The First National Conference on Software Engineering Applications
3-5 march 2009
اولین کنفرانس ملی
کاربردهای مهندسی نرم افزار لاهیجان

یک روش جدید برای نشان‌گذاری(Check Pointing) در فایل ثبات وقایع پایگاه‌داده

رضا فردوسی بیرامی[۱]
کارشناسی ارشد نرم افزار- دانشگاه آزاد شبستر
ferdousi.r@gmail.com

یوسف یکرنگ خسروشاهی[۲]
کارشناسی ارشد نرم افزار- دانشگاه آزاد شبستر
yousef_yekrang@yahoo.com

چکیده *- با توجه به اهمیت سیستم های پایگاه داده‌ای و لزوم کارکرد صحیح آن، صحت و تضمین کار کرد چنین سیستم هایی از مسائل اساسی به شمار میرود. و عملیاتی نظیر ثبت وقایع جهت باز یافت سیستم از حالت خرابی به یک حالت معتبر قبلی از طریق* log file *انجام می شود. که* checkpointing *نیز عملی برای جلو گیری از ازدیاد بیش از حد عمل بازیافت است. در این مقاله روش پیشنهادی جدیدی را برای عمل* checkpointing *بر روی* log file *مطرح می کنیم که مشکلات روش های قبلی را بهبود می دهد. این روش که با اهداف بهبود زمانی و بالا بردن توان عملیاتی سیستم در زمان ثبت* checkpoint *ارائه شده است که با عنوان* Partial checkpoint *معرفی خواهیم کرد.*

کلید واژه
- بازیافت , خرابی سیستم ,.log file , Checkpointing

۱- مقدمه

در این مقاله یک روش جدید برای ثبت رکورد checkpoint در داخل log file پیشنهاد شده است که با هدف حذف یا کاهش توقف سیستم درهنگام ثبت رکورد checkpoint معرفی شده است. این بحث را با مقدمه ای در مورد بازیافت سیستم های پایگاه داده ای که دچار خرابی شده اند , آغاز می کنیم .سپس با بررسی این رو ش و رسیدن به بحث Log File و Log Record وارد بحث checkpointing می شویم و با معرفی روشهای قبلی ثبت checkpoint , روش پیشنهادی خود را ارائه می دهیم. چها چوب کلی برای بیان مباحث ذکر شده به صورت زیر است:

در متن این مقاله در بخش ۲ بیان مختصری از مفهوم بازیافت و دلیل استفاده از این روش و انواع روشهای بازیافت را آورده ایم. در بخش ۳ به معرفی چک پوینت و انگیزه استفاده از آن و روشهای قبلی در مورد ثبت چک پوینت خواهیم پرداخت. در بخش ۴ یک روش پیشنهادی جدید برای عمل ثبت چک پوینت معرفی کرده ایم که از معایب روشهای قبلی می‌کاهد و مهمترین مزیت آن در جلوگیری از اتلاف وقت سیستم یا توقف سیستم برای ثبت چک پوینت است. در بخش ۵ خلاصه و جمع بندی مقاله آورده شده است و در بخش ۶ نیز نتیجه گیری خود را بیان کرده ایم. در پایان نیز با ذکر مراجع مقاله پایان پذیرفته است.

۲- بازیافت (Recovery)

در پایگاه داده ها، همواره تراکنشهای بسیاری در حال اجرا هستند که هر کدام در تلاش برای پایان یافتن، به سرعت از سیستم تقاضای داده و در برخی مواقع تقاضای به روز رسانی آن اطلاعات را می کنند. اما آنچه مهم است سلامت پایگاه داده در

