Cosmos: A Wiki Data Management System

Qinyi Wu Calton Pu Danesh Irani

College of Computing
Georgia Institute of Technology
Atlanta, GA
{qxw, calton, danesh}@cc.gatech.edu

ABSTRACT

Wiki applications are becoming increasingly important for knowledge sharing between large numbers of users. To prevent against vandalism and recover from destructive edits, wiki applications need to maintain the revision histories of all documents. Due to the large amounts of data and traffic, a Wiki application needs to store the data economically and retrieve documents efficiently. Current Wiki Data Management Systems (WDMS) make a trade-off between storage requirement and access time for document update and retrieval. We introduce a new data management system, Cosmos, to balance this trade-off. To compare Cosmos with the other WDMSs, we use a 68GB data sample from English Wikipedia. Our experiments show that Cosmos uses one-fifth of the disk space when compared to MediaWiki (Wikipedia's backend) and performs faster than other WDMSs at document retrieval.

Keywords

Revision control, social media

1. INTRODUCTION

Social media applications, such as wikis and blogs, are growing fast and attracting millions of users around the world to share and exchange knowledge. Wikipedia, ranked 7^{th} in overall web traffic [8], is the most popular of such sites. With over forty thousand requests a second and over thousands of new revisions created daily, it is critical that the underlying wiki data management system be efficient.

To recover from vandalism behavior [10, 17] and destructive edits, wiki systems have been maintaining the entire revision histories of their documents. Current approaches either use traditional version control systems (VCSs) or database management systems (DBMSs) for their data management. VCSs use disk space economically because they only store the differences between consecutive revisions of a document, but they are not suitable for retrieving earlier revisions of a document. On the other hand, DMBSs store revisions in their entirety allowing them to efficiently retrieve any version of a document, but use disk space inefficiently due to disregarding overlapped content between consecutive revisions. Using a data dump from English Wikipedia, our experiments show that the space consumption between these two approaches can be more than thirty times.

This paper introduces a new Wiki Data Management System called Cosmos. Cosmos balances the trade-off between storage requirement and access time. We use a data structure, called *partial persistent sequences* (PPSs) [19], to represent a document and its revision history. PPSs only maintain one copy of all the characters that have been inserted into a document. Furthermore, each character has a unique identifier, which is persistent over the time. A revision only maintains a selected set of identifiers, which can be used to restructure its content at anytime. PPSs use disk space economically due to its feature of one-copy per character. They can also be implemented efficiently on DBMSs to achieve fast document update and retrieval.

Our contributions in this paper consist of two parts:

- Design and prototyping of a Wiki Data Management System, *Cosmos*. Cosmos balances the trade-off between efficient storage and fast retrieval time provided by current systems. We implement the necessary functionality to support document retrieval and update.
- An experimental evaluation that compares Cosmos with the current systems by using a data dump from English Wikipedia [7]. We compare the performance of Cosmos on data storage and access time with two popular wiki systems. One is TWiki [6], which uses a VCS-based approach. The other is MediaWiki [3], which uses a DBMS-based approach. Our experiments show that Cosmos uses one-fifth of the disk space compared to the DBMS-based approach and provides the best performance for document retrieval.

Roadmap In Section 2, we first discuss the issue of data management in wiki systems, and then introduce the data structure–PPS and explain its application to WDMSs. Section 3 gives implementation detail of Cosmos. After that, we present our experimental results in Section 4. We survey related work in Section 5 and conclude in Section 6.

2. DATA MANAGEMENT IN WIKI SYSTEMS

2.1 Data Management

Wiki systems usually have some form of revision control for maintaining the revision histories of their documents. The common functionalities include putting a new document under revision control, checking in a new revision of a document, and checking out a particular revision. However, the revision control management of wiki systems is much simpler than those of source control in software configuration

management [12]. First, new updates can only be applied to the latest revision of a document. No branching is allowed in the revision history. Second, there is no concept of product space [12], which defines different versions of data objects and their relationships. Instead, each document independently maintains its revision history.

In wiki applications, a document consists of a sequence of characters. It can be updated by insert and delete operations. A new revision is created when a user commits his modification to the wiki data storage system. We assume each document has a document identifier *Did* so that we can decide whether two revisions belong to the same document. We also assume that each revision has a unique identifier *Rid*. Many revision control systems automatically generate revision numbers. For example, MediaWiki incrementally assigns a number to a new revision. In general, a wiki system uses the following three methods to manage its data:

- *creation<document name, content>*: creates a new document with the given content. This method creates a unique *Did* for the new document and a new *Rid* for this first revision.
- *update<Did, content>*: creates a new revision for the document identified by *Did*. This method creates a new *Rid* for the new revision.
- *retrieval<Did, Rid>*: retrieves the revision identified by *Rid* for the document *Did*.

