基于数据差异的CDP邻近时间点恢复

侯利曼 李战怀 胡 娜
(西北工业大学计算机学院 西安 710129)

摘 要 块级CDP系统无法提供有明确语义信息的可恢复时间点,用户难免需执行多次恢复才能获得有效数据。若对每次恢复都使用传统算法,会耗费大量时间与开销。目标时间点相邻较近的两次恢复,其有效数据间存在少量差异。将邻近时间点恢复划分为4种类型,给出一种用"位表"标记差异数据块、把多余数据剔除、缺少数据写入的"差异算法"。原型实验表明,该算法能够提供正确的邻近时间点恢复,其效率优于传统算法,且差异数据量更小,效率更高。

关键词 持续数据保护,邻近时间点,差异算法,位表
中图法分类号 TP334.5 **文献标识码** A

Neighboring Point Data Recovery for CDP Based on Data Gap

HOU Li-man LI Zhan-huai HU Na
(Department of Computer Science, Northwestern Polytechnical University, Xi'an 710129, China)

Abstract Block-level CDP system can't give clear recovery points with semantic information. Users have to execute data recovery several times to get correct data. Traditional algorithm for each data recovery will be time consuming and performance overhead costing. Target data of two neighboring point data recovery had little data gap. This paper divided neighboring point data recovery into four types, and presented a gap algorithm which eliminated more data and wrote lost data using bits table. It is proved by prototype experiment that the algorithm is correct and the less gap data, the more efficiency.

Keywords Continuous data protection, Neighboring point, Data gap algorithm, Bits table

随着计算机应用的发展,数据作为信息的载体,其存储媒介从纸张演变为电磁介质,大大方便了人们对信息的存取和携带。但在各种灾难面前,数据安全仍面临严峻挑战。持续数据保护技术(Continuous Data Protection,CDP)是一种专门应对病毒、人为误操作和软件故障等逻辑灾难(占数据损失原因的60%~80%)[1]的容灾技术。该技术持续捕捉对目标数据的更新操作,为每个更新数据打上时间戳,利用I/O书签(I/O bookmark)[2]或称为时间可寻址存储(Time Addressable Storage)技术备份起来;灾难发生后,利用时间点版本与备份记录间的映射找到记录,并依次写入目标数据,从而实现任意时间点的恢复。这种方法消除了备份窗口,实现了无缝的业务连续性。

持续数据保护技术可实现于数据块级、文件系统级或应用级别。基于块的数据保护系统可直接操作数据块,其备份和恢复效率都高于后两种技术[3],并具有与上层应用无关的优点,可构建面向多平台的CDP引擎,满足多种文件系统或数据库系统的数据保护需求。但这种实现的弊端是无法获得上层请求的语义信息,每个请求对于块设备来说,只有读写区别,这样也就无法提供具有明确语义信息[4]的可恢复时间点,故无法保证用户仅执行一次恢复操作即可获得目标数据,难免需要多次恢复。若对每次恢复都采用将目标时间点之前的备份记录写入目标卷的方法,随着备份时间的增加,恢复涉及的数据量也越多,这会耗费大量时间和读写开销。分析发现,这些恢复目标时间点具有相邻较近的特点,意味着相邻恢复的有效数据间存在大量冗余即少量差异。

本文的贡献是通过消除数据差异的方法实现邻近时间点的快速恢复。给出邻近时间点的4种划分,分析每种类型的恢复目标卷的数据差异,用"位表"标记差异数据块,将多余数据剔除,缺少数据写入的差异算法实现恢复,从而减少读、写开销,降低RTO[5](Recovery Time Object),提高恢复效率。

1 传统数据恢复

1.1 数据备份对象

备份的目的就是为了在灾难发生时能够进行数据恢复,减少数据损失。数据恢复算法是保证持续数据保护技术具有低RTO的关键。CDP系统的数据恢复算法与备份对象有关。常见的CDP备份对象有3种:新数据、旧数据和异或数据。3种备份对象各有优劣。与备份新数据相比,备份后两种数据会对每个更新请求增加一次读旧数据的开销,故备份时对上层应用影响较大。

1.2 传统数据恢复算法[6]

