Krake - ein MapReduce basiertes Crawlerframework

Thomas Nitschke

24. Mai 2012

Inhaltsverzeichnis

0.1 Einführung 2
0.2 Funktionsweise 2
    0.2.1 Crawlphase 2
    0.2.2 Verarbeitungsphase 2
0.3 Aufbau 3
    0.3.1 Queue 3
    0.3.2 Schlüsselverwaltung 3
    0.3.3 Crawlverzeichnis 4
0.4 Installation 4
    0.4.1 Konfiguration 5
    0.4.2 Schlüsselverwaltung 5
    0.4.3 Queue 6
0.5 Verwendung 6
    0.5.1 Datenexport und Verarbeitung 6
0.6 Dateiformate 7
    0.6.1 Contentdatei 7
    0.6.2 Webgraph 8
    0.6.3 URLMapping 8
0.7 Evaluierung 8

0.1 Einführung

Das Krakeframework ist ein verteilter Webcrawler der durch Anwendung des, durch das Apache Hadoop System (http://hadoop.apache.org) implementierte, MapReduce Paradigma während des Crawlvorgangs und der anschließenden Datenverarbeitung ein hohes Maß an Parallelität erreicht. Der Crawler bietet zusätzlich Funktionen zur Weiterverarbeitung und Aggregation der Daten zu einem Webgraphen und Inhaltsdatenbank. Das Krakeframework sollte nicht als fertiges Produkt gesehen werden, obwohl es als solches einsetzbar ist. Es ist viel mehr eine Basis für weitergehende Arbeiten, die Zugriff auf große, aus dem Web bezogene, Informationsmengen benötigen.

0.2 Funktionsweise

Dieses Kapitel bietet einen methodischen Überblick über die Arbeitsabläufe des Krakecrawlers und dient als Basis zum detaillierten Verständnis der einzelnen Komponenten. Der Arbeitsablauf des Frameworks ist zweiphasig. In der ersten, der Crawlphase, werden mehrere aufeinander folgende Crawlvorgänge durchgeführt, bis die gewünschte Datenmenge gesammelt wurde. In der folgenden Export- und Verarbeitungsphase werden die gesammelten Daten zunächst in Formate überführt, die eine effiziente Weiterverarbeitung ermöglichen, um dann zu einem Webgraphen und Inhaltsdatenbank aggregiert zu werden. Wie die einzelnen Prozesse der zwei Phasen angesprochen werden können und ineinander greifen, wird im Kapitel 0.5 dargelegt.

0.2.1 Crawlphase

Krake arbeitet während der Crawlphase in Iterationen, wobei jede Iteration einer Ausführung des "crawl"-Hadoop Jobs entspricht. In jeder Iteration wird eine bestimmte Anzahl URLs aus der Queue entnommen und verarbeitet. Die entstandenen Daten werden im Hadoop Distributed Filesystem (kurz: DFS) gespeichert und die neu entdeckten URLs der Queue hinzugefügt. Während des Crawlings wird über jede gesehene URL mit dem SHA1-Algorithmus eine Prüfsumme berechnet, die als Schlüssel für die unter dieser URL hinterlegten Daten fungiert. Der berechnete Schlüssel wird mit der verteilten Schlüsseldatenbank (siehe 0.3.2) abgeglichen und die URL, sofern der Schlüssel noch unbekannt ist, in die Queue (siehe 0.3.1) aufgenommen. Der Crawler verhindert damit weitestgehend das Crawlen von Inhalten, die bereits gesehen wurden.

0.2.2 Verarbeitungsphase

Die Verarbeitungsphase folgt auf die Crawlphase und schließt diese ab. Diese Phase erzeugt die sog. Contentdatei (siehe 0.6.1) und den fertigen Webgraphen (siehe 0.6.2). Da in dieser Phase nicht mehr alle Operationen parallel ausgeführt werden können und besonders bei der Zusammenführung von Daten eine hohe Last auf nur einem Rechner entstehen kann, ist die Verarbeitungsphase eine sehr arbeitsintensive Phase, für die evtl. weitere Ressourcen herangezogen werden müssen.

