den Followern vom Leader mitgeteilt – also ähnlich dem 2-Phasen-Commit-Protokoll. Man beachte, dass alle früheren Updates auch schon committ'et sind, da dieser Vorgang streng sequentiell abläuft. Nachdem ein Update committ'et ist, kann er auch auf die eigentliche Datenbank fortgeschrieben werden, wie in Abbildung 8.1 illustriert.

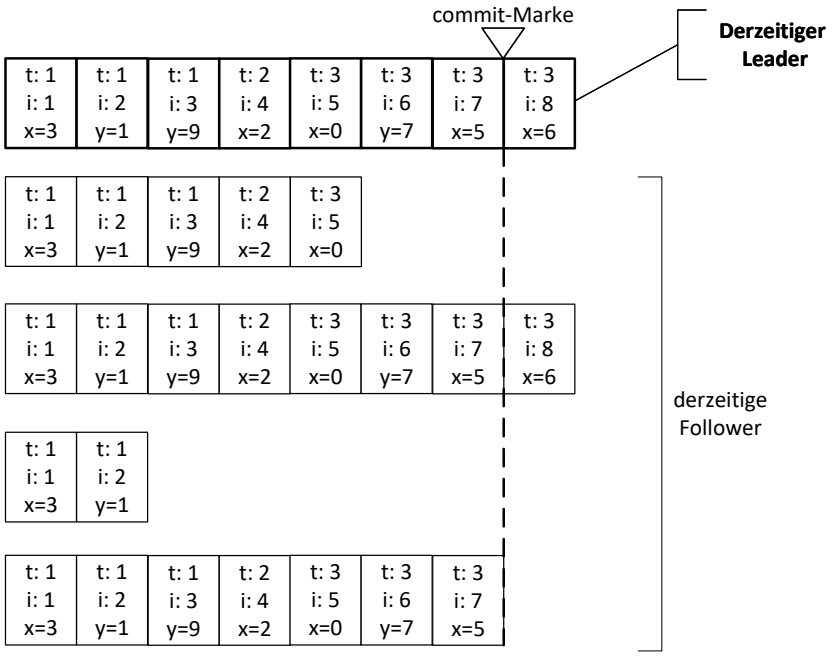

Abb. 8.4: Die verteilten Log-Dateien in Raft

8.2 JSON

XML gilt als sehr „verbos", so dass die eigentlichen Inhalte sich in relativ langen Dokumenten befinden. Deshalb hat sich in der letzten Zeit ein einfacheres Format im Datenbankbereich durchgesetzt: JSON. Das Akronym steht für JavaScript Object Notation, obwohl es sich um ein Programmiersprachen-unabhängiges Textformat handelt. Genau wie XML ist JSON ein hierarchisches Modell mit beliebig tief geschachtelten Unterobjekten. Aber anders als XML hat JSON nur noch Name/Wert- bzw. Key/Value-Paare als Informationseinheiten; die Unterscheidung in Attribute und Unterelemente entfällt. Als Strukturelemente gibt es in JSON also die *Objekte*, die mit geschweiften Klammern {…}

notiert werden und Arrays, die mit eckigen Klammern [...] gekennzeichnet sind. Arrays selbst können auch wieder Objekte enthalten. Anders als in XML werden Arrays ab 0 indexiert. Ein Objekt ist, anders als ein Array eine ungeordnete Liste von Key/Value-Paaren, wobei der Value durch einen Doppelpunkt vom Key abgetrennt ist. Als Key kommen nur Strings in Hochkommata infrage, wohingegen der Value-Teil „alles mögliche" sein kann, also eine Zahl, ein String, ein Boole'scher Wert, ein Objekt, oder ein Array.

Wir wollen die JSON-Datenmodellierung an einem Ausschnitt unseres vereinfachten Universitätsbeispiels der "Grösten Denker" demonstrieren:

```
{
    "Universität": {
      "Name": "Virtuelle Universität der Grösten Denker",
      "UniLeitung": {
        "Rektor": "Sokrates",
        "Kanzler": "Erhard" },
      "Fakultäten": [
         {"FakName": "Theologie",
          "Professoren": [ {
            "Name": "Augustinus",
            "Rang": "C3",
            "Raum": 309,
            "Vorlesungen": [
               {"VorlNr": "V5022",
                "Titel": "Glaube und Wissen",
                "SWS": 2 } ] } ] } ,
         {"FakName": "Physik",
          "Professoren": [
             {"Name": "Curie",
              "Rang": "C4",
              "Raum": 36 } ,
             {"Name": "Kopernikus",
              "Rang": "C3",
              "Raum": 310,
              "Vorlesungen": [
                 {"VorlNr": "V5023",
                  "Titel": "Alfonsinische Tafeln",
                  "SWS": 2 } ,
                 {"VorlNr": "V5024",
                  "Titel": "Astronomie",
                  "SWS": 2 } ] } ] } ] }
}
```

