

Reiser4

Seminar: Dateisysteme (SS 2007)

René Kraneis

Institut für Informatik
Ruprecht-Karls-Universität Heidelberg

29. Mai 2007

1 / 35

Gliederung

1 Einleitung

- Reiser4
- Design-Prinzipien
- Umsetzung

2 Grundlage: Bäume

- Binärer Suchbaum
- Balancierter Baum
- B*-Baum
- „Dancing Tree“

3 Layout

- Reiser4-Baum
- tree. [ch]
- znode. [ch]
- jnode. [ch]
- Weitere Elemente
- Suche im Baum

4 Plugins

5 „Politik“

6 Zusammenfassung

2 / 35

Gliederung

1 Einleitung

- Reiser4
- Design-Prinzipien
- Umsetzung

2 Grundlage: Bäume

- Binärer Suchbaum
- Balancierter Baum
- B*-Baum
- „Dancing Tree“

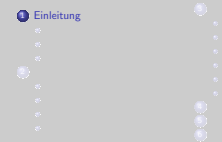
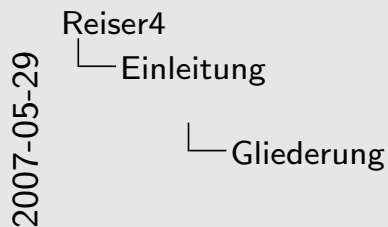
3 Layout

- Reiser4-Baum
- tree.[ch]
- znode.[ch]
- jnode.[ch]
- Weitere Elemente
- Suche im Baum

4 Plugins

5 „Politik“

6 Zusammenfassung

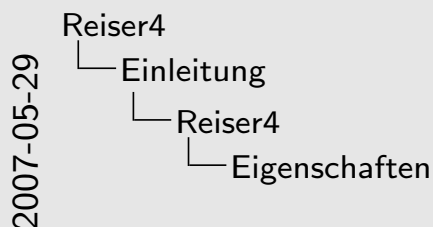


Eigenschaften

Reiser4 implementiert:

- B*-Bäume (ähnlich HFS, abgewandelt zu „Dancing Trees“)
- Journaling mit „Wandering Logs“
- „Allocate-on-flush“ (entspricht „delayed allocation“ bei XFS)
- Transaktionen
- Plugins

4 / 35



Eigenschaften

- Reiser4 implementiert:
- B*-Bäume (ähnlich HFS, abgewandelt zu „Dancing Trees“)
 - Journaling mit „Wandering Logs“
 - „Allocate-on-flush“ (entspricht „delayed allocation“ bei XFS)
 - Transaktionen
 - Plugins

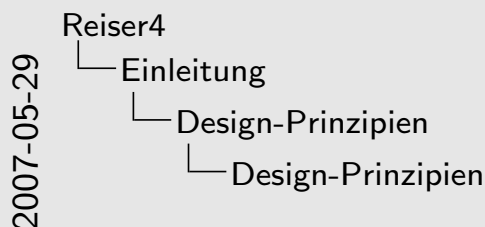
- Zu Bäumen und Plugins gleich mehr ...
- „Wandering Logs“ – Der Journaleintrag (es werden sowohl Daten als auch Metadaten geschützt) wird dort erzeugt, wo später auch die eigentliche Transaktion liegen soll; Änderungen werden von unten nach oben durch den Baum gereicht, so dass die letzte Änderung (schreiben des letzten Zeigers) atomar erfolgt.
- „Allocate-on-flush“ besagt, dass Änderungen der Datenstrukturen auf der Platte erst dann erfolgen sollen, wenn tatsächlich geflusht wird und nicht vorher. Eine Datei mit kurzer Lebensdauer wird so niemals auf der Platte gewesen sein.
- Transaktionen sind atomar ausgeführte Operationen. Entweder sie erfolgen vollständig oder nicht.

Design-Prinzipien

*Equal Source Code Access Is A Civil Right*¹

- Patente: Anreiz zum Teilen von Wissen geben, dafür zeitlich begrenztes Monopol auf dieses
- Pervertierung durch Softwarepatente
- „Software-Society“
- aber: „freier Code“ reicht nicht aus

¹http://www.namesys.com/v4/v4.html#civil_right



Design-Prinzipien

*Equal Source Code Access Is A Civil Right*¹

- Patente: Anreiz zum Teilen von Wissen geben, dafür zeitlich begrenztes Monopol auf dieses
- Pervertierung durch Softwarepatente
- „Software-Society“
- aber: „freier Code“ reicht nicht aus

¹http://www.namesys.com/v4/v4.html#civil_right

Statements, die vielleicht nicht notwendigerweise in einen Vortrag zu Dateisystemen gehören, mir persönlich aber wichtig sind.