0.3 Aufbau

Das Krake-Framework basiert auf dem Hadoop-MapReduce System und mySQL Server. Mit Hilfe dieser Basissysteme werden die drei Hauptkomponenten des Krakecrawlers gebildet:

- Queue
- Schüsselverwaltung
- Crawlverzeichnis

Im folgenden werden alle wichtigen Komponenten des Frameworks erklärt.

0.3.1 Queue

Die Queue arbeitet nach dem FIFO-Prinzip und speichert neu entdeckte URLs lokal auf dem jeweiligen Knoten, der sie während der Crawlphase entdeckt hat. Eine Verteilung der Queueinhalte unter den Knoten findet nicht statt. Dieser Ansatz gewährleistet eine schnelle Lese- und Schreibgeschwindigkeit, kann jedoch dazu führen, dass die Queues auf den Knoten unterschiedlich schnell wachsen. Mit Hinblick auf diese Einschränkung sollte besonders darauf geachtet werden, dass die erste URL-Menge, die beim ersten Crawl verwendet wird, der sog. "Seed", möglichst gleichmäßig auf alle Knoten verteilt wird.

Alle zur Queue gehörigen Dateien werden im, in der Konfigurationsdatei (siehe 0.4.1) festgelegten, Queueordner abgelegt. Die Queue besteht immer aus einer globalen Statedatei und mindestens einer Daten-Datei. Die Daten-Dateien speichern pro Zeile eine URL und werden aufsteigend benannt. D.h., die erste Daten-Datei, aus der als erstes während des Crawlens URLs entnommen werden, trägt immer den Namen "0". Sobald alle URLs aus dieser Datei abgearbeitet wurden, wird diese gelöscht und alle verbleibenden Daten-Dateien rücken um eine Stelle auf (Datei "1" wird zur neuen Datei "0").

Die Statedatei der Queue speichert in ihrer ersten Zeile die bereits bearbeiteten URLs aus der aktuellen Daten-Datei "0" und in den folgenden Zeilen die totale Anzahl an URLs in den Daten-Dateien. Eine Statedatei für 3 Daten-Dateien mit jeweils 1400, 3000, 50 URLs mit bereits 230 bearbeiteten URLs hätte folgenden Inhalt:

```
230
1400
3000
50
```

Da die Statedatei nur vor und nach jedem Crawlvorgang aktualisiert wird, sind die in ihr gespeicherten Daten auch nur zwischen Crawlvorgängen korrekt. Während eines Crawlvorgangs werden alle neuen URLs in der "new"-Datei gespeichert. Die "new"-Datei wird mit Beendigung des Crawls zu einer normalen Daten-Datei der Queue konvertiert.

0.3.2 Schlüsselverwaltung

Die Schlüsselverwaltung überprüft, ob eine URL bereits gecrawlt, in der Queue gespeichert oder noch unbekannt ist. Die Schlüsselverwaltung stellt damit eine zentrale Komponente des Crawlers dar und hat starken Einfluss auf die Verarbeitungsgeschwindigkeit des gesamten Systems.

Die Verwaltung besteht aus einer Reihe von mySQL-Servern, wobei jeder Server nur einen bestimmten Intervall von Schlüsselwerten verwaltet. Eine Lese- oder Schreiboperation für einen bestimmten Schlüssel wird daher immer nur an einen Server gerichtet. Die Anzahl der Schlüsselserver kann nicht beliebig gewählt werden, das Krakeframework geht davon aus, dass auf jedem Hadoop Verarbeitungsknoten auch ein mySQL-Server für die Schlüsselverwaltung installiert ist.

