Fachbereich Angewandte Informatik
an der Fachhochschule Bonn Rhein Sieg
Seminar: Verteilte und Parallele Systeme II

Seminarhausarbeit

Zugriffsmuster vorhersagen mittels Hidden-Markov-Modellen

25. Januar 2003

Themensteller:
Prof. Dr. Rudolf Berrendorf

Author:
Matthias Lübken
matthias@luebken.com

Inhaltsverzeichnis

1 Einleitung

In der allgemeinen Datenverarbeitung nimmt die Rechenleistung ständig zu, ob dies nun durch schnellere Prozessoren oder mehr parallel arbeitende Prozessoren erreicht wird. Insbesondere die Wissenschaft hat immer neuere Anforderungen und verlangt nach höherer Rechenleistung. Dadurch ergeben sich aber neue Probleme. Besonders für Anwendungen mit häufigen und großem In- und Output, ist der Zugang zum Dateisystem der Flaschenhals. Hier muss häufig der Prozessor auf neue Daten warten, bevor er weiter arbeiten kann.

Dieses Problem versucht man mit verschiedenen Strategien anzugehen. Eine Möglichkeit ist zum Beispiel die Daten geschickt auf einem parallelen Dateisystem zu verteilen und die Daten in einem möglichst schnellen Speicher (z.B. Cache) vorzuhalten. Doch hierbei stellt sich die Frage welche Daten wie verteilt und wie vorgehalten werden. Oder anders ausgedrückt wie greift meine Applikation auf meine Daten zu. Man spricht auch von einem Zugriffsmuster der Applikation auf die Daten.

Der einfachste Ansatz hierbei ist dem Entwickler der Applikation die Steuerung des verteilten Speichers zu überlassen. Hierfür gibt es bereits verschiedene APIs, bei dem der Entwickler das Verhalten in die jeweilige Applikation integrieren muss. Beispiele hierfür sind z.B. MPI-IO oder Intels PFS. Sicherlich kann der Entwickler den Zugriff auf zukünftige Daten am besten voraus sehen, nur stellt sich die Frage wie gut dieser Ansatz von dem Entwickler umgesetzt wird und wie komfortabel dieses System ist. Der Entwickler muss sich nun mit verteilten und lokalen Performance Problemen kümmern, und kann sich nicht mehr 100 % um die eigentliche Aufgabe seiner Applikation kümmern.

Ein weiterer Ansatz die hier besprochen wird, ist die Entscheidung dem Entwickler abzunehmen und Anhand der Lese - und Schreiboperationen die verschiedene Zugriffsmuster automatisch zur Laufzeit zu erkennen. So kann dann ein System, wenn es ein Zugriffsmuster erkannt hat, die Taktik des Dateisystems die Daten vorzuhalten und zu verteilen ändern.

Es gibt verschiedene Methoden die verschiedenen Zugriffsmuster zu erkennen. In dieser Ausarbeitung wird der Ansatz der versteckten Markov-Modelle (HMM) betrachtet. Hierzu werden diese zunächst im Abschnitt **2** eingeführt und dann im Abschnitt **2.1** unter dem Aspekt der Zugriffsmuster betrachtet. Anschliessend werden unter **3.1** lokale Zugriffsmuster beschrieben, also

Zugriffsmuster die lediglich ihre eigenen lokalen Daten haben, auf die sonst niemand zugreift. Unter **3.2** werden diese dann um globale Zugriffsmuster erweitert, bei denen die Daten verteilt sind und die verschiedenen lokalen Zugriffsmuster zusammen geführt werden müssen.

2 Markov Modelle

Markov Modelle dienen im Allgemeinen der Beschreibung und Modellierung von Zufallsprozessen. Wie bei anderen stochastischen Verfahren werden möglichst viele Referenzwerte gesammelt um ein einfaches mathematisches Modell / Objekt zu erzeugen, das das zu untersuchende System beschreibt um anschliessend Aussagen über dieses System machen zu können. Markov Modelle werden in verschiedenen Bereichen benutzt wie zum Beispiel in der Spracherkennung oder Prozessen aus der Wirtschaft (z.B. Bausparverträge).