本文选择备份新数据(基准参考数据模式[6])的方法实现持续数据保护。在系统创建时,需在备份卷上保存目标卷的初始全备份作为恢复的基础数据;备份过程中维护一个线性

到稿日期:2010-06-03 返修日期:2010-08-19 本文受国家863重大项目(2009AA01A404)资助。
侯利曼(1986—),女,硕士生,主要研究方向为数据存储技术、容灾技术,E-mail:mlhhlm@126.com;李战怀(1961—),男,博士,教授,主要研究方向为数据库理论与技术、网络存储等;胡 娜(1985—),女,硕士生,主要研究方向为数据存储技术。

的时间戳数据链(Time-Data List,TD链),记录更新日志。数据损坏后,系统根据用户提供目标时间点,将TD链上的有效数据写入目标卷,即可实现任意时间点的数据恢复。

上述方法即本文所述的传统恢复算法,假设恢复的目标时间点为 T_{target},该恢复算法的步骤如下:

①将初始全备份写入目标数据卷。

②假设 T_{first} 是TD链的初始备份时刻,TD链上满足 $T\in[T_{first},T_{target}]$ 的 T 构成一个时间升序的集合 $\{T_0,\cdots,T_i\}$,将对应的数据集合 $\{D_0,\cdots,D_i\}$ 中的所有元素按照 $0\rightarrow i$ 的顺序写入目标卷,重现更新过程,即可实现数据恢复。

为便于描述,假设初始全备份是TD链上的第一个节点,时间戳为 T_{base},那么该算法的读、写开销均为区间 $[T_{base},T_{target}]$ 所对应的数据量。

2 差异算法的关键技术

通过消除数据差异实现恢复的算法称为"差异算法"。假设目标数据卷在 T_1,T_2 时刻的数据状态分别为 S_1 和 S_2,S_2 相比 S_1 的数据差异有3种类型:多数据、少数据以及两者共存。若要将 S_2 变为 S_1,需把缺少数据写入目标卷,多余数据从目标卷剔除。故差异算法的关键技术即是数据写入、数据剔除。

2.1 数据写入

假设 S_2 比 S_1 少TD链上时间满足 $T\in(T_2,T_1]$ 所对应的数据 $D_{(T_2,T_1]}$,最简单消除差异的方法即按 $T_2\rightarrow T_1$ 顺序遍历TD链,将数据依次写入目标卷。但此方法的弊端是:无法避免对同一数据块的冗余写操作。假设对数据块 B 执行写操作的时间集合为 $SetT$,那么对 B 有效的写操作的时间 T 满足

$$T=\max(\{T\mid T\in SetT \text{ 且 } T\in(T_2,T_1]\})$$

由此式可看出,针对每个数据块只有距离目标时间点最近的写操作是有意义的。故可按时间逆序(递减)遍历TD链,用位表标记已写数据块,对每块仅执行一次写操作,从而提高写数据效率。此位表称为"覆写位表"。写入算法如图1所示,步骤如下:

①构建并初始化"覆写位表"。

②按 $T_1\rightarrow T_2$ 逆序遍历TD链数据,写前查看该块是否已写,对未写块执行写操作。写后在"覆写位表"中标记,当位表全被标记或遍历到达 T_2 时刻时,写入完成。目标卷从 S_2 变为 S_1 状态。

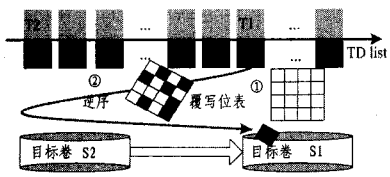

图1 "覆写位表"写数据

该算法的读开销为区间 $(T_1,T_2]$ 所对应的数据量 $D_{(T_1,T_2]}$。由于"覆写位表"对每个数据块仅执行一次写操作,故算法的写开销与数据在各数据块的分布有关。一般情况下差异数据量较小,故后文分析数据写入的写开销时按最大值记录,即 $D_{(T_1,T_2]}$。

2.2 数据剔除

假设 S_2 比 S_1 多TD链上时间满足 $T\in(T_1,T_2]$ 所对应的数据 $D_{(T_1,T_2]}$,要还原到状态 S_1 需将该数据剔除。目标卷不是由多个更新数据堆积的表层视图,而是覆盖写入的最终状态,故剔除的思想是将 $D_{(T_1,T_2]}$ 所对应的数据块在 $T1$ 时刻相应的数据重新写入目标卷。算法如图2所示,步骤如下:

①找出 $D_{(T_1,T_2]}$ 所对应的数据块:按 $T_1\rightarrow T_2$ 顺序遍历TD链,通过时间、数据和块号的映射关系,获得 $D_{(T_1,T_2]}$ 所覆盖的数据块号集合 Set_B。为便于操作,用位表记录 Set_B 中的元素,此位表称为"剔除位表"。

②将 T_1 时刻相应的数据写入目标卷:按 $T_1\rightarrow T_{base}$ 逆序遍历TD链,若数据所对应的块号 B_D 满足 $B_D\in Set_B$,则把该数据写入目标卷,B_D 从 Set_B 中删除。期间,若 $Set_B=\emptyset$,则恢复结束,否则执行至TD链 T_{base} 点。

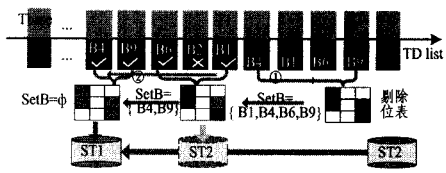

图2 "剔除位表"剔除数据

为便于后文描述,称第①步为构造剔除位表,第②步为重写剔除块。构造剔除位表需读数据 $D_{(T_1,T_2]}$。重写剔除块的写开销为剔除数据量(与 $D_{(T_1,T_2]}$ 的数据量相同),而读开销与数据在各个剔除数据块的分布有关,最差需读取区间 $[T_{base},T_1]$ 所对应的数据,如表1所列。

表1 数据剔除的读写开销

数据所在时间区间		读开销	写开销
构造剔除位表		$(T_1,T_2]$	0
重写剔除块	最优	$(T_1,T_2]$	$(T_1,T_2]$
	平均	$[T_{base},T_1]/2$	$(T_1,T_2]$
	最差	$[T_{base},T_1]$	$(T_1,T_2]$

3 邻近点数据恢复

用户使用CDP系统恢复数据时,难免需执行多次恢复操作。理想情况下,这些操作的目标时间点相邻较近,执行恢复操作的时间点也相邻较近。已知时间间隔 ΔT,其与对应目标卷的数据差异 ΔD 有如下关系:$\Delta D=\lambda\Delta T$ 且 $\lambda\geqslant 0$(λ 即平均写速率,当 ΔT 内没有写操作,$\lambda=0$)。若 $\lambda\neq 0$,ΔD 随 ΔT 增长而增长。当 ΔT 较小时,ΔD 也较小,故后续恢复没必要执行传统恢复算法,否则会将大量重复数据写入目标卷,增加读、写开销。而差异算法仅涉及差异数据 ΔD,故恢复效率较高。

定义1 CDP系统执行一次恢复操作后,经过 $\Delta T'$ 时间,系统再次执行恢复操作,称这次恢复为前一次恢复的"邻近时间点数据恢复"。假设两次恢复目标时间点间隔为 ΔT,理想情况下,$\Delta T'$ 和 ΔT 皆远小于系统正常运行的时间。

定义2 当相邻恢复操作间即 $\Delta T'$ 时间内有写操作时,称为"有间隙",反之称为"无间隙"。

为便于分析,有以下约定:$\langle T'\rightarrow T\rangle$ 表示一次数据恢复,即从 T' 时刻恢复至 T 时刻;$\langle T_1'\rightarrow T_1\rangle$ 为前一次恢复,$\langle T_2'\rightarrow T_2\rangle$ 为邻近恢复。由定义1得 $T_1'<T_2'$,$T_1\neq T_2$。假设 ST 表示目标数据卷在 T 时刻的数据状态,由定义2得,若"无间隙",则 $S_{T_1}=S'_{T_1}=S'_{T_2}$,3个时刻的数据状态一致;若"有间隙",则 $S_{T_1}=S'_{T_1}=S'_{T_2}-D_{(T_1',T_2')}$,$S'_{T_2}$ 比 S_{T_1} 和 S'_{T_1} 多了"间隙"数据。

邻近时间点恢复根据是否有"间隙"划分为两种类型。每

种类型根据两次恢复的目标时间点前后差异又可细分为：①时间点提前；②时间点推后。因此共需研究4种类型。

差异算法分为两步：①获得 S'_{T_2}，S_{T_2} 两状态的差异数据 ΔD；②用上文给出的关键技术，去除数据差异。针对每种邻近点恢复，后文均给出其与传统算法的读写性能比较。