2.2 Partial Persistent Sequences

A document is normally represented in a sequence data structure such as array or linked-list. Ordinary sequence data structures [13] are ephemeral in the sense that an update (i.e., insert and delete) destroys the old version. For example, in an array, an insert operation will change its structure to reserve the space for new elements, and a delete will reclaim the space for the deleted elements. However, wiki applications must maintain the old revisions of a document. In order to manage these revisions efficiently, we introduce a new data structure–partial persistent sequence (PPS). A PPS makes a sequence data structure persistent [14] in the sense that old revisions can always be accessed. We call it partially persistent because only the latest revision can be modified. For further detail, please refer to our technical report [19]. In this section, we describe its concept and supported operations to make this paper self-contained.

Conceptually, a partial persistent sequence (PPS) consists of a sequence of characters. Each character is uniquely indexed by a rational number. We call the rational position indexes *position stamps*. For example, the position stamps for "*abc*" are 0, 0.5, and 1 respectively. A PPS never deletes any elements. It has only one operation $INSERT$. For a newly inserted character, the PPS first locates the position stamps that are neighboring to the inserting point. Then it assigns a rational number that falls with the range of these two neighbors. In Figure 1, 0.3 and 0.4 are the neighbors. '*e*' is now indexed by 0.35. In the $INSERT$ operation, a new position stamp must fall within the range of its two neighbors in order to correctly record the order of characters in a document. Algorithms that are used to compute new position stamps are called *encoding schemes*. Examples of encoding schemes include dyadic rational numbers, which halving the interval between two neighbors, and Farey rational numbers, which choose mediant of two neighbors [18].

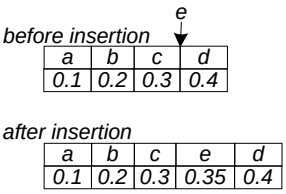

Figure 1: A PPS example

2.3 PPS Representation of a Document and its Revision History

We use a PPS to represent the entire content of a document's revision history. A document consists of two parts: *mapping* and *revision*. The mapping part records the correspondence between position stamps and their corresponding characters. The revision part contains the position stamp information for each revision. Even though the content of a document can be updated by insert and delete operations, the PPS never deletes anything. To correctly construct the content of a revision, the revision maintains an array of position stamps for those characters it contains. Figure 2 illustrates the idea. A document is represented by a PPS, which contains five position stamps. There are two revisions defined on it: Rid_i and Rid_j. Rid_i corresponds to the character sequence "$c_0c_1c_2c_4$", and Rid_j "$c_2c_3c_5$". To obtain a revision, we first obtain the array, and then sequentially concatenate the characters they point to.

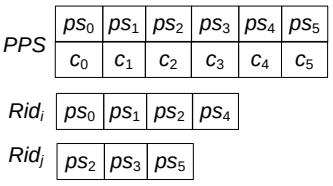

Figure 2: A document's PPS representation and two of its revisions

3. COSMOS-A WIKI DATA MANAGEMENT SYSTEM

Figure 3 shows the system architecture of Cosmos. It provides a library for wiki data management. Cosmos uses Berkeley DB [16] as its backend storage system. Berkeley DB is a general-purpose database engine that supports efficient data management on key/data pairs. In our implementation, all data are constructed in the form of key/data pairs. We describe our data schema in Section 3.1 and implementation issues in Section 3.2.

3.1 Data Schema

Figure 4 shows the data schemas used by Cosmos. The revision history of each document is managed by two tables: the *content table* and the *revision table*. The content table

