1, App1, activepg*> can be built as $\varphi(B)\nabla^{1}(activepg_t)=\theta(B)a_t$: where $activepg_t$ is the sample of *activepg* at time *t*.

The normal running models of each <*N, AppM, S*> are built by the same means. These models are used in fault detection and the results of fault detection are shown in section 4.2.

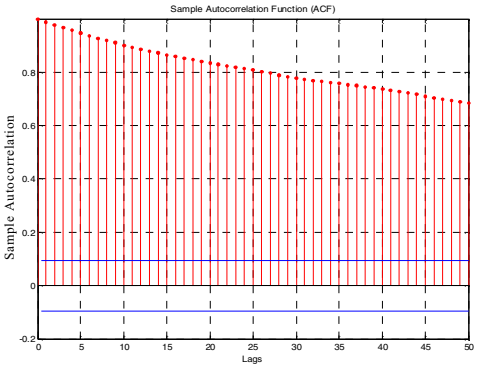
Figure 2. Building the normal running model of <node1, App1, activepg>



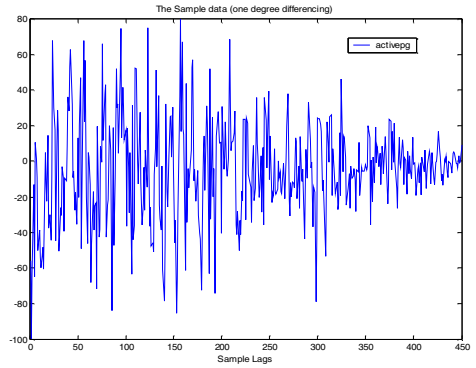
(a) The original sample data



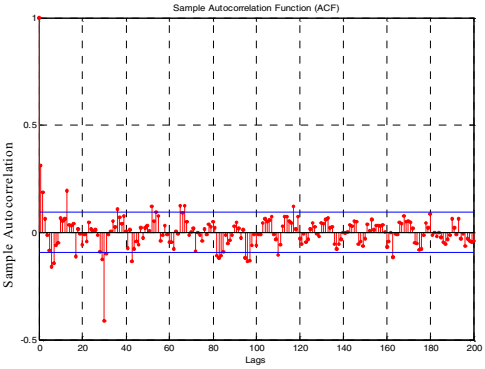
(b) The data after preprocess



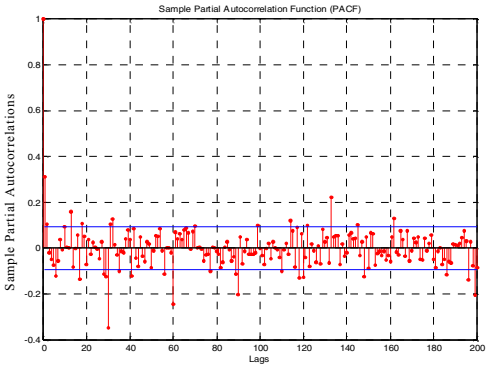
(b) The ACF of (b)



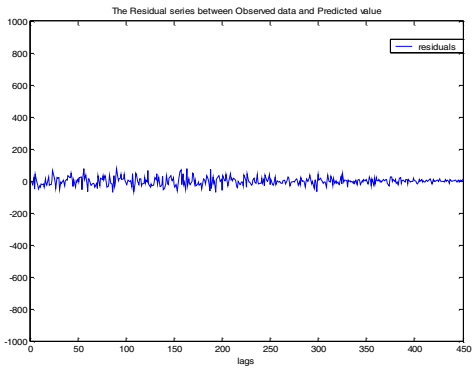
(d) Data after 1-pace differencing of (b)



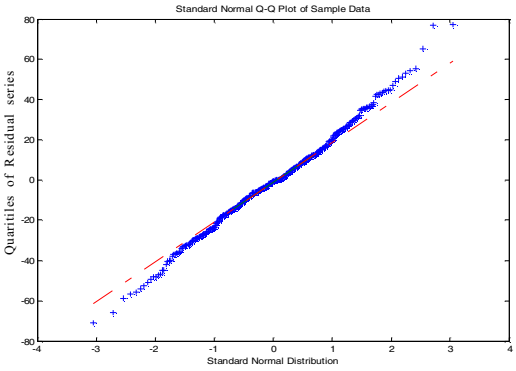
(e) The ACF ($k\leqslant 200$) of (d)