3.1 无间隙后邻近点恢复

定义3 当相邻恢复无间隙，且邻近恢复较前一次恢复目标时间点推后，即 $T_1<T_2$ 时，称为“无间隙后邻近点恢复”。

由于 $T_1<T_2$，故 S_{T_2} 比 S_{T_1} 多区间 $(T_1,T_2]$ 的数据，即

$$S_{T_2}=S_{T_1}+D_{(T_1,T_2]}\Rightarrow\Delta D=+D_{(T_1,T_2]}$$

恢复方法调用2.1节的数据写入算法构建“覆写位表”，将两状态的差异数据 $D(T_1,T_2]$ 写入目标卷。

由表2可知，差异恢复算法提高了读、写效率，且 $D_{(T_1,T_2]}$ 数据量越小，效率越高。

表2 无间隙后邻近点恢复读写开销比较

数据所在时间区间	读开销	写开销
传统算法	$[T_{base},T_2]$	$[T_{base},T_2]$
差异算法	$(T_1,T_2]$	$(T_1,T_2]$

3.2 无间隙前邻近点恢复

定义4 当相邻恢复无间隙，且邻近恢复较前一次恢复目标时间点提前，即 $T_1>T_2$ 时，称为“无间隙前邻近点恢复”。

由于 $T_1>T_2$，故 S_{T_2} 比 S_{T_1} 少区间 $(T_2,T_1]$ 的数据，即

$$S_{T2}=S_{T1}-D_{(T_2,T_1]}\Rightarrow\Delta D=-D_{(T_2,T_1]}$$

恢复方法调用2.2节的数据剔除算法顺序遍历区间 $(T_2,T_1]$ 构造剔除位表，逆序遍历区间 $[T_{base},T_2]$ 重写剔除块，将差异数据 $D_{(T_2,T_1]}$ 从目标卷剔除。

基于邻近时间点的假设，T_1 与 T_2 相邻较近，满足 $D_{(T_2,T_1]}<<D_{[T_{base},T_2]}$ 的可能性较大，故最差情况下，用读、写 $D_{(T_2,T_1]}$ 代替写 $D_{[T_{base},T_2]}$，性能更优，$D_{(T_2,T_1]}$ 越小，性能越优。读写开销如表3所列。

表3 无间隙前邻近点恢复读写开销比较

数据所在时间区间		读开销	写开销
传统算法		$[T_{base},T_2]$	$[T_{base},T_2]$
差异算法	最优	$2(T_2,T_1]$	$(T_2,T_1]$
	平均	$[T_{base},T_2]/+(T_2,T_1]2$	$(T_2,T_1]$
	最差	$[T_{base},T2]+(T_2,T_1]$	$(T_2,T_1]$

3.3 有间隙后邻近点恢复

定义5 当相邻恢复有间隙，且邻近恢复较前一次恢复目标时间点推后，即 $T_1<T_2$，称为“有间隙后邻近点恢复”。

已知两次恢复执行时间有 $T_1'<T_2'$，由于 $T_1<T_1'$ 且 $T_2<T_2'$，故这4个时间的先后关系有两种：①$T_1<T_1'<T_2<T_2'$；②$T_1<T_2<T_1'<T_2'$。若 $T_2\in(T_1',T_2')$ 即类型①，表示 T_2 为间隙内某一时刻，可称之为“间隙内邻近点恢复”，类型②称为“间隙外邻近点恢复”。

3.3.1 间隙内邻近点恢复

由于 $T_1<T_1'<T_2<T_2'$，故 S_{T_2} 比 S'_{T_2} 少区间 (T_2,T_2') 的数据，即

$$S_{T_2}=S'_{T_2}-D_{(T_2,T_2')}\Rightarrow\Delta D=-D_{(T_2,T_2')}$$

恢复需将数据 $D_{(T_2,T_2')}$ 剔除。调用2.2节的数据剔除算法遍历区间 (T_2,T_2') 构造剔除位表；由于恢复 $\langle T_1'\rightarrow T_1\rangle$ 的存在，间隙数据 $D_{(T_1',T_2')}$ 是基于状态 S_{T_1} 写入的，故“重写剔除块”时需逆序依次遍历 TD 链 $(T_1',T_2]$，$[T_{base},T_1]$ 两区间的数据，如图3所示。

