Vergleich dreier Implementationsvarianten für eine Analyse von Satellitenbildern

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Arts (B. A.)

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

eingereicht von: Robin Ellerkmann
geboren am: 25.04.1992
in: Berlin

Gutachter: Prof. Johann-Christoph Freytag, Ph.D.
Dipl.-Inf. Mathias Peters

eingereicht am: ..... verteidigt am: .....

Inhaltsverzeichnis

Kapitel 1

Einleitung

In den vergangenen Jahren war ein massiver Zunahme des generierten Datenaufkommens zu beobachten [EMC14]. Viele Projekte, Unternehmen und Institutionen haben Zugriff auf eine gewaltige Menge an Daten. Diese wächst immer schneller an. 2004 analysierte Google circa 100 Terabyte pro Tag [DG04]. Bis zum Jahr 2008 war die täglich zu analysierende Datenmenge bereits auf 20 Petabyte angewachsen [DG08]. Das Sloan Digital Sky Survey, das ein Viertel des Himmels astronomisch erkundet, hat seit 1998 insgesamt 116 Terabyte an astronomischen Daten gesammelt [YAJEA+00, AAP+15]. Jede Nacht kommen circa 250 Gigabyte neu hinzu Quelle: Herr Prof. Freytags VL, dort angegebene Quelle ist offline. Wie angeben?. Ein weiteres Beispiel ist das 1000 Genomes Project [Bak10], das zwischen 2008 und 2013 insgesamt 464 Terabyte Daten zum menschlichen Genom sammelte. Insgesamt werden die Datenmengen weiter stark zunehmen, für das Jahr 2020 wird eine weltweites Datenaufkommen von 44 Zettabyte prognostiziert [EMC14]. Diese Entwicklung offenbart diverse neue Herausforderungen bei der Speicherung, Verarbeitung und Analyse von Daten. Dabei spielt die möglichst schnelle Verarbeitung von stetig generierten Daten eine große Rolle. Diese muss im Gegensatz zur Verarbeitung bereits gespeicherter Daten abhängig vom aktuellen Datenaufkommen skalieren. Aktuelle Datenverarbeitungssysteme wie Apache Hadoop [Foue] und Apache Flink [Fouc] bieten diese Möglichkeit der Datenflussanalyse und ermöglichen eine flexible Analyse der Daten. Kern dieser Systeme ist eine Implementierung des Map-Reduce Paradigmas [DG08] sowie die Nutzung von *User Defined Functions*. Diese ermöglichen eine parallele Abarbeitung von Arbeitsschritten in einem direkten, azyklischen Graphen. Der DAG wird zuvor aus dem vom User bereitgestellten Quellcode erzeugt. Die durch diese Architektur erreichbare, massiv parallelisierbare Ausführung der Datenanalyse ermöglicht die Nutzung von Clustern. Somit wird eine skalierbare Infrastruktur genutzt, die wiederum eine skalierte Nutzung der Datenverarbeitungssysteme ermöglicht. Diese Systeme können auch die weiterhin wichtige Nutzung und Analyse von bereits konsistent gespeicherten Daten unter Nutzung des parallelen Verarbeitungsansatzes durchführen. Dies ist insbesondere deshalb notwendig, da traditionelle Datenbanksysteme große Datenmengen nicht immer in akzeptabler Form und Verarbeitungszeit verarbeiten können [Jac09]. Das Hauptproblem bei der Verarbeitung von großen Datenmengen auf einzelnen Maschinen entsteht, wenn die zu verarbeitende Datenmenge die Hauptspeichergröße übersteigt. In diesem Fall müssen die nicht in den Hauptspeicher speicherbaren Daten zur Verarbeitungszeit nachgeladen werden, was die Verarbeitungszeit aufgrund der unterschiedlichen Beschaffenheit der verschiedenen Speicherebenen extrem verlängert. Um diese SSpeicherklippeßu umgehen, werden zunehmend parallelisierbare Ansätze der Datenverarbeitung verfolgt. Im Rahmen dieser Bachelorarbeit sollen demzufolge ein traditioneller und ein massiv parallelisierbarer Ansatz bei der Verarbeitung von großen Datenmengen untersucht werden. So soll eine Abschätzung der Leistungsfähigkeit, Vorteile und Nachteile beider Ansätze ermittelt werden. Der Vergleich beider Ansätze wird am Beispiel ei-

nes Algorithmus zur Approximierung von Pixelzeitreihen durchgeführt. Dieser wird im Rahmen des Projekts GeoMultiSens[GP] zur Analyse der Veränderung der Flora in einer geographischen Region genutzt. Dabei werden durch Landsat-Satelliten Satellitenaufnahmen bereitgestellt, die nach der Aufbereitung durch vorgestellte Algorithmen ausschnittweise untersucht werden. Nach der Analyse werden anschließend mithilfe des Algorithmus auf Basis der approximierten Werte Prognosen zur weiteren Entwicklung der Flora der untersuchten Region gestellt. Dabei werden bei einer Analyse mehrere Szenenausschnitte derselben geographischen Region analysiert. Dabei müssen große Datenmengen verarbeitet werden, so dass sich die Nutzung eines aktuellen Datenverarbeitungssystems anbietet. Bei dieser Bachelorarbeit wird als Vertreter der massiv parallelisierbaren Datenverarbeitungssysteme Apache Flink genutzt.
Es werden drei unterschiedliche Implementierungen des Algorithmus untersucht, die sich hinsichtlich der eingesetzten Technologien und Programmiersprachen unterscheiden. Die zugrunde liegende Methodik, die der Algorithmus implementiert, ist bei allen untersuchten Varianten identisch. Als Basis wird die bereits implementierte und in der Praxis genutzte Python-Implementation genutzt. Sie sollte somit die untere Schranke der Leistungsmessungen darstellen. Die zweite und dritte Variante werden in Flink implementiert. Diese beiden Varianten unterscheiden sich bezüglich der genutzten Programmiersprache. Zur Implementierung der zweiten Variante wird Java-Schnittstelle von Flink genutzt, zur Umsetzung von Variante drei die Python-Schnittstelle. Schließlich werden alle drei Varianten unter identischen Bedingungen getestet. Dies bedeutet, dass sowohl die Testumgebung als auch die Testdaten identisch sein sollen. Dabei sollen alle Varianten sowohl auf einer leistungsfähigen Einzelmaschine als auch mit einem Cluster von Maschinen getestet werden. Ausgehend von den Untersuchungen und den ermittelten Ergebnissen soll nachfolgend eine Bewertung der drei Implementierungsvarianten des Algorithmus vorgenommen werden. Dabei sollen insbesondere die Größe der Ausgangsdatenmenge, die genutzte Hardware sowie die Größe der untersuchten Bildausschnitte in Bezug zu den Ergebnissen gesetzt werden.

Kapitel 2

Grundlagen

Die Basis für die Satellitenbildanalyse mittels Apache Flink bilden zum einen Konzepte aus der Geographie, zum anderen Strukturen und Vorgehensweisen aus der Informatik. Während die geographische Komponente insbesondere bei der Aufnahme der Satellitenbilder sowie bei der inhaltlichen Konzeption der Analysen sowie der Bereinigung der Daten vertreten ist, ist die Informatik für eine technisch korrekte und effiziente Umsetzung der geographischen Konzepte verantwortlich. Nachfolgend werden zuerst die geographischen Grundlagen der Fernerkundung erläutert. Dies umfasst insbesondere die technische Spezifikation der Aufnahmegeräte der eingesetzten Satelliten sowie wichtige Verfahren zur Datenaufbereitung sowie zur Datenanalyse. Anschließend wird ein Überblick über parallele Datenverarbeitungssysteme gegeben. Dieser umfasst auch die Eigenschaften von Big Data sowie eine konzeptuelle Beschreibung der Datenanalyseplattform Apache Flink. Abschließend wird Apache Flink aus der Entwicklerperspektive betrachtet. Dabei werden insbesondere Eigenschaften der Plattform beschrieben, die bei der Entwicklung von Programmen auf Basis von Flink von Bedeutung sind. Des weiteren wird die Programmiersprache Python betrachtet.