Bei dem zu untersuchenden Zufallsprozess geht man von einer zeitlichen Abfolge von messbaren Werten aus. Dabei beeinflussen sich die Messwerte in ihrer zeitlichen Abfolge gegenseitig. Man führt das System in bestimmte Zustände und Zustandsübergänge über. Zu bestimmten Zeitpunkten wechselt das System seinen Zustand anhand von bestimmten Wahrscheinlichkeiten. Zudem gibt es ein Ereignis das nach aussen hin beobachtet werden kann. Vergleichbar mit einer endlichen Maschine aus der theoretischen Informatik.

Bei den Observable (beobachtbare) Markov-Modellen (OMMs, Markovketten) wird durch dieses Ereignis der nächste Zustand festgelegt. Die Zustandsübergänge sind also transparent. Zudem sind die Zahl der Zustände bekannt.

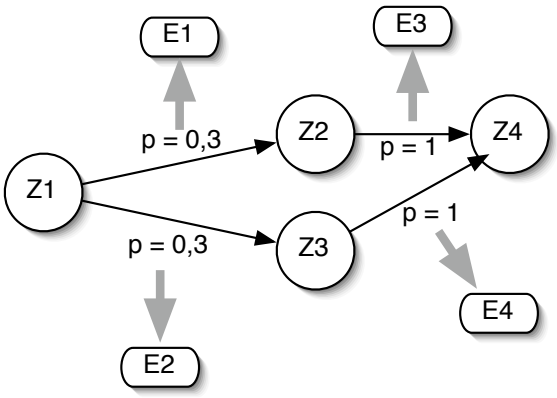

Abbildung 1: Ein allgemeines Markov-Modell. Mit Zuständen, Zustandsübergängen samt Wahrscheinlichkeiten und Ereignissen.

Anders ist dies bei den Hidden (versteckte) Markov-Modellen (HMMs). Hier gibt es verschiedene Übergänge für ein beobachtbares Ereignis. Oder anders gesagt, ein beobachtbares Ereignis gibt nicht unbedingt wieder in

welchem Zustand das System ist. Die Zahl der Zustände ist unbekannt sowie die Zustandsübergänge und sind nur analytisch zu schätzen.

2.1 Hidden Markov-Modelle bei Zugriffsmuster

Das Prinzip der Hidden Markov-Modelle bedeutet bei den Datei-Zugriffsmuster, dass man nicht in die Programmlogik schauen kann, die die verschiedene Datei-Zugriffe erzeugt. Statt dessen protokolliert man die bisherigen Zugriffe und die Programmlogik zu modellieren um so zukünftige Zugriffe vorauszusagen. Man kann auch sagen das Hidden Markov-Modelle die versteckte Logik lernen. Es existieren verschiedene Algorithmen um das Modell anhand der beobachteten Sequenzen zu trainieren, um dann die Wahrscheinlichkeit für zukünftige Sequenzen zu berechnen.

Hieraus ergibt sich eine wichtige Vorraussetzung bei dem Einsatz von Hidden-Markov-Modellen. Das Programm muss vor dem eigentlichen Einsatz einmal ausgeführt werden, damit das Markov-Modell die verschiedenen Zustände und die Übergänge lernen kann. Es existiert ein Hidden-Markov-Modell pro Datei die vom Programm eingelesen wird. Das Programm muss also auf den selben Daten mehrmals arbeiten, damit sich der Trainingsdurchlauf in der gesamten Performance rentiert. Zumindest sollten sich die Daten nur gering unterscheiden, damit ein Modell für verschiedene Daten genutzt werden kann. Gerade bei wissenschaftlichen Anwendungen werden häufig die selben Programme mit kleinen Änderungen in den Algorithmen auf den selben oder ähnlichen Daten immer und immer wieder ausgeführt.

Durch den Trainingsdurchlauf ergibt sich ein Vorteil gegenüber anderen Strategien. Wenn das Programm die jeweilige Datei benutzen möchte, wird das zugehörige Markov-Modell ausgewählt und man kann von Anfang an das beste Zugriffsmuster auswählen.