کنار پیشرفت کار است. برای مواقعی که تراکنشها به صورت عادی در حال کار هستند مراقبت های ویژه ای مثلاً با استفاده از سیستم قفل گذاری انجام می شود ولی هنگامی که سیستم در اثر برخی اتفاقات خاص دچار خرابی (Failure) می شود ما باید بتوانیم دوباره پایگاه داده خود را به حالت سازگار و یکپارچه قبلی باز گردانیم. در این موقع است که Data base Recovery System یا سیستم بازیافت پایگاه داده مطرح می شود.در برخی مواقع, ممکن است که سیستم با یک خرابی مثلاً Transaction Failure (خرابی تراکنش) یا System Crash (خرابی کل سیستم) و یا Disk Failure (خرابی دیسک) روبرو شود که در نهایت منجر به توقف سیستم شده و در چنین شرایطی نیاز پیدا می کنیم که از الگوریتم های Recovery برای بازگرداندن سیستم به یک حالت درست قبلی استفاده کنیم. این الگوریتم ها به گونه‌ای ساخته شده اند که بتوانند Transaction Atomicity , Database Consisteney و Durability (سازگاری پایگاه داده، یکپارچگی تراکنش ها وماندگاری) را تضمین کنند. الگوریتم Recovery دارای دو بخش هست که بخش اول اعمالی است که در پردازش نرمال تراکنشها انجام می شود و اطلاعاتی را ذخیره می کنند تا تضمین شود که اطلاعات کافی برای شروع به کار دوباره (در صورت بروز Failure) وجود دارد و بخش دوم اعمالی است که بعد از رخ دادن خرابی برای بازگردانی (recovery) محتوای پایگاه داده به وضعیتی که آن سه ویژگی بالا تضمین شود را شامل می شود. واضح است که بخش دوم وابسته به اطلاعاتی است که در بخش اول جمع آوری شده است.

ما برای ذخیره سازی اطلاعات در سیستم , یکی از سه نوع حافظه موجود را می توانیم انتخاب بکنیم که به ترتیب عبارتند از:

۱) حافظه فرّار که از System Crash جان سالم به در نمی برد.

۲) خافظه غیر فرّار که در System Crash مشکلی برایش بوجود نمی آید.

۳) حافظه پایدار که حافظه غیر فرّاری است که تحت هیچ شرایطی از بین نمی رود (ما وارد جزئیات ویژگی این نوع حافظه در این مقاله نمی شویم) اما بسیار گران قیمت است.

با توجه به ویژگیهایی که از سه نوع حافظه ذکر شده یاد شد، در سیستم های پایگاه داده ، پایگاه داده را در حافظه غیر فرار و اطلاعات لازم برای بازگردانی را ، در حافظه پایدار ذخیره می کنند. (از این موضوع بعداً استفاده خواهیم کرد.)[۱]

یکی از مواردی که عمل بازیافت می خواهد آن را تضمین کند، خاصیت Transaction Atomicity(یکپارچگی تراکنش) است که به این معنی است که ما یا باید همه دستورات تراکنش را انجام دهیم و یا اینکه هیچ کدام را انجام ندهیم، انجام دادن فقط برخی از تغییرات بر روی پایگاه داده، منجر به ورود سیستم به یک حالت ناسازگار خواهد شد. پس سیستم بازیافت یا Recovery system باید از انجام نشدن برخی اعمال و یا انجام مجدد برخی اعمال انجام شده درزمان خرابی, جلوگیری کند. برای اینکار قبلاً دو راهکار ارائه شده است که در بخش ۲-۱ به آنها خواهیم پرداخت.

۱- انواع روشهای بازیافت

در کل دونوع روش بازیافت وجود دارد[۱] که عبارتند از:

۱) Log-Based Recovery

۲) Shadow – Paging

روش دوم به خاطر برخی مشکلات مثل سربار و ... که بحث آن در مقاله ما جای ندارد , روش بهینه ای نیست و ما بر روی روش اول بحث خواهیم کرد. در روش Log-Based Recovery سیستم پایگاه داده از یک log file استفاده می کند که ترتیبی از اعمالی را نگهداری می کند که بر روی پایگاه داده انجام شده است. این log file باید در حافظه پایدار ذخیره شود تا بتوانیم هنگام بازیافت بعد از خرابی , به آن دسترسی پیدا بکنیم. لازم به توضیح است که هر سطر log file مربوط به یک عمل است که در پایگاه داده انجام شده است و به اصطلاح به آن log record گویند.

