

Rechnernetze und verteilte Systeme

Praxis 3: Distributed Word Count

Fachgruppe Telekommunikationsnetze (TKN)

Beginn Bearbeitung: 13.01.2025, 00:00 Uhr

Abgabe: **09.02.2025 23:59 Uhr**

Stand: 10. Januar 2025

Formalitäten

! Diese Aufgabenstellung ist Teil der Portfolioprüfung. Beachten Sie für Ihre Abgaben unbedingt die entsprechenden Modalitäten (siehe Anhang A).

In dieser Aufgabe wird ein verteilter Word-Count basierend auf einem Request-Reply Pattern in C (C11 Standard) implementiert. Hierzu wird Zero MQ (ZMQ)¹ genutzt. Im Rahmen der Aufgabe wird ein Distributor implementiert, welcher eine Datei von einem übergebenen Pfad einliest und die Word-Count Aufgabe auf eine variable Anzahl an Worker verteilt. Um die Wörter im Text zu zählen, implementiert jeder Worker die Map-Reduce Funktion. Nachdem die ganze Datei von den Workern verarbeitet wurde, gibt der Distributor alle gezählten Wörter mit absteigender Frequenz auf der Konsole aus. Falls Wörter die gleiche Frequenz haben, sollen sie alphabetisch sortiert werden.

1. Background

1.1. Map Reduce

In dieser Aufgabe wird die Map Reduce Funktion verwendet, um die Häufigkeit von Wörtern in einem Text zu ermitteln. Der Ablauf von Map Reduce ist in Abb. 1 gezeigt. Als Erstes wird ein langer Text in kleinere Teile (Chunks) unterteilt. Anschließend werden die vorkommenden Wörter über den Map Schritt in Key-Value Paare aufgeteilt. Der Key entspricht dem Wort und der Value der Frequenz des Wortes. Als einfaches Beispiel nutzen wir den folgenden Text: „Dieser Text ist ein Beispiel. Ein erstes Beispiel.“. Dieser wird dann zu den folgenden key-value Paaren zerlegt: (dieser,1), (text,1), (ist,1), (ein,2), (beispiel,2), (erstes,1). Danach folgt die Shuffle Phase, welche wir einfachheitshalber in dieser Aufgabe nicht Implementieren. Der Distributor sammelt nach dem map

¹<https://zeromq.org/>

Schritt alle Key-Value paare von den Workern. Angenommen es gäbe noch einen zweiten Worker, welcher „Ein weiterer Text zum mappen.“ als Eingabe bekommen hat, werden dem Distributor folgende Key-Value Paare zurückgegeben: (ein,1), (weiterer,1), (text,1), (zum,1), (mappen,1). In dieser Aufgabe werden jedoch nicht die Häufigkeiten als Zahl addiert. Sondern die Häufigkeit ist ein String bestehend aus Einsen dessen Länge der Häufigkeit entspricht. Somit wäre in unserem Fall die richtige Rückgabe von Worker 1: (dieser,1), (text,1), (ist,1), (ein,11), (beispiel,11), (erstes,1). Die Rückgabe von Worker 2 bleibt unverändert, da dort nur Einsen vorkommen.

Nachdem die Map phase komplett abgeschlossen ist und der Distributor alle Key-Value Paare gesammelt hat, wird reduce ausgeführt. Die gesammelten Key-Value Paare werden nun an die Worker übergeben, welche die Wörter so zusammenzählen, dass jedes Wort genau ein Mal in der Rückgabe vorkommt. In diesem konkreten Beispiel, nutzen wir jedoch nur einen Worker für reduce. Es werden nun die Häufigkeiten der Wörter zusammengezählt und als Zahl zurück an den Distributor gegeben. Der konkrete Input für reduce wäre: (dieser,1), (text,1), (ist,1), (ein,11), (beispiel,11), (erstes,1), (ein,1), (weiterer,1), (text,1), (zum,1), (mappen,1). Somit würde die Rückgabe von reduce wie folgt aussehen: (dieser,1), (text,2), (ist,1), (ein,3), (beispiel,2), (erstes,1), (weiterer,1), (zum,1), (mappen,1).

Zum Schluss sammelt der Distributor alle gezählten Wörter im combine Schritt. Als erstes, werden mehrfach vorkommende Wörter zusammengezählt und im Anschluss werden alle Wörter sortiert. Die Sortierung findet erst nach Frequenz statt und bei gleicher Frequenz alphabetisch. Zu guter Letzt, gibt der Distributor alle gezählten Wörter auf der Konsole aus.

Zusätzlich, können weitere Informationen zu Map Reduce hier nachgelesen werden.

Ein genauer Ablauf des Programms und wie Nachrichten konkret in dieser Aufgabe strukturiert sein müssen, finden Sie später in Abschnitt 2.5 und Abschnitt 2.7.