3 Klassifikation von Zugriffsmustern

Bei einem Zugriffsmuster geht es um das Muster mit dem eine Applikation auf die für sie notwendigen Daten zugreift. Wenn man dieses Muster vorhersagen und damit die Daten cachen / prefetchen kann, muss die Applikation nicht mehr auf den langsamen Dateisystemzugriff warten, sondern kann mit den schnelleren Cache arbeiten. Oder man kann zum Beispiel das Prefetching-Verhalten ändern. Wenn man herausfindet, dass der Zugriff völlig zufällig ist kann man unnötiges prefetchen ausschalten. Um es dann wieder gezielt einzuschalten wenn ein neues Muster erkennbar ist.

Dateisysteme sind meistens für ein bestimmtes Muster optimiert und verhalten sich schlecht, wenn sich dieses Muster ändert. So sind zum Beispiel UNIX Dateisysteme für sequentiellen Zugriff optimiert und verhalten sich schlecht bei zufälligem Dateizugriff.

3.1 Lokale Zugriffsmuster

Zunächst einmal betrachtet man die lokale Zugriffsmuster, also einem Zugriff von einem einzelnen Thread auf einen Datenbestand. Die verschiedenen Muster werden anhand von drei wesentlichen Merkmalen unterschieden und unterteilt. Diese Merkmale lassen sich mit geeigneten Methoden voraussagen und eignen sich Caching und Prefetching Methoden des Dateisystems zu beeinflussen.

- Read / write mix
  Wird nur lesend oder schreibend oder vermischt zugegriffen ?
- Sequentiality
  Ist der Zugriff in einer bestimmten Sequenz ?
- Request size
  Ist die Größe bekannt und verändert sich diese ?

Diese Merkmale lassen sich auf einem drei dimensionalem Koordinatenkreuz eintragen und damit würde ein Punkt in dem (positivem) Raum ein bestimmtes Zugriffsmuster beschreiben.

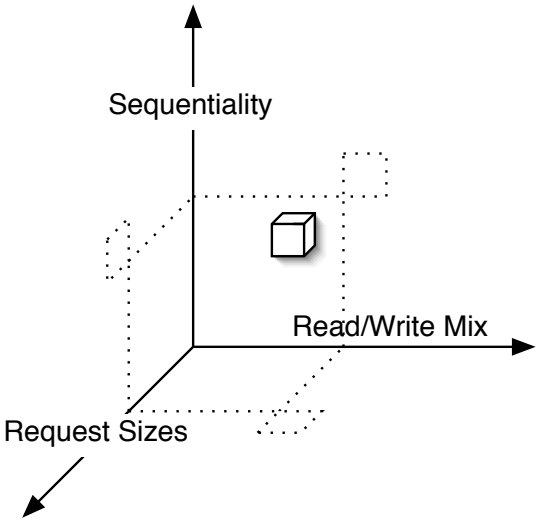

Abbildung 2: Der Kasten gibt ein bestimmtes Zugriffsmuster wieder. Zum Beispiel: Read Only / Uniform Request Size / 1-D Strided

Wie bei jedem Modell ist es auch hier wichtig und entscheidend für die effiziente Nutzung des Modells eine geeignete Abbildung der messbaren Größen auf die vorhanden Variablen zu finden. Bei Hidden Markov Modellen sind die Variablen: Zustand, Anzahl der Zustände und beobachtbare Ereignisse.

Für den Zustand nimmt man ein Segment einer Datei. Dies ist eine Region einer Datei mit variabler oder fester Größe. Idealerweise sollte die Größe auf die Cachegröße des Systems und der Zugriffsgröße der verwendeten Applikation angepasst sein. Innerhalb eine Segments geht man von einem sequentiellen Zugriff aus. Bsp: Eine 10 MB Datei hat 1280 K-Byte Blöcke wodurch das HM Modell 1280 Zustände hätte. Die beobachtbaren Ereignisse sind die Lese und Schreiboperationen der Applikation. Durch eine Lese oder Schreiboperation würde also unser System von einem aktuellen File-Block in den nächsten aktuellen File Block wechseln.

