(19) Bundesrepublik Deutschland
Deutsches Patent- und Markenamt

(10) **DE 10 2008 024 809 B3** 2009.11.19

(12) **Patentschrift**

(21) Aktenzeichen: **10 2008 024 809.6**
(22) Anmeldetag: **23.05.2008**
(43) Offenlegungstag: **–**
(45) Veröffentlichungstag der Patenterteilung: **19.11.2009**

(51) Int Cl.[8]: ***G06F 17/30*** (2006.01)
G06F 12/00 (2006.01)

(73) Patentinhaber:
Universität Konstanz, 78464 Konstanz, DE

(74) Vertreter:
Gramm, Lins & Partner GbR, 30173 Hannover

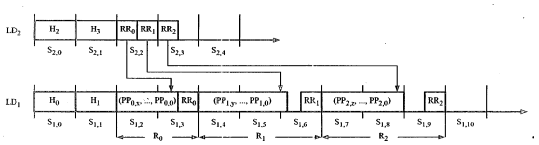
(72) Erfinder:
Kramis, Marc Ives Maria, Kreuzlingen, CH

(56) Für die Beurteilung der Patentfähigkeit in Betracht gezogene Druckschriften:

US	**71 11 233**	**B1**
US	**69 38 075**	**B1**

(54) Bezeichnung: **Verfahren zur Speicherung einer Mehrzahl von Revisionen von baumstrukturartig verknüpften Datenfamilienteilen**

(57) Zusammenfassung: Ein Verfahren zur Speicherung einer Mehrzahl von Revisionen (R_r) von baumstrukturartig verknüpften Datenfamilienteilen, die jeweils eine Anzahl von miteinander baumartig verknüpfter Datenseiten ($P_{r,p}$) mit einer zentralen Eltern-Datenseite ($PP_{r,0}$) und von dieser ausgehenden Kinder-Datenseiten ($PP_{r,x}$) haben, wird beschrieben. Für jede Revision (R_r) wird ein Haupt-Referenzzeiger (RR_r) auf die Eltern-Datenseite ($PP_{r,0}$) der jeweiligen Datenseite (P_r) in einer separat abgespeicherten Liste ($NDL_{r,p}$) bereitgestellt. Die Datenseiten (P_r) haben mindestens einen Seiten-Referenzzeiger ($PR_{r,p}$) auf logisch unmittelbar nachfolgend verknüpfte Kinder-Datenseiten ($PP_{r,x}$). Knotenänderungsinformationen über Änderungen von eine Referenz auf logisch unmittelbar nachfolgend verknüpfte Kinder-Datenseiten ($PP_{r,x}$) definierende Knoten der Struktur baumartig verknüpfter Datenfamilienteile werden für jede Revision (R_r) abgespeichert.



Beschreibung

[0001] Die Erfindung betrifft ein Verfahren zur Speicherung einer Mehrzahl von Revisionen von baumstrukturartig verknüpften Datenfamilienteilen, die jeweils eine Anzahl von miteinander baumartig verknüpfter Datenseiten mit einer zentralen Eltern-Datenseite und von dieser ausgehenden Kinder-Datenseiten haben, wobei für jede Revision ein Haupt-Referenzzeiger auf die Eltern-Datenseite der jeweiligen Datenseite in einer separat abgespeicherten Liste bereitgestellt wird und die Datenseiten mindestens einen Seiten-Referenzzeiger auf logisch unmittelbar nachfolgend verknüpfte Kinder-Datenseiten haben.

[0002] Informationen können digital hierarchisch strukturiert mit Hilfe des so genannten Extensible Mark-Up-Language XML abgespeichert werden. XML ist eine Aufzeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdateien. Das XML-Format wird insbesondere für den Austausch von Daten zwischen unterschiedlichen Computersystemen eingesetzt. Ein XML-Dokument besteht aus Elementen, Attributen, ihren Wertzuweisungen und dem Inhalt der Elemente, der aus Text oder aus untergeordneten Elementen bestehen kann. Die untergeordneten Elemente können ihrerseits wieder Attribute mit Wertzuweisungen und Inhalt haben. In einem XML-Dokument können Elemente mit und ohne Attribute vorkommen. Weiterhin können Elemente vorhanden sein, die ihrerseits viele andere Elemente enthalten. Weiterhin können Elemente lediglich Text oder sogar keinen Inhalt haben. Die Elemente, Attribute und Wertzuweisungen bilden eine Baumstruktur, die einer Verzeichnis- und Dateistruktur herkömmlicher Computersysteme ähnelt. Eine XML-Datei wird von einem so genannten XML-Paser abgearbeitet, indem der Hierarchie gefolgt wird, die sich durch die Baumstruktur der Daten ergibt. Jedes Elemente, jedes Attribut, jede Wertzuweisung an ein Attribut und jeder Zeicheninhalt eines Elements ist dabei ein eigener Bestandteil des Baums, der einzeln adressierbar ist. Diese Bestandteile werden auch als Knoten bezeichnet. Jedes XML-Dokument beginnt mit einem Wurzelknoten, der den Ursprung der Daten bildet. Der Wurzelknoten verweist auf ein äußerstes, den gesamten übrigen Inhalt der XML-Datei umfassendes Element, das heißt den obersten Knoten. An einen solchen Knoten schließen sich ein oder mehrere Kindknoten an, aus deren Sicht der übergeordnete Knoten einen Elternknoten bildet.

[0003] Mit einem solchen XML-Dokument, bei dem Eltern- und Kinderdatenseiten miteinander hierarchisch verbunden sind, lassen sich systemunabhängig Daten verwalten und insbesondere Revisionen von Datensammlungen so abspeichern, dass die Revisionshistorie nachvollziehbar ist. Dies ist insbesondere dann hilfreich, wenn eine Mehrzahl von Bearbeitern an demselben Objekt arbeiten, wie bspw. der Erstellung von Textdokumenten.

[0004] Es existieren eine Vielzahl von nativen XML-Datenbanken, die eine Enkodierung zur internen Repräsentation von XML-Daten als ungereihter, geordneter Baum mit der Randbedingung repräsentiert, dass die Ordnung der Kinder jedes Knotens stabil ist. Der enkodierte Baum wird in einem Speichersystem abgespeichert. Die nativen XML-Datenbanken machen den abgespeicherten Baum über ein Interface zugänglich.

[0005] Es existieren vier Hauptklassen zur Enkodierung. Das Enkodieren mit einem Schlüssel fester Größe weist jedem Knoten einen einzigartigen, nicht flüchtigen Schlüssel fester Größe zu. Ein Vergleich von zwei Schlüsseln reflektiert nicht notwendiger Weise die globale Reihenfolge der zwei aufeinander bezogenen Knoten. Jeder Knoten speichert Schlüssel seiner unmittelbaren Nachbarknoten. Aktualisierungen sind nicht skaliert, wenn ein Knoten eine große Anzahl von Kindern enthält. Das Enkodieren mit einem Schlüssel variabler Größe weist jedem Knoten einen einzigartigen, nicht flüchtigen Schlüssel variabler Größe zu. Einen Vergleich von zwei Schlüsseln reflektiert immer die globale Ordnung der zwei aufeinander bezogenen Knoten. In Abhängigkeit einer Einfügeposition kann die Größe des Schlüssels unbegrenzt ansteigen. Aktualisierungen sind für ständige Einfügungen in der angrenzenden Nachbarschaft eines einzelnen Knotens nicht skaliert.

[0006] Die Positionenkodierung weist jedem Knoten einen einzigartigen, flüchtigen Schlüssel fester Größe zu. Ein Vergleich von zwei Schlüsseln reflektiert immer die globale Ordnung der zwei aufeinander bezogenen Knoten. Der Schlüssel eines Knotens entspricht der aktuellen Position des Knotens, während Aktualisierungen die Position aller Knoten ändern, die den eingefügten oder gelöschten Knoten folgen. Aktualisierungen sind nicht skaliert, falls zusätzliche Indizes, wie ein Volltextindex involviert sind. Die vierte Art der Kodierung ist ein indexbasiertes Verfahren, bei dem eine oder mehrere Indizes für jeden Knoten vorhanden sind, um typische Abfragen zu beschleunigen. Hierzu muss das Original XML-Dokument für eine korrekte Rekonstruktion verfügbar gehalten werden. Aktualisierungen sind nicht skaliert, so dass das Original XML-Dokument und alle Knoten nach dem Einfügen oder Löschen von Knoten aktualisiert werden muss. Weiterhin existieren zwei grundlegenden Speichersystemklassen. Die erste Speichersystemklasse überschreibt existierende Daten bei der Ausführung von gegenwärtigen Aktualisierungen. Hierzu wird eine Baumkopie oder ein Feld üblicherweise teilweise oder vollständig im Speicher gehalten und während der Aktualisierung auf das Speichermedium zurückgeschrieben. Alte Daten werden teilweise oder vollständig überschrieben oder gelöscht.