(f) The PACF ($k\leqslant 200$) of (d)



(g) The residuals



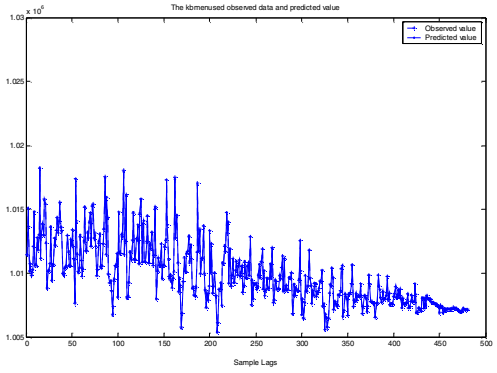
(h) The QQ plot for (g)

4.2 Results of fault detection

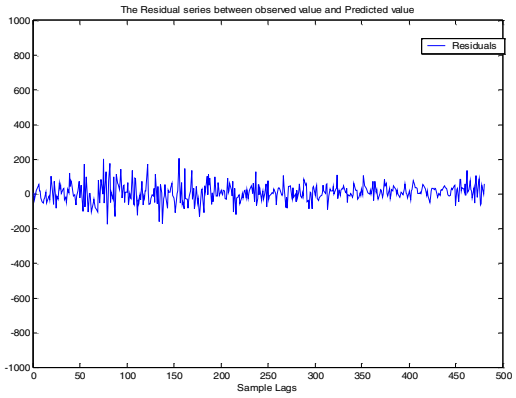
Once the normal running models are built, the fault detection process can be implemented by comparing the current running state with the normal running models. During the experiment, we will record the time when the FDF is 1 and the time when the fault is injected into system. The difference between these two times is the fault detection latency. The latency represents the sensitivity of fault detection mechanism for faults.

The fault injection technique is used to simulate the real-world faults in cluster system. First, run *App1* and inject memory faults on *node1*. The injection time is the 300-th lag (1500 seconds) after *App1* begins. Figure 3 gives the fault detection results based on the

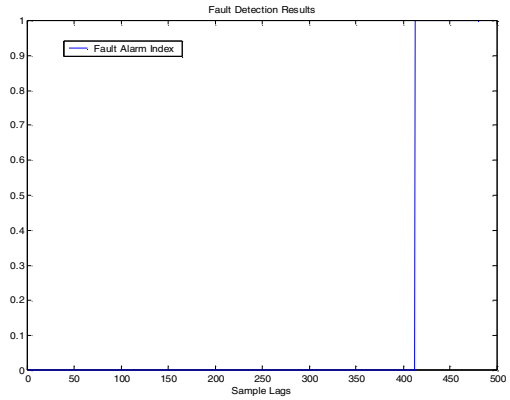
Figure 3. Fault detection results by normal running model of <node1, App1, kbmemused>



(a) kbmemused series



(b)The residual series



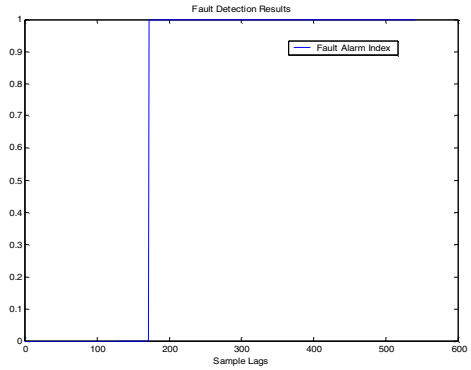
(c) The fault detection function

normal running model of *<node1, App1, kbmemused>*: in (a), the blue is the actual observed value and green is the predicted value by the normal running model; (b) is the residual series between observed value and predicted value; (c) is the results of fault detection function and the persistence

checking parameter w=50, the size of slide window n=120. From (c), the fault detection function is 1 from 413-th lag on. So, the fault detection latency is 113 lags (565 seconds).

Second, run *App2* and inject I/O faults on *node2*. The faults are injected when *App2* begins. Figure 4 gives the fault detection results based on the normal running model of <*node2, App2, Interrupt_ps*>: in (a), the blue is the actual observed value and green is the predicted value by the normal running model; (b) is the residual series between observed value and predicted value; (c) is the results of fault detection function and the persistence checking parameter w=50, the size of slide window n=120. From (c), the fault detection function is 1 from 172-th lag on and the fault detection latency is 172 lags (860 seconds).