2.1 Grundlagen der Satellitenbildanalyse

2.1.1 Fernerkundung mithilfe des Landsat-Satellitensystems

Als Fernerkundung wird „die Gesamtheit der Verfahren zur Gewinnung von Informationen über die Erdoberfläche oder anderer nicht direkt zugänglicher Objekte durch Messung und Interpretation der von ihr ausgehenden (Energie-) Felder“ [Deu12] verstanden. Fernerkundungssatelliten verfügen über verschiedene Aufnahmesysteme, die durch multispektrale Messungen von emittierter elektromagnetischer Strahlung eine berührungsfreie Beobachtung der Erdoberfläche ermöglichen. Bei der multispektralen Messung werden von Sensoren registrierte spektrale Signaturen einzelnen Bereichen des elektromagnetischen Spektrums zugeordnet. Das Resultat sind mehrere spektrumsspezifische, simultan aufgenommene Satellitenbilder, die nur das aufgefangene Licht eines spezifischen Spektralbereichs, auch Spektralband genannt, zeigen. Die Art und Qualität der Aufnahmesensoren ist dabei abhängig vom Typ des Satelliten.

Die Ausgangsdaten für die Untersuchungen in dieser Bachelorarbeit wurden von Satelliten des Landsat-Satellitensystems aufgenommen. Der erste Landsat-Satellit Landsat 1 wurde 1972 gestartet. Seitdem wurden die Sensoren und die Satelliten kontinuierlich weiterentwickelt. Aktuell sind Landsat 7 und, im Rahmen der Landsat Data Continuity Mission, Landsat 8 im Einsatz. Landsat 8 nutzt in der aktuellen Generation zwei verschiedene Instrumente zur Fernerkundung. Den Operational Land Imager (OLI) und die Thermal Infrared Sensors (TIRS).

Der OLI erfasst emittierte elektromagnetische Strahlung im Spektralbereich von 0,433 µm bis 1,390 µm unterteilt in acht Spektralkanäle sowie einen panchromatischen Kanal. Es werden mehr als 7000 Detektoren pro Spektralband genutzt, um eine bessere Bildqualität zu bieten als frühere Systeme [MSWI04]. Neben den klassischen Farbspektren Blau, Grün und Rot nutzt Landsat-8 ein weiteres Band, das speziell für die Fernerkundung von Küsten genutzt wird. Außerdem verfügt Landsat-8 über drei Infrarotbänder, die nahes und mittleres Infrarotlicht registrieren, sowie ein weiteres Infrarotband, das auf die Beobachtung von Cirruswolken spezialisiert ist. Der panchromatische Kanal registriert elektromagnetische Strahlung mit Wellenlängen von 0,500 µm bis 0,680 µm. Dieser Spektralbereich entspricht etwa dem des menschlichen Auges. Aufgrund des, im Vergleich zu den einzelnen Farbfrequenzbändern, breiten abgedeckten Spektralbereichs ist eine höhere Auflösung der Bilder möglich.

Die Thermal Infrared Sensors (TIRS) [Cha11] umfassen zwei Thermalkanäle. Diese erfassen im Gegensatz zu den Multispektralkanälen elektromagnetische Emissionen mit Wellenlängen zwischen 10,30 µm und 12,50 µm, also langwellige Infrarotstrahlung. Dies ist insbesondere für die Beobachtung von Wolken nützlich. Die Kantenlänge der einzelnen Pixel beträgt 100 Meter. Diese kann nachträglich auf 30 Meter angeglichen werden, um eine bessere Kompatibilität mit den Aufnahmen der Multispektralbänder zu gewährleisten.

Landsat 8 sendet pro Tag 400 Aufnahmen der Erdoberfläche, auch Szenen genannt, an die Bodenstation. Eine Aufnahme zeigt dabei eine geographische Region der Erde mit einer Ost-West-Ausdehnung von 185 Kilometer. Dies entspricht 100 nautischen Meilen. Die Nord-Süd-Ausdehnung einer Szene beträgt circa 174 Kilometer

Durchschnittlich wird jede Region der Erde alle 16 (?) Tage überflogen [IDB12].

Die von Landsat-Satelliten aufgezeichneten und übermittelten Bilder müssen jedoch vor der Durchführung von Analysen aufbereitet werden.

2.1.2 Aufbereitung und Analyse von Satellitenbildern

Die durch die Landsat-Satelliten aufgezeichneten und an die Bodenstationen übermittelten Szenen müssen vor ihrer Nutzung aufbereitet werden. Dadurch wird im Allgemeinen die Bildqualität verbessert, da externe Störfaktoren und eventuelle interne Fehlfunktionen ausgeglichen werden können. Es wird zwischen radiometrischen und die geometrischen Aufbereitungen unterschieden. Bei der radiometrischen Aufarbeitung werden digitale Werte wie zum Beispiel die Helligkeit der Szene angepasst.

Im Rahmen der geometrischen Aufbereitung sollen die Folgen einer eventuellen Fehlpositionierung des Satelliten korrigiert werden. Um die Szenen sinnvoll analysieren zu können, müssen sie korrekt und genau positioniert sein. Dies gilt insbesondere bei der Analyse einer Serie von Szenen derselben geographischen Region. Um eine normierte Positionierung der Szenen zu schaffen, werden aus jeder Szene, die einen Teil der zu analysierenden geographischen Region beinhaltet, quadratische Teile der Originalszene ausgeschnitten. Diese ausgeschnittenen Bereiche der ursprünglichen Szene werden Kacheln genannt. Dann wird jeder Pixel der Kachel auf die Zugehörigkeit zum Zielgebiet geprüft. Wenn ein Pixel relevant ist, wird er anhand seiner, aus der Position des Satelliten zum Aufnahmezeitpunkt ermittelten, Position in einem finalen Bild hinzugefügt. Dies garantiert eine einheitliche Grundlage für die Analyse der Zielregion.

Zusätzlich zu Fehlpositionierungen des Satelliten und den sich daraus ergebenden Abweichungen müssen möglicherweise noch weitere Störfaktoren durch die Aufbereitung gemindert werden. Hierzu zählen zum Beispiel durch die Atmosphäre verursachte Verschlechterungen, die aus Interferenzen innerhalb der Atmosphäre zwischen Erdoberfläche und dem Satelliten resultieren. Diese sollen korrigiert werden, um ein genaueres Satellitenbild zu erhalten. Techniken um diese Verbesserung zu erreichen sind beispielsweise das Strahlungstransfermodell, die bildbasierte atmosphärische Korrektur und die Histogramm-Minimum-Methode. Es ist individuell von der Szene und den zur

Verfügung stehenden Metadaten abhängig, mit welcher Methode die nützlichste Verbesserung erreicht werden kann.

Die Aufbereitung von Satellitenbildern muss vor einer wissenschaftlichen Analyse erfolgen, damit die Szenen unabhängig von Witterungseinflüssen, Atmosphäreninterferenzen, Fehlpositionierungen und sonstiger Störfaktoren untersucht werden können. Durch die zunehmend bessere Qualität von Satellitenbildern, die durch Fernerkundungsatelliten aufgezeichnet werden [MSWI04], können detailliertere Analysen getätigt werden. Jedoch steigt mit zunehmender Größe der Bilddateien auch der Rechenaufwand, um die Szenen aufzubereiten und zu analysieren. Mit zunehmender Datenmenge wird eine massiv parallelisierbare Vorgehensweise bei der Aufbereitung und der Analyse von Satellitenbildern attraktiver. Denn verteilte Systeme lassen sich meist kostengünstiger und flexibler erweitern als einzelne Maschinen, so dass das System bei einer unerwartet großen Datenmenge schnell erweitert werden kann. Dadurch lässt sich eine schnellere Ausführung der Prozesse erreichen.