روش log-based recovery به دو صورت می تواند انجام گیرد:

۱) Deffered Database modification

۲) Immediate Database Modification

با فرض اینکه دو عمل read (برای خواندن از پایگاه داده) و write(برای نوشتن در پایگاه) استفاده می شود:

Deffered Database Modification

در روش Deffered Database Modification همه write هایی که وجود دارد را به عقب می اندازیم به این صورت که اگر یک تراکنشی یک دستور write صادر کرد، log record آن را در log file می نویسیم ، اما خودش را در پایگاه داده اعمال نمی کنیم و همه write ها را به بعد از تکمیل جزئی تراکنش موکول می کنیم. حال اگر کامیت جزئی (partially commit)

پایان یافت ، log record ها را خوانده و شروع به انجام write ها بر روی پایگاه داده به صورت حقیقی می کنیم.

حال اگر سیستم crash کرد، به مرحله بازیافت کردن سیستم وارد می شود که نیازمند دو عمل redo و undo برای آن هستیم:

Redo(Ti) : انجام دادن مجدد تراکنشی که اعمال آن در log file است وآن اعمال باید در پایگاه داده انجام می شدند ولی ما مطمئن نیستیم که آنها در پایگاه داده تاثیر کرده اند یا نه.

Undo(Ti) : بازگردانی اعمال تراکنش Ti که اعمال آن در log file قرار دارد ولی به خاطر اینکه تراکنش مربوطه کامل نیست ، باید تغییرات مربوطه بازگردانی شود.

در روش Deffered هیچ نیازی به انجام دادن Undo نداریم. چون مطمئن هستیم که تا وقتی که تراکنش کامل نشود، بر روی پایگاه داده هیچ تغییری انجام نمی شود ولی برای اینکه مطمئن شویم که تغییراتی که باید بر روی پایگاه داده انجام می شدند، انجام شده اند باید Redo را انجام دهیم. یعنی ما از log file می خوانیم و چون صد در صد مطمئن نیستیم که در پایگاه داده نیز ثبت شده است این log recordها را برای انجام تغییرات در پایگاه داده استفاده می کنیم.

۲-۱-۲- روش Immediate Database Modification

در روش Immediate Database Modification هنگام شروع یک تراکنش ، رکورد < Ti, start > برای تراکنش Ti در log file ذخیره می شود.

در هر لحظه که تراکنش write انجام می دهد، write را مستقیماً در پایگاه داده البته در بافر انجام می دهیم.

از این رو ممکن است هنگام recovery کردن، عمل undo نیاز شود. البته باید این نکته را در نظر بگیریم که log record مربوطه باید قبل از این عمل در log file و در داخل حافظه پایدار ثبت شود. که این log record به صورت <Ti , X , V_1 , V_2 > است که V_1 , V_2 به ترتیب نشان دهنده مقدار قدیمی و جدید داده است که توسط تراکنش T_i تغییر می یابد.

البته باید در نظر داشت که این مقادیر قدیمی و جدید در هنگام recovery کردن نیاز خواهد بود . در پایان نیز اگر تراکنشی با موفقیت به پایان برسد، رکورد<T_i, commit> در log file ثبت خواهد شد.

پس هنگام recovery کردن دو کار نیاز است که انجام دهیم:

۱) Undo (T_i) : ذخیره کردن دوباره مقادیر قدیمی آیتم های داده ای که توسط تراکنش T_i آپدیت شده بودند و اکنون در log file رکورد<T_i, start> وجود دارد ولی رکورد <T_i, commit> ثبت نشده است انجام می دهیم.