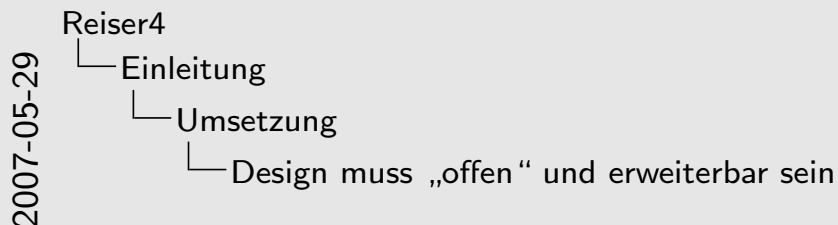
- Zeitlich begrenzte Patente sollen Investitionen unter der Voraussetzung schützen, dass nach Ablauf alle – also die Gesellschaft – Nutzen daraus ziehen können.
- Softwarepatente schützen nur die Investoren, schlimmer noch, schließen Konkurrenten vom Markt aus da zu oft „Ideen“ patentiert werden. (So können in den USA seit 1999 tatsächlich „Geschäftsideen“ patentiert werden – „State Street Bank“)
- Allerdings stellt „freier Code“ die Nutzbarkeit auch nicht sicher. Er muss nicht nur „technisch“, sondern auch „intellektuell“ zugänglich sein.

Design muss „offen“ und erweiterbar sein

*Software Libre Takes More Than A License — It Takes A Design*²

- Plugins – einfache Erweiterbarkeit
- „Orthogonalität von Features“
- Namespaces / Interfaces
- Datei- / Verzeichnis-Semantik, Probleme („Multics war besser“)

²http://www.namesys.com/v4/v4.html#software_libre



Design muss „offen“ und erweiterbar sein

*Software Libre Takes More Than A License — It Takes A Design*²

- Plugins – einfache Erweiterbarkeit
- „Orthogonalität von Features“
- Namespaces / Interfaces
- Datei- / Verzeichnis-Semantik, Probleme („Multics war besser“)

²http://www.namesys.com/v4/v4.html#software_libre

Plugins sollen es anderen Entwicklern erlauben, das Dateisystem auf einfache Weise um Fähigkeiten zu erweitern, ohne dass dazu Detailwissen über die Funktionsweise des Systems notwendig ist.

Die „Orthogonalität von Features“ soll sicherstellen, dass nicht Ähnliches ähnlich implementiert wird (im schlimmsten Fall durch Copy-Paste ;-)). Stattdessen sollen einfache Grundfunktionen abstrahiert und mehrere dieser Grundfunktionen zusammen verwendet werden (vgl. „Unix-Philosophie“: eine Aufgabe – ein Werkzeug).

Sauber definierte Schnittstellen stellen die Kommunikation von Objekten zwischen Namensräumen sicher. In Multics war eine Datei eine Sequenz von Elementen (z. B. Bytes, Verzeichniseinträge, ...) in Unix ist sie lediglich eine Sequenz von Bytes. Konsequenz ist, dass der „Standort“ in einem Unix-Verzeichnis nicht an einen Namen gebunden ist, sondern an einen Byte-Offset ...

Gliederung

1 Einleitung

- Reiser4
- Design-Prinzipien
- Umsetzung

2 Grundlage: Bäume

- Binärer Suchbaum
- Balancierter Baum
- B*-Baum
- „Dancing Tree“

3 Layout

- Reiser4-Baum
- tree.[ch]
- znode.[ch]
- jnode.[ch]
- Weitere Elemente
- Suche im Baum

4 Plugins

5 „Politik“

6 Zusammenfassung

2007-05-29

```
Reiser4
├── Grundlage: Bäume
│   └── Gliederung
```

Gliederung



```
Reiser4
├── Grundlage: Bäume
│   └── Gliederung
```

Überleitung:

- Bäume sind eine Möglichkeit hierarchisch strukturierte Daten darzustellen und erlauben ein einfaches Auffinden dieser Informationen.
- Da die Wahl der zugrundeliegenden Datenstruktur die Zugriffsgeschwindigkeit und -charakteristik (wie bei welchen Daten) nicht unwesentlich beeinflusst gehe ich hier auf die Grundlagen ein.

Bäume – Nomenklatur und Eigenschaften

Binärer Baum mit n Knoten

- Suchbaum: Knoten werden sortiert „eingehängt“
- Wurzel hat keinen, einen oder zwei Kindknoten
- innerer Knoten („Knoten“) hat einen oder zwei Kindknoten
- äußerer Knoten („Blatt“) hat keine Kindknoten
- Höhe $h \geq \log(n + 1)$, durchschnittlich $h = c \cdot \log n = \mathcal{O}(\log n)$
- Entartung – im schlimmsten Fall zur Liste (Höhe $h = n$)
- Komplexität von Operationen ist $\mathcal{O}(h)$

8 / 35

2007-05-29
Reiser4
├─ Grundlage: Bäume
│ └─ Binärer Suchbaum
│ └─ Bäume – Nomenklatur und Eigenschaften