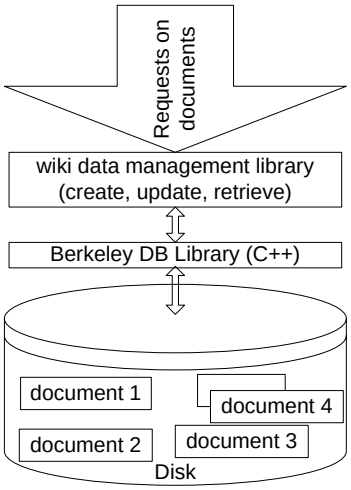

Figure 3: Architecture of Cosmos

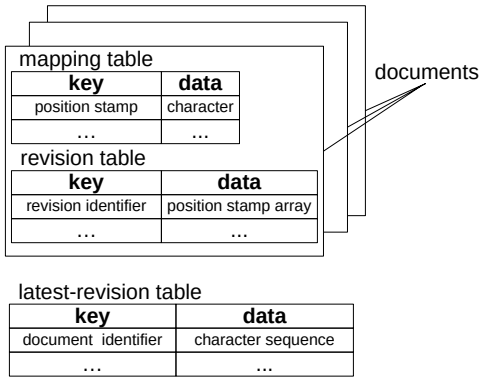

Figure 4: Data schemas used by Cosmos

stores the mapping between a position stamp and its corresponding character. The revision table stores the position stamps of each revision. To obtain a particular revision, Cosmos first obtains its position stamp array from the revision table, then look up the content table for their mappings. Since most retrieval requests ask for the latest revisions, we materialize the latest revisions of all documents in a separate table: the *latest-revision table*. Its keys are document identifiers *Did*s.

3.2 Implementation Issues

Indexing In our implementation, we use B+tree as the access method for the content table, and Hash as the access method for the revision table and the latest-revision table. These are two built-in access methods in Berkeley DB. We choose B+tree for the content table because the leaf nodes keep pointers to their neighbors to speed up sequential traversal. This is important to guarantee fast document retrieval. The ordering of position stamps is consistent to the sequential structure of a document. In a revision's position stamps, it is likely that the mappings of consecutive position stamps are stored sequentially in a B+tree's leave node. This reduces the amount of disk seek time and increases cache hits due to data locality. This is confirmed by our experiments for sequentially reading a document's revision history. For the revision table and the latest-revision table, we use hash access methods because it outperforms B+tree to serve random requests and maintains less internal information for the index structure.

Diff utility We use GNU *diff* [2] to compare the differences between two revisions. *diff* outputs differences between files line by line. Since a PPS creates a new position stamp for every character, it essentially requires a character-based *diff*-utility. In our earlier implementation, we created two temporary files, one character per line, and then executed *diff* to collect the output. However, this did not work well for two reasons. First, it is very expensive to convert a kilobyte-size file into a temporary file. Second, the *diff* utility does not output satisfactory result especially for those revisions that have significant difference. For example, if a revision contains one paragraph. Its next revision removes this paragraph and adds several new paragraphs. To find the longest common sequence between these two revisions, *diff* tries to match the characters in the old paragraph to the characters in the newly inserted paragraphs, which is not the intended result. The major problem is that the characters lose their context information after we convert them into a temporary file with one character per line. Therefore, in our current implementation, we choose the word-based granularity and construct the temporary files in the form of a word per line.

Limited precision bits Position stamps are rational numbers. A PPS can run out of precision bits for new characters due to the restrictions of the underlying computer architecture. There are two solutions to this problem. One is to use a specialized floating-point library [15] to overcome these restrictions, and the other is to create multiple PPSs for a document. If a document runs out of precision bits when creating a new revision, we create a new PPS based on the content of the new revision. Future edits will be redirected to the new PPS. Under this approach, each PPS has a unique identifier. A revision needs to maintain which PPS it belongs to. The experiments in Section 4 uses this strategy. The best strategy for solving the problem of limited precision bits will be conducted as future work.

Compressed position stamps in a revision Each revision records the positions stamps for all the character it contains. If we keep one position stamp per character, the size of position stamps would become larger than the content of the revision itself. Cosmos avoids this problem by storing position stamp information in a compact form. For the characters whose position stamps are assigned contiguously, we only store the position stamps of the left-most character and the right-most character and the distance between the consecutive characters. In our current implementation, newly inserted character sequence evenly divides the space between the two characters neighboring with the inserting point. For example, if we insert "*cde*" between '*a*' and '*b*' with position stamp being 0.1 and 0.9 respectively, the position stamps for the new characters would be $0.3, 0.5$, and 0.7 respectively. Instead of saving each of them individually, the new position stamps are saved in the compact form $< 0.1, 0.9, 0.2 >$, where 0.1 is the position stamp of the left-most character, 0.9 the right-most character, and 0.2 the distance between the consecutive characters.

4. EXPERIMENTS

We compare the storage requirements performance of Cosmos with two other wiki applications: MediaWiki and TWiki. We choose these for their different approaches in the underlying data management, which we detail below:

- *MediaWiki* uses MySQL as its backend storage system. MediaWiki maintains three tables: *page table*, *revision table*, and *text table*. The page table records the meta-information associated with a document such as the identifier of its latest revision and its byte length. The revisions of all the documents are stored in the revision table. When a new revision is created, it is automatically assigned a unique integral identifier. The revision table also records the meta-information such as update timestamp and username. The text table stores the real content of all the revisions. We follow MediaWiki's schema and import Wikipedia data dump into MySQL for our experiments.
- *TWiki* uses RCS [4] as its backend storage system. Since, only the latest revision is allowed to be updated, RCS has only one branch in its revision tree with each revision is incrementally labeled as $1.1, 1.2, \ldots$. In our experiments, we store the revision history of each document by creating a file with the document name and checking in all its revisions in chronological order.

