T-Cache: 시계열 배관 데이타를 위한 고성능 캐시 관리자
(T-Cache: a Fast Cache Manager for Pipeline Time-Series Data)

신 제 용 [†*] (Jeyong Shin) 이 진 수 [†*] (Jinsoo Lee) 김 원 식 [†*] (Wonsik Kim) 김 선 효 [†*] (Seonhyo Kim)
윤 민 아 [†] (Mina Yoon) 한 욱 신 [††] (Wook-Shin Han) 정 순 기 [††] (SoonKi Jung) 박 세 영 [††] (Se-Young Park)

요 약 지능형 배관 검사체(PIG)는 가스나 기름 배관 안을 지나가며 검사체에 장착된 여러 센서로부터 신호(센서 데이타로 불림)들을 취합하는 장치이다. PIG로부터 취합된 센서데이타들을 분석함으로써, 배관의 구멍, 뒤틀림 또는 잠재적으로 가스 폭발의 위험을 가지고 있는 결함들을 발견할 수 있다. 배관의 센서 데이타를 분석가가 분석을 할 때에는 주로 두 가지 분석 패턴을 사용한다. 첫 번째는 센서 데이타를 순차적으로 분석하는 순차적 분석 패턴이고, 두 번째는 특정한 구간을 반복해서 분석하는 반복적 분석 패턴이다. 특히, 센서 데이타를 분석할 때 반복적 분석 패턴이 많이 사용된다. 기존의 PIG 소프트웨어들은 사용자의 요청이 있을 때 마다 서버로부터 센서 데이타들을 오므로, 매 요청마다 네트워크 전송비용과 디스크 액세스 비용이 든다. 이와 같은 방법은 순차적 분석 패턴에는 효율적이지만, 분석 패턴의 대부분을 차지하는 반복적 분석 패턴에는 비효율적이다. 이와 같은 문제는 서버/클라이언트 환경에서 다수의 분석가가 동시에 분석을 할 경우에는 매우 심각해진다. 이러한 문제점을 해결하기 위해 본 논문에서는 배관 센서 데이타들을 여러 개의 시계열 데이타로 생각하고, 효율적으로 시계열 데이타를 캐싱 하는 T-Cache라 부르는 주기억장치 고성능 캐시 관리자를 제안한다. 본 연구는 클라이언트 측에서 시계열 데이타를 캐싱하는 최초의 연구이다. 먼저, 고정된 거리의 시계열 데이타들의 집합을 캐싱 단위로 생각하는 신호 캐시 라인이라는 새로운 개념을 제안하였다. 다음으로, T-Cache에서 사용되는 스마트 커서와 여러 알고리즘을 포함하는 여러 가지 자료구조를 제안한다. 실험 결과, 반복적 분석 패턴의 경우 T-Cache를 사용하는 것이 디스크 I/O측면과 수행 시간 측면에서 월등한 성능 향상을 보였다. 순차적 분석 패턴의 경우에도 T-Cache를 사용하지 않은 경우와 거의 유사한 성능을 보였다. 즉, 캐시를 사용함으로써 발생하는 추가 비용은 무시할 수 있음을 보였다.

키워드 : 캐싱, 시계열 데이타, 지능형 배관 검사체

Abstract Intelligent pipeline inspection gauges (PIGs) are inspection vehicles that move along within a (gas or oil) pipeline and acquire signals (also called sensor data) from their surrounding rings of sensors. By analyzing the signals captured in intelligent PIGs, we can detect pipeline defects, such as holes and curvatures and other potential causes of gas explosions. There are two major data access patterns apparent when an analyzer accesses the pipeline signal data. The first is a sequential pattern where an analyst reads the sensor data one time only in a sequential fashion. The second is the repetitive pattern where an analyzer repeatedly reads the signal data within a fixed range; this is the dominant pattern in analyzing the signal data. The existing PIG software reads signal data directly from the server at every user's request, requiring network transfer and disk access cost. It works well only for the sequential pattern, but not for the more dominant repetitive pattern. This problem becomes

