Speichern von STM-Aufnahmen

Fachgebiete Physik, Informatik Jan Sebastian Götte, 12n (Abiturjahrgang 2011)
s@2π.eu
Landesschule Pforta

Betreuender Lehrer: Uwe Krägefski 24. Oktober 2010
Bearbeitungszeitraum: September 2009 bis Oktober 2010
Abgabetermin: 25. Oktober 2010

Inhaltsverzeichnis

1. Anforderungen

Mit einer Aufnahme eines Rastertunnelmikroskops ist außer den eigentlichen, sich teils über mehrere Ebenen erstreckenden Bilddaten ein Satz so genannter Metadaten verbunden. Diese Metadaten enthalten Informationen wie den aufgenommenen Bereich der Probe, eine Identifikation der Probe, den verwendeten Tunnelstrom, den Zeitpunkt der Aufnahme und weiteres.

Ziel meiner Arbeit war und ist es, diese Metadaten in einfach durchsuchbarem Format gemeinsam mit den Ebeneninformationen der Aufnahme sowie etwaigen Notizen des Aufnehmenden zu speichern. Weiterhin sollen die Daten möglichst über ein Netzwerk abrufbar sein sowie einfach auf andere Computer kopierbar sein. Motivation?

2. Paradigmen der Datenspeicherung

Daten können in verschiedenen Formen gespeichert werden. Ich möchte hier die drei für die gegebene Fragestellung vielversprechendsten Ansätze aufzeigen.

2.1. Dateisysteme

2.1.1. Eigenschaften

Seit Menschengedenken ist das Dateisystem der Ablageort der Wahl für Daten aller Art. Es unterstützt die Speicherung der Daten in so genannten *Dateien*, binären Datenblöcken, die sich an einer bestimmten Position des Dateisystems befinden und über diese sowie einen *Namen* identifizierbar sind. Neben dem Namen können bei den meisten entwickelten Dateisystemen (zu

denen FAT32 nicht gehört) weitere Metadaten gespeichert werden, standardmäßig sind dies das letzte Modifikationsdatum der Datei, mithilfe so genannter *Extended Attributes* ist jedoch auch die Ablage von Informationen wie Informationen über eine STM-Messung denkbar.

2.1.2. Nachteile

Nachteilig bei der Verwendung eines Dateisystems als Ablage für STM-Messergebnisse ist, dass ein Dateisystem nicht per se netzwerkfähig ist, d.h. es ist nicht ohne Zusatzsoftware möglich, von mehreren Computern mit ihm zu arbeiten. Weiterhin ist ein Dateisystem besonders bei der Ablage großer Datenmengen effizient, diese Effizienz nimmt mit steigender Dateianzahl und sinkender Dateigröße ab. Das Duplizieren der gespeicherten Informationen zwecks Backup oder in dem Fall, dass ein Forscher die Messdaten mit nach Hause oder auf Reisen nehmen möchte, geschähe mit einem speziellen Programm wie *rsync* [2].

Würde man STM-Messdaten in großer Menge bei einer großen Anzahl darauf zugreifender Benutzer in einem Dateisystem speichern, wäre ein Dateisystem jedoch ein recht ineffizientes und somit effektiv langsames Mittel der Datenspeicherung. Heutige Dateisysteme können auf gemäßigter Computerhardware bei großer Datenmenge einige Tausend Dateioperationen (Metadaten lesen, Öffnen etc.) pro Sekunde ausführen. Läuft nun außer gezielten Anfragen an einzelne Dateien noch ein Suchprogramm, dass sämtliche Dateien nach Schlüsselwörtern durchforstet, kann das bei einigen zehntausend Dateien und einigen Nutzern zu einem „zähen“ Verhalten des Systems führen.

2.2. Relationale Datenbankmanagementsysteme (RDBMS)

2.2.1. Eigenschaften

In relationalen Datenbanken bzw. in genauerem Jargon relationalen Datenbankmanagementsystemen werden Daten in Form von Datensätzen in Tabellen gespeichert, dabei enthalten alle Datensätze zu einer „Spalte“ der Tabelle, die einem Attribut entspricht, einen Wert für diese Spalte vordefinierten Datentyps.