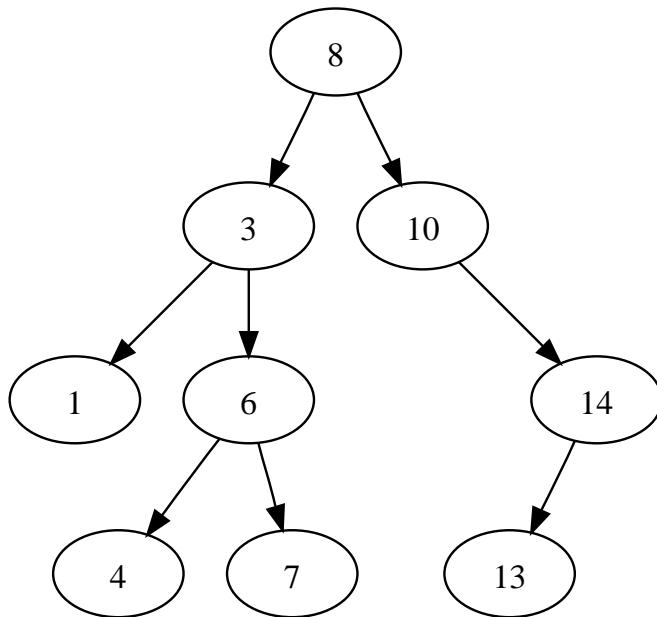
Bäume – Nomenklatur und Eigenschaften

- Binärer Baum mit n Knoten
- Suchbaum: Knoten werden sortiert „eingehängt“
 - Wurzel hat keinen, einen oder zwei Kindknoten
 - innerer Knoten („Knoten“) hat einen oder zwei Kindknoten
 - äußerer Knoten („Blatt“) hat keine Kindknoten
 - Höhe $h \geq \log(n + 1)$, durchschnittlich $h = c \cdot \log n = \mathcal{O}(\log n)$
 - Entartung – im schlimmsten Fall zur Liste (Höhe $h = n$)
 - Komplexität von Operationen ist $\mathcal{O}(h)$

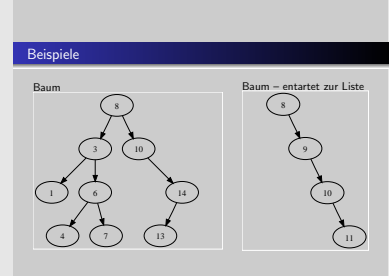
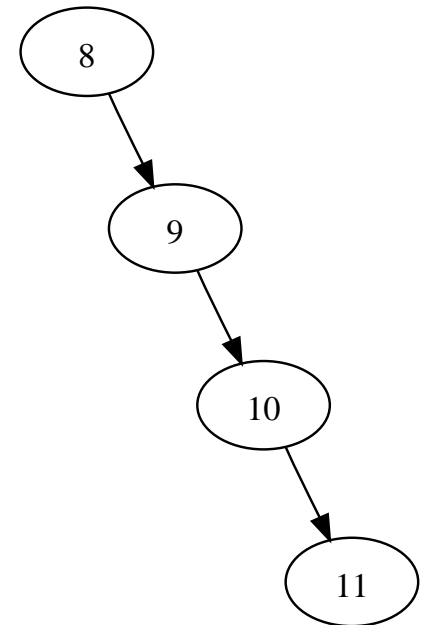
- Anmerkung: „Höhe“ ist hier die größte Entfernung zwischen Wurzel und Blatt plus eins (also die Anzahl der „Schichten“)
- Binärer Baum: Sortierung „von oben“ (wenn kleiner als aktueller Knoten links einsortieren, wenn größer rechts)
- Hinzufügen sortierter Daten führt zur Entartung
- „Operationen“ meint die bekannten Verfahren zum Suchen, Einfügen und Löschen von Elementen sowie die Rotation (diese hat allerdings eine konstante Laufzeit $\mathcal{O}(1)$)
- die Komplexität ist also mindestens logarithmisch und schlimmstenfalls linear

Beispiele

Baum



Baum – entartet zur Liste



- Binärer Suchbaum der Größe $n = 9$ (und Höhe $h = 4$) mit Wurzel 8, Knoten 3, 10, 6 und 14 und Blättern 1, 4, 7 und 13; beachte: $\log(n + 1) = \log(10) \approx 3,32$ („in etwa“ die tatsächliche Höhe)
- „sortierte Liste“ ($h = n = 4$, Wurzel 8, Knoten 9 und 10, Blatt 11)

Balance

Entartung ist also schlecht für die Performance

⇒ Lösung: balancierter Baum

- garantiert Höhe $h = \lceil \log(n + 1) \rceil$
- Verwendung der „Standardoperationen“ für binäre Bäume
- Re-Balancierung nach jeder Operation

10 / 35

2007-05-29
Reiser4
├─ Grundlage: Bäume
│ └─ Balancierter Baum
│ └─ Balance

Balance

Entartung ist also schlecht für die Performance
⇒ Lösung: balancierter Baum
• garantiert Höhe $h = \lceil \log(n + 1) \rceil$
• Verwendung der „Standardoperationen“ für binäre Bäume
• Re-Balancierung nach jeder Operation

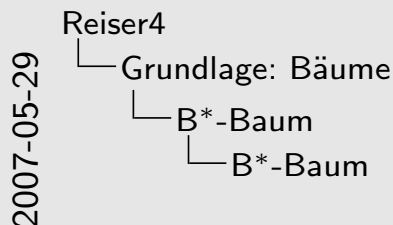
- Es gibt verschiedene Konzepte und Kriterien, einen Baum zu balancieren (hier: Höhe, aber auch „Gewicht“ der Teilbäume o. ä. möglich), zum Beispiel kann man zu „lange“ Teiläste rotieren. Hierauf gehe ich nicht weiter ein.
- Der Binärbaum hat in Schicht i maximal $n_i = 2^i$ Elemente und im Gesamten (bei h Schichten) maximal $n = \sum_{i=0}^h n_i = 2^h - 1$ Elemente. Durch Umformen zu $n + 1 = 2^h$ erhält man $\log(n + 1) = h$ für die Höhe des „gefüllten Baumes“ ⇒ Hinzufügen eines weiteres Element vergrößert auch die Höhe um eins.