4.1 Experiment Setup

Hardware configuration All experiments are conducted on a 64-bit GNU/Linux machine with Intel Core 2.83GHz CPU, 4GB RAM, and 1-Terabyte SATA hard disks.

Software configuration We used MediaWiki 1.13.5 with MySQL version 5.1.30 and TWiki with RCS 5.7.

Data set Wikipedia provides full-text access to all documents and their revision histories. We use a dump of the English Wikipedia (enwiki-20080103-pages-meta-history.xml.7z [7]) which is around 850GB in size. It has more than two hundred thousand documents, with over 35 million revisions in total. We use the WikiXRay parser[9] to import the data into a MySQL DBMS. We do not use the whole data set due to space and computation constraints. Instead, we uniformly sample 10 percent of documents and store their revisions in different systems. In the sampled data set, there are 20,039 documents and 3,053,829 revisions. The rest of experiments use this sampled data set.

4.2 Disk Space Consumption

In this experiment, we import the sampled data into the three systems and study their performance in terms of disk space consumption. Figure 5 shows the result with each bar representing the total disk space consumption for a particular system.

The bar labeled with "Raw" is the amount of disk space to store the sampled data in plain-text files. The amount of disk space in MediaWiki is slightly higher than Raw for two reasons. First, MediaWiki stores the entire content of each revision without considering the overlapped content between consecutive revisions. Therefore, it takes at least as much disk space as Raw. Second, MediaWiki uses some disk space to maintain an index structure for the text table. TWiki consumes the minimal amount of disk space because it only stores the delta difference between consecutive revisions. Cosmos takes one-fifth of disk space compared to MediaWiki, but five times more disk-space than TWiki. Theoretically, Cosmos only stores one-copy of every character, and revisions only store the position stamps of the characters in the revision. Since, for a new revision it takes less disk space to store the position stamps than the entire content, Cosmos is able to save disk space significantly. For this reason, the disk space of Cosmos should be close to that of TWiki. However, as discussed in Section 3.2, we have to flatten PPSs periodically. Therefore, it consumes more disk space than TWiki, but is still significantly better than MediaWiki.

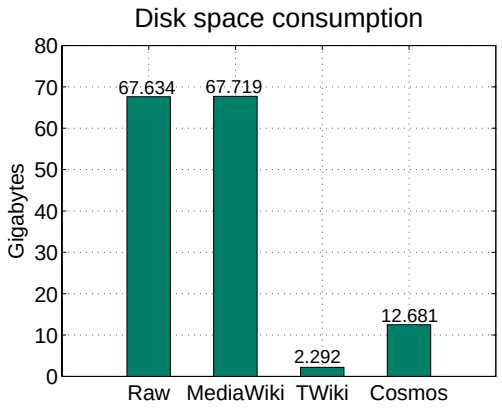

Figure 5: Total disk space consumption for the sampled data set

From the above experiment, we can see that the revisions contain a large portion of overlapped content. Otherwise, TWiki would not be only 3% of disk space compared to Raw. This is due to the fact that the size of a document normally gets stabilized after hundreds of revisions. New revisions may just paraphrase some sentences or correct some grammar errors. Therefore, the delta between consecutive revisions becomes very small compared to the whole content. Since both TWiki and Cosmos have the capability of storing only the delta difference between consecutive revisions, we expect they perform best for those documents that have long revisions. To confirm this observation, we categorize documents based on their revision length and import their revision history into these three systems. Figure 6 shows the results. We create three categories with the length of revision history being $(0, 100]$, $(100, 1000]$, and $(1000, +\infty]$. We can see that both TWiki and Cosmos perform better for the documents with long revision history, namely categories $(100, 1000]$, and $(1000, +\infty]$, which shows that they are especially useful for well-established documents.