The results of first and second experiments show the validity of our fault detection mechanism. But the fault detection latency is high (about 10 minutes), that's to say after the fault exist about ten minutes the

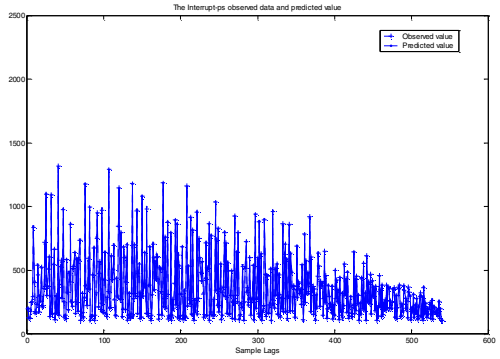


(c) The fault detection function

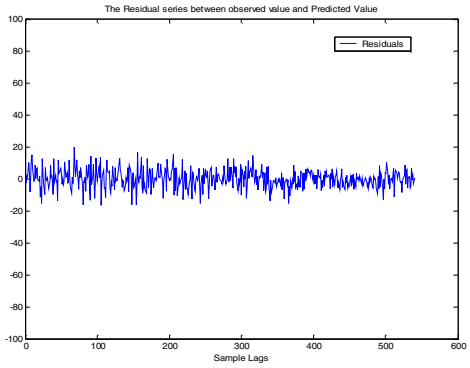
fault alarm just been send out. The main cause for the high latency is the size of slide window and the next experiment results give the relation between latency and the size of slide window.

In the last experiment, run *App3* and inject I/O faults on *node3*. The faults are injected when *App3* begins. Normal running model of <*node3, App3, Cswch_ps*> is $\varphi(B)\nabla^1 s_t = a_t$ where $\varphi(B)=1-0.05439B^{-1}$

Figure 4. Fault detection results by normal running model of <node2, App2, Interrupt_ps>

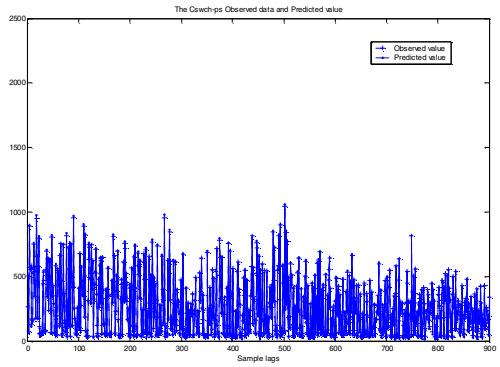


(a) Interrupt_ps series

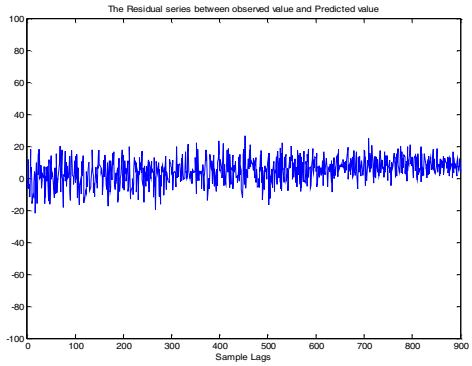


(b) The residual series

Figure 5. The residual series by normal running model of <node3, App3, Cswch_ps>



(a) Cswch_ps series



(b) The residual series

and is used to fault detection. The fault detection results for different size of windows (n=40,120,160 and w=50) show the impact of n on fault detection latency. In figure 5, (a) shows the observed value (blue) and the predicted value (green); (b) is the residual series.

Table 3 gives the fault detection results for different size of windows (n=40,120,160 and w=50). The results show that the fault detection latency decreases while the size of window increases. The main reason is that the LRT method is more sensitive with a larger sample space.

Table 3. The fault detection latency for different size of windows

The size of window (n)	Fault detection latency (lags, 1 lags = 5 seconds)
40	347
120	272
160	236

5. Conclusion

In large-scale cluster system, the fault detection before node failure is important. This paper proposes an proactive online fault detection method using the information collected from cluster monitoring system. First, building the cluster normal running models of performance parameters using the activity data when cluster runs without faults; Second, the difference between actual observed value and predicted value by normal running model is residual and use the residual time series as the symptom of fault; When the cluster system is behaving normally, all the residual series are white noise series; In the end, using the LRT method to test whether the residual series are white noise series and the fault detection function f is constructed based on the results of LRT and the persistence checking parameter w. If f=1, one fault alarm decision is made. The experiment results show that the method in this paper can detect the fault in system in good time and provides the ability to proactive fault management.