B*-Baum

Berücksichtigung der Zugriffscharakteristik von Festplatten

- Nicht mehr „Binärer“ sondern „ k -ärer“ Baum
- „Daten“ liegen nur in den Blattknoten \rightarrow höhere Verzweigung
- jeder Knoten (\neq Wurzel) ist mindestens zu $\frac{2}{3}$ gefüllt
- Knoten: variable Anzahl s von Schlüssel $k_1 \dots k_s$
- innerer Knoten: zusätzlich $s + 1$ Verweise auf Kindknoten
- Schlüssel: sind aufsteigend sortiert und teilen die Schlüsselbereiche der Unterbäume in $s + 1$ Teilbereiche

11 / 35



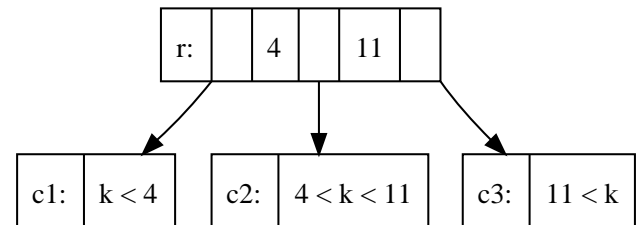
B*-Baum

- Berücksichtigung der Zugriffscharakteristik von Festplatten
- Nicht mehr „Binärer“ sondern „ k -ärer“ Baum
 - „Daten“ liegen nur in den Blattknoten \rightarrow höhere Verzweigung
 - jeder Knoten (\neq Wurzel) ist mindestens zu $\frac{2}{3}$ gefüllt
 - Knoten: variable Anzahl s von Schlüssel $k_1 \dots k_s$
 - innerer Knoten: zusätzlich $s + 1$ Verweise auf Kindknoten
 - Schlüssel: sind aufsteigend sortiert und teilen die Schlüsselbereiche der Unterbäume in $s + 1$ Teilbereiche

- Binärer Suchbaum ist für wahlfreien Zugriff (RAM) optimiert
- Hauptverzögerung: Anfahren einer Position auf der Festplatte
- Einmal positioniert, können sequenzielle Daten schnell gelesen werden
- Binäre Bäume liegen linearisiert auf Platte vor: viele wahlfreie Zugriffe notwendig
- B-Bäume auch linearisiert; aber: wegen k -facher Verzweigung viele „ähnliche“ oder korrelierte Elemente nacheinander
- Speicherbedarf wird v. a. durch die Nutzdaten dominiert, erhöhte Redundanz fällt nicht ins Gewicht
- Füllung mindestens $\frac{2}{3}$, um zu häufiges Balancieren zu vermeiden

B*-Baum – Beispiel

Die Teilbereiche können (wie hier) disjunkt sein oder ein gemeinsames Element haben (einfacher zu optimieren)



2007-05-29

- Reiser4
 - Grundlage: Bäume
 - B*-Baum
 - B*-Baum – Beispiel

B*-Baum – Beispiel

Die Teilbereiche können (wie hier) disjunkt sein oder ein gemeinsames Element haben (einfacher zu optimieren)

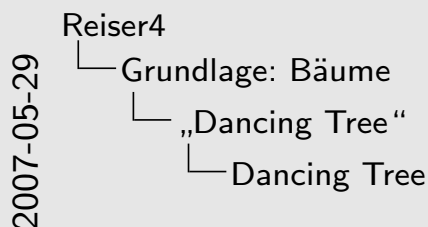


- Warum einfacher zu balancieren? Keine Motivation dazu gefunden.
- Die Grafik zeigt Wurzelknoten r mit Schlüsseln und Kindknoten c1, c2, c3 mit Bereichen, in denen mögliche Schlüssel liegen können.

Dancing Tree

Balancieren wird verzögert, bis auf Platte geschrieben wird
(Speicherbeschränkung oder Fertigstellung einer Transaktion)

- Schreiben dauert viel länger als Optimieren
- Es wird nicht so oft, dafür um so gründlicher optimiert



Dancing Tree

Balancieren wird verzögert, bis auf Platte geschrieben wird
(Speicherbeschränkung oder Fertigstellung einer Transaktion)

- Schreiben dauert viel länger als Optimieren
- Es wird nicht so oft, dafür um so gründlicher optimiert

- „Dancing Tree“ ist kein allgemeiner Begriff, sondern bezeichnet nur die Implementierung in Reiser4
- B*-Bäume fordern (wie Balancierte Bäume) nach jeder Operation eine Optimierung des Baumes
- Eine Transaktion bewirkt möglicherweise viele hundert Baum-Änderungen (Einfügen, Löschen);
nach jeder einzelnen zu optimieren ist sinnlos und kostet nur Zeit

Gliederung

1 Einleitung

- Reiser4
- Design-Prinzipien
- Umsetzung

2 Grundlage: Bäume

- Binärer Suchbaum
- Balancierter Baum
- B*-Baum
- „Dancing Tree“

3 Layout

- Reiser4-Baum
- tree. [ch]
- znode. [ch]
- jnode. [ch]
- Weitere Elemente
- Suche im Baum