2.2 Parallele Datenverarbeitungssysteme

Seit mehreren Jahren ist ein massiver Anstieg der global produzierten Datenmengen zu beobachten [EMC14]. Diese Menge an Daten ist mithilfe traditioneller Methoden der sequentiellen, stapelweisen Datenverarbeitung nicht effizient zu verarbeiten. Aus diesem Grund wird eine verteilte Verarbeitung von Daten in vielen Bereichen zunehmend populär. Dies gilt insbesondere für Daten, die gemäß der in Sektion 2.2.1 beschriebenen Kriterien als Big Data klassifiziert werden. Um eine parallele Verarbeitung von Big Data zu ermöglichen, wurden bestehende parallele Datenverarbeitungsmechanismen erweitert. Insbesondere das Map-Reduce Paradigma [DG04] bewirkte eine grundlegende Veränderung bei der Vorgehensweise zur Verarbeitung großer Datenmengen. In der Folge wurde Map-Reduce erweitert und flexibler einsetzbar. Diese Entwicklung wird in Sektion 2.2.2 erläutert. Eines der Systeme auf Basis von Map-Reduce ist Apache Flink [Fouc]. Es ermöglicht eine massiv parallelisierbare und echtzeitnahe Verarbeitung von großen Datenmengen. Die konzeptionelle Struktur von Apache Flink wird in der Sektion 2.2.3 beschrieben. Als Alternative zu parallelen Ansätzen existiert die bisherigen sequentiellen bzw. händisch parallelisierbaren Ansatz. Dieser wird am Beispiel der Programmiersprache Python in Sektion 2.2.4 kurz beschrieben.

2.2.1 Bedeutung und Eigenschaften von Big Data

Für das Jahr 2020 wird in der Folge der weltweit zunehmenden Generierung von Daten ein weltweites Datenaufkommen von 44 Zettabyte prognostiziert [EMC14]. Zusätzlich zu dieser schnell wachsenden Menge an verfügbaren Daten wächst auch der Bedarf diese nutzbringend zu analysieren. Insbesondere Forschungseinrichtungen und Unternehmen verfügen über immer größere Datenmengen und versuchen Erkenntnisse aus diesen zu gewinnen. Ein weiterer Teil dieser Daten wird durch die zunehmende Verbreitung des Internets der Dinge und die zunehmende Nutzung von Internetdiensten durch Konsumenten generiert. Traditionelle Methoden der Datenanalyse reichen jedoch nicht mehr aus, um diese Daten auszuwerten.

Dies resultiert aus den vier Eigenschaften, durch die Big Data definiert werden. Insbesondere die drei Charakteristika Volumen (engl. *volume*), Komplexität (engl. *variety*) sowie die echtzeitnahe Verfügbarkeit und schnelle Verarbeitung (engl. *velocity*) von Daten, die bereits 2001 von Dick Laney [Lan01] beschrieben wurden, erschweren die Verarbeitung mithilfe traditioneller Datenverarbeitungsmethoden. So können beispielsweise relationale Datenbanken unstrukturierte Daten nicht selbstständig kategorisieren bzw. strukturieren, da sie lediglich für die Verarbeitung von bereits strukturierten Daten konzipiert wurden. Hinzu kommt die nicht garantierte Zuverlässigkeit und

Einheitlichkeit der Daten (engl. *veracity*) [ZdP+12], die eine Strukturierung der Daten nach festen Mustern erschweren können. Im folgenden werden die vier Eigenschaften kurz erläutert.

Volume. Im Rahmen des generellen Anstiegs von zu verarbeitenden Datenmengen müssen Datenverarbeitungssysteme zunehmend mit großen Datenmengen umgehen. Dadurch entstehen neue Anforderungen bei der Speicherung und Verarbeitung der Daten. Zunehmend sind dabei einzelne, große Datenmengen von Bedeutung. Beispiele dafür sind unter anderem das Sloan Digital Sky Survey, das seit 1998 insgesamt 116 Terabyte an astronomischen Daten gesammelt hat [YAJEA+00, AAP+15] und das 1000 Genomes Project [Bak10], das zwischen 2008 und 2013 insgesamt 464 Terabyte Daten zum menschlichen Genom sammelte. Weitere Beispiele sind das CERN, dessen Large Hadron Collider täglich circa 1 Petabyte Daten produziert, und Google, das bereits im Jahr 2008 rund 20 Petabyte Daten pro Tag verarbeitete [DG08]. Evtl. ein oder zwei Beispiele weniger nehmen. Streichkandidaten: SDSS, 1000G.

Variety. Gesammelte Daten weisen vielfältige Datenstrukturen auf. Es werden nicht-strukturierte, semistrukturierte sowie strukturierte Daten gesammelt. Außerdem ist eine vorliegende Datenstruktur aufgrund von Inkompatibilität mit anderen Datenstrukturen möglicherweise schwierig in Bezug zu anderen Daten zu bringen. Ein Grund dafür ist der massive Anstieg an unterschiedlichen Datenquellen, deren erhobenen Daten nicht immer aufeinander abgestimmt sind. Daraus können sich Herausforderungen bei der Normierung von Daten ergeben. Denn Daten müssen teilweise selbstständig kategorisiert werden, oder komplett unstrukturiert gespeichert werden.

Velocity. Anwendungsfälle, die eine echtzeitnahe Verarbeitung von großen Datenmengen fordern, werden immer zahlreicher. Diese Verarbeitungsgeschwindigkeit ist aber nur umzusetzen, wenn die Datenverarbeitungssysteme mithilfe parallelisierter Architekturen auf eben solche ausgelegt sind, da die Daten teilweise sehr schnell verfügbar sein müssen. Ein Beispiel sind autonom steuernde Fahrzeuge, die nur bei sofortiger und schneller Auswertung von Sensordaten angemessen auf durch Sensoren ermittelte Hindernisse reagieren können. Hinzu kommen Anwendungsszenarien, bei denen zusätzlich ein hoher Datendurchsatz erforderlich ist.

Veracity. Gesammelten Daten sind weder garantiert korrekt noch garantiert komplett. Durch falsche Modellannahmen oder hohe Latenzen einiger Datenquellen kann es zu weiteren Unsicherheiten bezüglich der Validität der Daten kommen. Big Data sind folglich immer möglicherweise fehlerbehaftet. Analysesysteme müssen auf diesen Umstand insofern reagieren, dass nicht valide Daten automatisiert erkannt und aus der Analyse ausgenommen werden. Beispiel Aufgrund dieser Eigenschaften mussten in den letzten Jahren immer wieder neue Konzepte und Systeme entwickelt werden, um Big Data verarbeiten zu können.

2.2.2 Systeme zur massiv parallelen Datenverarbeitung

In Anbetracht des steigenden Bedarfs an Techniken, mit deren Hilfe Big Data verarbeitet werden können, wurden die Entwicklung neuer und die Weiterentwicklung bestehender Technologien und Konzepte im Bereich Big Data innerhalb der letzten Jahre vorangetrieben. Dazu zählen insbesondere massiv parallelisierbare Rechnerstrukturen in Verbindung mit neuartigen Datenverarbeitungssystemen, die diese Konzepte und Technologien verwenden um Big Data verarbeiten zu können.