[0007] Die zweite Speicherklasse führt Aktualisierungen mit Kopien beim Schreibvorgang ohne ein Überschreiben alter Daten aus. Hierbei wird ein Index sowie die Revision der Daten bereitgehalten. Ein neuer Index und neue Daten werden lediglich an die alten Informationen angehängt. Die Indexpunkte auf Daten entsprechen einer Revision und können damit entweder auf einen vollen Dateninhalt oder auf eine Sequenz von inkremmentalen Änderungen, das heißt auf den Unterschied zwischen den Dateninhalten zweier aufeinander folgende Revisionen zeigen, die zur Rekonstruktion der vollen Dateninhalte genutzt werden können.

[0008] Weiterhin existieren vier grundlegende Schnittstellenklassen, nämlich eine Schnittstelle zur Anwendungsprogrammierung (API = Application Programming Interface), zur integrierten Interaktion (Embedded Interaction) einer anwenderbezogenen Schnittstelle API für alleinstehende Interaktion (Stand Alone Interaction), eine anwenderbezogenen Schnittstelle API zur Web-basierten Interaktion sowie ein Text und/oder grafisches Nutzerinterface zur Mensch-Maschine-Interaktion.

[0009] C. Grün, A. Holupirek, M. Kramis, M. H. Scholl, M. Waldvogel: ”Pushing XPath Accelerator to its Limits”, in Proceedings of the First International Workshop and Performance and Evaluation of Data Management Systems, June 30, 2006 Chicago, Illinois, USA, beschreibt die Struktur von XML-Daten und Konzepte zur Enkodierung. Durch ein Set von Index-Tupel und Index-Blockstrukturen kann die Aktualisierungszeit reduziert werden. Die Dateistruktur wird mit einer Namensliste, einer Knotenliste und einer Werteliste verwaltet.

[0010] US 6,938,075 B1 offenbart ein Software-Verteilungssystem, bei dem zur Minimierung der bei der Softwareübertragung über ein Netzwerk genutzten Bandbreite nur die Datenpakete, die durch einen Knoten eines baumartig strukturierten Netzwerkes erforderlich sind, nur über die übergeordneten Verbindungen geschickt werden.

[0011] US 7,111,233 B1 offenbart ein Verfahren und ein System zur Modifikation von Programmapplikationen, wobei Daten im erweiterten Auszeichen- und Sprachenformat aus einer auf einem Computersystem laufenden Anwendung ausgegeben werden. Mit Hilfe einer Kontexttabelle, mit der ein Modell von Schreibeoperationen eine Altcomputeranwendung auf ein erweiterbares Auszeichen- und Sprachenschema abgebildet wird, wird das Verhältnis eines Schemaelementes der Ausgabedaten innerhalb des erweiterbaren Auszeichen- und Sprachenschemas und ein gegenwärtiger Kontext bestimmt. Weiterhin werden erweiterbare Auszeichnungssprachen-Tax für die Schemaelemente bis hinunter zum Schemaelement der Ausgabedaten geöffnet, um letztendlich die Daten und die geeigneten erweiterbaren Auszeichnungssprachen-Tax von einer Schreibervorrichtung in erweiterbarem Auszeichnungssprachenformat auszugeben. Die direkte Erzeugung von XML formatierten Daten von einem Altcomputersystem reduziert Reibungen in Informationsnetzwerken durch eine Vereinfachung des Informationstransfers.

[0012] Das der Erfindung zugrunde liegende Problem liegt in der effizienten Speicherung und Suche von Revisionen baumartig verknüpfter Datenfamilienteile, insbesondere unter dem Gesichtspunkt, dass die Größe und Struktur der verknüpften Datenfamilienteile von Revision zu Revision nicht vorhersagbar unterschiedlich ist.

[0013] Die Aufgabe wird mit dem Verfahren der eingangs genannten Art erfindungsgemäß gelöst durch Abspeichern von Knotenänderungsinformationen für jede Revision über Änderungen von eine Referenz auf logisch unmittelbar nachfolgend verknüpfte KinderDatenseiten definierende Knoten der Struktur baumartig verknüpfter Datenfamilienteile.

[0014] Es wird somit vorgeschlagen, für jede Datenseite einer Revision Informationen über die Änderungen bezogen auf die vorhergehende Revision in einer separaten Knotenliste als Knotenänderungsinformation abzuspeichern. Durch das Abspeichern der Knotenänderungsinformation für jede Revision wird eine weitere suchbare- und aus- wertbare Liste bereitgestellt, aus der unmittelbar die Änderungen der Knoten einer Datenseite erkennbar sind. Dort wird die Revision bereits anhand der Versionisierung der Baumsstruktur erkennbar, die über die abgespeicherten Knotenänderungsinformationen als Informationsfeld zugänglich ist. Dies hat den Vorteil, dass die Änderungsanweisung nicht erst umständlich über die Haupt- und Seitenreferenzzeiger ausgehend von einem Wurzelknoten ausgewertet werden müssen. Nicht persistent gemachte Seiten in einem Arbeitsspeicher können so aus persistenten Seitenteilen hergeleitet werden.

[0015] Die Knotenänderungsinformationen können das Hinzufügen neuer Knoten, das Löschen von Knoten und/oder die Änderung von Knoten kennzeichnen. Durch die separate Abspeicherung von Knotenänderungsinformationen in eine zusätzliche, die Änderung der Knoten einer Datenseite enthaltene Liste wird somit vermerkt, ob im Vergleich zur vorhergehenden Revision ein zusätzlicher Knoten hinzugefügt, gelöscht oder geändert worden ist.

[0016] Die Kinder-Datenseiten können die Änderung der entsprechenden Kinder-Datenseite an einer unmittelbar vorhergehenden Revision in Form von Differenzdaten enthalten. Dies hat den Vorteil, dass nicht der vollständige Inhalt der Datenseite abgespeichert werden muss, sondern lediglich die nicht binäre

Änderungsinformationen, was zu einem reduzierten Speicherbedarf führt und keine aufwändigen Berechnungen erfordert.

[0017] Die Datenseiten können aber auch den vollständigen Dateninhalt enthalten, der in der jeweiligen Revision optional modifiziert wurde.

[0018] In einer bevorzugten Ausführungsform sind die Hauptreferenzzeiger oder Kopien der Hauptreferenzzeiger auf einem weiteren Datenspeichermedium separat von den referenzierten Datenseiten abgelegt. Dies hat nicht nur den Vorteil der erhöhten Datensicherheit durch Redundanz, sondern ermöglicht auch einen schnelleren Datenzugriff, da die Suche in den Hauptreferenzzeigern auf dem separaten Datenspeichermedium parallel zum Auslesen oder Schreiben von Datenseiten erfolgen kann.

[0019] Denkbar ist aber auch, dass die Hauptreferenzzeiger oder Kopien davon in Sektoren eines Datenspeichermediums getrennt von den Sektoren desselben Datenspeichermediums abgelegt sind, in denen die referenzierten Datensätze gespeichert sind. Durch die Aufteilung in Sektoren wird sichergestellt, dass die baumartigen Strukturen der Daten zusammenhängend hintereinander abgelegt werden können, wie auch die Hauptreferenzzeiger, ohne dass die Speicherorte verwürftelt werden.

[0020] Weiterhin ist es vorteilhaft, wenn die Speicherung der Revision von Datenfamilien durch Verschlüsselung abgesichert wird. Durch Verschlüsselung sollte die Performance allerdings nicht wesentlich eingeschränkt werden. Es wird daher vorgeschlagen, ein Kryptographieverfahren basierend auf einem kodierten Hash-Nachrichten-Authentifizierungscode anzusetzen, bei dem eine kryptographische Hash-Funktion in Kombination mit einem geheimen Schlüssel eingesetzt wird. Zusätzlich wird eine symmetrische Verschlüsselung eingesetzt. Hierzu wird einer geheimer Masterschlüssel definiert und unabhängig von dem Masterschlüssel eine Zusatzkennung generiert. Diese Generierung der Zusatzkennung, die auch Salt genannt wird, wird auch als Salting bezeichnet. Aus dem Masterschlüssel und der Zusatzkennung wird ein Schlüssel abgeleitet. Dies wird auch als Stretching bezeichnet. Für jede Datenseite wird dann ein jeweils einzigartiger Ad-Hoc-Schlüssel und ein Zähler aus Ergebnissen der Ver-/Entschlüsselungsalgorhithmen übergeordneter Datenseite zur Verschlüsselung, Entschlüsselung und Authentifizierung mindestens der zugeordneten Hauptreferenzzeiger, Datenseiten und mindestens eines Kopfteils für die Gesamtheit der Revision der gespeicherten Datenfamilien abgeleitet.