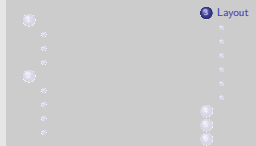
4 Plugins

5 „Politik“

6 Zusammenfassung

2007-05-29
Reiser4
└─ Layout
 └─ Gliederung

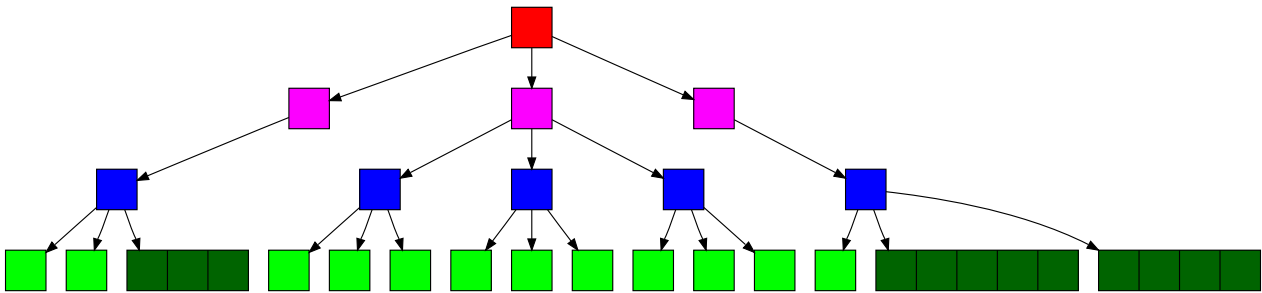
Gliederung



Nachdem wir nun so viel über Bäume erfahren haben und wissen, was es mit „Dancing Trees“ auf sich hat, wollen wir sehen, wie diese in Reiser4 verwendet werden.

Ich zeige hier zunächst eine graphische Darstellung zur Übersicht und dann Datenstrukturen, deren Entsprechung zu den eher theoretischen Konzepten des ersten Teils der Vorlesung ich erläutere.

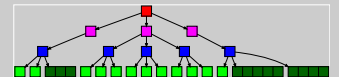
Reiser4-Baum



- Root-Knoten, Branch-Knoten, Twig-Twig, Leaf-Knoten
- Leaf, Extent
- Level nummeriert von 1 (Leaf) bis 4 (Root)
- „Fanout“ (Verzweigungsgrad) natürlich wesentlich höher

2007-05-29
Reiser4
├── Layout
│ └── Reiser4-Baum
│ └── Reiser4-Baum

Reiser4-Baum



- Root-Knoten, Branch-Knoten, Twig-Twig, Leaf-Knoten
- Leaf, Extent
- Level nummeriert von 1 (Leaf) bis 4 (Root)
- „Fanout“ (Verzweigungsgrad) natürlich wesentlich höher

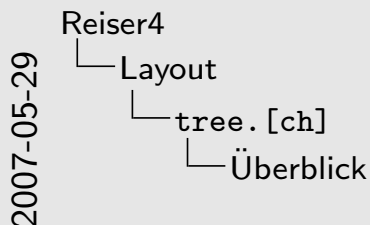
Bsp.: Fanout $k = 512$, Höhe $h = 4 \Rightarrow n = k^h - 1 = (2^9)^4 - 1 = 2^{36} - 1$
sind etwa 68 Mrd. Elemente. Den Fanout würde man so groß wählen,
dass ein Knoten nicht mehr als einen Block auf der Festplatte belegt.

Überblick

```

struct reiser4_tree {
    reiser4_block_nr    root_block;
    tree_level         height;
    __u64               estimate_one_insert;
    cbk_cache          cbk_cache;
    z_hash_table       zhash_table,    zfake_table;
    j_hash_table       jhash_table;
    rwlock_t           tree_lock,      dk_lock;
    spinlock_t         epoch_lock;
    __u64              znode_epoch;
    znode              *uber;          node_plugin    *nplug;
    struct super_block *super;
    struct { __u32 new_node_flags, new_extent_flags,
            paste_flags, insert_flags; } carry;
};
    
```

17 / 35



Überblick

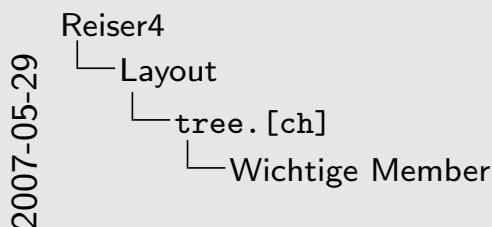
```

struct reiser4_tree {
    reiser4_block_nr    root_block;
    tree_level         height;
    __u64               estimate_one_insert;
    cbk_cache          cbk_cache;
    z_hash_table       zhash_table,    zfake_table;
    j_hash_table       jhash_table;
    rwlock_t           tree_lock,      dk_lock;
    spinlock_t         epoch_lock;
    __u64              znode_epoch;
    znode              *uber;          node_plugin    *nplug;
    struct super_block *super;
    struct { __u32 new_node_flags, new_extent_flags,
            paste_flags, insert_flags; } carry;
};
    
```

- repräsentiert den Reiser4-Baum, in dem alle Dateisystem- und Meta-Daten abgelegt werden
- ≠ Superblock, da es in diesem Dinge gibt, die nichts mit dem Baum zu tun haben (z. B. Mount-Optionen) und Dinge in einem Baum die nichts mit dem Superblock zu tun haben (z. B. der Baum der znodes)
- allerdings Teil des Superblocks
- wird an alle Baum-manipulierenden Methoden übergeben

Wichtige Member

- root_block: Datenblock
- height: Baumhöhe
- *uber: „Überknoten“
- *nplug: Knotenplugin
- *super: Superblock



Wichtige Member

- root_block: Datenblock
- height: Baumhöhe
- *uber: „Überknoten“
- *nplug: Knotenplugin
- *super: Superblock

Funktion der meisten gelisteten Member sollte offensichtlich sein:

- root_block verweist auf den Datenblock des Baumes,
- height ist die Höhe des Baumes,
- *uber ist ein Knoten, der über der eigentlichen Wurzel angeordnet ist um eine gültige Referenz auf diese zu haben auch wenn der Baum umstrukturiert wird,
- *nplug ist das mit diesem Knoten assoziierte Plugin und
- *super verweist auf den Superblock.