4.3 Latest Revision Retrieval

Wikipedia receives millions of requests daily [8]. Most requests retrieve the latest version of a document. Providing efficient access to this data has a direct impact on users' experience at Wikipedia. In this experiment, we measure the

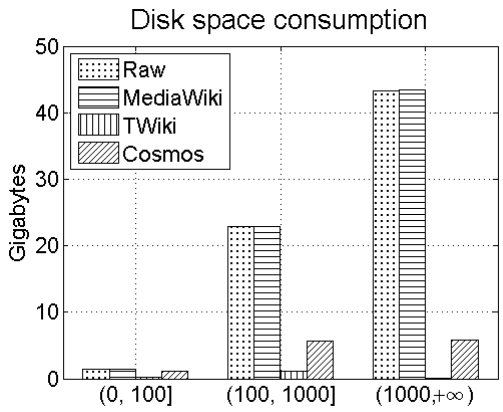

Figure 6: Disk space consumption for different categories

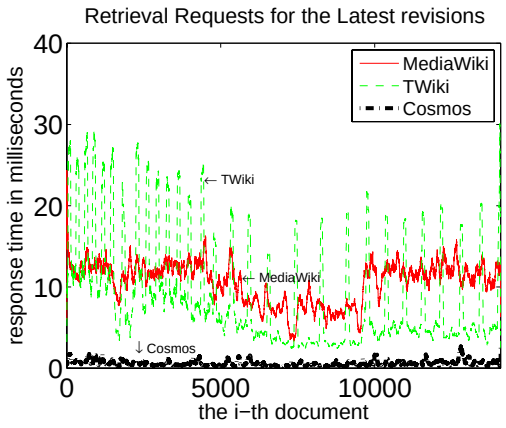

Figure 7: Response time for reading the latest revision of all the documents in the sampled data set. The x-axis is the i-th document in the sampled data set. The documents are ordered in ascending order according to the revision size.

performance of the three systems in serving these kinds of requests. Figure 7 shows the result. TWiki and MediaWiki have similar performance. (We draw MediaWiki in bold to differentiate these two curves.), but there are more fluctuations in the TWiki results. Cosmos performs best with a response time of a few milliseconds on average. We consider two reasons for this. First, Cosmos materializes the latest revisions of all documents in a separate table. Even though MySQL in MediaWiki indexes their revisions, the index structure is much larger than that of Cosmos because all the revisions (not just the latest one) are put in a single table. Therefore, Cosmos traverses a much smaller index structure than MySQL. Second, TWiki stores the latest revision of a document and its deltas in a single file. The cost of obtaining the latest revision is equal to the cost of reading this file from the underlying file system. Parsing this large file could contribute to the high cost in TWiki. A better understanding of this problem requires a look into the internal implementation of RCS in TWiki and we leave this as future work.

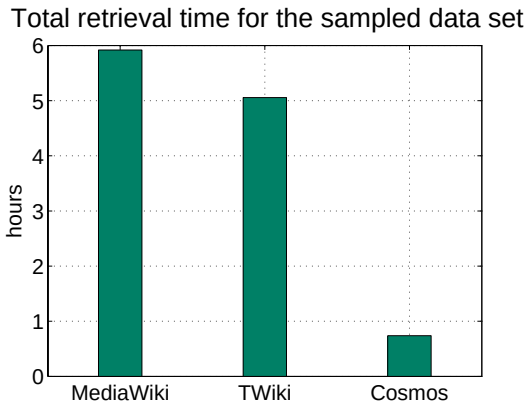

Figure 8: Total time for sequentially read all revisions of the sampled documents

4.4 Chronologically Retrieve the Revisions of a Document

Some applications have the requirement of reading the entire revisions of a document in chronological order. For example, a lot of research [10, 11, 17] has been conducted to study the statistical significance of documents for vandalism detection and their content evolvement. Considering most popular wiki systems contain thousands of documents with millions of revisions, it is important to support efficient revision retrieval to satisfy this requirement. In this section, we first present the experimental result for reading the entire revision history of the sampled data set. To obtain a deeper understanding on the performance of three systems we then present the results for three individual documents.

To compute the total retrieval time for the sampled data set, a workload generator processes one document at a time and sends retrieval requests for its revisions in chronological order to those systems. We collect timestamp information before and after the execution to compute the response time for each request. Finally, we sum up the response time of all the requests to obtain the total retrieval time. Figure 8 shows the result. Cosmos reduces the total retrieval time at a factor of eight compared to MediaWiki and a factor of seven compared to TWiki.