Als Sicherheitsmechanismus verfügt die Schlüsselverwaltung über einen Replikationsmechanismus der jeden Schlüssel redundant auf weiteren Schlüsselservern speichert (siehe 0.4.1). Dieser Mechanismus ersetzt jedoch nicht regelmäßige Backups aller Schlüsseltabellen. Die Zuweisung der Server zu einem Schlüsselintervall kann im Nachhinein nicht verändert werden, somit können auch keine neuen Server der Verwaltung hinzugefügt werden, ohne nicht alle Schlüsseltabellen neu aufzusetzen. Das Aufsetzen und Konfigurieren der Schlüsselverwaltung wird in Kapitel 0.4.2 erklärt.

0.3.3 Crawlverzeichnis

Das Crawlverzeichnis speichert alle heruntergeladenen Webseiten mit URL im Hadoop-Dateisystem. Der Pfad des Verzeichnisses wird in der Konfigurationsdatei festgelegt (siehe 0.4.1) und darf anschließend nicht verändert werden. Der Crawler erzeugt für jede Crawliteration einen Ordner im Crawlverzeichnis, der alle Daten dieses Crawls beinhaltet. Der Name des Ordners entspricht der POSIX-Zeit zum Startzeitpunkt des Crawls. Die Crawldaten sind somit immer chronologisch sortiert und einem eindeutigen Datum zuordenbar. Die Dateien innerhalb des Ordners unterliegen keiner Sortierung und enthalten die gecrawlten Webseiten jedes Hadoop-Knotens in einem einfachen Binärformat (siehe `org.apache.hadoop.io.SequenceFile`). Prinzipiell können ganze Ordner oder selektiv bestimmte Webseiten aus dem Crawlverzeichnis gelöscht werden, ohne die anderen Crawldaten zu beeinträchtigen. Es ist aber zu beachten, dass die gecrawlten Webseiten in der Schlüsselverwaltung weiterhin als gecrawlt vermerkt sind und sofern ein erneuter Crawl dieser Seiten gewünscht ist, dort ebenfalls entfernt werden müssen.

0.4 Installation

Bevor Krake installiert werden kann, muss zunächst ein vollständiges Hadoopsystem auf allen Knoten, die später zum Crawlen benutzt werden sollen, aufgesetzt werden. Die sog. spekulative Ausführung (siehe "mapred.map.tasks.speculative.execution" in der Hadoopkonfiguration) muss für das gesamte Hadoopsystem deaktiviert werden um Dateninkonsistenzen zu vermeiden. Der Classpath von Hadoop (siehe "conf/hadoop-env.sh" in Hadoop) sollte alle, für Krake erforderlichen Jars, enthalten:

- commons-cli (Teil von Hadoop)
- commons-httpclient (Teil von Hadoop)
- commons-logging (Teil von Hadoop)

- hadoop-core (Teil von Hadoop)
- log4j (Teil von Hadoop)
- zookeeper (Teil von Hadoop)
- junit4 (Teil von Hadoop)
- snakeyaml (siehe "http://code.google.com/p/snakeyaml/")
- mysql-connector-java (siehe "http://www.mysql.com/")

Anschließend muss auf den Knoten, die auch als Schlüsselserver (siehe 0.3.2) fungieren sollen, ein mySQL-Server mit einheitlichen Logindaten installiert werden. Die eigentliche Installation von Krake beschränkt sich auf das Kopieren der "krake.jar" auf den Cluster und das Einrichten der Konfigurationsdatei.

0.4.1 Konfiguration

Es wird dringend empfohlen, die Beispielkonfiguration "krake-config.yml" als Basis für eine eigene angepasste Konfiguration zu verwenden. Eine minimale Anpassung der Konfiguration an einen jeweiligen Cluster umfasst immer:

- Eintragen der Hostnamen in die Schlüsselserverliste
- Setzen der mySQL-Zugangsdaten
- Anlegen der Speicherpfade für die Queue
- Anlegen des Crawlverzeichnis

Die Konfigurationsdatei muss auf jeden Knoten kopiert werden und muss dort unter einem, für alle Knoten identischen, Pfad abgelegt werden. Dieser einheitliche Pfad wird durch die Umgebungsvariable "KRAKE_CONFIG_PATH" propagiert und sollte in der Shellkonfigurationsdatei (meist ".profile" im Homeverzeichnis) eingetragen werden.