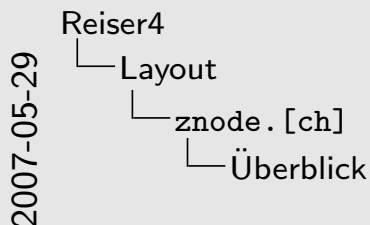
Weitere Members des Structs sind nötig für Caching-Strategien oder interne Nutzung (Locks u. ä.).

Überblick

```

struct znode {
    jnode          zjnode;
    parent_coord_t in_parent;
    znode          *left, *right;
    zlock          lock;
    int           c_count;
    node_plugin    *nplug;
    __u64          version;
    reiser4_key    ld_key, rd_key;
    __u16          level;
    __u16          nr_items;
} __attribute__((aligned(16)));
    
```

20 / 35



Überblick

```

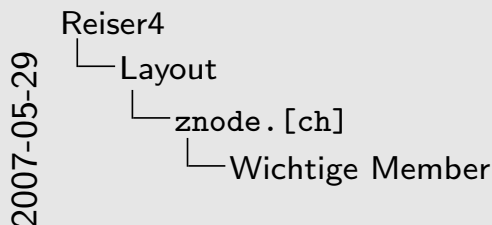
struct znode {
    jnode          zjnode;
    parent_coord_t in_parent;
    znode          *left, *right;
    zlock          lock;
    int           c_count;
    node_plugin    *nplug;
    __u64          version;
    reiser4_key    ld_key, rd_key;
    __u16          level;
    __u16          nr_items;
} __attribute__((aligned(16)));
    
```

- znode ist die Nur-Speicher-Repräsentation (oder der In-Speicher-Kopf) eines Baum-Knoten
- landet also nicht auf der Festplatte
- wird zudem zur Implementierung von Locking-Mechanismen und anderen erweiterten Funktionen benötigt (Hashes zur Leistungssteigerung u. ä.)

Wichtige Member

- zjnode eingebettete jnode
- in_parent->node Zeiger auf Elter
- *left, *right linker und rechter Nachbar
- c_count Anzahl der Kindknoten im Speicher
- *nplug mit diesem Knoten assoziiertes Plugin
- ld_key, rd_key linker und rechter begrenzender Schlüssel
- nr_items Anzahl der Elemente in dem Knoten

21 / 35



Wichtige Member

- zjnode eingebettete jnode
- in_parent->node Zeiger auf Elter
- *left, *right linker und rechter Nachbar
- c_count Anzahl der Kindknoten im Speicher
- *nplug mit diesem Knoten assoziiertes Plugin
- ld_key, rd_key linker und rechter begrenzender Schlüssel
- nr_items Anzahl der Elemente in dem Knoten

- Der Baum der znodes realisiert die „Dancing Tree“-Struktur in Reiser4
- zjnode repräsentiert die mit der znode verbundenen (Meta-)Daten
- in_parent->node, *left, *right stellen die Verzeigerung des Baumes sicher
- *nplug ist letztendlich für die Umsetzung der „Operationen“ auf den Knoten verantwortlich (delegiert diese z. B. an die in den Knoten enthaltenen Items)

Überblick

```
struct jnode {
    unsigned long    state;
    atomic_t         x_count, d_count;
    union { reiser4_block_nr z; jnode_key_t j; } key;
    union { z_hash_link z; j_hash_link j; } link;
    struct page      *pg;
    void             *data;
    reiser4_tree     *tree;
    struct list_head capture_link;
    struct rcu_head  rcu;
    reiser4_block_nr blocknr;
    reiser4_plugin_id parent_item_id;
} __attribute__((aligned(16)));
```

23 / 35

2007-05-29

- Reiser4
 - Layout
 - jnode.[ch]
 - Überblick

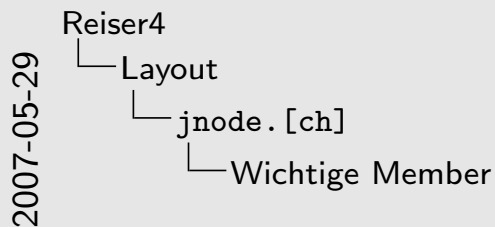
Überblick

```
struct jnode {
    unsigned long    state;
    atomic_t         x_count, d_count;
    union { reiser4_block_nr z; jnode_key_t j; } key;
    union { z_hash_link z; j_hash_link j; } link;
    struct page      *pg;
    void             *data;
    reiser4_tree     *tree;
    struct list_head capture_link;
    struct rcu_head  rcu;
    reiser4_block_nr blocknr;
    reiser4_plugin_id parent_item_id;
} __attribute__((aligned(16)));
```

- jnode „Journal Node“ – der eigentliche „Datencontainer“
- enthält insbesondere Transaktionsinformationen

Wichtige Member

- `x_count`, `d_count` Anzahl der Referenzen des `jnode` bzw. seiner Daten
- `*pg`, `*data` Zeiger auf die Seite des `jnode` bzw. diesen selbst



Solange noch Referenzen auf die Speicherseite gehalten werden, kann der `jnode` nicht aus dem Speicher entfernt werden.