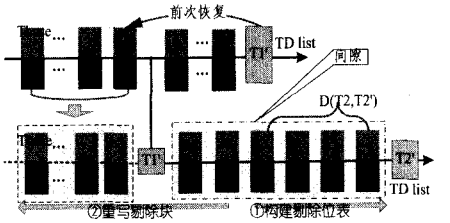

图3 间隙内邻近点恢复

由于重写剔除块时遍历区间改变，读写开销也做相应变化，如表4所列。

表4 间隙内邻近点恢复读写开销比较

数据所在时间区间		读开销	写开销
传统算法		$[T_{base},T_1]+(T_1',T_2]$	$[T_{base},T_1]+(T_1',T_2]$
差异算法	最优	$2(T_2,T_2')$	(T_2,T_2')
	平均	$[T_{base},T_1]/2+(T_1',T_2]+(T_2,T_2')$	(T_2,T_2')
	最差	$[T_{base},T_1]+(T_1',T_2]+(T_2,T_2')$	(T_2,T_2')

3.3.2 间隙外邻近点恢复

由于 $T_1<T_2<T_1'<T_2'$，当前目标卷状态 S'_{T_2} 是基于 S'_{T_1} 写入区间 (T_1',T_2') 的数据，而目标状态 S_{T_2} 是基于 S_{T_1} 写入区间 $(T_1,T_2]$ 的数据，故可得如下关系式：

$$\begin{cases}S'_{T_2}=D_{(T_1',T_2')}+S'_{T_1}\\ S_{T_2}=D_{(T_1,T_2]}+S_{T_1}\\ S'_{T_1}=S_{T_1}\end{cases}$$

$$\Rightarrow S_{T_2}=S'_{T_2}-D_{(T_1',T_2')}+D_{(T_1,T_2]}$$

因此，本次差异数据 $\Delta D=D_{(T_1,T_2]}-D_{(T_1',T_2')}$，恢复算法需将数据 $D_{(T_1',T_2')}$ 剔除，数据 $D_{(T_1,T_2]}$ 写入。数据剔除或写入的算法已由上述2.1节和2.2节给出，当需同时使用两种方法时，由于二者均有写操作，为避免写覆盖和对同一数据块的冗余写，需遵循一定规则。假设写入数据 $D_{(T_1,T_2]}$ 覆盖数据块集合为 Set_{write}，“剔除位表”标记数据块集合为 Set_{delete}，两集合有以下4种关系，如图4所示。

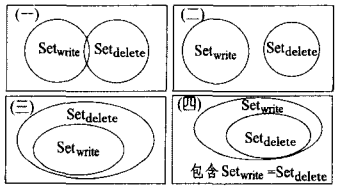

图4 剔除和写入数据块集合关系

在关系(四)下，由于写入数据块包含剔除数据块，故写入同时即完成剔除。其它3种关系下，写入数据 $D_{(T_1,T_2]}$ 之后仍需要遍历 TD 链区间 $[T_{base},T_1]$ 完成后续剔除。故该恢复需判断写入及剔除数据间关系。恢复如图5所示，步骤如下：

①写入数据 $D_{(T_1,T_2]}$，获得 Set_{write}，建初值全“0”的“覆写位表”，“1”位标记 Set_{write} 中元素；

②根据数据 $D_{(T_1',T_2')}$ 获得 Set_{delete}，构造初值全“1”的剔除位表，“0”位标记 Set_{delete} 中元素；

③判断 Set_{write} 与 Set_{delete} 的关系，获得未剔除数据块：两位表执行“或”操作，若其结果为全“1”，即图4所示的关系(四)，则恢复结束；否则将或操作结果赋值给“剔除位表”，更新 Set_{delete}；

④逆序遍历区间$[T_{base},T_1]$，重写Set_{delete}中数据块。

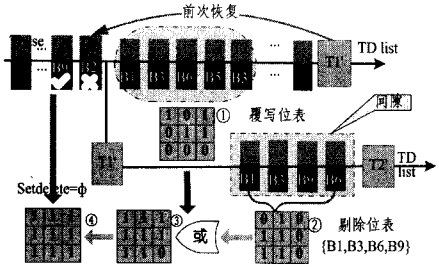

图5 间隙外邻近点恢复

该算法的读写开销需同时考虑数据的写入与剔除。最差情况时两集合的交集为空(即关系(二))，需遍历$[T_{base},T_1]$重写剔除块，如表5所列。

表5 间隙外邻近点恢复读写开销比较

数据所在时间区间		读开销	写开销
传统算法		$[T_{base},T_2]$	$[T_{base},T_2]$
差异算法	最优	$(T_1,T_2]+(T_1',T_2')$	$(T_1,T_2]$
	平均	$[T_{base},T_1]/2+(T_1,T_2]+(T_1',T_2')$	$(T_1,T_2]+(T_1',T_2')/2$
	最差	$[T_{base},T_2]+(T_1',T_2')$	$(T_1,T_2]+(T_1',T_2')$