* 4명의 저자는 본 논문에 동일하게 공헌하였음
† 정 회 원 : 경북대학교 컴퓨터공학과
jyshin@www-db.knu.ac.kr
jslee@www-db.knu.ac.kr
wskim@www-db.knu.ac.kr
shkim@www-db.knu.ac.kr
mayoon@www-db.knu.ac.kr
†† 종신회원 : 경북대학교 컴퓨터공학과 교수
wshan@knu.ac.kr
(Corresponding author임)
skjung@knu.ac.kr
seyoung@knu.ac.kr
논문접수 : 2007년 9월 6일
심사완료 : 2007년 10월 16일

very serious in a client/server environment where several analysts analyze the signal data concurrently. To tackle this problem, we devise a fast in-memory cache manager, called T-Cache, by considering pipeline sensor data as multiple time-series data and by efficiently caching the time-series data at T-Cache. To the best of the authors' knowledge, this is the first research on caching pipeline signals on the client-side. We propose a new concept of the signal cache line as a caching unit, which is a set of time-series signal data for a fixed distance. We also provide the various data structures including smart cursors and algorithms used in T-Cache. Experimental results show that T-Cache performs much better for the repetitive pattern in terms of disk I/Os and the elapsed time. Even with the sequential pattern, T-Cache shows almost the same performance as a system that does not use any caching, indicating the caching overhead in T-Cache is negligible.

Key words : Caching, Time series data, Intelligent PIG

1. Introduction

Several million kilometers of oil and gas pipelines have been installed underground throughout the world [1], and around 30,000 to 40,000 kilometers of new pipelines are added every year. Pipelines may span over several thousands of kilometers [2] and may wear out due to heat, gas pressure or just poor installation.

Several techniques [3] have been used in the past to inspect long pipelines. Among the techniques, the intelligent PIG (pipe inspection gauge) [4] is used by many major pipeline inspection companies [4-7]. A PIG is equipped with more than one hundred sensors which are installed on the circumference of the PIG. The PIG collects the sensor data which is then evaluated after the inspection using special analyzer software. Figure 1 shows an intelligent PIG.

We notice that there exist two major patterns when an analyst accesses the pipeline signal data. One is a sequential pattern where an analyst reads sensor data sequentially only once. The other pattern is a repetitive pattern where an analyzer reads the signal data in a fixed range repeatedly, which is a dominant pattern in analyzing the signal data. This frequently happens whenever an analyst sees doubtful signal data and must repeat a sequence for further analysis. Figure 2 shows a screenshot of our analyzer software.

Figure 1 An intelligent PIG [8]

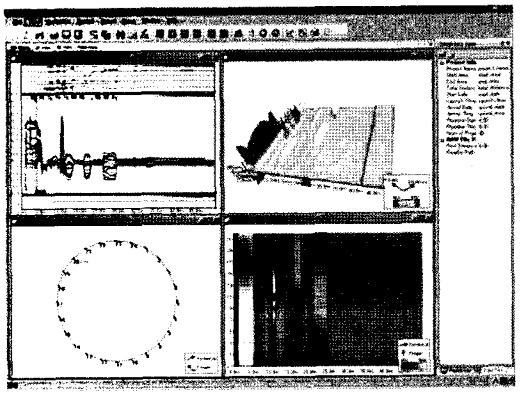


Figure 2 A screenshot of our analyzer software

Regardless of which pattern is used, all the major PIG analyzer software reads signal data directly from the server at the user's request, which only works well for a sequential pattern. Retrieving a repetitive pattern can causes delays for the analyst. We note that the OS-level caching does not help improve system performance in the client/server environment where several analysts analyze the signal data concurrently. In this paper, we solve this problem by exploiting the client-side caching technique of object-oriented databases. We propose a fast cache manager, called T-Cache, by considering pipeline sensor data as multiple time-series data and by caching efficiently the time-series data at T-Cache.

Our contributions are summarized as follows: 1) we propose a new notion of the time-series cache line and regard the cache line as a fetch unit and as a caching unit. To the best of the authors' knowledge, this is the first work caching pipeline signals on the client-side. 2) We propose detailed data structures and algorithms used in T-Cache. 3) Experimental results show that T-Cache performs much better for the repetitive pattern in terms of server calls and the elapsed time. Results show caching overhead in T-Cache is negligible. Even with the sequential pattern, T-Cache shows nearly the same performance as a system that does not use any caching.