In der Konfigurationsdatei sind zunächst im Unterpunkt "servers" alle Verarbeitungsknoten des Clusters einzutragen. Unter "dfs_primary" muss der DFS Pfad zum primären DFS Knoten gesetzt werden. Anschließend sollten die Zugangsdaten für die mySQL-Server im Unterpunkt "key_storage" angepasst werden. Im Unterpunkt "queue" muss "local_dir" auf ein Verzeichnis verweisen, das auf allen Knoten im lokalen Dateisystem existiert und für den Hadoop-Prozess les- und schreibbar ist. Eine Auflistung aller weiteren Konfigurationsparameter und deren Bedeutung kann der Beispielkonfiguration "krake-config.yml" entnommen werden.

0.4.2 Schlüsselverwaltung

Nach Einrichten der mySQL-Zugangsdaten in der Konfiguration muss auf jedem mySQL-Server die entsprechende Datenbank angelegt werden. Die zugehörige Tabelle kann anschließenden automatisch von Krake durch den Aufruf

```
hadoop jar krake.jar krake.keys.KeyStorage
```

erzeugt werden. Alternativ kann dies manuell auf jeden Server durch die äquivalenten SQL-Statements geschehen:

```
DROP TABLE IF EXISTS krakekeys;
CREATE TABLE krakekeys (hash CHAR(40) PRIMARY KEY, keyinfo TINYINT);
```

.

0.4.3 Queue

Sofern ein Queue-Verzeichnis bereits angelegt wurde (siehe 0.4.1), kann als letzter Schritt die Queue jedes Knoten mit einem initialen "Seed", d.h. einer ersten URL-Menge, gefüllt werden. Dies kann maschinell oder manuell geschehen und beschränkt sich auf das Anlegen zweier Textdateien im Queueordner:

URL-Liste Textdatei mit Namen "0". Speichert eine URL pro Zeile.

Statedatei Textdatei mit Namen "state". Die erste Zeile darf nur aus einer 0 bestehen, die zweite Zeile enthält die Anzahl der URLs in Datei "0".

Das genaue Format für die Queuedateien ist unter 0.3.1 dokumentiert. Es gilt zu beachten, dass die Queue lokal für jeden Knoten ist, d.h. bei der Verteilung des Seeds auf die Knoten ist auf Überlappungsfreiheit und eine möglichst gleichmäßige Verteilung zu achten.

0.5 Verwendung

Sind alle in 0.4 angeführten Schritte durchgeführt worden, kann der Crawler mit dem Aufruf

```
hadoop jar krake.jar krake.Crawler
```

gestartet werden. Alternativ kann durch Angabe des Zusatzparameters "iters=N" direkt ein Crawl mit N Iterationen gestartet werden.

Bei Verfolgung des Crawlfortschritts kann oft beobachtet werden, dass der Hadoop-Job abbricht, obwohl die Fortschrittsanzeige noch nicht 100% erreicht hat. Dieses Verhalten ist normal und stellt keinen Fehler dar. Der Crawler verarbeitet pro Knoten nur die in der Konfiguration unter "batch_size" angegebene Menge an Webseiten. Dieser Wert kann, besonders bei einer kleinen Queue, nicht immer erreicht werden.