3.4 有间隙前邻近点恢复

定义6 若相邻恢复有间隙，且邻近恢复较前一次恢复目标时间点提前，即$T_1>T_2$，则称之为“有间隙前邻近点恢复”。

由于$T_1>T_2$，当前目标卷状态S_{T_2}'是基于S_{T_1}'写入区间(T_1',T_2')的数据，故有关系式：

$$\begin{cases} S_{T_2}'=D(T_1',T_2')+S_{T_1}' \\ S_{T_2}=S_{T_1}-D(T_2,T_1] \\ S'_{T_1}=S_{T_1} \end{cases}$$

$$\Rightarrow S_{T_2}=S_{T_2}'-D_{(T_1',T_2')}-D_{(T_2,T_1]}$$

差异数据为$\Delta D=-D_{(T_1',T_2')}-D_{(T_2,T_1]}$，故本次恢复需将两段数据剔除：顺序遍历区间$(T_2,T_1]$，$(T_1',T_2')$构造剔除位表，逆序遍历区间$[T_{base},T_2]$重写剔除块。

基于邻近点的定义，$\Delta T'$和ΔT皆远小于系统正常运行的时间，那么满足$(D_{(T_2,T_1]}+D_{(T_1',T_2')})<<D_{[T_{base},T_2]}$的可能性较大，即差异算法更优；当此条件不满足时，需比较二者数据量，选择恢复算法，读写开销如表6所列。

表6 有间隙前邻近点恢复读写开销比较

数据所在时间区间		读开销	写开销
传统算法		$[T_{base},T_2]$	$[T_{base},T_2]$
差异算法	最优	$(T_2,T_1]+(T_1',T_2')$	$(T_2,T_1]+(T_1',T_2')$
	平均	$[T_{base},T_2]/2+(T_2,T_1]+(T_1',T_2')$	$(T_2,T_1]+(T_1',T_2')$
	最差	$[T_{base},T_1]+(T_1',T_2')$	$(T_2,T_1]+(T_1',T_2')$

4 评估与实验

由于CPU处理时间远小于I/O时间，I/O次数是影响系统开销的主要因素，故本文通过分析恢复算法的读写开销进行性能评估。第1.2节已说明传统算法的读写开销均为$D_{[T_{base},T_{target}]}$。差异算法通过消除数据差异实现恢复，其开销与差异数据量ΔD有关。由上述读、写开销表可得：

①差异算法仅需要数据写入时，由于写入数据必满足$\Delta D\subset D_{[T_{base},T_{target}]}$，故差异算法比传统算法更优。

②差异算法仅需要数据剔除时，其写开销为ΔD，而其读开销由两部分构成：构造剔除位表、重写剔除块。构造剔除位表需读取ΔD大小的数据。由于数据在不同数据块分布的不确定性，重写剔除块的读开销无法确定，最差情况下，它与传统算法的一样，这样差异算法的读开销相比传统算法多ΔD。忽略读、写开销本身的差异，当满足$2\Delta D<D_{[T_{base},T_{target}]}$时，若两算法写开销的差值大于构造剔除位表的读开销，则差异算法更优，否则传统算法更优。

③差异算法需同时使用数据写入与剔除时，由于写入同时或许会剔除部分数据，这与数据分布相关，故考虑最差情况。此时，差异算法的写开销为ΔD，读开销比传统算法的多间隙数据$\Delta D'$(有$\Delta D'\subset\Delta D$)。那么当满足$(\Delta D+\Delta D')<D_{[T_{base},T_{target}]}$时，差异算法更优，否则传统算法更优。

为了验证差异算法的正确性与执行效率，作者利用驱动开发模型(Windows Driver Model，WDM)[7]在文件系统以下、逻辑卷[8]以上嵌入有CDP功能的过滤驱动程序，完成原型实验。实验环境为一台运行WindowsXP系统的普通PC，备份卷和目标卷均为普通IDE硬盘。实验设定恢复目标时间点，通过修改前一次恢复的目标时间点，配置不同类型的邻近时间点恢复，测试不同差异数据量与恢复时间的关系。实验表明，差异算法可以将目标卷恢复到正确的数据状态，其恢复结果与传统算法的一致，效率如图6所示。