۲) Redo (Ti) : انجام دوباره اعمالی که توسط تراکنش Ti انجام شده اند و در log file قرار دارند و دلیل آن اینست که رکوردهای شروع و پایان یا <Ti , Start> و <Ti , Commit> در log file وجود دارد. پس مطمئن هستیم که این تراکنش با موفقیت تمام شده است ولی مطمئن نیستیم که تمامی تغییرات مربوطه که در log file ثبت شده اند, در فایل اصلی پایگاه داده نیز ثبت شده است یاخیر.

۳- معرفی Checkpint و روش های قبلی آن

فرض کنید ما دارای یک log file بسیار بزرگ هستیم که تمامی رکوردهایی که در آن ثبت شده اند، در پایگاه داده که در حافظه غیر فرّار قرار دارد اعمال شده اند و تنها ۲٪ از رکوردهای ثبت شده در آن فقط در log file و بافر تغییر کرده اند وقبل از اینکه در پایگاه داده و در دیسک سخت تغییر بکنند، سیستم دچار خرابی شده است حال اگر سیستم بازگردانی انجام دهد, بخاطر این ۲٪ باید ۹۸٪ دیگر را به صورت اضافی در سیستم انجام دهیم اما سیستم مدیریت پایگاه داده، نمی خواهد که این کارهای بیهوده را انجام دهد ویا به عبارتی , می خواهد که از دوباره کاری جلوگیری کند. بنابراین بحث نشان گذاری مطرح می شود . که عاملی برای فهمیدن اینکه تغییرات کدام تراکنشها در پایگاه داده ذخیره شده اند و کدام قبل از ذخیره در پایگاه داده و هنگامی که فقط در بافر بر روی اطلاعات تغییر کرده بودند، سیستم Crash کرده است. پس با فهمیدن این مورد، ما به راحتی record های قبلی را کنار می گذاریم و فقط رکوردهایی را که هنوز اطمینان به انجام صد درصد آنها در پایگاه داده نیست انجام می دهیم.

برای اینکار روش چک پوینتینگ به صورت برای علامت زدن به log file استفاده می شود که در بخش ۳-۱ و ۳-۲ دو روش قبلی که برای اینکار اجرا می شد را بررسی خواهیم کرد.

۳-۱- نشان گذاری ساده

ساده ترین روش برای گذاشتن چک پوینت در یا علامت گذاری بر روی log file به صورت زیر انجام می شود :

۱) کلیه تراکنشهای سیستم را متوقف کن.

۲) کلیه log record هایی که در حافظه اصلی هستند را در داخل log file بنویس (البته اگر فرض کنیم که کلیه log record ها به طور مستقیم در log file و در داخل حافظه پایدار نوشته می شوند، این مرحله نیاز نیست).

۳) همه بلوک های با فری که شامل اطلاعات تغییر یافته پایگاه داده هستند را در داخل پایگاه داده (که در دیسک سخت یا حافظه غیر فرار قرار دارد) را بازنویسی کن.

۴) حال یک رکورد <check point , L> در داخل log file بنویس که L لیست تراکنشهای فعال سیستم است.

۵) سیستم را دوباره با تراکنشهای متوقف شده بکار بگیر.

این مراحل که تمام شد ، سیستم دوباره به اجرای کار خود به صورت عادی ادامه می دهد. اما این چک پوینت چگونه استفاده می شود؟

هنگامی که سیستم دچار خرابی شد و می خواهد بازگردانی انجام دهد هیچ کدام از تراکنشهایی را که قبل از Chekpoint بوده اند را در نظر نمی گیریم. فقط تراکنشهایی که در لیست L هستند و در ضمن تراکنشهایی که بعد از آن بوجود آمده اند را در نظر گرفته و آنهایی را که نیاز به Undo کردن باشند، (تراکنشهای نا تمام)undo کرده و آنهایی را که نیاز به Redo کردن باشند (تراکنشهایی که با موفقیت پایان پذیرفته اند) را Redo می کنیم. البته ما نمی خواهیم که وارد بحث چگونگی انجام بازیافت و استفاده از checkpoint شویم..