The rest of this paper is organized as follows. Section 2 describes existing work related to data analysis software. Section 3 presents the architecture and detailed data structures of T-Cache. Section 4 presents the results of performance evaluation. Section 5 concludes the paper.

2. Related Work

There are several major commercial PIG equipment companies including BJ Pipeline Inspection Services [4], PII (Pipeline Integration International) [9] and 3P Service [10]. By far the most popular analyzer software used in the industry today is GEODENT [4,6], Vectra view [5] and Lina view [7].

Currently, no companies have opened their detailed techniques or source code for the PIG analyzer software. However, by reading some fixed sized data repetitively, we can easily conjecture whether they use a caching technique.

Our finding is that the performance of reading fixed size signal data for the first time is the same as reading data repetitively. This indicates that the existing analyzer software does not use caching techniques and instead reads signal data directly from the server per the user's request. In a client/server environment, the OS-level caching does not help to improve the system performance.

In object-oriented databases, object caching techniques [11,12] are used on the client side. Similar to our approach, their methods also maintain several hash tables to keep track of objects in cache. They cache once read objects as often as they can. However, their caching and fetch unit is an object, and thus, we can not directly use the object cache for our environment. This is because we need to access signal data fast in a bulk fashion.

In reference [13], we introduce the overall description for a scalable pipeline data processing framework. In this paper, we focus on efficient caching mechanisms for the framework.

3. T-Cache

In this section, we first introduce the overall architecture of our analyzer software in Section 3.1, and explain implementation issues including detailed data structures and algorithms in Section 3.2.

3.1 Overall Architecture

Our analyzer software consists of the two components, namely the data manager and the visualizer. The data manager parses and stores raw pipeline data into a time-series store, and provides fast access to the time-series store using T-Cache. As for time-series storage, we can use either our specialized time-series store or a commercial relational DBMS. The visualizer allows users to visualize the sensor data with various views such as fisheye, histogram and wave views. We do not discuss the details of the visualization module, which is beyond our scope. Figure 3 shows the system architecture of our analyzer software.

The data manager consists of the following modules: 1) cleanser, 2) loader, 3) time-series chunk reader, and 4) time-series cache (T-Cache). After we acquire data from a PIG, we first clean raw signal data by removing some erroneous data. For this purpose, our cleanser progressively reads raw signal data and corrects errors based on the cleansing rules. We then load raw signal data cleaned to a time-series database. For initial loading, we do bulk loading. After that, the visualizer can access the time-series signal data by accessing T-Cache, which is in charge of caching part of the whole time-series data. Whenever data requested is not found in T-Cache, we read the data from disk using time-series chunk reader. As the name of the module indicates, it reads chunks of time-series data for fast performance. The chunk reader can be implemented using either a proprietary module or a relational DBMS. However, we prefer to use our specialized module since it provides an order of magnitude faster access than commercial RDBMS.

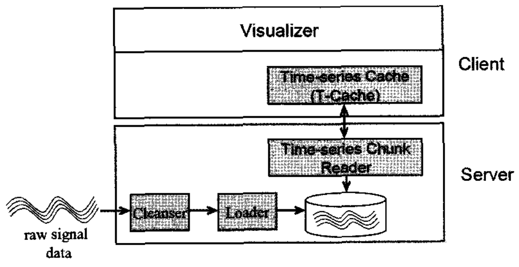


Figure 3 Overall system architecture of our analyzer software

3.2 Implementation of T-Cache

Before we discuss implementation details of T-Cache, we first formally define the time-series cache line as the following:

Definition 1 The *time-series cache* line is a list of signal segments for a fixed in a list of signal segments for a fixed interval. A signal segment consists of signal time-series data for a specific sensor. The time-series cache line is used as a fetch unit and as a replacement unit.

Figure 4 shows the structure of the time-series cache line. Here, the header contains the starting and ending distance of the data in the time-series cache line.

Although users request specific signal data for a certain distance, T-Cache retrieves corresponding time-series cache line from the server as a chunk and returns a pointer to the specific signal data. Doing this allows us to provide users fine level access to data without sacrificing performance.

