2009 International Conference on Computer Engineering and Applications
IPCSIT vol.2 (2011) © (2011) IACSIT Press, Singapore

Study on the Lightweight checkpoint based rollback recovery mechanism

Zhang Lei [1,3] +, Wang Rui [2] Dai Hao[3], Ma Mingkai[3] and Li Xianghong[4]

[1] Institute of Command Automation PLA University of Science and Technology
[2] Train base of General Staff Team 5
[3] Institute of China Electronic Equipment System Engineering Company
[4] Bureau of Xinhua News Agency Communication Technology

Abstract. The cluster service recovery mechanism is very important in the study of survivable server cluster. After analyzing the related work, this paper point out the deficiency of the existed recovery mechanisms, and propose the Lightweight checkpoint based rollback recovery mechanism that aims to enhance the recovery performance, the experiment result validate that the performance excelled existed methods.

Keywords: Rollback recovery, Cluster, Service survivability.

1. Introduction

With the development of the network technology and the increase of users' requirement, the continuity of the network service provided by the cluster must be enhanced. Some applications such as electronic business require very high quality of service, so the survivable server cluster system must have not only the ability of resistance, but also the ability of recovery when necessary. The key application data should be integrated and the recovery mechanism should be transparent to the users, finally the application program affected very little.

Checkpoint based Rollback Recovery (CRR) method is a common fault-tolerant technology in software/hardware systems. CRR [1] means that the system or the process start to run anew from the right point of the status which has been saved before the fault time. In the server cluster systems, it is necessary of the checkpoint information to recovery the fault service. The reasons that cause the service interruptive may be different in the study of fault-tolerant and survivability, but the service survivability study doesn't focus on the causations, so the CRR also can be used in the service survivability study. Although the rollback recovery technology provided the reliability, it increases the complexity of the system.

Checkpoint [2] is the copy of the application program status, which is saved in the steady storage; we assume that the steady storage never fails. The checkpoint information can also be sent to the storage of the other cluster node in order to recovery the service in that node, this method is usually used in the cluster system and decrease the recovery time.

2. Checkpoint Mechanism

2.1. Checkpoint Information

It is flexible to set checkpoint, some applications can recovery from very little information, some other applications can recovery from the special checkpoint information such as special time, and the information

+ Corresponding author. Tel.: +861066820285; fax: +861066820258.
E-mail address: zhanglei_isme@163.com.

may be very small. To decrease the overhead of the program, we can decrease the number of checkpoint or the size of the checkpoint files.

In the ordinary application programs, checkpoints need to save these contents as follows[3]: (1) process data field, content in the user stack; (2) context related items, include program counter, process or status register, status point, etc; (3) active file information, include file descriptor, access mode, file size, read/write point, etc; (4) related signal information, include shield code, stack point, process function handler, and suspended signal flag; (5) user file content, register content. These content of the process status can be used in single point server, but in the server cluster systems, the checkpoint information need include: (6) cluster view of the server node; (7) backup and interplay information among cluster nodes; (8) the time and content when receive/send messages; (9) special information related to the application.

We improve the traditional checkpoint methods, we record as least information as possible when setting checkpoint, but these information can be used when recovery the process. For example, for the send and receive message pair, we only record one of them, when the program runs correctly, the other message is also correct. We save the checkpoint in data field, like table 1:

Table1: lightweight checkpoint information data field

Exec_state	Pro_ctl	Mem_add	Msg_state
File_state	Cluster_view	Reserved	

There are seven fields in the lightweight checkpoint, these content are as follows:

Exec_state: process running status; include context transition information, register and stack point information;

Pro_ctl: process control status, process id, process relationship;

Mem_add: memory and address status; include content in the stack, data field related to the process;

Msg_state: record of sending message and message in cache;

File_state: the file state, include the file list which process opened, handler and cache information;

Cluster_view: the status of the cluster, include the node role, backup and interplay information among cluster nodes;

Reserved: some field to save something related to the special applications.

These are the whole content of the lightweight checkpoint; the first three items can recovery a process sometimes, so this can hardly reduce the burden which checkpoint bring on.