Weitere Elemente

Weitere Spezifizierung durch Plugins realisiert

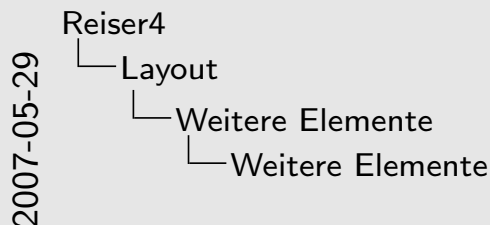
Beispiele für Items ...

- `static_stat_data`
- `cmpnd_dir_item`
- Extent- und Knotenzeiger
- Kleine Dateien

... und Units darin

- Dateimetadaten
- Verzeichniseintrag
- Extents
- Bytes

25 / 35



Weitere Elemente

Weitere Spezifizierung durch Plugins realisiert

Beispiele für Items ...

- `static_stat_data`
- `cmpnd_dir_item`
- Extent- und Knotenzeiger
- Kleine Dateien

... und Units darin

- Dateimetadaten
- Verzeichniseintrag
- Extents
- Bytes

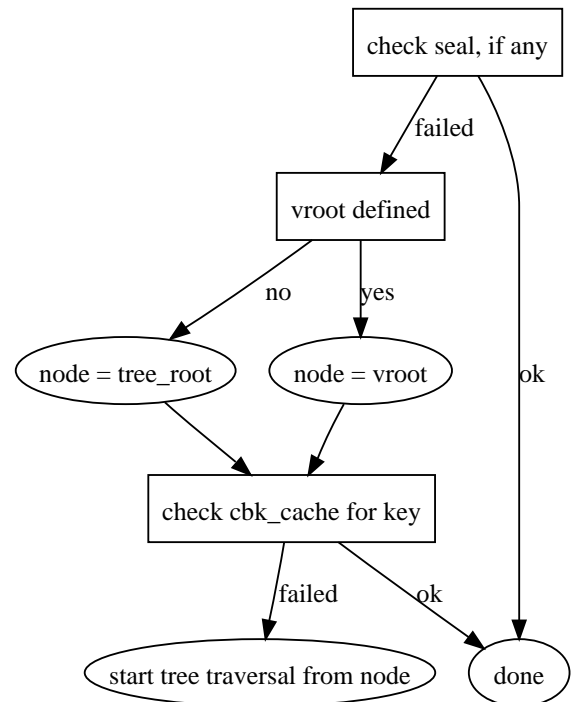
Einander gegenübergestellt sind Items und die darin enthaltenen Units. Eine Unit kann, wie der Name schon sagt, nicht über mehrere Items aufgeteilt werden.

- `static_stat_data` entspricht Dateimetadaten
- `cmpnd_dir_item` entspricht Verzeichniseinträgen
- Extentzeiger zeigen auf unformatierte Blätter (enthalten „Rohdaten“ eines Objektes)
- Knotenzeiger zeigen auf formatierte Knoten
- Extents entsprechen BLOBs, liegen allerdings auf der Ebene der Blätter und nicht darunter

Suche im Baum

Binäre Suche vermeiden, daher:

- Seals
- Virtual Roots
- Look-Aside-Cache



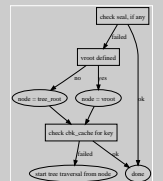
26 / 35

2007-05-29
Reiser4
├── Layout
│ └── Suche im Baum
│ └── Suche im Baum

Suche im Baum

Binäre Suche vermeiden, daher:

- Seals
- Virtual Roots
- Look-Aside-Cache



Binäre Suche ist stark CPU-lastig und zerstört Caches.

- seals: „weiche“ Baumzeiger. `znode.version` wird bei jeder Änderung erhöht. Diese wird in das `seal` kopiert. `znode ∈ Cache` und Versionsnummer OK \Rightarrow verwenden, sonst suchen.
- sowohl `tree.znode_epoch` als auch `znode.version` erhöhen, um falschen Positive zu vermeiden (nach dem Löschen eines Knotens könnte ein neuer zufällig dessen Versionsnummer erhalten)
- `vroot` eines Objekts (z. B. Datei) ist der von der Wurzel am weitesten entfernte Knoten, sodass noch alle Teile des Objektes in einem Baum mit dieser virtuellen Wurzel liegen.
- `cbk_cache` (Coord-by-Key) ist eine kleine Liste von Knoten, auf die kürzlich zugegriffen wurde (LRU-Strategie)

Suche in den Knoten selbst (nach Items/Units) \Rightarrow Plugins

Gliederung

1 Einleitung

- Reiser4
- Design-Prinzipien
- Umsetzung

2 Grundlage: Bäume

- Binärer Suchbaum
- Balancierter Baum
- B*-Baum
- „Dancing Tree“

3 Layout

- Reiser4-Baum
- tree.[ch]
- znode.[ch]
- jnode.[ch]
- Weitere Elemente
- Suche im Baum

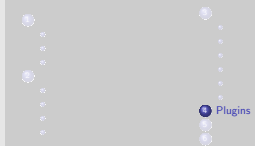
4 Plugins

5 „Politik“

6 Zusammenfassung

2007-05-29
Reiser4
├── Plugins
│
└── Gliederung

Gliederung



Nachdem nun die grundlegende Struktur von Reiser4 geklärt ist, folgt nun ein Blick auf die (wichtigsten) vorhandenen Plugins, wie zum Beispiel auch das Node-Plugin, das ja eben unter Anderem die Balancierung des Baumes umsetzt.