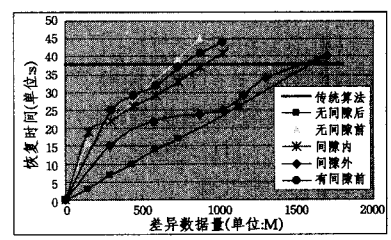

图6 差异恢复与传统算法恢复效率比较

传统算法恢复到该目标时间点需要读、写1.66G数据量，大约需要38s。而差异恢复效率与数据量有关，数据量越小，效率越高。设图中差异恢复曲线与传统恢复曲线的交点所对应的差异数据量为临界值，由图可得：无间隙后邻近点恢复的横纵坐标关系近似呈一条直线，临界值最大，因为它只需要数据写入，读、写开销均为差异数据量；间隙外邻近点恢复需要数据写入和数据剔除，二者数据有一些交集，减少了读、写开销，故临界值次大；其它3种类型都只需要数据剔除，故3条曲线布局较为集中，临界值较小。差异数据量小于临界值时，选择差异算法恢复效率更高。

基于邻近时间点的定义，$\Delta T'$和ΔT皆远小于系统正常运行的时间，则满足ΔD远小于临界值的可能性较大，采用差异算法恢复更优。

结束语 本文对邻近时间点划分了4种类型，研究目标卷在不同时间点的数据差异，用“剔除位表”记录剔除数据块，用“覆写位表”防止对同一数据块的冗余写操作，实现快速恢复。通过对读写开销的分析，差异算法在邻近时间点恢复中可有效避免传统恢复算法中大量读写开销，从而提高了恢复效率，减少了RTO。实验也表明，本算法能够满足用户多次、有效的数据恢复，其效率与差异数据量有关，差异数据量越小，效率越高。

CDP系统的所有恢复都基于对TD链的操作，链条上任

意一个节点出错或任意两个节点顺序颠倒，都会导致恢复数据不正确，使长期部署的 CDP 系统失效。通过添加少量快照可提高 TD 链的可靠性[3]，其邻近时间点恢复算法也需随之做相应调整，这一点有待进一步研究。

参 考 文 献

[1] Yang Q, Xiao W, Ren J. TRAP-Array: A disk array architecture providing timely recovery to any point-in-time[C]// Proc of the Int Symp on Computer Architecture. New York: ACM, 2006

[2] 王彦龙，李战怀，徐娟. 基于块的数据库系统连续数据保护[J]. 计算机研究与发展，2006，43(Suppl.)：168-173

[3] 李旭，谢长生，等. 一种改进的块级连续数据保护机制[J]. 计算机研究与发展，2009，46(5)：762-769

[4] 喻强. 基于 iSCSI 连续数据保护系统的研究和实现[D]. 北京：清华大学，2008

[5] Keeton K, Santos C, Beyer D, et al. Designing for disasters[C]// Proc. of 3rd Conference on File and Storage Technologies. San Francisco, CA, 2004

[6] 魏冰璐. 一种持续数据保护系统的设计与实现[D]. 上海：复旦大学，2008

[7] Walter O. Programming the Microsoft Windows Driver Model [M]. USA: Microsoft Press, 2000

[8] 王彦龙，李战怀，郑然. 多平台数据容灾系统的研究与设计[J]. 计算机应用研究，2007，24(2)：215-218

(上接第 128 页)

4. 在由 Babel 生成的 impl 文件中添加端口的具体实现代码。可以在该文件中包含头文件以链接库文件，比如可以通过包含 mpi. h 头文件来调用 mpi 通信函数，与其他构件通信。其中，自定义的方法中可以封装 mpi 代码，但代码中不可含有 mpi 初始化、结束化语句，使框架进行初始化、结束化工作。SetServices 为与框架交互部分，应给出提供和使用的端口信息。在 sles_Jacobi_Impl. cc 文件中实现 setServices 方法。