0.5.1 Datenexport und Verarbeitung

Die unter Kapitel 0.2.2 beschriebene Verarbeitungsphase wird mit dem Aufruf für einen ersten Datenexport eingeleitet. Dieser erste Schritt liefert das sog. URLMapping (siehe 0.6.3) und die Contentdatei (siehe 0.6.1). Das URLMapping ordnet jeder URL einen weiteren Schlüssel, sie sog. NodeID, zu. Die NodeID entspricht später der Knotennummer im exportierten Webgraph und wird hauptsächlich für alle weiteren Verarbeitungsschritte genutzt. Der erste Export wird mit

```
hadoop jar krake.jar krake.Exporter
```

gestartet.

Soll zusätzlich ein Webgraph erzeugt werden, so wird zunächst mit dem "Linker" die Vernetzung der Webseiten aus der Crawldaten extrahiert und als vorläufiger Webgraph gespeichert:

```
hadoop jar krake.jar krake.Linker
```

Der finale Webgraph wird schließlich durch

```
hadoop jar krake.jar krake.Finalizer
```

produziert.

Da die Exportprozesse niemals bereits bestehende Daten überschreiben, müssen, sofern Schritte wiederholt werden sollen, vorher die Ausgabedaten dieser Schritte umbenannt oder gelöscht werden (siehe 0.4.1).

0.6 Dateiformate

Krake erzeugt während der Datenverarbeitungsphase eine Reihe von Ausgabedateien, die über Klassen in `krake.file` und `krake.graph` angesprochen werden können. Da unter bestimmten Umständen der manuelle Zugriff auf diese Daten durch andere Programme nötig ist, werden in diesem Kapitel die Dateien und die entsprechenden Dateiformate erklärt. Bei allen binären Formaten ist darauf zu achten, dass sie von der JavaVM geschrieben wurden und daher auch auf LittleEndian-Maschinen alle primitive Datentypen (Integer, Double usw.) im BigEndian-Format gespeichert werden. Die Dateiformate der Queue (siehe 0.3.1) und Konfiguration (siehe 0.4.1) werden in ihren jeweiligen Kapiteln behandelt.

0.6.1 Contentdatei

Die Contentdatei ist das Resultat des ersten Exports (siehe 0.2.2) und speichert in kompakter Form den Inhalt aller gecrawlten Webseiten. Die Contentdatei kann über die Klassen in `krake.file` sowohl als bidirektionaler Datenstrom als auch als Random Access Datenbank benutzt werden. Um einen direkten Zugriff auf einzelne Inhalte zu ermöglichen, wird zusätzlich zur Contentdatei eine Offsetdatei benötigt die als Sprungtabelle in die Contentdatei dient. Die Offsetdatei wird im selben Exportschritt wie die Contentdatei erzeugt.

Die Contentdatei besteht aus einer Sequenz von komprimierten Webseiten. Jede Webseite wird durch einen 4 Byte signed Integer eingeleitet, der die Größe des folgenden komprimierten Datenblocks angibt. Dieser Datenblock kann mit `java.util.zip.Inflater` zum ursprünglichen Quelltext der Webseite dekomprimiert werden. Auf den Datenblock folgt ein weiterer 4 Byte signed Integer, der nochmals die Größe des gerade gelesenen Datenblocks angibt. Diese Speichermuster wiederholt sich für alle Webseiten in der Contentdatei bzw. bis das Ende der Datei erreicht ist.

Die Offsetdatei speichert für jede NodeID ein Byteoffset in die Contentdatei, das auf den Anfang des ersten Size-Integers zeigt. In der Offsetdatei sind daher immer Paare von 4 Byte signed Integer (NodeID) und ein 8 Byte signed Integer (Offset) gespeichert.