اما نکته بسیار مهمی که وجود دارد , این است که هنگامی که ما در حال ثبت Cheekpoint در log file هستیم، نمی توانیم هیچ تراکنشی را در سیستم به صورت فعال داشته باشیم. همانگونه که در بند (۱) از چک پونیتینگ اشاره شده است , ابتدا همه تراکنشها را متوقف می کنیم و سپس شروع به انجام عمل چک پونیتینگ می کنیم و در آخر نیز تراکنشها را مجدداً راه اندازی می کنیم. مشکل اینجاست که ممکن است سیستمی باشد که کاربران اجازه توقف را به سیستم ندهند یا اینکه لااقل کاربران انتظار اینکه سیستم آنها را منتظر نگه دارد را ندارند. حال اگر در یک سیستم بزرگ که Cheekpoint آن , زمان قابل ملاحظه ای طول می کشد تا از این روش Cheekpoin استفاد بکنیم، مثل اینکه کل سیستم را متوقف کرده ایم . بنابراین روشهای دیگری برای چکپوینتینگ ارائه شد که از توقف سیستم به طور کامل جلوگیری می کرد. که از جمله آنها می توان به Fuzzy check point اشاره کرد. که در بخش ۳-۲ به آن می پردازیم.

۳-۲- Fuzzy Checkpointing

چک پونیتینگ فازی به صورت زیر عمل می کند.

۱) جلوی همه به روز رسانی ها (Update) توسط تراکنشها را بگیر. یعنی اینکه تراکنشهایی که در حال تغییر دادن یا به روز رسانی نیستند را کاری نداشته باش.

۲) یک چک پوینت به صورت <Checkpoint , L> مستقیم در درون log file حافظه پایدار بنویس.

۳) یک لیست M برای بلوک های میانگین تغییر یافته در نظر بگیر.

۴) اکنون به تراکنشهایی که جلویشان را گرفته بودیم ، اجازه ادامه دادن بده.

۵) همه بلوک های تغییر یافته که در لیست M هستند را با رعایت موارد زیر در داخل پایگاه داده در حافظه غیر فرار , بازنویسی کن.

* بر روی بلوک ها نباید در حین این عمل , به روز رسانی انجام شود.

* باید قانون WAL رعایت شود. قانون WAL (Write Ahead Loging) می گوید که قبل از اینکه یک بلوک از داده در حافظه اصلی (بافر) به پایگاه داده در حافظه جانبی نوشته شود. باید log record هایی که وابسته به آن بلوک هستند , به داخل log file در حافظه پایدار نوشته شود.

۶) یک اشاره گر (برای راحت تر پیدا کردن آخرین check point در هنگام بازیافت) با نام last checkpoint برای نشان دادن آخرین چک پوینت بر روی دیسک ذخیره می کنیم.

البته هنگام بازیافت کردن همانند روش قبلی عمل می کنیم فقط روش گذاشتن چک پوینت فرق کرد که آن هم برای جلوگیری از توقف سیستم است. این روش بسیار بهتر از روش قبلی است. چون در طی مدت زمان قابل ملاحظه ای که سیستم در حال انجام عمل چک پوینت است تراکنشهایی که عمل به روز رسانی انجام نمی دهند، می توانند اجرا شوند و این خود امتیاز بسیار بزرگی است ولی با این حال تراکنشهای به روز رسان به حالت تعویق رفته اند و این باعث می شود که اگر سیستم ما به گونه ای باشد که تعداد تراکنشهای به روز رسان زیاد باشد تا حدودی مثل روش قبلی چک پونیتینگ شود. این دلیل باعث شد تا ایده جدیدی برای گذاشتن چک پوینت در log file بوجود آید که از تاخیر زیاد درسیستم در هنگام ثبت چکپوینت جلوگیری می کند واین روش پیشنهادی ما در بخش ۴ معرفی

شده است.