Die Entwicklung paralleler Rechnerstrukturen begann in den 1970er Jahren im Rahmen der Konstruktion von Computern mit mehreren kleinen Prozessoren (*Computer with multiple miniprocessors*) [Bel71, WB72]. Die Entwicklung nutzbarer parallel arbeitender Computer begann in den 1980er Jahren [Sei85]. Hinzu kam die Konzeption und Umsetzung parallelisierter Algorithmen [BH85]. Seitdem schritt die Weiterentwicklung parallelisierter Architekturen und Konzepte mit steigendem Tempo fort [TW12]. Während früher einzelne Maschinen mit parallel geschalteten Komponenten zur Bearbeitung aufwändiger Datenverarbeitungsaufgaben eingesetzt wurden, werden aktuell vermehrt Computercluster eingesetzt. Diese bestehen aus mehreren Maschinen, die

mithilfe eines losen Netzwerks verbunden sind und so einen virtuellen Supercomputer darstellen [HDF13]. Aufgrund der Beschaffenheit der Computercluster lässt sich die Anzahl an zusammengeschlossenen Maschinen flexibel definieren. Auf diese Weise kann die Rechenleistung eines solchen Netzwerks kontinuierlich an die Anforderungen angepasst werden. Dies schafft optimale Voraussetzungen für die Verarbeitung von Big Data, da parallelisierte Strukturen einfacher erweitert werden können. So lässt sich die Kapazität des Clusters an den Umfang der zu analysierenden Daten anpassen. Einschränkend müssen aber auch die Grenzen von parallelisierten Verarbeitungsstrukturen berücksichtigt werden. Laut Amdahls Gesetz kann die Beschleunigung der Ausführungsgeschwindigkeit von Programmen durch eine parallele Ausführung maximal linear zur Anzahl der Prozessorkerne ansteigen. Dies resultiert aus Programmteilen, die zwangsweise sequentiell durchgeführt werden müssen. Jedes Programm besitzt solche Programmteile, etwa Speicherallokationen oder der sequentielle bzw. synchrone Zugriff von Threads auf geteilte Ressourcen. Gustafsons Gesetz notwendig?

Für eine effiziente Nutzung physischer paralleler Strukturen müssen parallelisierbare Algorithmen verwendet werden. Ein prägendes Konzept, das die Implementierung solcher Algorithmen ermöglicht, ist das 2004 veröffentlichte Map-Reduce System [DG04]. Inspiriert durch ein ähnliches Konzept aus der funktionalen Programmierung ermöglicht es die nebenläufige Berechnung von großen Datenmengen. Darüber hinaus bietet es eine selbstständige Korrektur von bei Ausfällen von Netzwerkknoten verlorenen Daten. Dabei nutzt es ein verteiltes Dateisystem, wie beispielsweise das Google File System [GGL03], das auf einem Computercluster ausgeführt wird.

Bei jedem Map-Reduce Programm müssen die zwei Funktionen zweiter Ordnung *map* und *reduce* vom User als *User-defined function* implementiert werden. Das Map-Reduce System parallelisiert dann die UDFs map und reduce auf Teilmengen der zu verarbeitende Datenmenge. Wichtig ist dabei die zu berücksichtigende Struktur des Programms, die genau eine map-Funktion sowie genau eine auf die map-Funktion folgende reduce-Funktion voraussetzt.

Bei der Ausführung eines Map-Reduce Programms werden die Netzwerkknoten des genutzten Clusters gemäß einer Master-Worker Architektur genutzt. Ein Master-Knoten weist den unterschiedlichen Worker-Knoten map- bzw. reduce-Aufgaben zu. Zusätzlich erhält der Worker-Knoten eine Referenz auf den Speicherort der zur verarbeitenden Daten. Des weiteren koordiniert der Master-Knoten die Worker und prüft sie in regelmäßigen Abständen auf korrekte Funktionalität. Sollte einer der Worker-Knoten nicht mehr funktionsfähig sein, wird er aus dem restlichen Verarbeitungsprozess ausgeschlossen. Die Aufgaben, die der Knoten bereits erfüllt hat werden als unerledigt gekennzeichnet und von anderen Workern nochmals ausgeführt.

Wie in Abbildung 2.1 abgebildet, erhält jeder Worker, der eine Map-Aufgabe ausführt, eine Teilmenge der zu verarbeitenden Gesamtdatenmenge als Eingabe. Die Daten müssen dazu in Form von Schlüssel-Wert Paaren gespeichert sein. Diese werden dann gemäß der vom Nutzer spezifizierten UDF verarbeitet. Als Ausgabe werden keine, ein oder mehrere Schlüssel-Wert Paare durch die map-Funktion produziert und im lokalen Speicher der jeweiligen Worker gespeichert. Aufgrund der Aufteilung der Gesamtdatenmenge in kleinere, von einander unabhängig prozessierbare Mengen, kann die Map-Phase parallelisiert ablaufen. Dabei werden auf jedem Netzwerkknoten des Clusters verschiedene Teilmengen der Eingabedaten verarbeitet.

Nach Beendigung der Map-Phase folgt die Shuffle-Phase. Während dieser werden die durch die Map-Funktion produzierten Schlüssel-Wert Paare gemäß ihrer Schlüssel gruppiert. Wenn der Speicherbedarf für alle Schlüssel-Wert Paare einer Gruppe größer ist als der verfügbare Speicher auf einem Worker werden mehrere Gruppen mit demselben Schlüsselwert gebildet. Jede Gruppe wird anschließend auf verfügbare Worker übertragen, um die anschließende Anwendung der reduce-Funktion zu ermöglichen.

Die reduce-Funktion wird auf jede in der Shuffle-Phase erzeugten Gruppe von Schlüssel-Wert Paaren angewendet. Die spezifizierte UDF wird auf die Werte aller Schlüssel-Wert Paare einer Gruppe angewendet. Als Ausgabe wird entweder keine, ein oder mehrere Schlüssel-Wert Paare

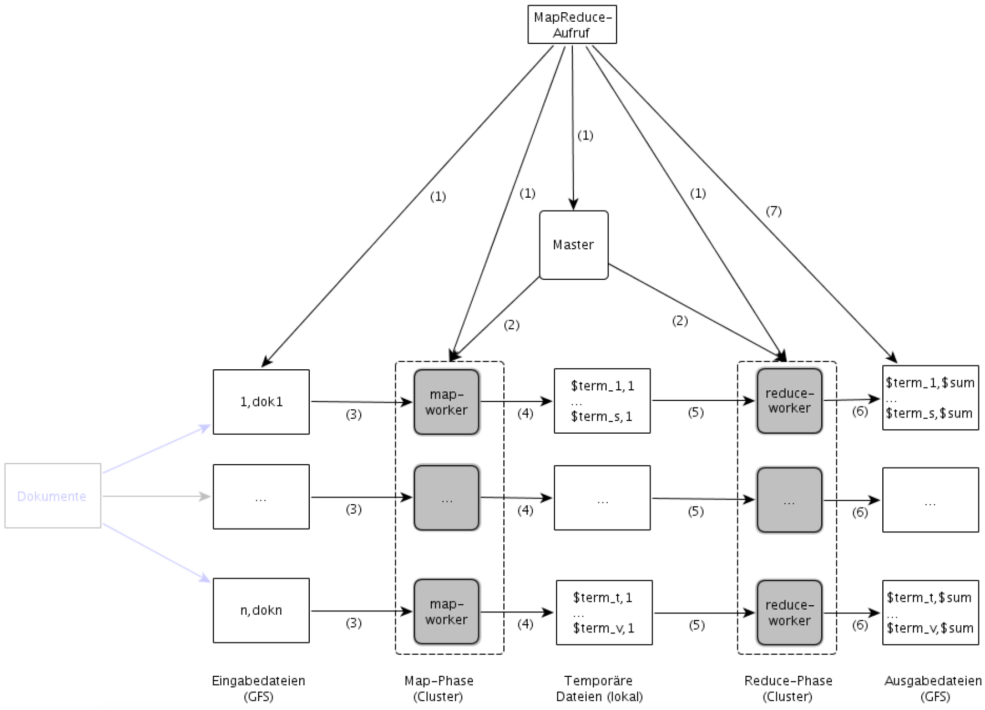


Abbildung 2.1: Systemübersicht eines Map-Reduce System [BR09]

produziert.

Die Gesamtausgabe des Map-Reduce Programms besteht aus allen durch reduce-Funktionen produzierten Schlüssel-Wert Paaren.