2.2. Set checkpoint

In the early days, when to set checkpoint, the running process must be suspended [4] during this time, the process continue to run after finishing the checkpoint. This method guarantee the checkpoint information coherent among the whole cluster, but it influenced the usual process running, and the overhead is also large, it is not fit for these applications that have many requires and large datum. In order to prevent the checkpoint mechanism to influence the usual applications, some researchers build new thread to set checkpoint specially, this thread can communicate with the main process, and record the status of every thread in the checkpoint time, save the messages, make the checkpoint file as least as possible. Meantime, other threads can continue running the usual processes.

Based on the BLCR [5] (Berkeley Lab's Linux Checkpoint/Restart), we improve the setting policy of the checkpoint, add the lightweight mechanism, and reduce the overhead of the checkpoint. In order to guarantee the reliability of the recovery method, the storage must be absolutely reliable.

There are two phases of setting the lightweight checkpoint, in the first phase, the application initialize with a thread recall function, this recall function generate a recall thread, when to set checkpoint, this thread stop blocking, the other threads in the process continue to run their own program, the file handler is used to save checkpoint information. When the last thread recalls to the cr_checkpoint status, the recall function return to the kernel status, and the process enter the second phase.

In the second phase, system calls send checkpoint signal to other threads, after all the threads entering kernel status, the checkpoint information begins to be recorded. After the threads write their own status into the checkpoint file, the system returns to the user space, continue executing other program codes.

2.3. Rollback recovery

Using the checkpoint information recovery the application process, we mainly rely on the ioctl call to read checkpoint file data. The rollback recovery means to re-execute process from the last checkpoint time when the system fails.

Under Linux system, first recall the function do_fork() to generate process, then using clone() to generate all threads that the process needed, startup recall process to run these threads. The first thread reads the checkpoint file, recovery the shared items and its own id, register and signal status, other threads gradually recovery the saved status, at last, recovery the relationship of these threads, then, the process returns from the kernel status, after all the processes recovery, other application codes begin to execute.

3. Optimal Checkpoint Intervals

When there are fewer failures, and the running environment is good, setting too many checkpoints will influence the usual application performance. On the other hand, if there are high load, and the probability of failure will be high, small checkpoint intervals may bring on large recovery overhead, because the process must re-execute far from the failure time. We will decide the optimal checkpoint intervals in this section.

3.1. Time spending of setting checkpoint

We consume that the server node failures in the cluster present a Poisson distributing, the failure probability is λ. The time spending of a single checkpoint under no failure conditions is CS; the checkpoint duration is L. The time spending of recovery is CR. Notice that the CS and CR are constant as to the special application. In the period of a checkpoint, the program executing time increases T+CS, the time spending is CR+(L-CS)+(T+CS)=CR+T+L.

Consume that the application executing time is W, under the checkpoint mechanism, the average real executing time of the program is T(W), the task is divided by several checkpoints, denotes by K, the average time spending of the checkpoint is t, so T(W)=K*t.

Service duration C can be calculated as: C=W/T(W).

3.2. Status transformation

There are all two kinds of status in the cluster nodes, one is normal status, running the process, include setting the checkpoint, another status is failure, the process rollback to recovery. Figure 2 is the status transformation.

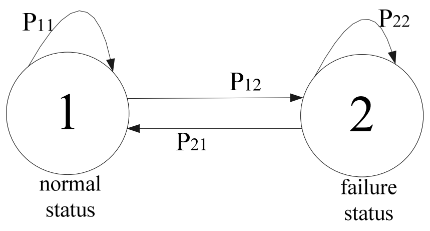


Figure 1. status transformation

In figure 1, status 1 is the normal status, and status 2 is the failure status, the letters on the arrow side are the transformation probability. The two status transform model reduce the complexity of the system status, and it is easy to program.