0.6.2 Webgraph

Der Webgraph speichert die Vernetzung der gecrawlten Webseiten untereinander ohne dabei zusätzliche Daten wie Inhalt oder Metainformationen zu beinhalten. Der Graph wird als Ausgabe des letzten Schrittes in der Export- und Verarbeitungsphase erzeugt (siehe 0.2.2) und kann mit den entsprechenden Klassen unter `krake.graph` verwendet werden. Das Format der Webgraphdatei orientiert sich an der klassischen Adjazenzlistenrepräsentation von Graphen: Die Webgraphdatei beginnt mit einem 4 Byte signed Integer der die Anzahl der Knoten im Graph speichert. Es folgt die Kantenliste des ersten Knoten (NodeID=0), danach die Kantenlisten des zweiten Knoten (NodeID=1) usw.. Eine Kantenliste beginnt mit zwei 4 Byte signed Integer. Der erste Integer "N" speichert die gesamte Anzahl der Kanten für diesen Knoten, der zweite Integer "I" die Anzahl der In-Kanten für diesen Knoten. Es folgenden "N" viele NodeIDs der adjazenten Knoten als 4 Byte signed Integer. Die ersten "I" NodeIDs dieser Liste stellen dabei die Inlinks und alle folgenden "N-I" vielen NodeIDs die Outlinks dar.

0.6.3 URLMapping

Das URLMapping ordnet jedem Schlüssel einer gecrawlten Webseite seine NodeID im Webgraphen und jeder NodeID seinen Schlüssel zu. Das Mapping wird daher immer genutzt, sobald Informationen aus Inhalt und Webgraph verknüpft werden sollen. Das Mapping besteht aus einem sortierten Array der Schlüssel aller gecrawlten Webseiten, die Position eines Schlüssels im Array spiegelt seine NodeID wieder. Eine Auflösung NodeID zu Schlüssel kann damit in konstanter Zeit vollzogen werden, wohingegen zur Auflösung eines Schlüssels in eine NodeID eine binäre Suche durchgeführt wird. Entsprechend der einfachen Speicherstruktur ist auch das Dateiformat des Mapping einfach gehalten. Die Datei beginnt mit einem 4 Byte signed Integer, der die Anzahl der gespeicherten Schlüssel hält, gefolgt von allen Schlüsseln im Mapping (jeweils 20 Byte). Ein URLMapping wird in Krake durch die Klasse `krake.file.URLMapping` repräsentiert.

0.7 Evaluierung

Da das Krakeframework während seiner Einsatzphase noch mehrfache Überarbeitungen durchlaufen hat, ist die erzielte Verarbeitungsgeschwindigkeit leider nur schwer vergleichbar oder verlässlich einzuschätzen. Hinzu kommt, dass vergleichbare Messungen aufgrund von Unterschieden in der Netzwerkanbindung der eigenen oder Gegenseite fast nicht möglich sind. Während der Einsatzphase haben sich hingegen andere Features wie Robustheit des Crawlers und der Datenformate sowie eine Fehlertoleranz als wichtige Merkmale für das Framework herauskristalisiert.

Bei jedem längerfristigem Crawl muss mit dem Ausfall von Komponenten oder der teilweisen Korruption von Daten durch Defekte gerechnet werden. Die Rückführung des Systems in einen konsistenten Zustand erfordert den Einsatz von dedizierten, speziell für diesen Fall geschriebenen, Tools. Krake begünstigt das schnelle Entwickeln dieser Tools durch seine einfachen Speicherformate und die eingebauten Datenredundanz in der Dateisystem- und Schlüsselverwaltung. Während des aktiven Einsatzes des Krakeframeworks kam es mehrfach zu kritischen Datenkorrumptionen oder Ausfällen durch Hardware- oder Softwarefehler.

In fast allen Fällen konnten die betroffenen Daten komplett wiederhergestellt oder zu großen Teilen rekonstruiert werden.

Krake stellt ein flexibles und skalierbares Crawlerframework dar, dass sich durch mehrmonatigen Einsatz bewährt hat. Aufbauend auf Krake und MapReduce kann in wenigen Schritten eine Suchmaschine aufgesetzt werden (siehe Beispielquellcode in `krake.search`) oder eine tiefgreifende Analyse von Graph- oder Inhaltsmerkmalen vorgenommen werden.