Aufgrund der sehr genau festgelegten Programmstruktur, die zur Nutzung von Map-Reduce eingehalten werden muss wird der Anwender gezwungen einen Datenverarbeitungsprozess auf mehrere aufeinanderfolgende Map-Reduce Instanzen zu verteilen, wenn dieser zu komplex ist.

Seit seiner Einführung im Jahr 2004 wurde das Map-Reduce Paradigma erweitert und bietet nun mehr Flexibilität, da es um weitere Funktionen zweiter Ordnung ergänzt wurde, die zusätzliche Datentransformationen ermöglichen. Aufgrund des Einsatzes von Map-Reduce in Hadoop [Foue] und anderen Systemen wie beispielsweise Spark [ZCF+10] bleibt es aber eines der dominanten Paradigmen bei der Verarbeitung großer Datenmengen. Auch Apache Flink [Fouc] nutzt eine ähnliche Funktionsweise, um die parallelisierten Datenströme zu verwalten.

Neben den bereits erwähnten Systemen Apache Hadoop und Apache Flink existieren weitere populäre Systeme, beispielsweise Apache Spark [Foua, ZCF+10], Apache Storm [Foub, Jon13] oder Asterix [Ast, AKL+12]. Diese unterscheiden sich bereits in ihrer Architektur. Während Apache Hadoop das Map-Reduce Paradigma sehr fixiert umsetzt und somit dem Nutzer nur eingeschränkte Mittel zur Spezifizierung seines Programmes lässt, setzen Systeme wie Apache Spark und Apache Flink das Map-Reduce Paradigma flexibler ein. Nachfolgend wird ein kurzer Überblick über die Funktionsweise der genannten Systeme gegeben.

Apache Hadoop. Apache Hadoop implementiert das Map-Reduce System im Rahmen eines freien Projekts. Dabei können sowohl die Map- als auch die Shuffle- und Reducephase vom Nutzer individuell konfiguriert werden. Als freier Ersatz für GFS nutzt Hadoop das Dateisystem Hadoop Distributed File System (HDFS). In der Basisversion ermöglicht Hadoop lediglich Stapelverarbei-

tungen, echtzeitnahe Datenverarbeitung ist nicht vorgesehen.

Apache Spark. Apache Spark ist eine allgemeine Datenverarbeitungsplattform. Es beinhaltet mehr Funktionen zweiter Ordnung als Hadoop, es gibt also mehr Möglichkeiten verschiedene Datentransformationen auszuführen. Im Gegensatz zu Hadoop ermöglicht es neben der Stapelverarbeitung weitere Anwendungen wie zum Beispiel maschinelles Lernen. Des weiteren ist ebenfalls möglich Spark mit anderen Bibliotheken zu ergänzen. Diese Eigenschaften machen Spark gegenüber Hadoop für viele Anwendungszecke überlegen.

Apache Storm. Apache Storm ist ein Datenverarbeitungssystem, das speziell auf die schnelle echtzeitnahe Verarbeitung von Daten ausgelegt ist. Ein Storm-Programm hat die Form eines direkten, azyklischen Graphs, dessen Kanten Datenströme darstellen. Ein Storm-Programm kann als datentransformierende Pipeline bezeichnet werden. Ein Storm-Programm entspricht einer Implementierung eines abstrakten Map-Reduce-Jobs. Im Unterschied zu einem Map-Reduce Job läuft ein Storm-Programm, bis es beendet wird. Dies unterstreicht den Einsatzzweck für echtzeitnahe Verarbeitung von Daten, die ohne Zeitbegrenzung generiert werden.

Asterix.

2.2.3 Apache Flink

Ein Datenverarbeitungssystem, das auf eine massiv parallelisierte Verarbeitung von großen Datenmengen spezialisiert ist, ist Apache Flink. Es ging 2014 aus Stratosphere hervor [Bra], einem Forschungsprojekt, das seit 2010 kooperativ von Forschern verschiedener Universitäten entwickelt wurde [BEH+10, ABE+14]. Seit Januar 2015 ist Apache Flink ein Top-Level Projekt der Apache Software Foundation [Fou15]. Ähnlich wie andere Systeme ermöglicht es Apache Flink bereits vorhandene Daten in einem *Batch*-Verfahren zu analysieren. Darüber hinaus können jedoch auch in Echtzeit zu verarbeitende Daten im Rahmen eines *Streaming*-Verfahrens prozessiert werden.

Die Hauptkomponenten von Apache Flink sind die Flink-Laufzeitumgebung (engl. *Flink Runtime*) und der Flink-Optimierer (engl. *Flink Optimizer*). Darüber hinaus verfügt Flink über Programmierschnittstellen für die Hochsprachen Java, Python und Scala, mithilfe derer Nutzer Flink-Programme spezifizieren können.

Flink implementiert das vom Map-Reduce System bekannte Schema von Funktionen zweiter Ordnung und gekapselten nutzerdefinierten Funktionen erster Ordnung. Dabei beschreibt die Funktion zweiter Ordnung die Art der Datentransformation, die UDF definiert die vom Nutzer spezifizierte Verarbeitungslogik, die auf die Daten angewendet werden soll. Im Gegensatz zum Map-Reduce System verfügt Flink jedoch nicht nur über die Funktionen zweiter Ordnung Map und Reduce, sondern bietet insgesamt 15 verschiedene Datentransformationen. Diese werden als Operatoren bezeichnet. Weitere wichtige Operatoren neben Map und Reduce sind beispielsweise Cross, Match und CoGroup, die in Abbildung 2.2 dargestellt sind. Jeder Operator besteht also aus jeweils einer spezifischen Transformation sowie einer eingebetteten UDF. Zusätzlich besitzen Operatoren eine Eingabe- sowie eine Ausgabedatenmenge in Form eines Datenstroms. Ein Flink-Programm wird als ein Datenfluss in Form eines direkten und azyklischen Graphen dargestellt, dessen Knoten Operatoren und dessen Kanten Datenströme darstellen. Ref auf Abbildung eines Beispieldatenstroms.. Dieser wird *Operator DAG* genannt.

Der Flink-Optimierer erhält den mithilfe einer Programmierschnittstelle spezifizierten Operator DAG als Eingabe. Die Aufgabe des Flink-Optimierers ist die Umwandlung des Operator DAG in einen parallelisiert ausführbaren Datenfluss, den *OptimizedPlan*. Der OptimizedPlan soll möglichst stark parallelisierbar sein, damit eine effiziente Verarbeitung des OptimizedPlans durch die Flink-Laufzeitumgebung ermöglicht werden kann. Um einen OptimizedPlan zu erhalten werden abhängig von den verwendeten Operatoren programmspezifische Optimierungen vorgenommen. Dabei wird für jeden Operator auf Basis seiner Eingabedaten festgelegt, mit welcher internen Implementierung der Operator ausgeführt wird. Ein Beispiel dafür wäre die Ausführung eines Join-Operators

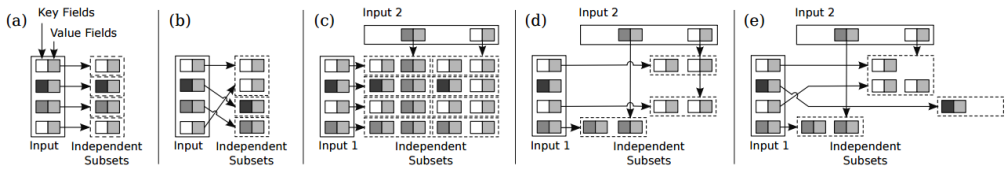


Abbildung 2.2: Funktionen zweiter Ordnung: (a) Map, (b) Reduce, (c) Cross, (d) Match, and (e) CoGroup [HPS+12]

wahlweise als Sort-Merge-Join oder als Hash-Join. Weitere Optimierungen umfassen die Wahl des Datenaustauschs zwischen den Operatoren sowie die Auswahl der Programmpunkte, an denen Zwischenergebnisse gespeichert werden. Außerdem wird versucht Daten, die durch einen Operator sortiert oder partioniert wurden, mit derselben Sortierung weiterzuverwenden, um weitere Sortier- oder Partionierungsprozesse zu vermeiden. Ist der optimale OptimizedPlan ermittelt wird er zu einem *JobGraph*, einem direkten azyklischen Graphen bestehend aus Operatoren und Zwischenergebnissen als Knoten, umgewandelt. Ein Bespiel für einen JobGraph wird in Abbildung 2.3 gezeigt. Dieser wird dann von der Flink-Runtime verarbeitet.