[0021] Das Verschlüsselungsverfahren kann beispielsweise Gebrauch machen von dem standardisierten Advanced-Encryption-Standard, z. B. CTR-AES-256 und dem Hash-Authentifizierungsverfahren, wie z. B. HMAC-SHA-256. Das Auslesen der gespeicherten Revisionen von Datenseiten kann vorzugsweise erfolgen durch die Schritte:

a) Verifizieren der Übereinstimmung gespeicherter Kopien eines Kopfteils für die Gesamtheit der Revision gespeicherter Datenfamilien;

b) Verifizieren der Übereinstimmung einer in einem separaten Datenspeichermedium oder auf einem separaten Sektor eines Datenspeichermediums abgelegten Kopie eines Hauptreferenzzeigers für die aktuelle Revision mit einem Hauptreferenzzeiger der aktuellen Revision, die auf einem Sektor des Datenspeichermediums zusammen mit den zugehörigen Datenseiten der Revision abgelegt sind,

c) Zugreifen auf Datenseiten gewünschter Revisionen über im Hauptzeiger für die entsprechende Revision und den Seitenreferenzzeigern enthaltenen Zeigern und

d) Auswerten von Knotenänderungsinformationen für jede Revision über Änderungen von einer Referenz auf logisch unmittelbar nachfolgend verknüpfte Kinder-Datenseiten definierende Knoten.

[0022] Das Zugreifen auf eine gewünschte Revision kann mittels des binären Suchalgorithmus über die Liste der Hauptreferenzzeiger erfolgen, wobei gleichermaßen an Hand der Revisionsnummer und/oder des im Hauptreferenzzeiger gespeicherten Zeitstempels gesucht werden kann.

[0023] Die Erfindung wird nachfolgend anhand von Ausführungsbeispielen mit den beigefügten Zeichnungen näher erläutert. Es zeigen:

[0024] Fig. 1 – Speicherschema zur Speicherung einer Mehrzahl von Revisionen baumstrukturartig verknüpfter Datenfamilien mit zwei separaten Datenspeichern;

[0025] Fig. 2 – Speicherschema der Speicherung einer Mehrzahl von Revisionen von baumstrukturartig verknüpfter Datenfamilien unter Nutzung von zwei Sektoren eines einzigen Datenspeichers;

[0026] Fig. 3 – beispielhafte Darstellung des logischen und physischen Seiten- und Seitenteilbaums für vier Revisionen;

[0027] Fig. 4 – Darstellung des Zustandsdiagramms eines Verfahrens zur Speicherung einer Mehrzahl von Revisionen von baumstrukturartig verknüpfter Datenfamilien;

[0028] Fig. 5 – Flussdiagramm des Verfahrens zur Serialisierung einer im Speicher abgelegten Datenseite auf einen Festspeicher.

[0029] Fig. 6 – beispielhafte Darstellung einer Art

zur Abspeicherung von Knoten.

[0030] Fig. 1 lässt die Speicheraufteilung unter Verwendung von zwei logischen Datenspeichern LD_1 und LD_2 erkennen. Jeder Datenspeicher LD_d mit d = 0, 1, ... n besteht aus einem Feld von Sektoren $S_{d,s}$ mit der maximalen Anzahl von max(s) von Sektoren. Jeder Sektor kann beispielsweise 512 Byte breit sein. Die Datenspeicher LD_d unterstützen einen zufälligen Schreib- und Lesezugriff auf jeden Sektor $S_{d,i}$, wobei die Datenspeicher für Zufallslesen (random-read) und sequentiellen Schreibaktivitäten optimiert sein sollte. Der benötigte Speicherplatz auf den Datenspeichern LD_d wächst ständig zu jeder Zeit durch Hinzufügen weiterer Sektoren, wobei die Anzahl von Schreibzugriffen auf jeden Sektor $S_{d,i}$ begrenzt sein kann.

[0031] Es ist erkennbar, dass alle Daten und Metadaten auf dem primären logischen Datenspeicher LD_1 abgelegt werden. In dem in der Fig. 1 dargestellten Ausführungsbeispiel werden die Metadaten zudem auf dem zweiten logischen Datenspeicher LD_2 abgelegt, um die Speicherung und das Auslesen von Revisionen sowie die Suche von Revisionen zu beschleunigen. Durch die redundante Abspeicherung in dem zweiten Datenspeicher LD_2 kann der Dateninhalt des zweiten Datenspeichers LD_2 jederzeit aus dem ersten Datenspeicher LD_1 rekonstruiert und auf Konsistenz geprüft werden. Inkrementale Backups des Dateninhalts auf dem ersten und/oder zweiten Datenspeicher LD_1 und/oder LD_2 können synchron oder asynchron auf einem oder mehreren anderen weiteren Datenspeichern LD_d zur Datensicherung abgelegt werden. Als Metadaten sind zunächst sogenannte Kopfdaten H_h (header) vorgesehen, die für das Salting eine Zusatzkennung SLT_h (salt) von beispielsweise 32 Byte, eine Konfiguration $CONF_h$ von beispielsweise 448 Byte und ein Token HT_h von 32 Byte enthalten, der zur Authentifizierung der Konfiguration $CONF_h$ genutzt wird. Der Token HT_h entspricht dem Authentifizierungsalgorithmus $HMAC\text{-}SHA\text{-}256_k$ ($CONF_h$). Der Algorithmus ist aus dem Secure-Hash-Standard: Federal Information Processing Standards Publication 180-2, National Institute of Standards and Technology, August 2002, aus dem Advanced Encryption Standard (AES): Federal Information Processing Standards Publication 197, National Institute of Standards and Technology, November 2001 und aus „The Keyed Hash-Message Authentication code (HMAC): Federal Information Processing Standards Publication 198, National Institute of Standards and Technology, March 2002, hinreichend bekannt.

[0032] Die Zusatzkennungen SLT_h, d. h. die ersten 32 Byte des Headers H_h werden nicht verschlüsselt. H_h wird zweifach auf dem ersten Datenspeicher LD_1 als H_0 und H_1 und zweifach auf dem zweiten Datenspeicher LD_2 als H_2 und H_3 abgelegt.

[0033] Als Metadaten werden weiterhin Revisions-Referenzen RR_r abgespeichert, die eine Seitenteilreferenz $PPR_{r,0}$ (z. B. 32 Byte) und das Token RRT_r (z. B. 32 Byte) enthält, das zur Authentifizierung der Seitenteilreferenz $PPR_{r,0}$ verwendet wird. Das Token RRT_r entspricht dem Authentifizierungsalgorithmus $HMAC\text{-}SHA\text{-}256_k$ ($PPR_{r,0}$). Weiter enthalten die Revisions-Referenzen (RR_r) Angaben über den Autor sowie den Zeitpunkt der Erstellung der Revision R_r.

[0034] Um ausgehend von der aktuellsten Revision $R_{max(r)}$ die vorhergehende Revisionsreferenz $RR_{max(r)-1}$ auf dem zweiten Datenspeicher LD_2 aufzufinden, wird eine binäre Suche auf den Sektoren $S_{2,2,}$ ... $S_{2,max(s)-1}$ durchgeführt. Jedes Mal wird der Median des Sektors $S_{2,s}$ ausgewählt, da angenommen wird, dass dieser einige Seitenteilreferenzen $PPR_{r,p}$ bei einem Byte-offset von Null enthält. Falls das Token RRT_r gleich $HMAC\text{-}SHA\text{-}256_k$ ($PPR_{r,0}$) ist, wird die Suche in die Richtung nach rechts fortgeführt und andernfalls in die Richtung nach links. Falls dort einige Revisionsreferenzen $RR_{max(r)-1}$ auf die Revision $R_{max(r)-1}$ vorhanden sind, wird die binäre Suche diese schließlich selektieren.

[0035] Falls der zweite Datenspeicher LD_2 nicht verfügbar ist, wird wiederum eine binäre Suche mit dem ersten Datenspeicher LD_1 wie vorstehend beschrieben ausgeführt. Falls der Median keine Revisionsreferenz $RR_{r,p}$ auf eine Revision bei einem Byte-Offset von Null in einem Sektor $S_{s,1}$ zurückgibt, wird die Suche in Richtung nach rechts und links wie folgt fortgeführt. Die Revisionsreferenz $RR_{r,b}$ wird in den Sektoren $S_{1,m+kj-1,}$ ..., $S_{1,m+kj}$ ausgehend von $k_o = 1$ gestartet. Falls sie nicht gefunden wird, erfolgt eine Anpassung von $kj = -2 \cdot k_{j-1}$, bis k_j eine konfigurierbare Grenze erreicht. Anschließend wird wie oben beschrieben verfahren.