2.2.2. Nachteile

RDBMS sind nicht für die Speicherung großer Datenblöcke wie den Ebeneninformationen der STM-Messungen ausgelegt. Obwohl theoretisch möglich, ist die Ablage solcher Datenmengen in RDBMS nicht elegant.

Ein RDBMS setzt eine klar definierte Datenstruktur voraus. Bei der Erstellung der Tabellen der Datenbank muss genau bekannt sein, welche Felder benötigt werden und welche Beziehungen zwischen diesen Feldern abgefragt werden. Bei STM-Messdaten, die wahrscheinlich von verschiedenen Geräten sowie verschiedenen Proben kommen, wäre das jedoch eine starke Einschränkung, da sich mehrere Mikroskope sowie mehrere Proben in den zu Verfügung stehenden Datenfeldern unterscheiden können. Möglichkeiten, die gewollte Flexibilität z.B. durch Ablegen von XML-formatierten Daten in einem Datenfeld sind nicht besonders performant sowie relativ unelegant.

Die Stärken eines RDBMS liegen in der extrem hohen Abfragegeschwindigkeit bei einfachen Tabellen, da das Datenformat einer STM-Messung jedoch wesentlich komplexer ist, dürfte eine Umsetzung desselben in einer relationalen Datenbank schwierig sein.

2.3. Dokumentenbasierte Datenbanken

2.3.1. Eigenschaften

Dokumentenbasierte Datenbanken (prominente Vertreter sind Lotus Notes, MongoDB und CouchDB) speichern Daten in Form von Informationseinheiten ohne feste Struktur, die dem Namen nach als Dokumente bezeichnet werden.

Die Dokumente werden zumeist in Suchbäumen organisiert, was einen schnellen Zugriff anhand vorher festgelegter Parameter ermöglicht. Zum schnellen Zugriff auf einzelne Attribute (Eigenschaften) der Dokumente verwenden dokumentenbasierte Datenbanken Indizes, meist ihrerseits Suchbäume, in denen Verweise auf die Dokumente oder die Dokumente selbst nach bestimmten Kriterien geordnet abgelegt werden.

2.3.2. Vorteile

Im Fall der Ablage von STM-Messergebnissen, die ein recht komplexes, verschachteltes Datenformat vorweisen, dass sich von Messung zu Messung in Details unterscheiden kann, sind solche Datenbanken sehr gut geeignet.

3. Die konkret verwendete Software

3.1. Überblick

Das verwendete System besteht aus CouchDB, einer dokumentenbasierten Datenbank, dem Apache Webserver sowie dem Browser auf dem Clientcomputer.

Der Apache-Webserver liefert die statischen HTML-Seiten sowie Stylesheets und Skripte für die Webanwendung an den Clienten aus und leitet Anfragen an die Datenbank weiter.

3.2. Warum CouchDB

CouchDB ist ein recht bekannter und auch schon weit gediehener Vertreter der dokumentenbasierten Datenbanken. CouchDB ist dadurch, dass es konzeptuell auf JavaScript und der JavaScriptObjectNotation (JSON) basiert, für die Erstellung von Webanwendungen prädestiniert, da es somit zwischen Browser und Datenbank keine „Sprachbarriere“ zu überwinden gibt (SQL–PHP–HTML).

3.3. Die Indexstrategie von CouchDB

In CouchDB hat jedes Dokument eine datenbankweit eindeutige ID. Diese ID ist in dem hier beschriebenen System eine UUID[1]. Die Dokumente in der Datenbank sind in einem B-Tree anhand ihrer IDs geordnet gespeichert, was anhand der IDs einen sehr schnellen Zugriff auf sie ermöglicht.

Um den Zugriff anhand anderer Attribute der Dokumente zu ermöglichen, kann man so genannte *Views* erstellen. Views werden in Designdokumenten, Dokumenten in der Datenbank, die eine spezielle ID besitzen, spezifiziert und ordnen die Dokumente nach dem MapReduce-Algorithmus ihrerseits nach gewählten Kategorien in B-Trees. Pro Designdokument existiert ein Suchbaum, den sich mehrere View teilen können.