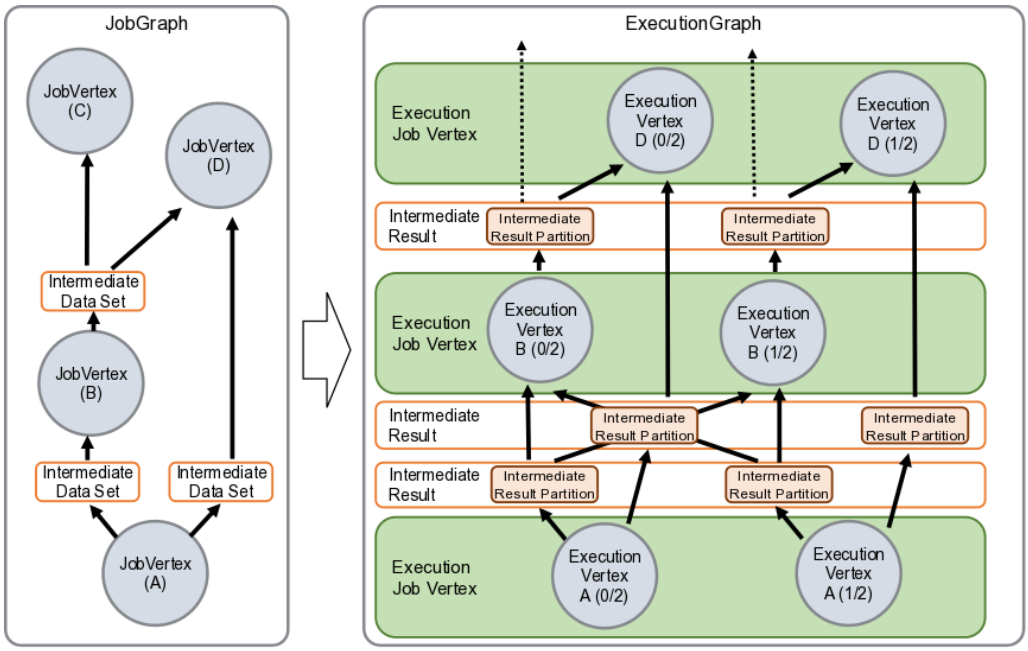


Abbildung 2.3: Modell eines Jobgraphs und des dazugehörigen ExecutionGraphs

Flink nutzt ebenso wie das Map-Reduce System eine Master-Worker Architektur zur Organisation des Netzwerkes, auf dem das Flink-Programm ausgeführt wird. Abbildung 2.4 zeigt eine Übersicht der Systemstruktur von Flink. Der Master-Knoten, auch *Jobmanager* genannt, verwaltet dabei eine variable Anzahl an Worker-Knoten, *TaskManager* genannt. Gemäß der Master-Worker Architektur weist der JobManager den einzelnen TaskManagern Programmteile zu, die diese dann ausführen. Der Status der Ausführung wird vom JobManager überwacht. Fällt ein TaskManager aus werden dessen Aufgaben von anderen Worker-Knoten übernommen bzw. wiederholt. Eine Besonderheit des Flink-Systems ist der Datenaustausch zwischen Workern in Form von Datenströmen

untereinander.

Wird ein Flink-Programm ausgeführt, wird nachdem die Optimierung beendet ist das System gemäß nutzerdefinierter Prämissen wie dem Grad der Parallelität initialisiert. Der JobManager erhält den vom Flink-Optimizer generierten JobGraph als Eingabe. Dieser wird, wie in Abbildung 2.3 dargestellt, in einen parallel strukturierten *ExecutionGraph* transformiert. Für jeden Knoten des Jobgraphs wird ein *ExecutionJobVertex* erstellt. Dieser verantwortet die Ausführung des durch den Knoten repräsentierten Operators sowie die Weitergabe der, aus der Anwendung des Operators auf die Eingabedaten resultierenden, Zwischenergebnisse. Dabei werden n *ExecutionVertices* genutzt, wobei n dem vom Nutzer festgelegten Grad der Parallelität entspricht. Jeder ExecutionVertex übernimmt die Ausführung einer der parallelisierten Verarbeitungsinstanzen des Operators. Dazu wird ein vom JobManager bestimmter Worker-Knoten genutzt. Der ExecutionJobVertex garantiert die komplette Ausführung des Operators auf der gesamten Eingabedatenmenge, indem der Status jedes ExecutionVertex verfolgt wird. Sollte einer der ExecutionVertices ausfallen wird die gerade verarbeitete Ausführungsinstanz an andere ExecutionVertices delegiert. Darüber hinaus beinhaltet der ExecutionGraph die Zwischenergebnisse, die zwischen der Anwendung verschiedener Operatoren erstellt werden. Dadurch ist jederzeit bekannt, ob die Bedingungen für die Ausführung eines weiteren Operators gegeben ist, oder ob noch nicht alle Eingabedaten in Form von Zwischenergebnissen verfügbar sind.

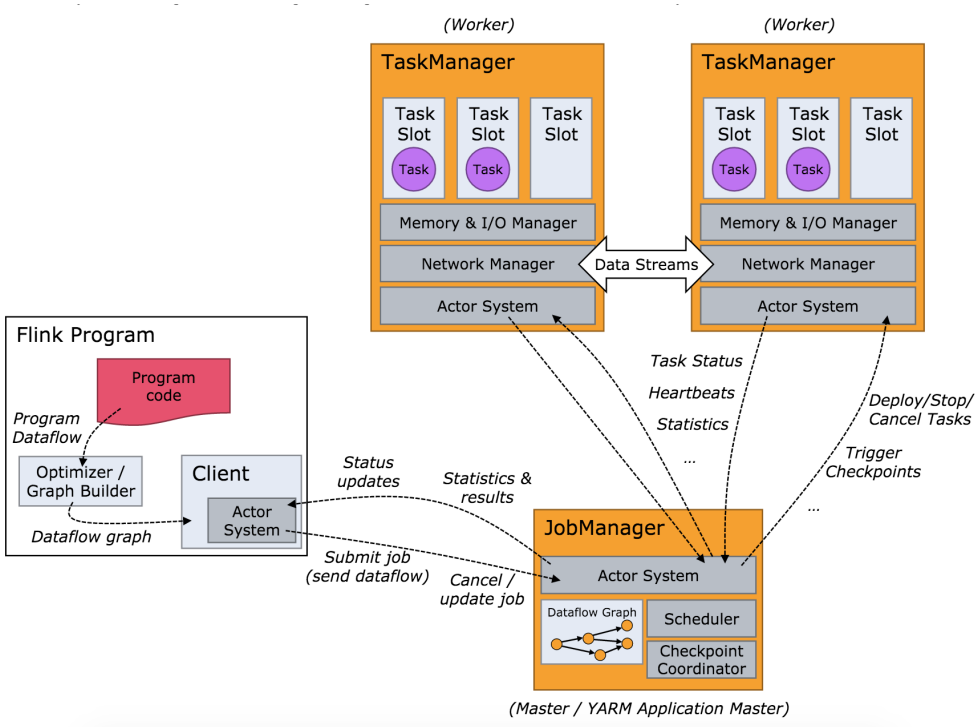


Abbildung 2.4: Systemübersicht bei der Ausführung eines Flink-Programms [Foud]