[0036] Aus logischer Sicht besteht eine im ersten Datenspeicher LD_1 abgelegte Revision R_r aus einem Baum von Datenseiten. Die Datenseiten sind beginnend mit der Wurzel-Datenseite $P_{r,0}$ aufgezählt. Anfänglich beinhaltet die Revision R_r alle Datenseiten von der vorhergehenden Revision R_{r-1}. Nachfolgende Änderungen werden nur auf Kopien der Datenseiten vorgenommen und nicht bei den Originalen. Die Kopien sind nur für die Revision R_r sichtbar und in rückwärtiger Ordnung als Datenseitenteile $PP_{r,0}$ abgespeichert. Falls die gesamte Datenseite gespeichert wird, ist die Kopie des Seitenteils als Seiten-Snapshot $PS_{r,p}$ bezeichnet. Falls nur eine Änderung zur letzten Revision gespeichert wird, ist dies als Seitenänderung $PD_{r,p}$, d. h. als nicht binäre Differenz oder Delta bezeichnet.

[0037] Aus Fig. 1 ist erkennbar, dass für jede Revision R_r eine Datenseitenreferenz RR_r vorgesehen ist, die die Datenseite $P_{r,0}$ gleich $PP_{r,0}$ referenziert, die ein Seiten-Snapshot sein muss. Alle anderen Datensei-

ten werden durch eine Seitenreferenz PR_(r, p) referenziert, die entweder eine einzelne Seiten-Snapshot $PS_{r,p}$ oder eine Sequenz von Seitenänderungen $PD_{r,p}$ referenziert, wobei die Seitenänderungen auf einen anfänglichen Datenseiten-Snapshot $PS_{r,p}$ angewandt werden, um eine Datenseite $PP_{r,p}$ abzuleiten. Der Seitenteilzähler $PPC_{r,p}$ bezeichnet die Anzahl der Seitenänderungen, die auf die anfängliche Seiten-Snapshots anzuwenden sind. Jede Seite wird dabei an eine Revision r und eine Seitennummer p gebunden, die die Seite eindeutig von allen anderen Seiten unterscheidet. Ein Seitenteil erhält somit über alle Revisionen dieselbe Seitennummer. Jede Seitenteilreferenz in einer Seitenreferenz PR_(r, p) ist mit der Revisionsnummer, während dieser dieses Seitenteil erstellt wurde, assoziiert. Eine Seitenreferenz mit dem Seitenzähler $PPC_{r,p}$ gleich Null kennzeichnet eine gelöschte Seitenreferenz.

[0038] Eine nicht persistent gemachte Seite $P_{r,p}$ enthält zwei Felder, das Seitenreferenzfeld $PRA_{r,p}$ sowie das Knotenfeld $NDA_{r,p}$. Beide Felder sind das Ergebnis aller Änderungen aus allen Seitenteilen, die auf die Seite $P_{r,p}$ bezogen sind. Die Größe der beiden Felder ist fest oder für jede Seite entsprechend ihrer entsprechenden Position in dem Seitenbaum definiert. Die maximale Größe der beiden Felder ist auf beispielsweise 256 limitiert. Jeder Eintrag in eines der Felder ist durch seine absolute Position n markiert. Ist n negativ, so bedeutet das, dass der Eintrag in einer früheren Revision geändert wurde. Der Eintrag θ ist nicht verwendet. Das erste Feld, d. h., das Seitenreferenzfeld $PRA_{r,p}$ speichert die n-te Seitenreferenzen $PR_{rp'}$, ..., PR_{rp^*}, die auf die Kinder-Datenseiten $P_{r,p'}$ ..., P_{r,p^*} zeigen. Das zweite Feld, d. h. das Knotenfeld $NDA_{r,p}$ speichert Knoten $ND_{p,n}$

[0039] Ein Knoten $ND_{p,n}$ besteht aus einer z. B. ein Byte großen Position im Knotenfeld $NDA_{r,p}$ einem Knotentyp von z. B. einem Byte und einer Nutzlast variabler Größe. Der Knotentyp kleiner 0 kennzeichnet einen gelöschten Knoten und enthält keine Nutzlast. Daher sind beispielsweise 127 verschiedene Knotenkinder verfügbar.

[0040] Ein Seitenteil $PP_{r,p}$ ist entweder ein Seiten-Snapshot $PS_{r,p}$ oder eine Seitenänderung $PD_{r,p}$. Ein Seiten-Snapshot $PS_{r,p}$ speichert das Ergebnis aller vergangenen Änderungen sowie der Änderungen während der Revision R_r. Die Knotenänderung $PD_{r,p}$ enthält lediglich die während der Revision R_r vorgenommenen Änderungen.

[0041] Das persistent gemachte Seitenteil $PP_{r,p}$ enthält zwei Listen variabler Größe, die die an der Datenseite $P_{r,p}$ während der Revision R_r vorgenommenen Änderungen reflektieren. Jede Liste enthält eingangs die Listengröße von einem Byte. Die maximale Größe beider Listen ist beispielsweise auf 256 begrenzt. Die erste Liste, die sogenannte Seitenreferenzliste $PRL_{r,p}$ speichert Änderungen an jeglicher Seitenreferenz $PR_{r,p'}$ ..., PR_{r,p^*}, die auf die Kinderseiten $P_{r,p'}$ ..., P_{r,p^*} zeigen, wobei $p < p' < ... < p^*$ ist. Die zweite Liste, die sogenannte Knotenliste $NDL_{r,p}$ speichert Änderungen an jeden Knoten $ND_{r,p,n}$. Falls das Seitenteil $PP_{r,p}$ ein Seiten-Snapshot $PS_{r,p}$ ist, speichert die Knotenliste $NDL_{r,p}$ auch den aktuellen Zustand vom Knoten, falls dieser nicht während der Revision R_r modifiziert wurde.

[0042] Erkennbar ist aus Fig. 1, dass die Referenzen RR_r auf die Revisionen gefolgt von den Headern H_2 und H_3 hintereinander mit der ersten Referenz RR_0 auf die erste Revision R_0 abgespeichert werden.

[0043] In dem ersten Datenspeicher LD_1 werden die Revisionen R_0, R_1, ..., $R_{max(r)}$ ebenfalls hintereinander unter Ausnutzung eines oder mehrerer Sektoren $S_{d,s}$ abgelegt. Dabei werden die Daten einer Referenz RR_r in den genutzten Sektoren jedoch rückwärts abgelegt, wobei an dem Ende des letzten Sektors zunächst die Referenz RR_r auf die jeweilige Revision R_r abgelegt wird. Die Seitenteile $PP_{r,p}$ der Revision R_r werden beginnend vom Anfang des ersten Sektors $S_{d,s}$ der Revision R_r mit der letzten Kinder-Datenseite, d. h. dem Datenseitenteil $PP_{r,x}$ bis zum Wurzel-Datenteil $PP_{r,0}$ abgelegt, so dass die Sequenz der Datenteile $PP_{r,p}$ mit dem Wurzel-Datenteil $PP_{r,0}$ endet.

[0044] Fig. 2 lässt eine andere Ausführungsform für die Speicherung einer Mehrzahl von Revisionen R_r von baumstrukturartig verknüpften Datenfamilien unter Verwendung eines einzigen logischen Speichermediums LD_0 erkennen. Die auf die Seitenteile verweisenden Kopien der Revisionsreferenzen RR_r auf die Revisionen R_r sowie die Kopien H_2 und H_3 der Header H_0 und H_1 werden in Sektoren $S_{0,max(s)-i}$ getrennt von den Sektoren $S_{0,i}$ für die Revisionen R_r abgelegt.

[0045] Während der Datenspeicherbereich für die Header H_0, H_1 und die Revisionen R_r sequentiell beginnend von einem ersten Sektor $S_{0,0}$ hintereinander in Vorwärts-Reihenfolge abgelegt werden, erfolgt die redundante Speicherung der Metadaten in Rückwärts-Richtung beginnend mit den Headern H_2 und H_3 am Ende der Sektoren für den zweiten Speicherbereich, d. h. am Ende des Sektors $S_{0,max(s)-1}$ wie in Fig. 2 skizziert ist.

[0046] Fig. 3 lässt die Struktur der logischen und physischen Abspeicherung von Seitenbäumen und Seitenteilbäumen für vier aufeinanderfolgende Revisionen R_0, R_1, R_2 und R_3 erkennen.