But, the method in this paper is just a fault detection and not fault diagnosis. If abnormal behavior is detected, the next step is locating the accurate cause of fault. In the future work, we will pay more attention on two questions:

- Extract the more sensitive symptom of fault in cluster system. In this paper, only using the residual series of performance parameters as the symptom of faults. In the future, we will select the more sensitive symptom for faults and reduce the fault detection latency, reduce the computation complexity at same time.
- Online fault diagnosis method. Finding the root cause of fault and give the correctly healing advice to system administrator.

6. References

[1]. Gao Wen. The design and analysis method of high availability in the server consolidation system: *[Ph.D. dissertation]. Beijing: Institute of Computing Technology, Chinese Academy of Sciences*. 2001.

[2]. Vittorio Castelli, Richard E. Harper, Philip Heidelberger, Steven W. Hunter, Kishor S. Trivedi, Kalyanaraman Vaidyanathan, William P. Zeggert: Proactive management of software aging. *IBM Journal of Research and Development* 45(2): 311-332 (2001)

[3]. R. Sahoo, A. Oliner, I. Rish, M. Gupta, J. Moreira, S. Ma, R. Vilalta, A. Sivasubramaniam. Critical Event Prediction for Proactive Management in Large-scale Computer Clusters. *In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* , pages 426-435, August 2003.

[4]. D. Dasgupta, K. KrishnaKumar, D. Wong, M. Berry: Negative Selection Algorithm for Aircraft Fault Detection. *3rd International Conference on Artificial Immune Systems Catania, Sicily.(Italy)* September 13-16 2004.

[5]. J. Wesley Hines, D. W. Miller and B. K. Hajek. Merging Process Models With Neural Networks for Nuclear Power Plant Fault Detection and Isolation, *The proceedings of the 9th Power Plant Dynamics, Control and Testing Symposium*, Knoxville, TN, 1995.

[6]. Y. Huang, C. Kintala, N. Kolettis, N.D. Fulton. Software rejuvenation: analysis, module and applications.

Proc. 25th International Symposium on Fault-Tolerant Computing, June 1995, pp. 381 – 390.

[7]. S. Garg, A. van Moorsel, K. Vaidyanathan, and K. Trivedi, A Methodology for Detection and Estimation of Software Aging, *Proceedings of the 9th International Symposium on Software Reliability Engin*eering, Paderborn, Germany, November 1998, pp. 282–292.

[8]. Lei Li, Kalyanaraman Vaidyanathan, Kishor S. Trivedi: An Approach for Estimation of Software Aging in a Web Server. *Proc. 2002 Int'l Symp. on Empirical Software Eng.*, pp.91–100, IEEE CS Press, 2002.

[9]. Hughes, G.F., Murray, J.F., Kreutz-Delgado, K. and Elkan, C. Improved Disk Drive Failure Warnings. *IEEE Transactions on Reliability*, September 2002.

[10].J. F. Murray, G F Hughes, "Hard Drive Failure Prediction Using Non-parametric Statistical Methods" (coauthor), *International Conference on Artificial Neural Networks*, Istanbul, June 26-29, 2003.

[11].Greg Hamerly and Charles Elkan. Bayesian approaches to failure prediction for disk drives. *In Proceedings of the eighteenth international conference on machine learning*, pages 202-209. Morgan Kaufmann, San Francisco, CA, 2001.

[12].J. Moreira, "System Management in The BlueGene/L Supercomputer", *Proc. Int'l. Parallel and Distributed Processing Symposium*. Los Alamitos, CA, IEEE Computer Society. 2003, p. 8., April 2003.

[13].Top 500 Superconputer Sites. System Info: Dawning 4000A, Opteron 2.2 GHz, Myrinet. http://www.top500.org/sublist/System.php?id=7036. 2004.6

[14].Chen Yi, Meng Dan, Zhan Jian-Feng, Zen Ning, Design and implement of federation based trading service, *Computer Engineering and Applications*, 2004.

[15].Meng Dan, Zhan Jianfeng, Wang Lei, Tu Bibo, Zhang Zhihong. Fully integrated cluster operating system: Phoenix. *Journal of Computer Research and Development*, 2004.6.