8.2.1 JSONpath

Mit JSONpath scheint sich eine standardisierte Anfragesprache für JSON-Dokumente im Datenbankbereich durchzusetzen. Leider weicht die Syntax doch deutlich von XPATH Pfadausdrücken ab; auch wenn es semantisch sehr ähnlich ist. So verwendet man in Pfadausdrücken in JASONPath für die Adressierung eines Kind-Objekts den Punkt statt eines Backslash. Beispielsweise wird der Rektor der Uni durch diesen JSONPath-Ausdruck ermittelt:

```
$.Universität.UniLeitung.Rektor
```

Hierbei steht das Dollarzeichen $ für das Wurzelobjekt. Alternativ zur Punkt/Dot-Notation kann man auch eckige Klammern verwerden, dann muss der Key aber in Hochkommata gesetzt werden, wie hier:

```
$['Universität']['UniLeitung']['Rektor']
```

Der rekursive Abstieg, also die *descendant-or-self*-Achse wurde in XPATH abgekürzt durch „//“. Aber in JSONpath verwendet man zwei Punkte, wie in diesem Programmbeispiel:

```
$..Professoren..Vorlesungen..Titel
```

Hier werden die Titel aller von Professoren gehaltenen Vorlesungen ausgegeben. Man beachter, dass dieser rekursive Abstieg also auch durch die zwischengelagerten Arrays iteriert – die Vorlesungen sind in unserem Beispieldokument den Professoren in einem (geschachtelten) Array zugeordnet ebenso wie die Fakultäten in einem Array der Universität zugeordnet sind.

Will man nur die Vorlesungen von Kopernikus ausgeben, so benötigt man ein Filterprädikat, das mit einem Fragezeichen eingeleitet wird. Das Prädikat steht dann in (…)-Klammern. In der Regel wird das Prädikat auf der Basis des schon erreichten Objekts formuliert, was mit dem „Klammeraffen“ notiert wird. Es entspricht in XPATH also dem **self** oder, abgekürzt, dem Punkt.

```
$..Professoren[?(@.Name=='Kopernikus')]..Vorlesungen..Titel
```

Das Ergebnis für unsere Beispiel-Datenbank ist dann das Array mit den beiden Vorlesungstiteln.

```
[
  "Alfonsinische Tafeln",
  "Astronomie"
]
```

Will man nur die wichtigste, also die erste Vorlesung von Kopernikus ausgeben, muss man die Indexposition 0 abfragen

```
$..Professoren[?(@.Name=='Kopernikus')]..Vorlesungen[0].Titel
```

und erhält ein Array mit nur einem Eintrag:

```
[
  "Alfonsinische Tafeln"
]
```

Man kann auch Array-Bereiche angeben. Wenn man beispielsweise die ersten beiden Vorlesungen von Kopernikus ausgeben will, geht das wie folgt:

```
$..Professoren[?(@.Name=='Kopernikus')]..Vorlesungen[0:2].Titel
```

Das vorhesehbare Ergebnis ist:

```
[
  "Alfonsinische Tafeln",
  "Astronomie"
]
```

Wenn man explizit durch alle Elemente eines Arrays iterieren will, kann man das mit [*] ausdrücken. Bei den Bereichsangaben der Array-Indizes kann man auch die untere bzw. auch die obere Schranke weglassen, wie beispielsweise [:1] wenn man nur das erste Element haben möchte:

```
$.Universität.Fakultäten[*].Professoren[?(@.Name=='Kopernikus')]..
                                                    Vorlesungen[:1].Titel
```

Wiederum haben wir ein vorhersehbares Ergebnis als Ausgabe:

```
[
  "Alfonsinische Tafeln"
]
```

Wie bereits erwähnt kann man statt der Dot-Notation auch die eckigen Klammern für die Traversierung zum Kind-Objekt verwenden, muss dann aber den Key in Hochkomma angeben:

```
$['Universität']['Fakultäten'][*]['Professoren']
```

```
[?(@['Name']=='Kopernikus')]..['Vorlesungen'][:1]['Titel']
```

Das Ergebnis ist natürlich wieder wie zuvor:

```
[
  "Alfonsinische Tafeln"
]
```

Wenn man die wichtigste (also jeweils erste) Vorlesung aller Professoren ermitteln geht das wie folgt:

```
$['Universität']['Fakultäten'][*]['Professoren']..
                                  ['Vorlesungen'][0]['Titel']
```

oder ohne rekursiven Abstieg könnte man es so formulieren:

```
$['Universität']['Fakultäten'][*]['Professoren'][*]
                                ['Vorlesungen'][0]['Titel']
```

Das Ergebnis für unsere Beispiel-Datenbank sieht dann so aus:

```
[
  "Glaube und Wissen",
  "Alfonsinische Tafeln"
]
```