[0047] Die erste Revision R_0 basiert auf der Ursprungs-Datenseite $P_{0,0}$, die mit der Revisionsreferenz RR_0 referenziert wird. Die Ursprungs-Datenseite $P_{0,0}$ zeigt ihrerseits auf die Datenseite $P_{0,1}$.

[0048] Als Änderungen der revidierten Datenseite $P_{0,1}$ wurden die Änderungen 0A und 1B vorgenommen. Diese werden in dem Seitenteil $PP_{0,p}$ gekennzeichnet, so dass aus diesem Seitenteil $PP_{0,p}$ die Änderungen in der Revision sofort ersichtlich ist.

[0049] Der physische Seitenteil-Baum besteht aus der Revisionsreferenz RR_0, die auf die Kopie der Seite $P_{0,0}$, d. h. den Seiten-Snapshot $PS_{0,0}$ verweist, der seinerseits wieder auf dieser den Seiten-Snapshot $PS_{0,1}$ mit den Änderungen verweist.

[0050] In der folgenden Revision R_1 wurde eine weitere Datenseite hinzugefügt, so dass die Datenseite $P_{1,0}$ nunmehr in die Datenseiten $P_{1,1}$ und $P_{1,2}$ verzweigt. Der logische Seitenbaum ist beginnend von der Revisionsreferenz RR_1, die auf die Wurzel-Datenseite $P_{1,0}$ der Revision R_1 verweist, nunmehr baumartig so strukturiert, dass die Datenseite $P_{1,0}$ zwei Kinder-Datenseiten $P_{1,1}$ und $P_{1,2}$ enthält.

[0051] Als Änderungen wurden in der Kinder-Datenseite $P_{1,1}$ die Änderung „2C" und „0" und in der zweiten Kinder-Datenseite $P_{1,2}$ „0D" und „1E" vorgenommen. Diese Änderungen sind in dem Seitenteil $PP_{1,p}$ entsprechend vermerkt, wobei die Revision auch noch die Änderungen des vorhergehenden Seitenteils $PP_{0,p}$ der ersten Revision R_0 enthält.

[0052] Aus der Darstellung des physischen Seitenteil-Baums in Fig. 3 ist erkennbar, dass die revidierte Kinder-Datenseite $PD_{1,1}$ mit der Kinder-Datenseite $PS_{0,1}$ verwandt ist. Diese Kinder-Datenseite der Revision R_1 enthält allerdings keine vollständigen Dateninhalte, sondern nur die geänderten Daten, so dass der vollständige Inhalt der Kinder-Datenseite aus der Kinder-Datenseite $PS_{0,1}$ der vorhergehenden Revision R_0 und den Änderungsinformationen abgeleitet werden kann.

[0053] In der folgenden Revision R_2 ist der Datenbaum unverändert geblieben. Es wurden daher keine Datenseiten gelöscht oder hinzugefügt. Als Änderungen wurden „1B'" und „0D'" in den Kinder-Datenseiten $P_{2,1}$ und $P_{2,2}$ vorgenommen. Diese Änderungen sind zusammen mit der Änderungshistorie in den Seitenteilen PP_0, $PP_{1,p}$ und $PP_{2,p}$ verzeichnet.

[0054] Aus dem physischen Seitenteil-Baum ist erkennbar, dass die erste Kinder-Datenseite wiederum nur die Änderungen zur vorhergehenden Kinder-Datenseite $PD_{1,1}$ als Änderungsinformation $PD_{2,1}$ enthält. Auch die zweite geänderte Kinder-Datenseite $PD_{2,2}$ enthält nur die Änderungsinformation in Form der Deltas zur vorhergehenden Datenseite $PS_{1,2}$ der vorhergehenden Referenz R_1.

[0055] Bei der nächsten Revision R_3 wurde die erste Kinder-Datenseite $P_{3,1}$ zum vorhergehenden Revision $P_{2,1}$ nicht geändert. Allerdings wurde aus den Änderungsinformationen $PD_{1,1}$ und $PD_{2,1}$ auf der Basis des letzten verfügbaren Snapshots PS0,1 ein Snapshot $PS_{3,1}$ abgeleitet, der nunmehr den vollständigen Dateninhalt der Kinder-Datenseite $P_{3,1}$ enthält.

[0056] In der zweiten Kinder-Datenseite $P_{3,2}$ wurde im Vergleich zur vorherigen Revision eine Änderung „0D" vorgenommen. Diese Kinder-Datenseite $P_{3,2}$ wurde physisch nur als Änderungs-Datenseite $PD_{3,2}$ abgespeichert.

[0057] Weiterhin wurde eine dritte Kinder-Datenseite $P_{3,3}$ mit den Änderungen „0F" und „1G" hinzugefügt.

[0058] Fig. 4 lässt ein Zustandsdiagramm bei der Durchführung des Verfahrens zur Speicherung einer Mehrzahl von Revisionen von baumstrukturartig verknüpften Datenfamilienteilen erkennen.

[0059] In dem sogenannten „Wipe"-Zustand werden beispielsweise alle Sektoren einmal mit Zufallsdaten überschrieben, die nicht kodiert sind.

[0060] In dem Initialisierungs-Zustand „init" werden die Datenspeicher LD_1, LD_2 und ein Masterschlüssel MK durch den Nutzer ausgewählt. Weiterhin wird eine Zusatzkennung SLT_h unter Verwendung des Salting-Prozesses gewählt und ein Schlüssel K aus dem Masterschlüssel MK und der Zusatzkennung SLT_h abgeleitet. Ein Token HT_h wird auf den durch den HMAC-SHA-256_k ($CONF_h$)-Algorithmus vorgegebenen Wert gesetzt. Die n-te Kopie des Headers H_h wird auf einen durch die Bedingung SLT_h verknüpft mit CTR-AES-256_k ($CONF_h$) verknüpft mit HT_h vorgegebenen Wert gesetzt.

[0061] Die Header H_0, H_1, H_2 und H_3 werden dann synchron geschrieben und verifiziert.

[0062] Im Falle eines Verifikationsfehlers wird ein Neustart des Initialisierungs-Zustands veranlasst.

[0063] Der sogenannte Start-Zustand „Start" erfolgt nach der Initialisierung. Hierbei werden der Datenspeicher LD_1, der Datenspeicher LD_2 und der Masterschlüssel MK durch den Nutzer bereitgestellt. Dann werden die Header H_0, H_1, H_2 und H_3 ausgelesen und die Zusatzkennung SLT_h durch Vergleich mit den in den Headern H_0, H_1, H_2 und H_3 abgelegten Zusatzkennungen verifiziert. Ein Knotenschlüssel wird aus dem Masterschlüssel MK und der Zusatzkennung SLT_h abgeleitet und die Header H_0, H_1, terschlüssel MK und der Zusatzkennung SLT_h abgeleitet und die Header H_0, H_1, H_2 und H_3 zur Überprüfung des Tokens HT_h mit dem Algorithmus HMAC-SHA-256_k($CONF_h$) verifiziert.

[0064] Weiterhin wird die Revisionsreferenz $RR_{max(r)-1}$ auf den zweiten Datenspeicher LD_2 gesucht

und durch Vergleich mit der auf dem ersten Datenspeicher LD_1 abgelegten Revisionsreferenz $RR_{max(r)-1}$ verifiziert. Ein Fehler bei der Verifikation verursacht einen Übergang in den Wiederherstellungs-Zustand „recover".

[0065] Im Wiederherstellungs-Zustand „recover" wird der Header H wiederhergestellt. Dies kann anhand der verfügbaren Kopien des Headers erfolgen. Weiterhin wird der Inhalt auf dem ersten und/oder zweiten Datenspeicher LD_1 und LD_2 unter Verwendung der verfügbaren redundanten Daten wiederhergestellt. Nach der Wiederherstellung der Information erfolgt ein Sprung in den Stop-Zustand „stop".

[0066] Der Ausführungs-Zustand „run" wird jedes Mal durchgeführt, wenn eine Revision übergeben wird. Hierbei werden folgende Punkte abgearbeitet.

[0067] Im ersten Punkt wird die letzte Revision $R_{max(r)}$ synchron auf dem ersten Datenspeicher LD_1 geschrieben. Im folgenden Schritt wird die Revisionsreferenz $RR_{max(r)}$ auf die letzte Revision $R_{max(r)}$ auf den ersten Datenspeicher LD_1 verifiziert. Sodann wird die Revisionsreferenz $RR_{max(r)}$ synchron auf den zweiten Datenspeicher LD_2 geschrieben und dort verifiziert. Anschließend wird im fünften Schritt die Nummer der verfügbaren Revision $R_{max(r)}$, d. h. die fortlaufende Nummer der aktuellen Revision inkrementiert auf max(r) + 1.