3.3. Optimal checkpoint intervals

Expected checkpoint intervals can be solved using the two-state discrete Markov chain presented in figure 1. As to the Poisson process, the transition rate is λ, in time t, the rate of events are λt, x(t) denotes the numbers of the events, then

$$p(x(t)=k)=\frac{(\lambda t)^k}{k!}e^{-\lambda t} \quad ,k=0,1,2\cdots \tag{1}$$

Each transition(x, y), from state X to state Y in the Markov chain, has an associated transition probability P_{xy} and a cost W_{xy}, these variables can be calculated as follows one by one.

$$P_{11}=e^{-\lambda(T+CS)} \tag{2}$$ $$W_{11}=T+CS \tag{3}$$ $$P_{12}=1-P_{11}=1-e^{-\lambda(T+CS)} \tag{4}$$

According to the transition probability P_{12} and condition probability, we can get the time cost W_{12} as follows:

$$\begin{aligned} W12&=\int_0^{T+CS} t\cdot\frac{\lambda\cdot e^{-\lambda t}}{1-e^{-\lambda(T+CS)}}dt \\ &=[1-e^{-\lambda(T+CS)}]^{-1}\cdot\int_0^{T+CS}-t\cdot de^{-\lambda t} \\ &=\frac{1}{\lambda}-\frac{(T+CS)e^{-\lambda(T+CS)}}{1-e^{-\lambda(T+CS)}} \end{aligned} \tag{5}$$

$$\begin{aligned} W22&=\int_0^{T+L+CR} t\cdot\frac{\lambda\cdot e^{-\lambda t}}{1-e^{-\lambda(T+L+CR)}}dt \\ &=[1-e^{-\lambda(T+L+CR)}]^{-1}\cdot\int_0^{T+L+CR}-t\cdot de^{-\lambda t} \\ &=\frac{1}{\lambda}-\frac{(T+L+CR)e^{-\lambda(T+L+CR)}}{1-e^{-\lambda(T+L+CR)}} \end{aligned} \tag{9}$$

$$P_{21}=e^{-\lambda(T+L+CR)} \tag{6}$$ $$W_{21}=T+L+CR \tag{7}$$

$$P_{22}=1-P_{21}=1-e^{-\lambda(T+L+CR)} \tag{8}$$

From state 2 to state 1, the process finishes the rollback recovery, the average time spending is T+L+CR, note that CR is the average recovery time, so we get formula (5)-(9). Now, we get all the average time spending of the state transition, let Ψ denote the average application process running time, according the time spending and their probability, we get Ψ as follows,

$$\begin{aligned} \Psi&=P11\cdot W11+P12(W12+\frac{P22}{1-P22}W22+W21) \\ &=\frac{1}{\lambda}(1-e^{-\lambda(T+CS)})\cdot e^{\lambda(T+L+CR)} \\ &=\frac{1}{\lambda}e^{\lambda(L-CS+CR)}(e^{\lambda(T+CS)}-1) \end{aligned} \tag{10}$$

$$\gamma=\frac{1}{\lambda T}e^{\lambda(L-CS+CR)}(e^{\lambda(T+CS)}-1) \tag{11}$$

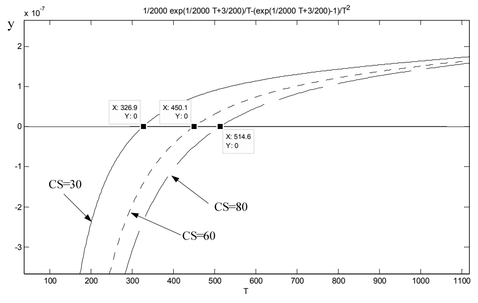
We introduce checkpoint time efficiency $\gamma=\frac{\Psi}{T}$; note that T is the average checkpoint intervals, then

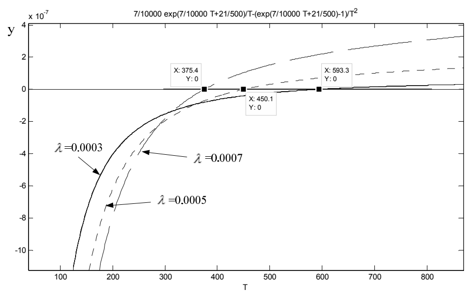
The proper T should satisfy the following formula(12) and (13),

$$\begin{aligned} \frac{\partial\gamma}{\partial T}&=\frac{\partial}{\partial T}[\frac{1}{\lambda T}e^{\lambda(L-CS+CR)}(e^{\lambda(T+CS)}-1)]=0 \\ &\Rightarrow\frac{\partial}{\partial T}[\frac{e^{\lambda(T+CS)}-1}{T}]=0 \end{aligned} \tag{12}$$

$$\begin{aligned} \frac{\partial\gamma}{\partial T}&=\frac{\lambda Te^{\lambda(T+CS)}-e^{\lambda(T+CS)}+1}{T^2}=0 \\ &\Rightarrow(\lambda T-1)e^{\lambda(T+CS)}+1=0 \end{aligned} \tag{13}$$