۴- ارائه روش جدید ثبت Check point

مسائلی که تا کنون مطرح شد، مسائلی بودند که قبلاً وجود داشتند و اشاره به بحث Recovery(بازیافت) و در سایه آن بحث Checkpointing و انواع روشهای ارائه شده آن می کرد. همانطور که ملاحظه شد روش سنتی Checkpointing هنگامی که می خواست رکورد <checkpoint , L> را به log file اضافه کند در طول زمان قابل ملاحظه ای تراکنشها را متوقف می کرد و مانند این بود که کل سیستم در حال توقف است و این در سیستم هایی که امکان ایستادن وجود ندارد، غیر قابل قبول است برای حل این مساله روش Fuzzy checkpoint مطرح شده بود که همانطور که ملاحظه شد، تا حدودی از این اتفاق جلوگیری می کرد ولی در سیستم هایی که دارای تراکنشهای به روز رسان بیشتری بودند قابلیت زیاد خوبی نداشت چرا که تقریباً چیزی شبیه به Checkpoint ساده بود و تنها کمی از آن بهتر بود. ما برای اینکه بتوانیم سیستم را از این مشکل در هنگام Checkpoint نویسی نجات دهیم , روش جدیدی را ارائه کرده ایم. در این روش تا حد ممکن از توقف سیستم جلوگیری می شود و تنها در موارد بسیار معدودی ممکن است سیستم متوقف شود.. این روش در بخش ۴-۱ معرفی شده است.

۴-۱- روش پیشنهادی

فرض کنید که سیستم می خواهد در Log file، Check point بگذارد و تراکنشهایی بروی داده هایی در حال کار هستند . روش پیشنهادی به صورت زیر عمل میکند :

۱) همه log record هایی را که در حافظه اصلی هستند را به داخل log file که درون حافظه پایدار قرار دارد, انتقال می دهیم و تا پایان عمل checkpoint, همه log record ها را مستقیما" در حافظه پایدار می نویسیم.

۲) یک رکورد به صورت <partial checkpoint , L1> در log file ثبت می کنیم. که L1 برابر لیست تراکنشهای فعال سیستم در آن لحظه است.

در حین گامهای ۱ و ۲ اجازه می دهیم که تراکنشها به کار خود به صورت عادی ادامه دهند. منتهی هر تراکنشی که کارش تمام می شود، باید تغییرات خود را درپایگاه داده اعمال کند.

۳) بعد از بازه زمانی t، یک لیستی از تراکنشهای فعال سیستم که در حال کار هستند را می گیریم. فرض کنید که این لیست برابر L2 است اگر L1-L2 برابر تهی شود, به این معنی است که سیستم تمامی تراکنشهایی را که در لحظه Partial checkpoint لیست کرده است را تمام کرده است . در غیر اینصورت تراکنشهایی از لیست L1 وجود دارند که هنوز پایان نیافته اند. بنابر این ,دوباره بعد از t بازه زمانی دیگر , لیست L3 را تهیه می کنیم و L1-L3 را بررسی میکنیم. این کار تا زمانی ادامه می یابد که هیچکدام از تراکنشهای لیست L1 در سیستم باقی نمانده باشند.

۵) فرض کنید که در لیست nام هیچکدام از تراکنشهای لیست L1 وجود ندارد و یا به عبارتی دیگر , L1-Ln تهی است. پس کار تمامی تراکنشهای زمانPartial Checkpoint در سیستم پایان پذیرفته است. ما با تاثیر تمامی تغییرات آنها در پایگاه داده البته با رعایت قانون WAL و ترتیب رکوردهای این تراکنشها و تراکنشهای دیگر , مطمئن می شویم که تراکنشهای زمان Partial Check point پایان یافته اند و تغییرات آنها در پایگاه داده انجام شده است .و از این جمله می توانیم به این نتیجه برسیم که تمامی رکوردهای قبل از رکورد Partial Checkpoint رکوردهایی هستند که تغییرات آنها در پایگاه داده اعمال شده اند. پس حال به سراغ رکورد <Partial Check point , L1> می رویم و آن را تبدیل به <Check point , L> می کنیم که L برابر همان L1 است.