To better understand on their performance, we conduct another set of experiments to measure the total retrieval time for individual documents. We choose a document based on two factors: the length of its revision history and its size. The documents at Wikipedia have variable-length of revision history. Some have few than a hundred of revisions, while others have thousands of revisions. The sizes of documents also vary, ranging from several kilobytes to hundreds of kilobytes. To choose documents that are representative, we draw the histogram of data size for documents in the sampled data set and obtain the information for what kinds of documents count for the major content. From Figure 9, we can see that documents with the revision length ranging between 500 and 3000 accounts for the major portion of data size. In our experiments, we randomly choose three documents with revision length around 500, 1500, and 3000, namely "Roman Emperor", "Northern Ireland", and

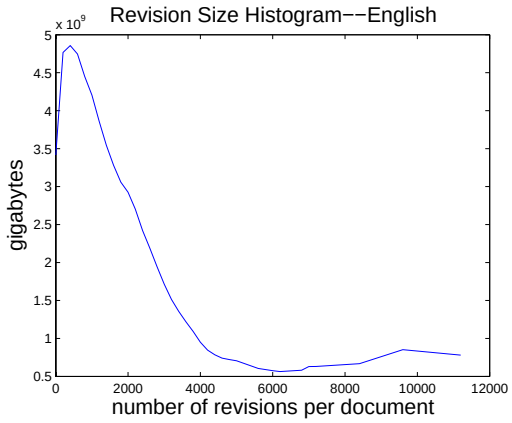

Figure 9: Histogram of revision size

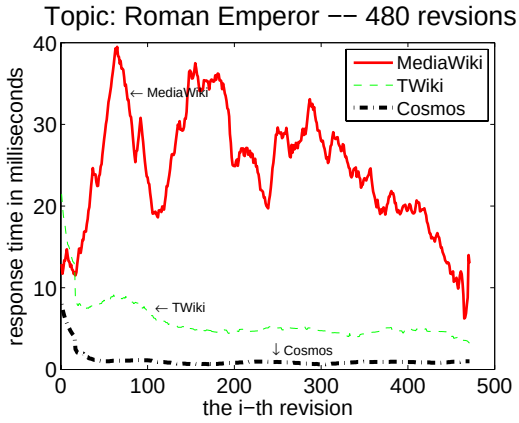

Figure 10: Response time for sequential retrieval requests

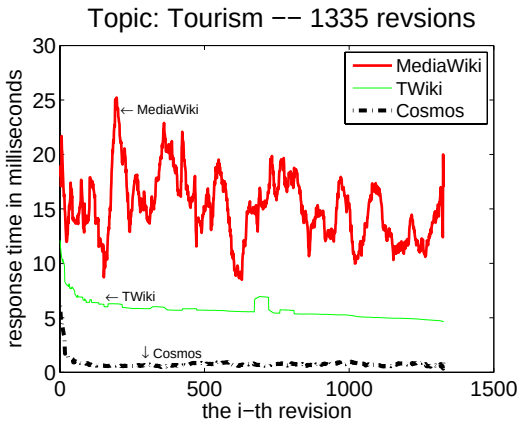

Figure 11: Response time for sequential retrieval requests

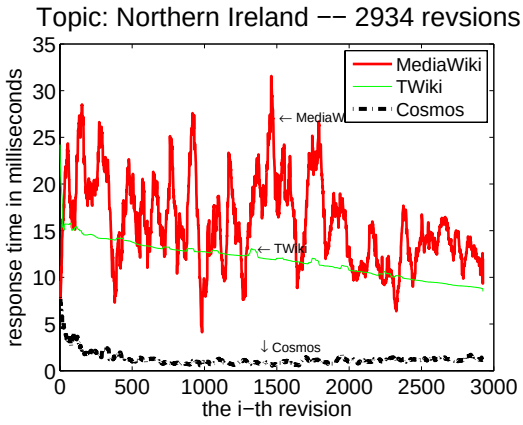

Figure 12: Response time for sequential retrieval requests

"Tourism".

We have two observations based on the results in Figure 10-12:

- As the length of revision history increases, the performance of TWiki gets closer to MediaWiki.
- Both TWiki and Cosmos perform worse than MediaWiki in reading the first revision of a document. For example, in the "Roman Emperor" document, it takes Cosmos 78 milliseconds to get a revision, TWiki 225 millisecond, and MediaWiki 74 millisecond. But after the first revision, both Cosmos and TWiki perform at an order of magnitude fast than MediaWiki.

There are three possible reasons for these results:

- Cosmos stores one-copy for each character in a document. Each revision only stores the position stamps for the characters it contains. Since the size of position stamps is much smaller than the actual content of a revision, Cosmos reads much less data than that of MediaWiki, which reduces disk I/O cost significantly.
- TWiki only materializes the content for the latest revision of a document. Old revisions are represented in the form of delta, i.e. as differences between revisions. To get a particular revision, TWiki applies the deltas backward to reconstruct that revision. Therefore, it is slowest to get the oldest revision. That explains why the curves of TWiki go down slightly in all three cases. The response time of the first revision of TWiki being two orders of magnitude higher than other revisions is because it has to read the latest revision and applies the delta difference all the way back to the very first revision. After that, the data needed for constructing the rest of revisions is in memory. Since the cost of applying delta to the latest revision is much faster than the cost of reading data from disk, it is efficient to get the rest of revisions. However, the performance of TWiki becomes worse as the length of revision history increases. As can be seen from Figure 10 to Figure 12, the average response time of TWiki increases. because the computation cost of applying delta to the latest version of a document starts to become influential.
- MediaWiki has to read a much larger data set than the two other systems, it suffers from high I/O cost.