[0068] Ein Fehler bei der Überprüfung führt zu einer zweiten Serialisierung beginnend mit Schritt 1 des Ausführungs-Zustands „run".

[0069] Wiederum erfolgt am Ende ein Sprung in den Stop-Zustand, bei der neue Änderungen nicht länger zugelassen werden. Bislang nicht abgespeicherte Änderungen werden wie in dem Ausführungs-Zustand beschrieben abgespeichert.

[0070] Falls in dem Ausführungs-Zustand der fünfte Punkt der Inkrementierung der Nummer für die aktuelle Revision nicht innerhalb einer einstellbaren Zeit erreicht wurde, erfolgt sofort ein Stop.

[0071] Fig. 5 lässt das Verfahren der Serialisierung einer im Speicher abgelegten Datenseite auf ein Gerät, wie z. B. eine Hard Disk, als Flussdiagramm erkennen.

[0072] Eine im Speicher abgelegte Datenseite wird serialisiert. Die serialisierten Datenseitenteile werden komprimiert und die komprimierten Datenseiteteile werden authentifiziert, verschlüsselt und anschließend auf das Gerät, wie beispielsweise eine Hard Disk oder einen anderen Datenträger, geschrieben.

[0073] Das Auslesen von Datenseitenteilen aus dem Gerät erfolgt, indem die Dateninhalte zunächst gelesen und die verschlüsselten Datenseitenteile entschlüsselt und authentifiziert werden. Die komprimierten Datenseitenteile werden anschließend dekomprimiert und liegen als serialisierte Datenseitenteile nunmehr unverschlüsselt vor. Nach einer Deserialisierung kann die Datenseite dann in einem Speicher, insbesondere einem flüchtigen Datenspeicher RAM, abgelegt werden.

[0074] Fig. 6 lässt eine Skizze für eine beispielhafte Art zur Abspeicherung von Knoten erkennen.

[0075] In der ersten Spalte ist der Zustand der Datenseiten $P_{3,0}$ zur Zeit der Revision R_3 dargestellt. Die Seite $P_{3,0}$ entspricht der Seitenkopie $PS_{3,0}$. Der Knoten bei der Position 1 wurde vor der Revision R_3 gesetzt. Der Knoten ist vom Typ 22 und hat den Wert <example>. Der Knoten bei der Position 2 wurde vor der Revision R_3 gesetzt. Der Knoten ist vom Typ 33 und hat den Wert „text". Der Knoten an der Position 6 wurde während der Revision R_3 gelöscht. Der gelöschte Knoten war vom Typ 33.

[0076] In der zweiten Spalte ist der Zustand der Datenseite $P_{4,0}$ zur Zeit der Revision R_4 dargestellt. Die Änderungen sind in der Differenz-Datenseite $PD_{4,0}$ abgelegt. Der Knoten bei der Position 1 wurde vor der Revision R_4 gesetzt. Der Knoten ist vom Typ 22 und hat den Wert <example>. Der Knoten bei der Position 2 wurde während der Revision R4 gesetzt. Der Knoten ist nun vom Typ 33 und hat den Wert „new text".

Patentansprüche

1. Verfahren zur Speicherung einer Mehrzahl von Revisionen (R_r) von baumstrukturartig verknüpften Datenfamilienteilen, die jeweils eine Anzahl von miteinander baumartig verknüpfter Datenseiten (P_r) mit einer zentralen Eltern-Datenseite ($PP_{r,0}$) und von dieser ausgehenden Kinder-Datenseiten ($PP_{r,x}$) haben, wobei für jede Revision (R_r) ein Haupt-Referenzzeiger (RR_r) auf die Eltern-Datenseite ($PP_{r,0}$) der jeweiligen Datenseite (P_r) in einer separat abgespeicherten Liste bereitgestellt wird und die Datenseiten (P_r) mindestens einen Seiten-Referenzzeiger ($RR_{r,p}$) auf logisch unmittelbar nachfolgend verknüpfte Kinder-Datenseiten ($PP_{r,x}$) haben, gekennzeichnet durch Abspeichern von Knotenänderungsinformationen für jede Revision über Änderungen von eine Referenz auf logisch unmittelbar nachfolgend verknüpfte Kinder-Datenseiten ($PP_{r,x}$) definierende Knoten der Struktur baumartig verknüpfter Datenfamilienteile.

2. Verfahren nach Anspruch 1, dadurch gekennzeichnet, dass die Knotenänderungsinformationen das Hinzufügen neuer Knoten, das Löschen von Knoten und/oder die Änderung von Knoten kennzeichnen.

3. Verfahren nach Anspruch 1 oder 2, dadurch

gekennzeichnet, dass Kinder-Datenseiten (PPr, x) die Änderung der entsprechenden Kinder-Datenseite (PPr, x) einer unmittelbar vorhergehenden Revision in Form von Differenzdaten (PDr, x) enthalten.

4. Verfahren nach einem der vorherigen Ansprüche, dadurch gekennzeichnet, dass Datenseiten (Pr) den vollständigen, optional in der jeweiligen Revision modifizierten Dateninhalt enthalten.

5. Verfahren nach einem der vorhergehenden Ansprüche, dadurch gekennzeichnet, dass die Hauptreferenzzeiger (RRr) oder Kopien davon auf einem weiteren Datenspeichermedium (LD_2) separat von den referenzierten Datenseiten (Pr) abgelegt sind.

6. Verfahren nach einem der Ansprüche 1 bis 4, dadurch gekennzeichnet, dass die Hauptreferenzzeiger (RRr) oder Kopien davon in Sektoren (S0,s) eines Datenspeichermediums (LD0) getrennt von den Sektoren (S0,s) desselben Datenspeichermediums (LD0) abgelegt sind, in denen die referenzierten Datensätze (Pr) gespeichert sind.

7. Verfahren nach einem der vorhergehenden Ansprüche, gekennzeichnet durch Absichern der Speicherung der Revisionen (Rr) von Datenfamilien durch Verschlüsselung, indem ein geheimer Masterschlüssel (MK) definiert und unabhängig von dem Masterschlüssel (MK) eine Zusatzkennung (SLT) generiert (Salting) wird und aus dem Masterschlüssel (MK) und der Zusatzerkennung (SLT) ein Schlüssel (K) abgeleitet wird (Stretching), sowie für jede Datenseite (PPr, x) ein jeweils einzigartiger Ad-Hoc-Schlüssel (N) und ein Zähler (CTR) aus Ergebnissen der Ver-/Entschlüsselungsalgorithmen übergeordneter Knoten zur Verschlüsselung, Entschlüsselung und Authentifizierung mindestens der zugeordneten Hauptreferenzzeiger, der Datenseiten und mindestens eines Kopfteils für die Gesamtheit der Revisionen (Rr) der gespeicherten Datenfamilien abgeleitet wird.

8. Verfahren nach einem der vorhergehenden Ansprüche, gekennzeichnet durch Auslesen von gespeicherten Revisionen (R_r) von Datenseiten (P_r),
gekennzeichnet durch
– Verifizieren der Übereinstimmung gespeicherter Kopien eines Kopfteils (H_h) für die Gesamtheit der Revision (R_r) gespeicherter Datenfamilien,
– Verifizieren der Übereinstimmung einer in einem separaten Datenspeichermedium (LD_2) oder auf einem separaten Sektor ($S_{0,s}$) eines Datenspeichermediums (LD_0) abgelegten Kopie eines Hauptreferenzzeigers (RR_r) für die aktuelle Revision ($R_{max(r)}$) mit einem Hauptreferenzzeiger (RR_r) der aktuellen Revision ($R_{max(r)}$), die auf einem Sektor ($S_{0,s}$) des Datenspeichermediums (LD_2) zusammen mit den zugehörigen Datenseiten (P_r) der Revision (R_r) abgelegt sind,
– Zugreifen auf Datenseiten ($PP_{r,x}$) gewünschter Revisionen (R_r) über im Hauptreferenzzeiger (RR_r) für die entsprechende Revision (R_r) und den Seitenreferenzzeigern ($PR_{r,p}$) enthaltenen Zeigern und
– Auswerten von Knotenänderungsinformationen für jede Revision (R_r) über Änderungen von eine Referenz auf logisch unmittelbar nachfolgend verknüpfte Kinder-Datenseiten ($PP_{r,x}$) definierende Knoten der Struktur baumartig verknüpfter Datenfamilienteile.