۶) حال اگر فرضی که در گام پنجم در نظر گرفتیم. درست در نیامد و چندین تراکنش در لیست L1 بودند که در سیستم برای مدت زمان بسیار طولانی مثلاً زمان M*t باقی ماندند که ما نمی خواهیم partial Checkpoint بیشتر از این زمان طول بکشد. در این صورت نیز راه حل ساده ای وجود دارد و آن اینکه آن چند تراکنش لیست L1 را که هنوز در سیستم باقی مانده اند, لحظه ای متوقف کرده تغییرات آنها را در log record و پایگاه داده اعمال کرده و سپس چک پوینت خود را به صورت <Check point , L> تبدیل می کنیم. البته در این صورت , تنها تراکنشهایی که در سیستم برای مدت زمان بسیار طولانی اجرا شده اند متوقف می شوند، نه کل تراکنشهای به روز رسان.

باید در نظر داشت که وابسته به سیستم و موقعیت سیستم , باید بازه زمانی t که در بالا به آن اشاره کردیم, تعیین شود. بنابراین t یک ثابت زمانی است که وابسته به سیستم های مختلف , میتواند متغییر باشد.

حال اگر سیستم در زمان عمل Partial Checkpoint دچار خرابی شود, سیستم در هنگاه بازیابی دارای یک <partial checkpoint ,L1> خواهد بود که کامل نشده است. بنابراین , ما آخرین چک پوینتی را که قبل از partial checkpoint

نوشته شده بود و به عبارت بهتر یک partial checkpoint کامل است , در نظر می گیریم و با آن کار می کنیم.

۵- جمع بندی و خلاصه

در این مقاله اشاره ای به خرابی ها در سیستم های پایگاه داده ای شده است و سپس روش های بازیافت(recovery) که برای بازگشت از خرابی به یک حالت معتبر قبلی است را یاد آوری کرده ایم و عمل نشان گذاری و روش های قبلی ایجاد Check point را که برای جلوگیری از ازدیاد کار در زمان بازیابی است , بررسی کرده ایم. در نهایت نیز روشی جدید برای عمل چک پوینتینگ ارائه شده است. که روشی بهینه تر از دو روش قبلی است.

۶- نتیجه گیری

یک روش جدید برای عمل ثبت Checkpoint در Log file پیشنهاد شده است که از روشهای قبلی بهینه تر است و می توان در سیستم های پایگاه داده ای که امکان ایست آنها وجود ندارد و یا توقف کردن تراکنش های فعال باعث بوجود آمدن مشکل خواهد شد, بکار برد.

سپاسگزاری

با سپاس از زحمات استاد ارجمند, جناب آقای دکتر محمد رضا فیضی درخشی .

مراجع

[1] Abraham Silberschatz & Henry F.Korth & S.Sudarshan , DataBase System Conceps.

[2]F. Cristian and F. Jahanian, “A timestamp-based checkpointing protocol for long-lived distributed computation,” in Proc. 10th IEEE Symposium on Reliable Distributed Systems, 30 Sep-02 Oct, 1991, pp. 12–20.

[3] N. Neves and W. Fuchs, “Using time to improve the performance of coordinated checkpointing,” in Proc. 2nd IEEE International Computer Performance and Dependability Symposium, September 4-6, 1996, pp. 282–291.

[4] S. George, I. Chen, and Y. Jin, “Movement-based checkpointing and logging for recovery in mobile computing systems,” in Proc. MobiDE’06 5th ACM International Workshop on Data Engineering for Wireless and Mobile Access, June, 2006, pp. 51–58.

[5] J. Ahn and C. Hwang, “Low-cost fault-tolerance for mobile nodes in mobile IP based systems,” in Proc. IEEE International Conference on Distributed Computing Systems Workshop, April 16-19, 2001, pp. 508–513.

[6] J. Ahn, S. Min, and C. Hwang, “A causal message logging protocol for mobile nodes in mobile computing systems,” Future Generation Computer Systems**,** 2004, May, vol. 20, no. 4, pp. 663–686.