[1] UUIDs sind aus Zufallszahlen generierte Zeichenfolgen. Das Prinzip dahinter ist, dass die verwendeten Zufallszahlen so absurd groß sind, dass auch bei exzessiver Verwendung von UUIDs die Chance, dass man eine bereits vorhandene UUID generiert, irrelevant klein ist. UUIDs taugen somit in verteilten Systemen, deren Komponenten über den Status der anderen Komponenten jeweils keine Informationen besitzen, als eindeutige IDs.

3.4. Die CouchDB-API

CouchDB ist eine NOSQL-Datenbank. Das bedeutet, dass CouchDB zur Abfrage der Daten kein SQL erfordert, dass im Fall komplex strukturierter Dokumente auch nicht gut geeignet ist. CouchDB verwendet als Abfragemethode ein HTTP-basiertes REST-API. REST (REpresentational State Transfer) ist ein Konzept, nach dem in diesem Fall die abzufragenden Informationen in der URL kodiert werden und so dem Dienst, in diesem Fall CouchDB, übermittelt werden. Die CouchDB-API ist nicht rein REST-basiert, teilweise werden Abfragedaten auch per HTTP-POST-Request im Body (dem Inhaltsteil eines HTTP-Daten-„Paketes“) übertragen.

Ein API-Aufruf läuft in CouchDB prinzipiell folgendermaßen ab:

1. Der Client sendet eine HTTP-Anfrage an CouchDB, die notwendigen Parameter werden in der URL sowie unter Umständen im Request-Body übergeben.

2. CouchDB bearbeitet die Anfrage und sendet das Resultat in JSON gekapselt zurück an den Client.

3.5. JSON

JSON, die JavaScript Object Notation, ist ein einfaches Format zur Repräsentation baumartig strukturierter Daten. JSON ist am ehesten mit XML zu vergleichen. Es ist diesem prinzipiell dahingehend ähnlich, dass die ausgedrückten Beziehungen ähnlich sind, es ist jedoch sehr viel einfacher. Die deutschsprachige Wikipedia äußert sich über JSON folgendermaßen:

> JSON kennt Objekte, Arrays, Zeichenketten, Zahlen, boolesche Werte (true, false) und null. Daten können beliebig verschachtelt werden, beispielsweise ist ein Array von Objekten möglich. [4]

JSON ist zum Datenaustausch zwischen clientseitigem JavaScript und dem Server sehr beliebt, da es valides JavaScript ist und so zum Parsing in JavaScript kein zusätzlicher Code notwendig ist, wodurch es bezüglich der CPU-Ressourcen sparsam ist.

CouchDB verwendet JSON sowohl intern zum Speichern der Daten als auch zur Kommunikation mit dem Client. Die Dokumente liegen im JSON-Format vor und sämtliche Ausgaben erfolgen in JSON.

4. Die Programmarchitektur

Das eigentliche Programm lebt innerhalb eines Scriptes in einer HTML-Datei. Dieses „Haupt-Script“ lädt nach Anfrage des Benutzers einzelne Knoten (die durch das Programm verwalteten Informationseinheiten) aus der Datenbank (in der diese durch Dokumente dargestellt werden) und rendert diese. Jeder Knoten enthält ein Attribut names `node_type`, das angibt, welchen Typs der Knoten ist.

Auf dem Server sind in einem Verzeichnis so genannte Renderer abgelegt, kleine JavaScript-Scripts, die jedes aus dem JavaScript-Objekt eines oder mehrerer Knotentypen HTML generieren, dass dem Benutzer angezeigt werden kann. Beim Start der Anwendung (Web-Application) lädt das Haupt-Script alle auf dem Server in einer Indexdatei eingetragenen Renderer, die sich daraufhin registrieren. Sobald das Haupt-Script nun beim Laden eines Knotens auf einen bis dato unbekannten Typ stößt, fragt es bei jedem geladenen Renderer ab, ob dieser den Knotentyp rendern kann. Sobald es dabei auf einen Renderer stößt, der sich den entsprechenden Knotentyp zu rendern fähig zeigt, wird dieser für diesen Knotentyp als Standardrenderer eingetragen.