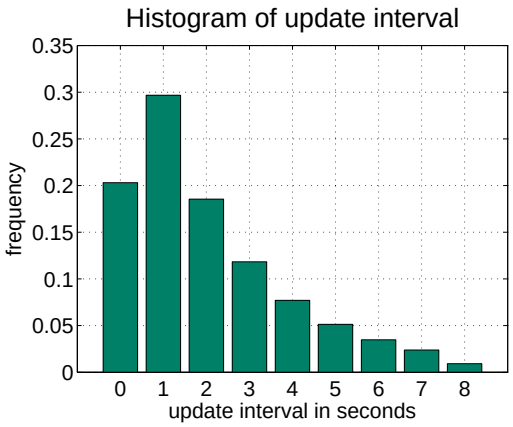

Figure 13: Histogram of update interval

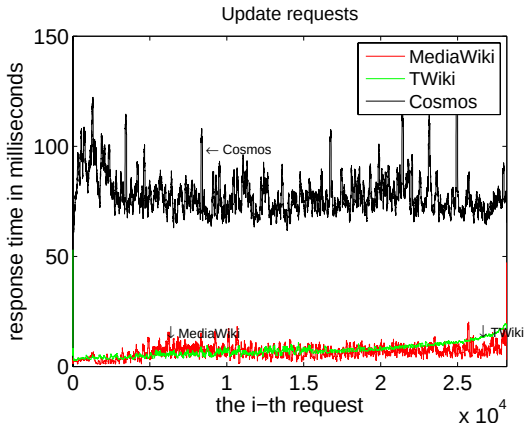

Figure 14: Response time per revision for writing 1500 documents with 23455 revisions

Another reason is that revisions of a document are not stored sequentially on disk. MediaWiki uses B+tree to index its revisions and use the revision identifiers as keys. Because the way MediaWiki organizes its data, revisions from the same document are not assigned contiguous identifiers, which increases disk seek time significantly.

4.5 Update Requests

Users create many new documents and revisions at Wikipedia. Figure 13 shows the update frequency of Wikipedia in 2007. The x-axis is the elapsed time between consecutive updates measured in seconds. On average, there is a new update every two seconds.

In our experiments, we measure the three systems' performance on update requests by composing 1500 documents with total 28193 revisions. A workload generator sequentially sends update requests to these three systems. Each time, the workload generator randomly chooses a document and sends an update request to create a new revision for that document. We collect the timestamp information between and after the execution to calculate the response time. The result is shown in Figure 14. MediaWiki and TWiki perform better among the three systems in terms of response time. It takes almost constant time to insert a new entry into MediaWiki. TWiki is slightly worse because it needs to read the content of current version and compute the delta difference. By comparison, Cosmos has a much higher cost than other two systems due to the current implementation for diff utility. Cosmos calls diff utility from command-line by starting a system shell in C++, then parses the result file. Furthermore, word-based diff is more expensive than the line-based diff because it includes additional cost of converting files to temporary files with one word per line. We will further investigate the impact of diff utility on the performance of update requests as the future work. From Figure 14, we can see that the average response time of Cosmos is around 80 milliseconds, which is enough to support the current workload at Wikipedia. Therefore, Cosmos is able to provide satisfactory performance for update requests.

5. RELATED WORK

Data structures for text sequences Many kinds of data structures to represent text sequences have been proposed in the literature [13] such as linked lists, and gap arrays. These data structures are used to represent a document in memory and support efficient insert and delete operations during an editing session. When the editing session is over, the document is stored on disk as a sequence of characters. They do not address the concern of version management.

Version control systems Version control systems are used to automate the storing, retrieval, logging, identification, and merging of revisions. Traditional VCSs (such as CVS [1], RCS [4], and Subversion [5]) are mainly used for archiving purpose and not target on these online version management systems that need to not only support version control, but also support a large amount of traffic for document update and retrieval. TWiki [6] reports a performance impact if there are more than 20,000 pages put under RCS. Therefore, many wiki systems choose a DBMS to support its version control by relying on its mature technique in indexing, replication and clustering. In the DBMS-based approach, the versions of a document are stored independently on disk and are indexed by access methods such as B+tree and Hash. An update request creates a new version. A retrieval request reads a version by using its version identifier. The DBMS-based approach supports fast read and write. However, it disregards the overlapping content between consecutive versions and a major portion of disk is used for storing redundant data. Cosmos balance the tradeoff between these two approaches and support efficient document update and retrieval with much less disk space.