We use ezplot() function in Matlab program to figure the relationship between T and y, note that y is $\frac{e^{\lambda(T+CS)}-1}{T}$, we get the result





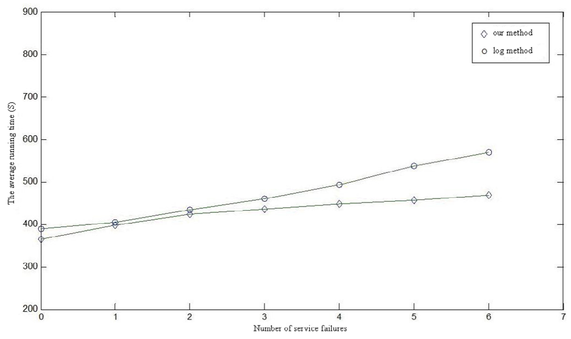
(a) (b)

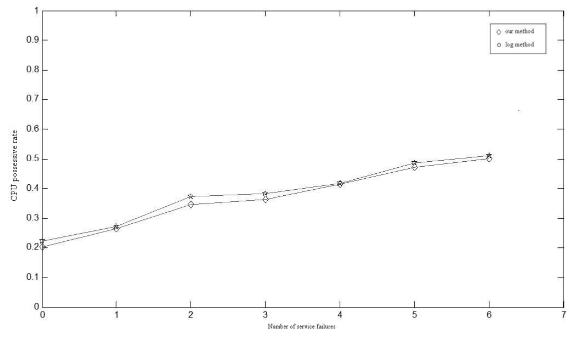
Figure 2. the optimal checkpoint intervals

Figure 2(a) is the differential coefficient function of y, CS are 30, 60, 80 respectively, the proper T is the point where y is zero, we conclude that as the CS increase, the proper T is increasing. From figure 2(b), we conclude that as λ increase, the proper T is decreasing, so the proper T should be decided according to the special conditions.

4. performance study

In order to evaluate the performance of the lightweight checkpoint mechanism, we utilize the Charm++ language to edit the parallel system, and implement the checkpoint based rollback recovery method. We choose net-Linux, the compiler is GNU. This stage can call the initial program of the operation system; the method of the paper is embedded in the application program to evaluate its performance.





(a) (b)

Figure 3. recovery time performance comparing

Figure 3(a) is the comparing result of recovery method and the traditional log based method, under several system failures, the average running time of lightweight checkpoint based rollback recovery is smaller than the log based method. When λ is constant, the recovery method in this paper is efficient than the pessimistic log based recovery method, not only the average running time is smaller, but also the CPU resource is economic.

5. Conclusion

We improved the traditional checkpoint based recovery method, pre-digested the checkpoint file, this made the checkpoint setting process and recovery easy and efficient to perform. The recovery process can choose the lightweight checkpoint information field data, which can reduce the recovery time very much. The experiment result showed that this method consume less materials and more efficient.

6. References

[1] E. N. Elnozahy, L. Alvisi, Y. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. ACM Comput. Surv. 34(3): 375-408, 2002.

[2] Vaidya, N.H., Impact of checkpoint latency on overhead ratio of a checkpointing scheme, IEEE Transactions on Computers, Volume 46, Issue 8, Aug. 1997: 942-947.

[3] Logging Service, http://docs.sun.com/source/816-6687- 10/logging.html. 2008-12-30.

[4] Tannenbaum T, Litzkow M. The Condor Distributed Processing System [J]. Dobb's Journal, 1995, 20(2): 40-48.

[5] J. Duell, P. argrove, E. Roman, "The Design and Implementation of Berkely Lab's Linux Checkpoint/Restart", 2002-10-13.