Der Benutzer kann einen anderen Renderer auswählen, der dann als neuer Standardrenderer für den aktuellen Knotentyp verwendet wird. Jeder Renderer bekommt als Argument zum Knoten die gewünschte Rolle, die er rendern

soll. Momentan wird hier zwischen `show` zum normalen Anzeigen des Knotens und `summary` zum Rendern einer Kurzfassung (Titel, wichtigste Attribute) unterschieden. Jeder Renderer kann darüber hinaus Code enthalten, um einen leeren Knoten des Typs, den er rendert zu erstellen. Jeder Renderer muss beim Rendern der `show`-Rolle auch eine Bearbeitungsmöglichkeit für die Daten rendern.

4.1. Das Datenformat

Allen Knotentypen sind einige Angaben gemein. Zu diesen Angaben gehören der „Besitzer“ des Knotens, das letzte Veränderungsdatum sowie ein Datenfeld, das in einen Teil für die „Summary“ und einen für den „Rest“ gegliedert ist. Hier werden die eigentlichen Daten gespeichert, in Summary lediglich der zum Rendern einer Kurzfassung notwendige Teil, in Rest das vollständige Dokument. Das hierbei verwendete Datenformat ist vollständig dem Renderer überlassen.

4.2. Tagging

Zur Organisation der Knoten besteht die Möglichkeit, einen Knoten mit einem anderen zu „taggen“. Tagging ist eine „Many-to-Many“-Beziehung, mehrere Knoten können dasselbe Tag besitzen und ein Knoten kann mehrere Tags besitzen. Mittels Tagging können verschiedene Strukturen entstehen:

1. Es *können* wie in einem Dateisystem Baumstrukturen (Hierarchien) entstehen, bei denen ein Knoten genau ein Tag hat, ein Knoten jedoch von mehreren Knoten als Tag verwendet werden kann.

2. Eine weitere Möglichkeit ist, dass ein Knoten auch mehrere Tags haben kann, jedoch in keiner Beziehung sein eigenes Tag ist (d.h. dass auch kein Knoten, mit dem er getaggt ist, ihn als Tag besitzt etc.). Es liegt

eine Polyhierarchie vor, mathematisch ist das ein gerichteter azyklischer Graph.

3. Die letzte, freiste und komplizierteste Möglichkeit ist, dass jeder Knoten beliebige Beziehungen zu anderen Knoten hat. Der hierbei entstehende gerichtete Graph kann zyklisch sein.

Um durch diese Graphen zu navigieren, kann der Benutzer ein Tag auswählen. Ihm werden nun alle Knoten angezeigt, die dieses Tag besitzen. Die konkrete Umsetzung sieht so aus, dass ein CouchDB-View definiert ist, in dem sämtliche Knoten in ihrer Kurzfassung (als JSON-Objekte ohne das Attribut mit den vollständigen Daten) nach ihren Tags geordnet sind. Möchte sich der Benutzer nun alle Knoten ansehen, die mit einem bestimmten Knoten getaggt sind, wählt er diesen Knoten aus, worauf eine Anfrage an CouchDB gestartet wird, deren Ergebnis in einem temporären Knoten gespeichert wird. Dieser temporäre Knoten hat den Knotentyp `node_list`. Er existiert nicht in der Datenbank sondern nur lokal und wird nun mit einem Renderer zu einer Graphischen Repräsentation der Knoten umgewandelt. In einer einfachen Form sieht das so aus, dass die vom jeweiligen Knotenrenderer gerenderten Kurzfassungen der Knoten wie bei einem Dateimanager in einer Liste untereinander angezeigt werden.