2.2.4 Python

Python ist eine quelloffene und universell einsetzbare Programmiersprache, die seit 1989 existiert und fortwährend weiter entwickelt wird. Prägende Eigenschaften der Sprache sind unter anderem

eine dynamische Typisierung von Variablen, eine simpel gehaltene Syntax und die Erweiterbarkeit durch Module und Bibliotheken. Es ist auch möglich Python-Code durch C- beziehungsweise C++-Bibliotheken zu erweitern [Mar06]. Dies ermöglicht eine verkürzte Ausführungszeit eines Programms, insbesondere bei rechenintensiven Programmabschnitten. Ein Schwachpunkt von Python im Bezug auf die schnelle Verarbeitung großer Datenmengen ist die nicht auf automatisierte Parallelisierung ausgelegte Architektur. Daraus resultiert eine unzureichende Skalierbarkeit, sobald Daten, deren Größe die Arbeitsspeichergröße der ausführenden Maschine übersteigt, verarbeitet werden müssen. Dies ist insbesondere im Zuge der oben aufgezeigten Entwicklung hin zu größeren zu verarbeitenden Datenmengen relevant. Entwicklungen, die eine bessere Parallelisierung und damit einhergehend eine bessere Skalierbarkeit von Python Projekten zum Ziel haben, sind jedoch meist nicht universal anwendbar. Darüber hinaus wird die Funktionalität dieser Ansätze durch Module bereitgestellt und existiert somit nicht als Grundfunktion in Python sondern muss projektweise hinzugefügt werden. Dies erhöht den zu leistenden Anpassungs- und Entwicklungsaufwand im Vergleich zu anderen Systemen, die mit Rücksicht auf diese Anwendungsfälle konzipiert wurden.

Kapitel 3

Beschreibung und Umsetzung des Algorithmus zur Analyse von Pixelzeitreihen

3.1 Beschreibung des Algorithmus zur Analyse von Pixelzeitreihen

Beschreibung der Vorgehensweise bei der Analyse (Zhu, SVR), Ziel der Analyse, Entwicklungsgeschichte der Analysetechnik

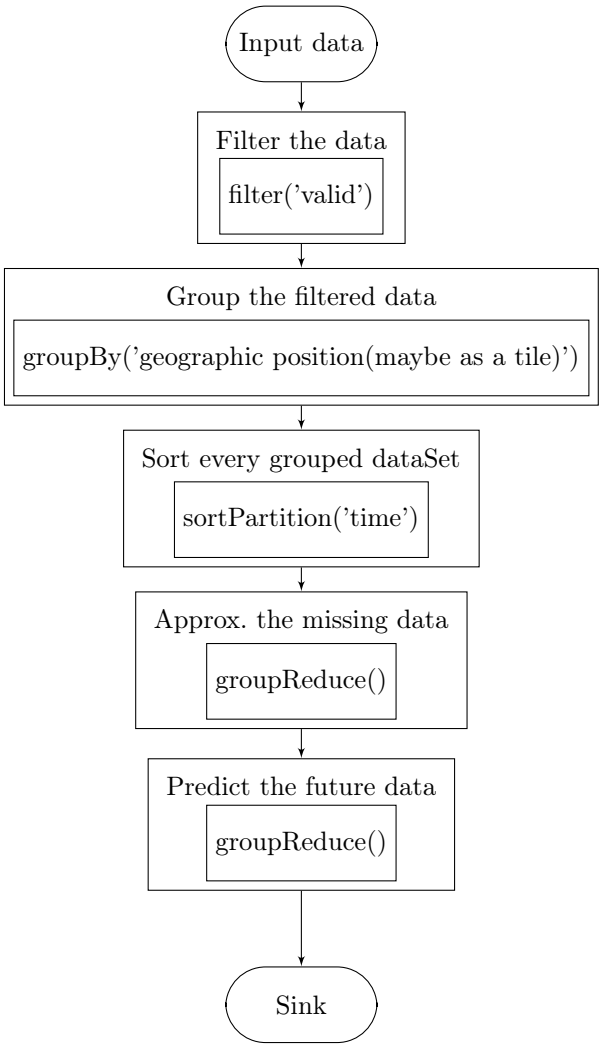
3.2 Umsetzung des Algorithmus mit Apache Flink

3.2.1 Nutzung der Java-Programmierschnittstelle

3.2.2 Nutzung der Python-Programmierschnittstelle

3.3 Umsetzung des Algorithmus in Python

Die Analyse von Satellitenbildern erfordert die Verarbeitung großer Mengen komplexer Rohdaten, die nahezu in Echtzeit verfügbar sind. Aufgrund dieser Charakteristika handelt es sich bei dieser Analyse um ein Big-Data Problem. Denn alle vier Kriterien, die ein solches charakterisieren sind erfüllt.
Bei der Analyse von Satellitenbildern sind die Merkmale Datengröße und Datenkomplexität sowie die schnelle Verarbeitung der Daten von Bedeutung. Abhängig von der Anzahl der genutzten Bilder sind die zu verarbeitenden Datenmengen sehr groß. Ein Bild besitzt im Regelfall abhängig vom Satellitenmodell, das die Aufnahme gemacht hat, eine Größe von 750 Megabyte bis zu 1,5 Gigabyte. Um eine Entwicklung zu untersuchen werden jedoch viele dieser Bilder in die Untersuchung mit einbezogen, so dass die zu verarbeitende Datenmenge kontinuierlich ansteigt. Dieser kontinuierliche Anstieg entsteht dadurch, dass aktuell mehrere Satelliten mit der Fernerkundung der Erde fortfahren und so in kurzen Intervallen neue Bilder zur Verfügung stehen, die im Rahmen der Analyse verwendet werden sollen. Quelle.


Input data
Filter the data
filter('valid')
Group the filtered data
groupBy('geographic position(maybe as a tile)')
Sort every grouped dataSet
sortPartition('time')
Approx. the missing data
groupReduce()
Predict the future data
groupReduce()
Sink

Kapitel 4

Evaluierung

4.1 Versuchsbeschreibung

Beschreibung + Begründung für meine Versuchsbedingungen

4.2 Auswertung

Beschreibung und Bewertung der Ergebnisse meiner Untersuchungen

Kapitel 5

Fazit

Fazit und Ausblick z.b. Vergleich mit anderen Untersuchungen

Literaturverzeichnis

[AAP+15] Shadab Alam, Franco D. Albareti, Carlos Allende Prieto, F. Anders, Scott F. Anderson, Timothy Anderton, Brett H. Andrews, Eric Armengaud, Ãric Aubourg, Stephen Bailey, and et al. The eleventh AND twelfth data releases of the sloan digital sky survey: Final data from sdss-iii. *The Astrophysical Journal Supplement Series*, 219(1):12, Jul 2015.

[ABE+14] Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, and et al. The stratosphere platform for big data analytics. *The VLDB Journal*, 23(6):939,964, May 2014.

[AKL+12] Sattam Alsubaiee, Young-Seok Kim, Chen Li, Nicola Onose, Pouria Pirzadeh, Rares Vernica, Jian Wen, Yasser Altowim, Hotham Altwaijry, Alexander Behm, and et al. Asterix. *Proceedings of the VLDB Endowment*, 5(12):1898,1901, Aug 2012.

[Ast] Asterixdb website. `https://asterixdb.ics.uci.edu/`.

[Bak10] Monya Baker. Next-generation sequencing: adjusting to data overload. *Nat Meth*, 7(7):495,Äì499, Jul 2010.

[BEH+10] Dominic Battré, Stephan Ewen, Fabian Hueske, Odej Kao, Volker Markl, and Daniel Warneke. Nephele/pacts: a programming model and execution framework for web-scale analytical processing. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 119–130. ACM, 2010.

[Bel71] C Gordon Bell. C. mmp: the cmu multiminiprocessor computer, requirements and overview of the initial design. 1971.

[BH85] A. Borodin and J.E. Hopcroft. Routing, merging, and sorting on parallel models of computation. *Journal of Computer and System Sciences*, 30(1):130,Äì145, Feb 1985.