The smart cursor is provided for transparent access to time-series data either on cache or on the server. By using the smart cursor, users need not to care whether data pointed to by a smart cursor is in the cache or not. If the data requested exists in the cache, the physical pointer in the smart cursor can immediately access the data in the cache. Otherwise, T-Cache loads the relevant time-series cache line containing the data requested to the cache and makes the smart cursor point to the data in the cache.

To facilitate replacement of time-series cache lines, we use a concept of the {\it resident segment descriptor} (RSD), which points to a segment in a time-series cache line. Smart cursors can point to only RSDs, not directly to a signal segment. This enables us to replace signal segments from the cache at any time by invalidating pointers in RSDs. Although our current implementation replaces all segments in a time-series cache line, theoretically, only some signal segments in a time-series cache can be replaced out. Figure 5 shows the relationship of the smart cursors and the RSD.

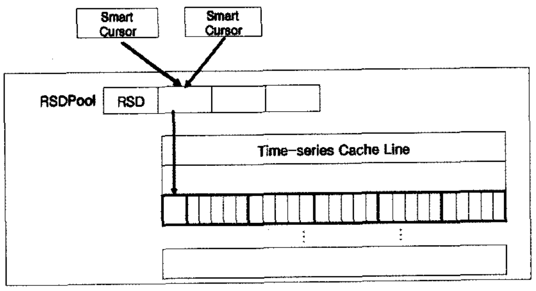


Figure 5 Relationship between smart cursors and RSDs

When we first load a time-series cache line from the server, we need to remember a pair of values for the starting distance of the cache line and the physical address of the time-series cache line in the cache. For this purpose, T-Cache maintains an in-memory hash table called the cache line table. When users want to access a cache line from cache, we first look up the cache line table without blindly calling the server.

T-Cache also maintains an additional in-memory hash table called the reverse cache line table. Here, keys in the hash table are pointers to cache lines, and values in the hash table are pointers to RSDs.

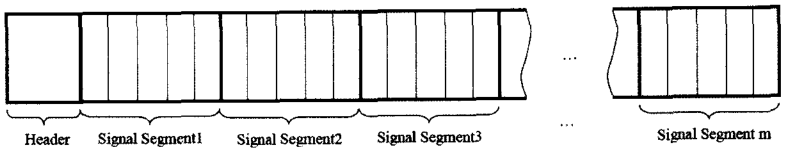


Figure 4 Time-series cache line

When we replace a cache line, we need to invalidate all the RSDs pointing to the cache line. This hash table accelerates to find those RSDs.

We note that we use red black trees [14] for the two in-memory hash tables. Red black trees provide the best possible guarantees for various worst case operations.

Example 1 Figure 6 shows various data structures used in T-Cache. As you see here, a smart cursor points to a RSD r_1 that in turn points to a time-series cache line t_1, the cache line table stores the pair of addresses, the starting address and in-memory address of t_1, and the reverse cache line table stores a pair of t_1 and r_1.

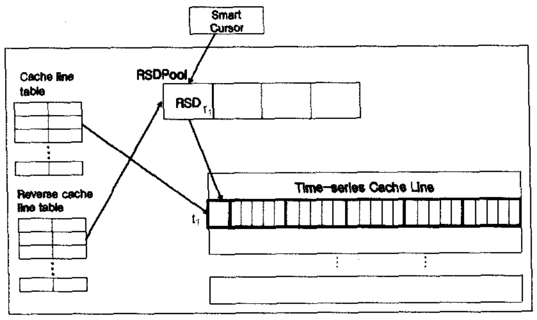


Figure 6 Various data structures used in T-Cache

4. Performance Evaluations

In this section we test our proposed approach with a comprehensive set of experiments. We describe the setup for the experiments in Section 4.1 and present the results of the experiments in Section 4.2.

4.1 Experimental setup

We use real data that an intelligent PIG collected from the pipeline spanning from Haechon to Daemi (about 45 kilometers long). The size of the data is about 3.5 Gbytes. All the experiments are done on Windows Server 2003 with Pentium3 851 MHz PC.