4.3. Der Listenrenderer

Der Listenrenderer ist ein sehr einfaches Stück Code. Er überführt die Liste der Knoten in einer for-Schleife in einen HTML-Baum, dessen Blätter die in `<div>`-Elementen eingeschlossenen vom jeweiligen Knotenrenderer gerenderten Kurzfassungen der Knoten sind.

4.4. Der Notizrenderer

Als einen Beispielrenderer habe ich den Notizrenderer programmiert. Meine ursprüngliche Intention beim Schreiben dieses Programms war, es als verbesserte digitale Notizverwaltung zu verwenden. Der Notizrenderer definitert den Knotentyp `note`. Er enthält auch ein Callback, das vom Haupt-Script verwendet werden kann, um einen neuen, leeren Knoten dieses Typs zu generieren.

Im `summary`-Datenteil des Knotens werden bei Notizen die Länge in Wörtern sowie die ersten 10 Wörter der Notiz gespeichert. Die Zusammenfassung wird gerendert, indem die Wortanzahl und die im `summary`-Teil des Knotens gespeicherten ersten 10 Wörter in `<div>`-Blöcke gesetze werden. Für die Anzeige des gesamten Knotens wird der gesamte Text in einem `<div>`-Block angezeigt. Bei einem Klick darauf wird dieser Block mit etwas JavaScript in eine `<textarea>` umgewandelt. Nun kann der Benutzer die Notiz bearbeiten. Klickt er zum Übernehmen der Änderungen auf den ebenfalls gerenderten `Speichern`-Button, wechselt die Anzeige wieder zum herkömmlichen `<div>`-Element und die Änderungen werden in der JSON-Repräsentation des aktuellen Knotens gespeichert. Zuletzt wird noch eine Funktion aufgerufen, die dem Haupt-Script mitteilt, dass eine Bearbeitungsaktion abgeschlossen wurde und der Knoten somit in der Datenbank gespeichert werden kann.

4.5. Der Renderer für strukturierte Daten

Eine Verallgemeinerung dieses Prinzips ist ein (noch nicht programmierter) Renderer, der die JSON eines Knotens nimmt und sämtliche angegebenen Felder in ihrer Hierarchie anzeigt. Solche Darstellungen werden beispielsweise in der Firefox-Erweiterung *Firebug* [1] sowie in *Futon* [3], dem integrierten graphischen Datenbankmanager von CouchDB verwendet.

In der so erstellten Übersicht aller Attribute eines Knotens können diese bearbeitet sowie neue hinzugefügt werden. Die Metadaten der STM-Messung finden hier ihren Platz. Die Ebeneninformationen werden in einem der üblichen Dateiformate abgelegt und als Anhang (*Attachment*) des den Knoten repräsentierenden Dokumentes in CouchDB hochgeladen. Hierbei müsste die zum Exportieren der Daten genutze Anwendung die Aufgabe übernehmen, Vorschaubilder für die Ebenen in einem webüblichen Grafikformat (`jpg`, `png` etc.) zu generieren, sodass der Benutzer schon in der Übersicht einen Blick auf die eigentliche Messung werfen kann.

5. Zusammenfassung und Fazit

Der bisherige Stand der Webanwendung (die im übrigen mangels besserer Vorschläge den Namen *Taskforce* trägt) ist, dass diese wiewohl noch nicht für den Alltagsgebrauch geeignet doch schon die Hauptfunktionalität besitzt. Der aktuellste Quelltext kann vom geneigten

Anhang

A. Literaturverzeichnis

[1] Firebug, 06. Mai 2010. `https://addons.mozilla.org/de/firefox/addon/1843/`.

[2] Linux Man Page. rsync(1), 31. Dez. 2009. `man rsync`.

[3] CouchDB Wiki. Getting started with Futon, 13. Aug. 2010. `http://wiki.apache.org/couchdb/Getting_started_with_Futon`.

[4] Deutschsprachige Wikipedia. JavaScript Object Notation, 24. Okt. 2010. `http://de.wikipedia.org/wiki/JSON`.

B. Glossar

NOSQL – `No(t Only) SQL` – Datenbank, die neben oder statt SQL andere Abfragesprachen zur Verfügung stellt