4.3 运行构件程序

安装好 ccaffeine 后，可以在 shell 下使用 ccafe-single 或 ccafe-batch 命令启动框架运行构件程序，前者以单进程启动框架，允许用户交互式执行；后者以批处理方式处理文件中的一组命令，以完成实例化构件、连接端口、运行等操作。ccafe-batch 可以调用 mpich 进行初始化、结束化，可以用多进程启动框架，以并行执行。以 ccafe-batch 命令调用 mpi 并行执行求解线性方程组程序的命令如下：

mpirun-p4pg pgfile /ccaffeine-0. 5. 10/bin/ccafe-batch--ccafe-rc /ccaffeine/sles. rc

其中 pgfile 指定 ccafe-batch 位置和进程数。

localhost 0 /ccaffeine-0. 5. 10/bin/ccafe-batch

localhost 1 /ccaffeine-0. 5. 10/bin/ccafe-batch

--ccafe-rc 选项指定批处理程序 ccafe-batch 要处理的文件：sles. rc。

在 ccafe-gui 下，可以直观地看到已实例化的构件和构件间通过端口的调用关系。图 3 的 Arena 中显示了实例化的构件和连接好的端口，图中红线连接的 4 个构件构成了一个 SOR 解法器。

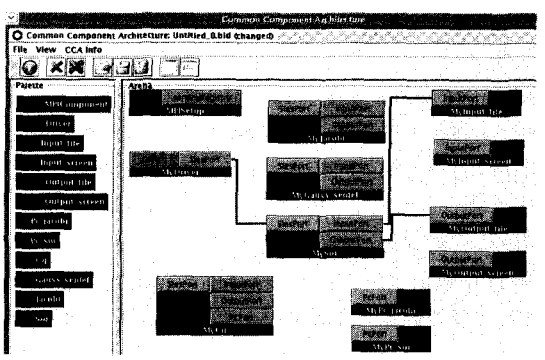

图 3 ccafe-gui 下实例化构件以及构件间调用关系图

4.4 构件化程序性能测试

为测试构件化程序与使用 c 或 fortran 结合 mpi 的并行程序的开销，设计实现以下实验。设一维椭圆偏微分方程为：

$$u_{xx}=0, x\in\Omega=[0,1] \tag{1}$$

其中，Dirichlet 边界条件为：

$$u(x)=x, x\in\partial\Omega \tag{2}$$

对求解区域[0,1]使用网格划分，用空间间隔为 $1/(n-1)$ 的长度把求解区域网格化。用 $u_i(i=0,\cdots,n-1)$ 表示有限差分近似值。对式(1)进行 3 点有限差分近似，得：

$$u_{i-1}-2u_i+u_{i+1}=0 \tag{3}$$

考虑边界值条件 $x_0=0$ 和 $x_{n-1}=1$。测试 n 取不同值时使用 c+mpi 程序与构件程序的执行时间($n=50$ 和 $n=500$ 时，由于通信时间已大于计算时间，故未比较此时的并行程序执行时间)。

从表 1 中可以看出，在方程个数为 50 时，构件程序与手工程序相比开销较明显，当方程个数为 500 时，二者执行时间差别已很小。当方程个数为 500000 时，二者串并行执行时间几乎相同。分析原因，构件程序比手工程序开销多在于引入多余的函数调用，而随着方程个数的增加，计算时间加大，函数调用时间占总执行时间的比例越来越小，甚至可以忽略。

表 1 c+mpi 程序与构件程序的执行时间比较

n	手工程序	构件程序	手工并行程序	构件并行程序
50	0.250(ms)	0.608(ms)	\	\
500	32.609(ms)	33.676(ms)	\	\
500000	282.5(s)	283.3(s)	166.18(s)	167.62(s)

结束语 本文介绍了 CCA 的概念和如何基于 CCA 开发构件程序。分析了基于 CCA 制作构件、管理构件和构建程序的步骤及实现原理。在此基础上设计实现了一款偏微分方程线形解法器，用户可以基于该解法器系统所提供的构件方便地构建程序以实现线性方程组求解，也可以进行扩展，继承已有的端口开发新的求解构件或输入输出构件。

构件化科学计算软件开发还处于探索阶段，CCA 是构件化科学计算领域的一个良好范例。

参 考 文 献

[1] CCA Forum homepage[OL]. http://www. cca-forum. org

[2] Bernholdt D E, Allan B A, Armstrong R. A Component Architecture for High-Performance Scientific Computing[J]. International Journal of High Performance Computing Applications, 2006, 20(2): 163-202

[3] 谭祎炙，黄春，赵克佳. 基于 Babel 的构件程序设计[J]. 计算机科学，2006，33(12)：235-237