[BR09] Stefan Bethge and Astrid Rheinländer. Mapreduce. Seminararbeit, Humboldt-Universität zu Berlin, 2009.

[Bra] Mikio Braun. The future of big data (according to stratosphere/flink). `http://blog.mikiobraun.de/2014/06/future-big-data-flink-stratosphere.html`.

[Cha11] Anju Chaudhary. Thermal infrared sensors. *Encyclopedia of Snow, Ice and Glaciers*, page 1156, 2011.

[Deu12] Deutsches Institut für Normung e.V. *Photogrammetrie und Fernerkundung - Begriffe*, 8 2012. Rev. 3.

[DG04] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. page 13, 2004.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce. *Communications of the ACM*, 51(1):107, Jan 2008.

[EMC14] EMC². The digital universe of opportunities. Technical report, EMC², 2014.

[Foua] Apache Software Foundation. Apache spark website. `https://spark.apache.org/`.

[Foub] Apache Software Foundation. Apache storm website. `https://storm.apache.org/`.

[Fouc] Apache Software Foundation. Flink website. `https://flink.apache.org/`.

[Foud] Apache Software Foundation. Flink website - general architecture and process model. `https://ci.apache.org/projects/flink/flink-docs-master/internals/general_arch.html`.

[Foue] Apache Software Foundation. Hadoop website. `https://hadoop.apache.org/`.

[Fou15] Apache Software Foundation. The apache software foundation announces apache™ flink™ as a top-level project. `https://blogs.apache.org/foundation/entry/the\_apache\_software\_foundation\_announces69`, January 2015.

[GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29, Dec 2003.

[GP] GfZ-Potsdam. Geomultisens website. `http://www.geomultisens.gfz-potsdam.de/`.

[HDF13] Kai Hwang, Jack Dongarra, and Geoffrey C Fox. *Distributed and cloud computing: from parallel processing to the internet of things*. Morgan Kaufmann, 2013.

[HPS+12] Fabian Hueske, Mathias Peters, Matthias J. Sax, Astrid Rheinländer, Rico Bergmann, Aljoscha Krettek, and Kostas Tzoumas. Opening the black boxes in data flow optimization. *Proceedings of the VLDB Endowment*, 5(11):1256,1267, Jul 2012.

[IDB12] James R. Irons, John L. Dwyer, and Julia A. Barsi. The next landsat satellite: The landsat data continuity mission. *Remote Sensing of Environment*, 122:11,21, Jul 2012.

[Jac09] Adam Jacobs. The pathologies of big data. *Communications of the ACM*, 52(8):36, August 2009.

[Jon13] M Tim Jones. Process real-time big data with twitter storm. *IBM Technical Library*, 2013.

[Lan01] Doug Laney. 3d data management: Controlling data volume, velocity and variety. *Application Delivery Strategies published by META Group Inc.*, Feb 2001.

[Mar06] Alex Martelli. *Python in a Nutshell. A Desktop Quick Reference*. O'Reilly, second edition edition, 2006.

[MSWI04] B.L. Markham, J.C. Storey, D.L. Williams, and J.R. Irons. Landsat sensor performance: history and current status. *IEEE Transactions on Geoscience and Remote Sensing*, 42(12):2691,2694, Dec 2004.

[Sei85] Charles L Seitz. The cosmic cube. *Communications of the ACM*, 28(1):22–33, 1985.

[TW12] Arthur Trew and Greg Wilson. *Past, present, parallel: a survey of available parallel computer systems*. Springer Science & Business Media, 2012.

[WB72] William A Wulf and C Gordon Bell. C. mmp: a multi-mini-processor. In *Proceedings of the December 5-7, 1972, fall joint computer conference, part II*, pages 765–777. ACM, 1972.

[YAJEA+00] Donald G. York, J. Adelman, Jr. John E. Anderson, Scott F. Anderson, James Annis, Neta A. Bahcall, J. A. Bakken, Robert Barkhouser, Steven Bastian, Eileen Berman, William N. Boroski, Steve Bracker, Charlie Briegel, John W. Briggs, J. Brinkmann, Robert Brunner, Scott Burles, Larry Carey, Michael A. Carr, Francisco J. Castander, Bing Chen, Patrick L. Colestock, A. J. Connolly, J. H. Crocker, Istvan Csabai, Paul C. Czarapata, John Eric Davis, Mamoru Doi, Tom Dombeck, Daniel Eisenstein, Nancy Ellman, Brian R. Elms, Michael L. Evans, Xiaohui Fan, Glenn R. Federwitz, Larry Fiscelli, Scott Friedman, Joshua A. Frieman, Masataka Fukugita, Bruce Gillespie, James E. Gunn, Vijay K. Gurbani, Ernst de Haas, Merle Haldeman, Frederick H. Harris, J. Hayes, Timothy M. Heckman, G. S. Hennessy, Robert B. Hindsley, Scott Holm, Donald J. Holmgren, Chi hao Huang, Charles Hull, Don Husby, Shin-Ichi Ichikawa, Takashi Ichikawa, Zeljko Ivezifá, Stephen Kent, Rita S. J. Kim, E. Kinney, Mark Klaene, A. N. Kleinman, S. Kleinman, G. R. Knapp, John Korienek, Richard G. Kron, Peter Z. Kunszt, D. Q. Lamb, B. Lee, R. French Leger, Siriluk Limmongkol, Carl Lindenmeyer, Daniel C. Long, Craig Loomis, Jon Loveday, Rich Lucinio, Robert H. Lupton, Bryan MacKinnon, Edward J. Mannery, P. M. Mantsch, Bruce Margon, Peregrine McGehee, Timothy A. McKay, Avery Meiksin, Aronne Merelli, David G. Monet, Jeffrey A. Munn, Vijay K. Narayanan, Thomas Nash, Eric Neilsen, Rich Neswold, Heidi Jo Newberg, R. C. Nichol, Tom Nicinski, Mario Nonino, Norio Okada, Sadanori Okamura, Jeremiah P. Ostriker, Russell Owen, A. George Pauls, John Peoples, R. L. Peterson, Donald Petravick, Jeffrey R. Pier, Adrian Pope, Ruth Pordes, Angela Prosapio, Ron Rechenmacher, Thomas R. Quinn, Gordon T. Richards, Michael W. Richmond, Claudio H. Rivetta, Constance M. Rockosi, Kurt Ruthmansdorfer, Dale Sandford, David J. Schlegel, Donald P. Schneider, Maki Sekiguchi, Gary Sergey, Kazuhiro Shimasaku, Walter A. Siegmund, Stephen Smee, J. Allyn Smith, S. Snedden, R. Stone, Chris Stoughton, Michael A. Strauss, Christopher Stubbs, Mark SubbaRao, Alexander S. Szalay, Istvan Szapudi, Gyula P. Szokoly, Anirudda R. Thakar, Christy Tremonti, Douglas L. Tucker, Alan Uomoto, Dan Vanden Berk, Michael S. Vogeley, Patrick Waddell, Shu i Wang, Masaru Watanabe, David H. Weinberg, Brian Yanny, and Naoki Yasuda. The sloan digital sky survey: Technical summary. *The Astronomical Journal*, 120(3):1579, 2000.

[ZCF+10] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, volume 10, page 10, 2010.

[ZdP+12] Paul Zikopoulos, Dirk deRoos, Krishnan Parasuraman, Thomas Deutsch, James Giles, and David Corrigan. *Harness the Power of Big Data The IBM Big Data Platform*. McGraw-Hill Osborne Media, 2012.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Weiterhin erkläre ich, eine ...arbeit in diesem Studiengebiet erstmalig einzureichen.

Berlin, den 23. Oktober 2015 .....

Statement of authorship

I declare that I completed this thesis on my own and that information which has been directly or indirectly taken from other sources has been noted as such. Neither this nor a similar work has been presented to an examination committee.

Berlin, 23rd October 2015 .....