Persistent data structures Various persistent data structures have been proposed in the literature due to their usefulness in a variety of applications such as computational geometry and programming languages [14]. The PPS data structure is one kind of persistent data structures. It never deletes the content of old versions and preserves necessary position information so that any version can be dynamically reconstructed out of the structure. Strictly speaking, PPSs are partially persistent in that all versions can be accessed but only the latest version can be updated compared to fully persistent data structures that updates can be applied to

any of the revisions. We take the first initiative to make a sequential data structure persistent and apply it to version control for document management. A distinctive feature of our approach is that it is *declarative*. Versions are represented are key/data pairs, which can be efficiently managed by any key/data management systems such as Berkeley DB. By comparison, old approaches are *constructive*. They need to maintain necessary timestamp information and pointers to the data in old versions. Reconstructing an old version requires a traversal of the structure by following the pointers based on the meta-information.

6. CONCLUSION

We propose a new wiki data management system, Cosmos, to achieve a balance between low disk-space consumption and efficient document retrieval and update. We present its design and implementation, based on partial persistent sequences, as well as demonstrate its performance using a representative sample (68GB) of Wikipedia data. Our experiments show Cosmos consumes one-fifth of the disk-space and achieves an order of magnitude speed-up in document retrieval when compared to MediaWiki (stored on the MySQL relational database). When compared to version control systems, although Cosmos consumes five times as much disk-space, it decreases the sequential revision access time by a factor of five.

We note that the document update time for Cosmos is higher than that of MediaWiki and TWiki, but at about 100ms it is well within the human reaction time. We plan to improve this by investigating other options for word based diff as well as different encoding schemes for the partial persistent sequence data structure.

7. REFERENCES

[1] CVS. `http://www.nongnu.org/cvs/`.

[2] GNU diffutils. `http://www.gnu.org/software/diffutils/`.

[3] MediaWiki. `http://www.mediawiki.org/wiki/MediaWiki`.

[4] RCS. `http://www.gnu.org/software/rcs/`.

[5] Subversion. `http://subversion.tigris.org/`.

[6] TWiki. `http://twiki.org/`.

[7] Wikipedia data dump download. `http://download.wikimedia.org/enwiki/`.

[8] Wikipedia statistics. `http://en.wikipedia.org/wiki/Most_viewed_article`. [Online; accessed March-2009].

[9] WikiXRay. `http://meta.wikimedia.org/wiki/WikiXRay_Python_parser`. [Online; accessed March-2009].

[10] Thomas B. Adler and Luca de Alfaro. A content-driven reputation system for the wikipedia. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 261–270, New York, NY, USA, 2007. ACM Press.

[11] Luciana S. Buriol, Carlos Castillo, Debora Donato, Stefano Leonardi, and Stefano Millozzi. Temporal analysis of the wikigraph. In *WI '06: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 45–51, Washington, DC, USA, 2006. IEEE Computer Society.

[12] Reidar Conradi and Bernhard Westfechtel. Version models for software configuration management. *ACM Comput. Surv.*, 30(2):232–282, 1998.

[13] Charles Crowley. Data structures for text sequences. `http://www.cs.unm.edu/~crowley/papers/sds/sds.html`.

[14] J R Driscoll, N Sarnak, D D Sleator, and R E Tarjan. Making data structures persistent. In *STOC '86: Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 109–121, New York, NY, USA, 1986. ACM.

[15] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. Mpfr: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw.*, 33(2):13, 2007.

[16] Michael A. Olson, Keith Bostic, and Margo I. Seltzer. Berkeley db. In *USENIX Annual Technical Conference, FREENIX Track*, pages 183–191, 1999.

[17] Reid Priedhorsky, Jilin Chen, Shyong (Tony) K. Lam, Katherine Panciera, Loren Terveen, and John Riedl. Creating, destroying, and restoring value in wikipedia. In *GROUP '07: Proceedings of the 2007 international ACM conference on Supporting group work*, pages 259–268, New York, NY, USA, 2007. ACM.

[18] Vadim Tropashko. Nested intervals tree encoding in sql. *SIGMOD Rec.*, 34(2):47–52, 2005.

[19] Qinyi Wu and Calton Pu. Partial persistent sequences and their application to collaborative editing. Research Report GIT-CERCS-09-07, Georgia Institute of Technology, 2009.