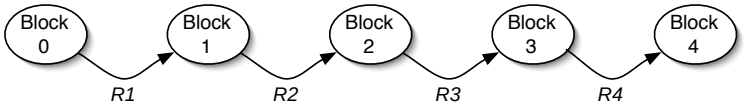

Abbildung 3: Ein sequentielles Lesen von vier Blöcken.

Abbildung 3 ist ein Beispiel für sequentielles Lesen. Dabei ist alles beobachtbar da bei der gegebenen Beobachtung das System nur in einem Zustand sein kann. Daher würde man hier auch von einer Markov-Kette sprechen.

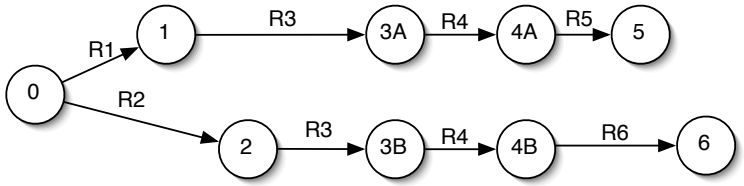

Abbildung 4: Verschiedene Zustände für ein mögliches Ereigniss

Anders ist dies in der Abbildung 4. Hier gibt es für ein beobachtbares (z.B. R4) zwei mögliche Zustände (z.B. 4A oder 4B). Und jenach Zustand zwei unterschiedliche Blöcke die als nächstes gelesen werden. Dieses Modell merkt sich also, wie das aktuelle Programm zu dem Block 3 und 4 gekommen ist und kann nun entscheiden, welches Strategie die günstigste wäre für die nächste Blöcke und ob gegebenenfalls die Cache-Strategie gewechselt werden muss.

Die oben vorgestellten Merkmale wird in den folgenden Stufen unterschieden:

Read / Write Read Only, Write Only, Read-Update-Write, Read/Write Mix

Sequentiality Sequential, 1-D Strided, 2-D Strided, Variably Strided

Request Size Uniform, Variable

So würde zum Beispiel das Zugriffsmuster aus Sequential nach 1-D Strided wechseln, wenn die Wahrscheinlichkeit dass die zukünftige Zugriffe eher 1-D Strided sind einen gewissen Grenzwert überschreitet. Wenn nun das aktuelle Zugriffsmuster besagt man soll einen bestimmten Block prefetchen, und das aktuelle Markov-Modell aber die Wahrscheinlichkeit für diesen Block als gering ausgibt und diese Wahrscheinlichkeit unter einem bestimmten Grenzwert fällt, kann ein anderes Zugriffsmuster ausgewählt werden, welches einer höhere Wahrscheinlichkeit für den nächsten Block ergibt.

3.2 Globale Zugriffsmuster

Bis jetzt wurden nur lokale Zugriffsmuster betrachtet, zum Beispiel nur die Zugriffe eines Prozesses eines parallelen Programms. Bei globalen Zugriffsmustern wird zusätzlich beachtet dass bei parallelen Systemen sich die unterschiedlichen lokalen Zugriffsmuster gegenseitig beeinflussen. Häufig können angemessene Entscheidungen nur so getroffen werden, die mit einfachen lokalen Zugriffsmuster nicht gemacht werden können. Zum Beispiel müsste eine gemeinsame Initialisierungs-Datei von jedem parallelen Prozess komplett gelesen werden und die eigentliche Daten die bearbeitet werden, müssen auf die verschiedene Prozessoren und deren Caches verteilt werden. Um solche Entscheidungen zu treffen ist ein globales Wissen notwendig.

Globale Klassifikation von Zugriffsmister ist nicht trivial, da lokale Zugriffsmuster sich in Zeit und Raum überschneiden können. Die zeitliche Koordination wird dadurch gesichert dass jedes lokale Zugriffsmuster in bestimmten Zeitintervallen sich mit den anderen überschneiden muss. So gilt ein globales Zugriffsmuster für alle lokalen Zugriffsmuster die während der Zeit des globalen Musters gestartet und beendet wurden. Räumliche Koordination wird mittels einer bestimmten Zugriffsalgebra zugesichert. So ist zum Beispiel das globale Zugriffsmuster Read-Only, wenn alle lokalen Zugriffsmuster Read-Only sind. Es sind aber auch noch differenziertere Unterscheidungen möglich, die hier nicht beschrieben werden.