Table 1 shows parameters used in the experiments. We use both sequential and repetitive patterns. The sizes of the repetitive patterns are 300~2100 m with incremental increase of 300 m. We also varied the size of the cache; 0~Mbytes, 25~Mbytes, 50~Mbytes and 100~Mbytes. As a reference, 500 m of pipeline data is approximately 25 Mbytes.

Table 1 Parameters used in the experiments

type	value
pattern	sequential, repetitive
repeating distance	300, 600, 900, ⋯ , 2100 m
cache size	0 (= no cache), 25, 50, 100 Mbytes

4.2 Performance Evaluations

For the repetitive pattern, the performance of T-Cache is strongly affected by the physical distance being repeated and the size of cache. Figure 7 shows the results of performance for the repetitive pattern as we vary the physical distance being repeated and the size of cache. With the cache size being 25 Mbytes, we can store 500 meters worth of pipeline data. Thus, with a cache size larger than the size of the data accessed repetitively, the number of times the server is called is not changed regardless of how many times we repeat the data. However, if the size of the data being repeated exceeds the size of the cache, the performance degrades gracefully. Compared with the RDBMS-based repository, T-Cache provides by up to 31.9

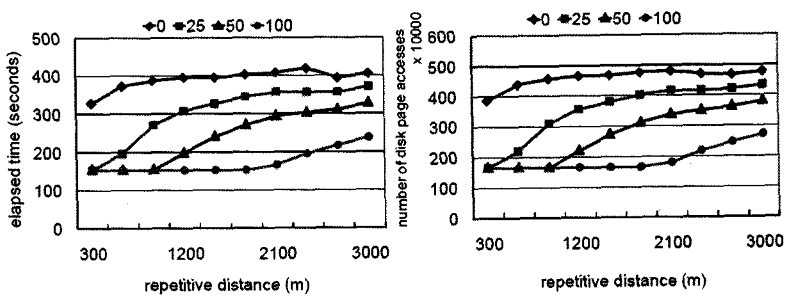


Figure 7 Experimental results for the repetitive pattern

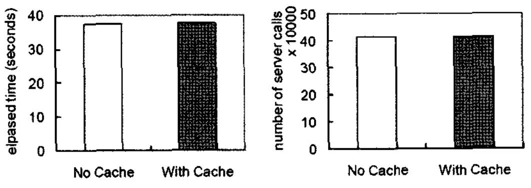


Figure 8 Experimental results for the sequential pattern

times speedup.

Regarding the sequential pattern, any caching technique can not enhance performance, but degrade performance due to caching overhead. With robust data structures and algorithms for T-Cache, however, very little overhead is caused, even for the sequential pattern.

Figure 8 shows experimental results of the sequential pattern when the cache size is 100 Mbytes. As we expect, the number of times the server is called is the same regardless if we use caching or not. Regarding the elapsed time, T-Cache is almost the same as for the system that does not use any caching. This shows our caching maintenance overhead is almost negligible.

We can conclude that T-Cache is robust and effective for both sequential and repetitive patterns. T-Cache performs much better especially for the repetitive pattern type. In our experiment, we do not consider the communication cost in a client/server environment. However, in a client/server environment, effective caching using T-Cache can further reduce communication cost.

5. Conclusions

In this paper, we proposed a fast cache manager, called T-Cache, by considering pipeline sensor data as multiple time-series data and by efficiently caching the time-series data using T-Cache. Unlike the conventional PIG analyzer software, we use T-Cache to efficiently cache data, which allows us to avoids reading signal data directly from the server for each user's request. We also provided detailed data structures and algorithms used for T-Cache. Experimental results showed that T-Cache performs much better for the repetitive pattern in terms of the server calls and the elapsed time. Even with the sequential pattern, T-Cache shows almost the same performance as a system that does not support time-series data caching, indicating caching overhead in T-Cache is negligible.

References

[1] SCADA in the Energy Industry -- A Janus View, EnergyPulse, 2004. (also available at http://www.newton-evans.com/news/EnergyPulseArticle.pdf)

[2] http://www.businessweek.com/magazine/content/02_17/b3780129.htm.

[3] R. Agthoven, "Ultrasonic Inspection of Riser a New and Simple Approach," *European Conference on Non-Destructive Testing (ECNDT)*, Vol.3, No.11, Nov. 1998.