9. Verfahren nach Anspruch 8, gekennzeichnet durch Zugreifen auf gewünschte Revisionen (R_r) mittels eines binären Suchalgorithmus über die Liste der Hauptreferenzzeiger (RR_r), wobei eine Suche an Hand der Revisionsnummer und/oder eines im Hauptreferenzzeiger (RR_r) gespeicherten Zeitstempels erfolgt.

Es folgen 5 Blatt Zeichnungen

Anhängende Zeichnungen

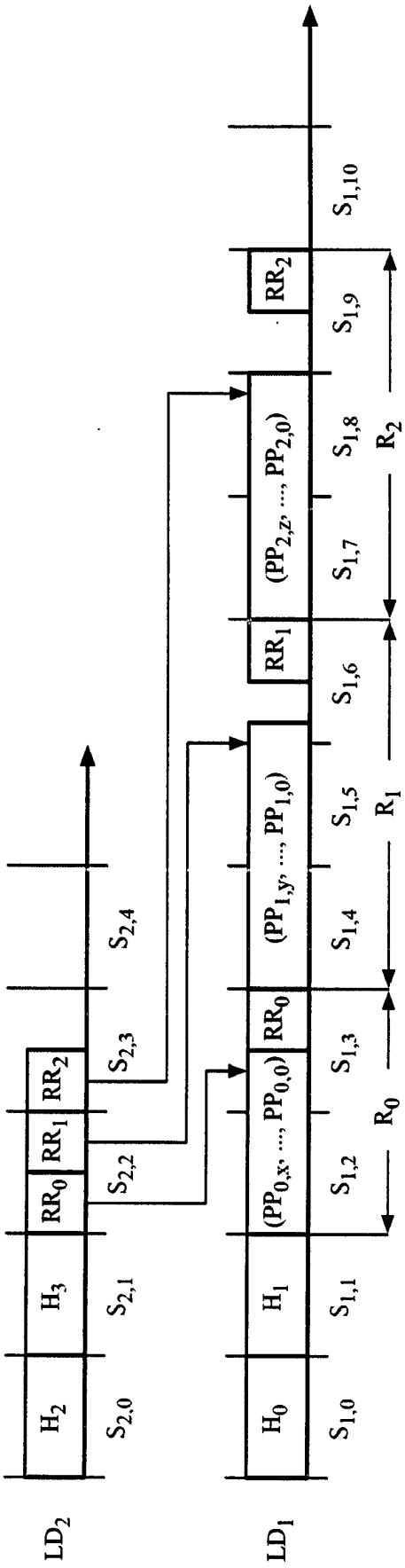


Fig. 1

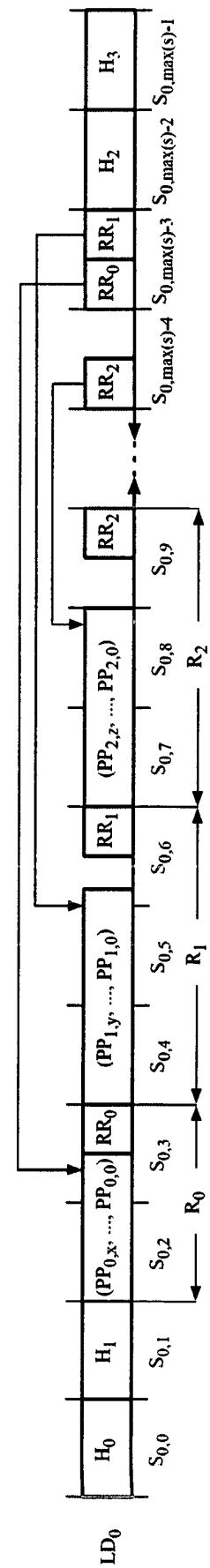


Fig. 2

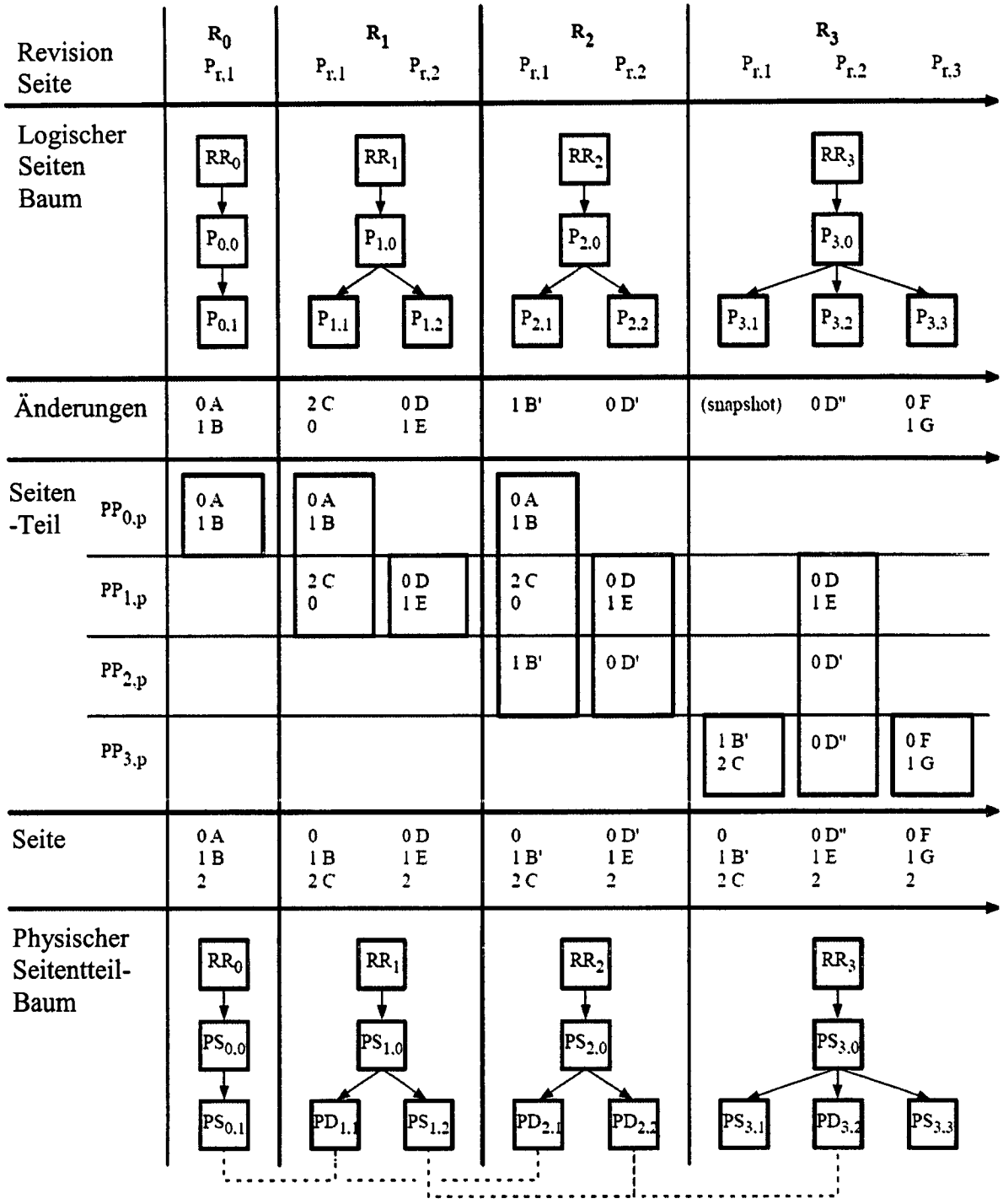

Revision
Seite
R0
Pr.1
R1
Pr.1
Pr.2
R2
Pr.1
Pr.2
R3
Pr.1
Pr.2
Pr.3
Logischer Seiten Baum
RR0
P0.0
P0.1
RR1
P1.0
P1.1
P1.2
RR2
P2.0
P2.1
P2.2
RR3
P3.0
P3.1
P3.2
P3.3
Änderungen
0 A
1 B
2 C
0
0 D
1 E
1 B'
0 D'
(snapshot)
0 D"
0 F
1 G
Seiten -Teil
PP0.p
PP1.p
PP2.p
PP3.p
Seite
0 A
1 B
2
0
1 B
2 C
0 D
1 E
2
0
1 B'
2 C
0 D'
1 E
2
0
1 B'
2 C
0 D"
1 E
2
0 F
1 G
2
Physischer Seitentteil-Baum
RR0
PS0.0
PS0.1
RR1
PS1.0
PD1.1
PS1.2
RR2
PS2.0
PD2.1
PD2.2
RR3
PS3.0
PS3.1
PD3.2
PS3.3