4 Performance Tests

Die folgenden Tests wurden mittels einem Portable Parallel File System (PPFS) durchgeführt. Dabei handelt es sich um eine Bibliothek die zwischen dem Dateisystem und der Applikation arbeitet. Die Applikation greift über das PPFS auf die Daten zu und eröffnet so die Möglichkeit die verschiedenen Eigenschaften der jeweiligen Applikation abzufangen und nach eigenen Vorstellungen das Verhalten (z.B. das Cacheverhalten) des Dateisystems zu verändern.

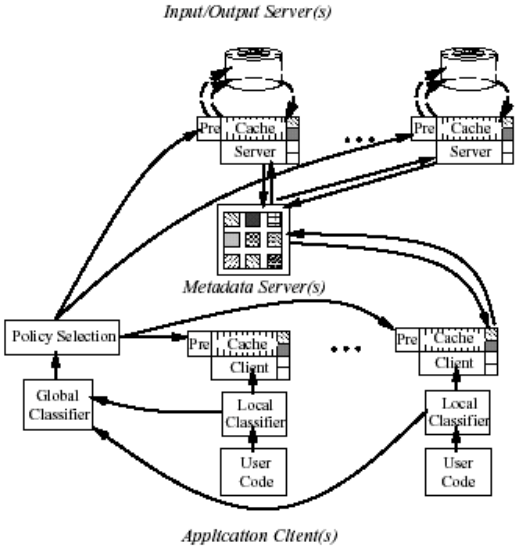

Abbildung 5: Der Aufbau von PPFS

4.1 Lokale Tests

In diesem Test wird auf eine 40 Millionen Byte große Datei mit zwei Zugriffsmustern zugegriffen. Die erste Hälfte der Datei wird sequentiell gelesen, die andere Hälfte zufällig. Der ganze Test wird drei mal ausgeführt. Einmal ohne jede Klassifikation der Zugriffsmuster also nur über das eigentliche Dateisystem, einmal mittels HMM und einmal mittels Neuralen Netzen.

Beobachtungen:

- Die periodischen Senken der Kurven beim sequentiellen Lesen sind Zeitpunkte wo der jeweilige Cache nicht voll war und die Applikation auf das Dateisystem warten musste.
- Die HMM erkennt die Änderung des Zugriffsmusters früher als ANN
- Das Zugriffsmuster ändert sich von sequentiell nach zufällig. HMM und ANN schalten darauf hin den Cache ab und haben danach einen wesentlich besseren Durchsatz als das pure Dateisystem.