Überblick

```
$ mkfs.reiser4 --version 2>&1 | head -n 1
mkfs.reiser4 1.0.6
$
$ mkfs.reiser4 --print-plugins 2>/dev/null | \
> wc --lines
53
$
```

29 / 35

2007-05-29
Reiser4
└─ Plugins
 └─ Überblick

Überblick

```
$ mkfs.reiser4 --version 2>&1 | head -n 1
mkfs.reiser4 1.0.6
$
$ mkfs.reiser4 --print-plugins 2>/dev/null | \
> wc --lines
53
$
```

Soviel Zeit bleibt natürlich nicht mehr, deshalb beschränke ich mich auf einige grundlegende oder interessante Plugins

Beispiele

File-Plugins

- reg40 Regular file
- dir40 Directory
- sym40 Symlink
- sp140 Special file
- ccreg40
Cryptcompress

Item-Plugins

- stat40 StatData
- cde40 Compound directory Entry
- nodeptr40 Node pointer
- extent40 Extent file body
- plain40 Plain file body
- ctail40 Compressed file body

2007-05-29

Reiser4
└─ Plugins
 └─ Beispiele

Beispiele

- | File-Plugins | Item-Plugins |
|---|--|
| <ul style="list-style-type: none">• reg40 Regular file• dir40 Directory• sym40 Symlink• sp140 Special file• ccreg40
Cryptcompress | <ul style="list-style-type: none">• stat40 StatData• cde40 Compound directory Entry• nodeptr40 Node pointer• extent40 Extent file body• plain40 Plain file body• ctail40 Compressed file body |

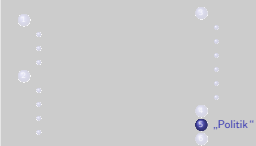
Beispiele für grundlegende Plugins. Die Funktionsweise dieser Plugins wird noch durch weitere modifiziert. Beispielsweise wird die Art der Kompression u. a. von den Plugins lzo1 und gzip1 gesteuert, oder die Schlüsselvergabe in einem Verzeichnis von verschiedenen Fibration-Plugins (lexic_fibre [lexikalische Sortierung], dot_o_fibre [*o zusammen], ext_1_fibre und ext_3_fibre [Sortierung nach ein- bzw. dreibuchstbiger Dateierweiterung]).

Gliederung

- 1 Einleitung
 - Reiser4
 - Design-Prinzipien
 - Umsetzung
- 2 Grundlage: Bäume
 - Binärer Suchbaum
 - Balancierter Baum
 - B*-Baum
 - „Dancing Tree“
- 3 Layout
 - Reiser4-Baum
 - tree.[ch]
 - znode.[ch]
 - jnode.[ch]
 - Weitere Elemente
 - Suche im Baum
- 4 Plugins
- 5 „Politik“
- 6 Zusammenfassung

2007-05-29
Reiser4
└─ „Politik“
 └─ Gliederung

Gliederung



- Plugins
- Documentation/CodingStyle
- „Mordsache Reiser“

2007-05-29
Reiser4
└─ „Politik“

- Plugins
- Documentation/CodingStyle
- „Mordsache Reiser“

- Unnötige Abstraktionen?
- hitzige, z. T. persönliche Diskussionen auf LKML zwischen Hans Reiser und Christoph Hellwig (Reviewer für FS-Code)
- Status unbekannt; Zukunft von Namesys

Gliederung

- 1 Einleitung
 - Reiser4
 - Design-Prinzipien
 - Umsetzung
- 2 Grundlage: Bäume
 - Binärer Suchbaum
 - Balancierter Baum
 - B*-Baum
 - „Dancing Tree“
- 3 Layout
 - Reiser4-Baum
 - tree.[ch]
 - znode.[ch]
 - jnode.[ch]
 - Weitere Elemente
 - Suche im Baum
- 4 Plugins
- 5 „Politik“
- 6 Zusammenfassung

2007-05-29
Reiser4
└─ Zusammenfassung
 └─ Gliederung

Gliederung



- spezifizierte Schnittstellen \Rightarrow gute Wartbarkeit
- Plugins \Rightarrow leicht erweiterbar
- B*-Bäume \Rightarrow gute Performance bei vielen kleinen Dateien
- Technische Innovation \Leftrightarrow „menschliche Komponente“



[Reiser4](#)

<http://namesys.com/v4/v4.html>



[Reiser4 Quelltext](#)

<ftp://ftp.namesys.com/pub/reiser4-for-2.6/2.6.21/reiser4-for-2.6.21.patch.gz>



[„Bäume in der Wikipedia“](#)

<http://en.wikipedia.org/wiki/B-tree>

http://en.wikipedia.org/wiki/B*-tree

http://en.wikipedia.org/wiki/Dancing_tree