Fig. 3

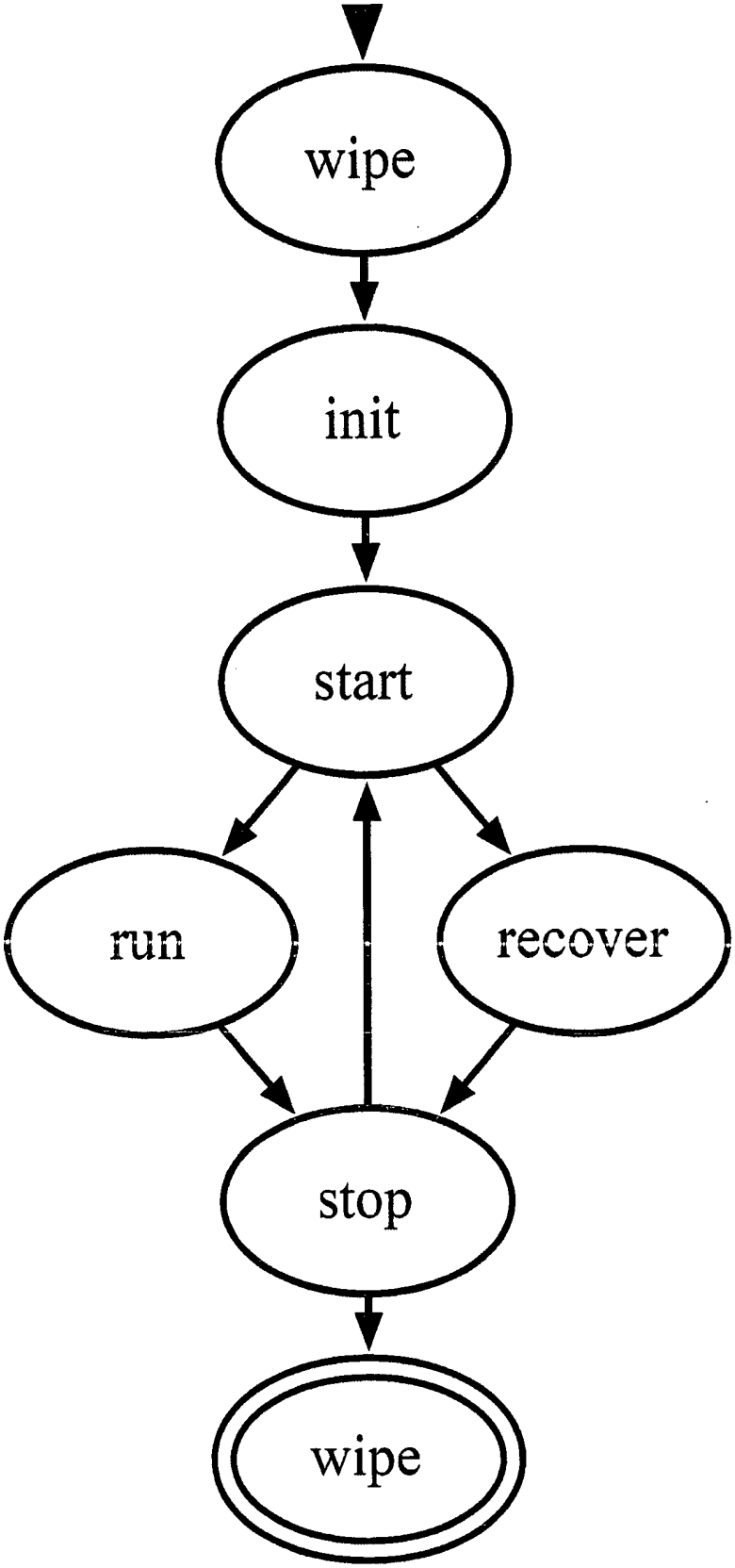


Fig. 4

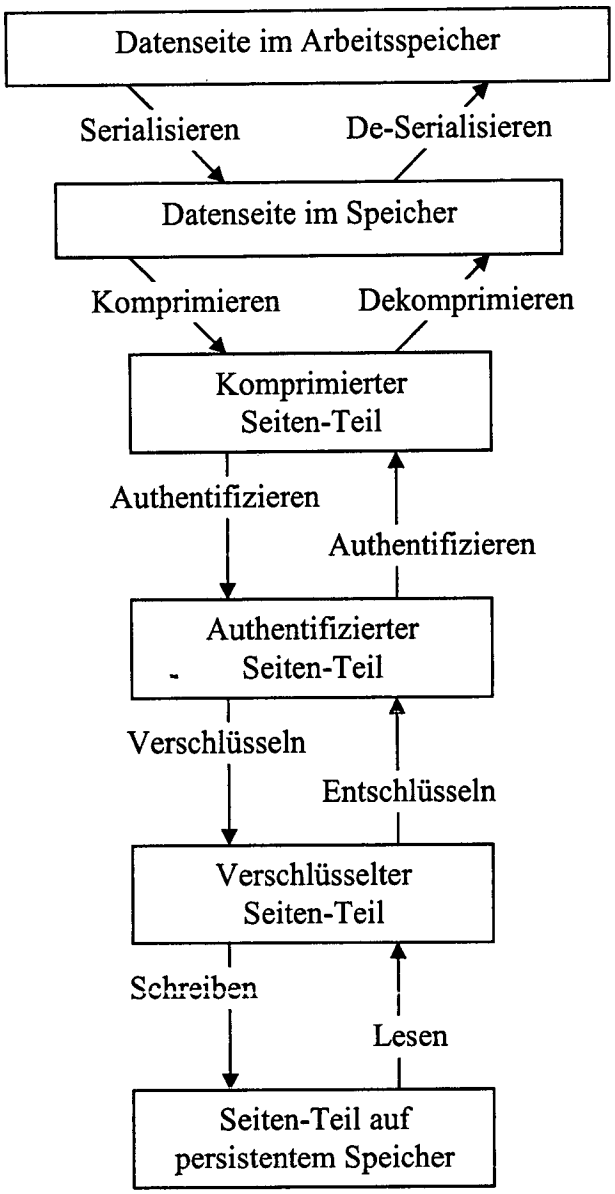

Datenseite im Arbeitsspeicher
Serialisieren
De-Serialisieren
Datenseite im Speicher
Komprimieren
Dekomprimieren
Komprimierter Seiten-Teil
Authentifizieren
Authentifizieren
Authentifizierter Seiten-Teil
Verschlüsseln
Entschlüsseln
Verschlüsselter Seiten-Teil
Schreiben
Lesen
Seiten-Teil auf persistentem Speicher

Fig. 5

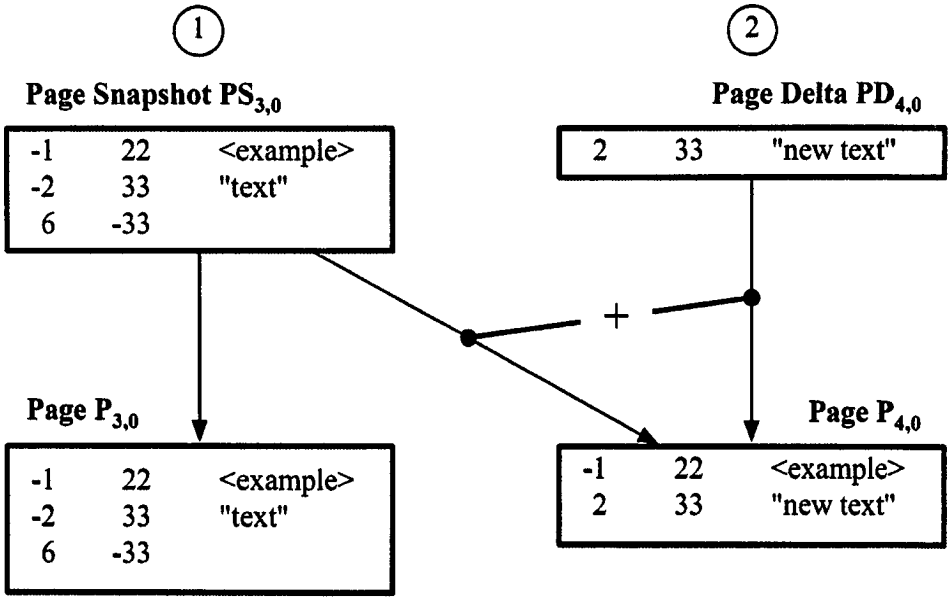

1
2
Page Snapshot PS3,0
Page Delta PD4,0
-1 22 <example>
-2 33 "text"
6 -33
2 33 "new text"
+
Page P3,0
Page P4,0
-1 22 <example>
-2 33 "text"
6 -33
-1 22 <example>
2 33 "new text"

Fig. 6