[4] BJ Pipeline Inspection Services, GEODENT/GEODISPLAY Software Manual, 1997.

[5] S. Westwood and D. Hektner, Data Integration Ensures Integrity, BJ Services company, Mar. 2003.

[6] P. Michailides et al., NPS 8 Geopig: Inertial Measurement and Mechanical Caliper Technology, BJ Services company, June 2002.

[7] http://www.tuboscopepipeline.com/Products.htm

[8] D.K. Kim et al., "Development of the Caliper System for a Geometry PIG Based on Magnetic Field Analysis," *KSME International Journal*, Vol. 17, No.12, pp. 1835-1843, 2003.

[9] http://www.gepower.com/pii.

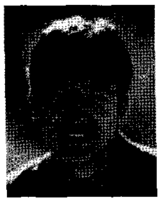
[10] http://www.3p-services.com.

[11] W. Han, K. Whang, and Y. Moon, "A Formal Framework for Prefetching Based on the Type-Level Access Pattern in Object-Relational DBMSs," *IEEE Transactions on Knowledge and Data Engineering*, Vol.17, No.10, pp. 1436-1488, Oct. 2005.

[12] A. Kemper and D. Kossman, "Dual-Buffering Strategies in Object Bases," In *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, 1994.

[13] W. Han and et al., "A Scalable Pipeline Data Processing Framework Using Database and Visualization Techniques," In *Proc. Int'l Conf. on Intelligent Computing*, pp. 334-344, Aug. 2007.

[14] T. H. Cormen et al., Introduction to Algorithms, MIT Press and McGraw-Hill, 2001


신 제 용
2005년 경북대학교 컴퓨터공학과 졸업(공학사). 2007년 경북대학교 컴퓨터공학과 졸업(이학석사). 2007년~현재 ITMEX SYI 근무

이 진 수
2006년 경북대학교 컴퓨터공학과 졸업(공학사). 2006년~현재 경북대학교 컴퓨터공학과 대학원 재학 중(석사 과정). Database system, Similarity search, Progressive optimizer

김 원 식
2003년 계명대학교 컴퓨터공학과(학사) 2005년 경북대학교 컴퓨터공학과(석사) 2006년~현재 (주) 다음커뮤니케이션 메일개발팀. 관심분야는 메인 메모리 데이타베이스, 대용량 메일 시스템

김 선 효
2003년 안동대학교 멀티미디어공학 졸업(공학사). 2005년 경북대학교 컴퓨터공학 졸업(공학석사). 2007년~현재 동양대학교 직원

윤 민 아
2006년 경북대학교 컴퓨터공학과 졸업(공학사). 2006년~현재 경북대학교 컴퓨터공학과 석사과정

한 욱 신
1994년 경북대학교 컴퓨터공학과 졸업(공학사). 1996년 한국과학기술원 전산학과 졸업(이학석사). 2001년 한국과학기술원 전산학과 졸업(공학박사). 2003년~현재 경북대학교 컴퓨터공학과 조교수

정 순 기
1990년 경북대학교 컴퓨터공학과 졸업(공학사). 1992년 한국과학기술원 전산학과 졸업(이학석사). 1997년 한국과학기술원 전산학과 졸업(공학박사). 1997년~1998년 University of Maryland, Research Associate. 2001년~2002년 University of Southern California, Research Associate. 1998년~현재 경북대학교 컴퓨터공학과 부교수. 1999년~현재 ㈜아이디스 기술고문. Virtual Reality, Computer Vision, Computer Graphics, HCI

박 세 영
1980년 경북대학교 전자공학과 졸업(공학사). 1982년 한국과학기술원 전한학과 졸업(이학석사). 1989년 프랑스 파리 7대학 전산학과 졸업(공학박사). 1982년~2000년 ETRI 연구부장. 2000년~2003년 서치캐스트(주) 대표이사. 2003년 정보통신연구진흥원 전문위원. 디지털 콘텐츠 및 SW솔루션 분야 PM. 과학기술부 과학기술혁신본부의 차세대 성장동력 사업단장. 2005년~현재 경북대학교 컴퓨터공학과 교수, 산학협력중심대학사업단 겸무