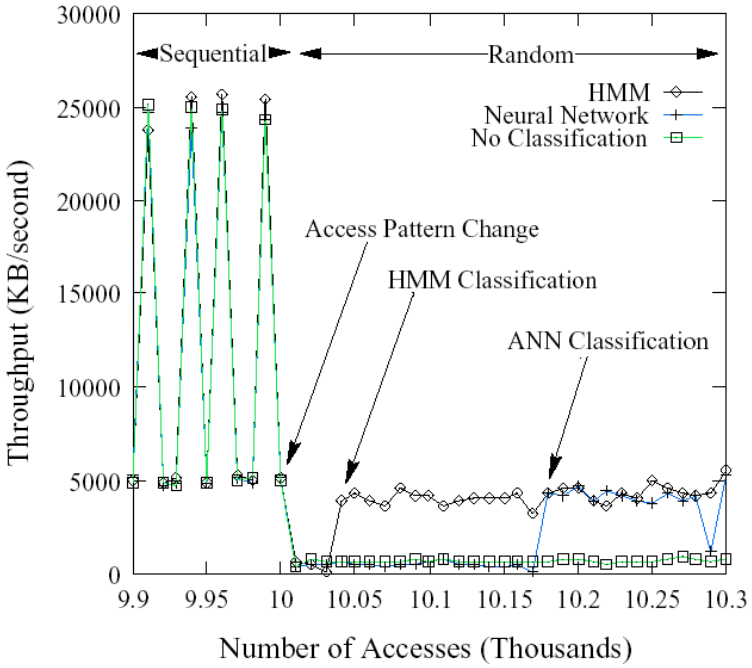

Abbildung 6: Verschiedene Zugriffsmuster adaptieren

4.2 Globale Tests

Wie bei den lokalen Tests wurde auch hier das PPFS auf Intels PFS aufgesetzt. PFS ist ein paralleles Dateisystem, welches Daten auf einem verteilten Speicher verteilt. Dabei gibt es Platten als Input-Knoten und Platten als Output-Knoten. Bei normalem Gebrauch würde der Entwickler die verschiedenen Modi von PFS selektieren um die aktuelle Zugriffe zu optimieren. Im folgenden werden folgende Modi benutzt:

M_UNIX : Der default Unix Modus mit Input / Output Atomarität
Die Datei Operationen werden auf einer "first-come, first-served" Basis ausgeführt.

M_ASYNC : Asynchrones Lesen und Schreiben und bewahrt nicht die Input/Output Atomarität, diese muss die Applikation garantieren.

M_GLOBAL : Alle Prozesse greifen auf die gleichen Daten zu.

Weitere Modi: M_LOG, M_SYNC, M_RECORD, M_GLOBAL

QCRD:

QRCD steht für Quantum Chemical Reactions Dynamics und untersucht elementare chemische Reaktionen. Bei diesem parallelen Programm werden die Daten (Matrizen) auf die verschiedenen Knoten verteilt und mit dem gleichen Code berechnet.

Wir betrachten die erste Phase von QRCD, in der 64 Prozessoren koordiniert 13 globale Matrizen schreiben, die sich überschneiden. Mit dem default M_UNIX Modus braucht der In und Output 10,6 Prozent der gesamten benötigten Zeit. Wenn man M_ASYNC wählt wird diese Zeit auf weniger als einem Prozent gesenkt.

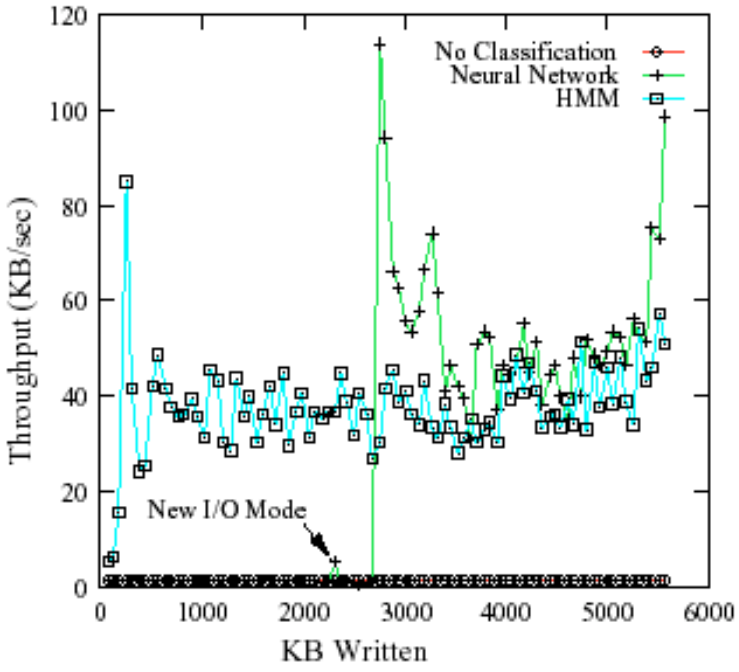

Abbildung 7: QCRD mit den verschiedenen Strategien

Wie vorher sieht man auch hier, dass HMM einen Vorsprung vor ANN hat, da es bereits zu Beginn weiss was für ein Zugriffsmuster vorherrschen wird und wann es wechseln wird. Und in diesem Fall ist dieser Vorteil auch noch gewichtiger.

Literatur

[1] Tara M. Madhyastha Daniel A. Reed
Input/Output Access Pattern Classification Using Hidden Markov Models (1997)

[2] http://www.cs.umd.edu/projects/hpsl/io/sio-hlapi/currentAPIs.html

[3] http://www-pablo.cs.uiuc.edu/Project/IO/iomodes.htm