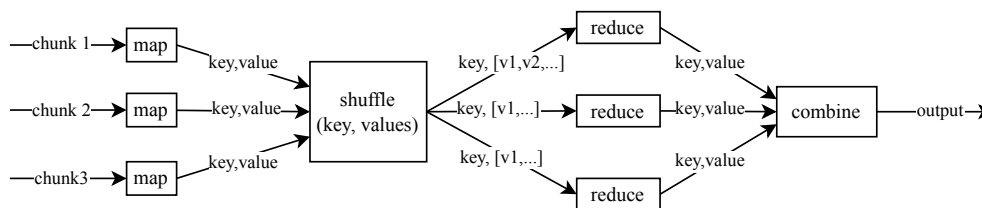


Abbildung 1: Map Reduce Ablauf

1.2. ZMQ

ZMQ ist eine einfach zu handhabende Nachrichtenbibliothek mit vordefinierten Strukturen für bestimmte Anwendungszwecke. In dieser Aufgabe wird der Request Reply Pattern genutzt. Es wird ein Distributor erstellt, welcher mehrere Requests an eine Vielzahl an Arbeitern schickt, welche dann Map Reduce ausführen. Der Startpunkt für diese Aufgabe ist das ZMQ C Beispiel. Informationen über die verschiedenen Nachrichtentypen und

über ZMQ befinden sich hier. Das Request Reply Pattern zu verstehen genügt für diese Aufgabe. Die API Dokumentation von ZMQ befindet sich hier.

1.3. Threading

In dieser Aufgabe wird threading benötigt, um das volle Potential der Arbeitsverteilung zu nutzen. Es wird empfohlen, sich mit pthreads vertraut zu machen. Einige hilfreiche Ressourcen finden sich hier und hier.

1.4. Datenstrukturen

Es ist empfehlenswert für diese Aufgabe, sich auch mit den grundlegenden Datenstrukturen wie Listen, Hashmaps und Bäumen auseinander zu setzen. Diese können Ihnen beim Lösen dieser Aufgabe behilflich sein.

2. Task

Als Startpunkt für diese Aufgabe können Sie das ZMQ C Beispiel mit der libzmq Bibliothek nutzen. Hier geht es zum Beispiel. Bitte wählen Sie C als Programmiersprache und libzmq als Library.

2.1. Installation

Um den „zmq.h“ Header in Ihrem Programm nutzen zu können, müssen Sie zuerst die libzmq3-dev Bibliothek auf Ihrem lokalen Ubuntu System installieren. Dies geht mit folgenden Befehl:

```
1 | sudo apt install libzmq3-dev
```

Listing 1: Installation libzmq3-dev

! Auf den Ubuntu 20 EECS Rechnern ist die libzmq3-dev Bibliothek bereits installiert und kann sofort genutzt werden.

Um die benötigten Python-Module zu installieren nutzen Sie die „requirements.txt“ mit folgenden Befehl:

```
2 | pip install -r requirements.txt
```

Listing 2: Installation Python-Module

2.2. Interpretation Wörter

In dieser Aufgabe werden alle Buchstabenketten als ein Wort interpretiert. Jegliche trennende Zeichen zwischen Wörtern, z.B. Leerzeichen, Bindestriche, Klammern, Zahlen, etc., werden diese als zwei verschiedene Wörter interpretiert. Zusätzlich sind Wörter nicht von Groß- und Kleinschreibung abhängig. Alle Wörter sollen in Kleinschreibung gezählt werden.

2.3. Programm Aufruf

Als Erstes definieren wir die Namen für die beiden aufzurufenden Programme. Der Distributor wird `zmq_distributor` und der Worker `zmq_worker` genannt. Der Distributor erwartet eine `.txt` Datei als Eingabe und die Ports der Worker, welche ≥ 1 sein müssen. Der Worker Aufruf benötigt nur die Ports, auf welchen die Worker hören sollen, welche wieder ≥ 1 sein müssen. Ein Beispielaufruf von Distributor und Worker ist in Listing 3 und Listing 4 zu sehen.

```
3 | ./zmq_distributor <file.txt> <worker port 1> <worker port 2> ... <worker port n>
```

Listing 3: Distributor call

```
4 | ./zmq_worker <worker port 1> <worker port 2> ... <worker port n>
```

Listing 4: Worker call

2.4. Arbeitsverteilung

Nachdem die Eingabedatei erfolgreich gelesen wurden, muss die Aufgabe auf die Worker aufgeteilt werden. Dies passiert durch den Distributor, welcher die Datei in Chunks unterteilt und an die Worker zum Verarbeiten schickt. Um die Aufgabe zu verteilen soll threading benutzt werden, da die Request-Reply Aufrufe warten bis die Aufgabe beendet ist und die Antwort zurück kommt.

2.5. Protokollstruktur

Es werden String-basierte (**ASCII kodiert**) Nachrichten zwischen Distributor und Worker ausgetauscht, welche wie folgt aussehen:

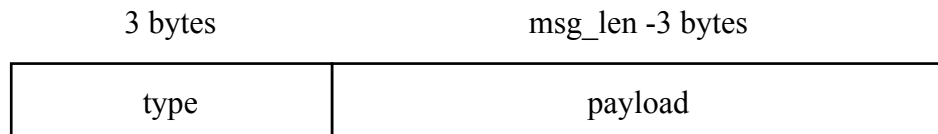


Abbildung 2: Message layout

Hierbei liegt die maximale Nachrichtenlänge bei 1500 Byte, somit $msg_len = 1500$. Die verfügbaren Nachrichtentypen sind in Tabelle 1 definiert und erläutert. Die vom Worker zurückgesendete Antwort soll ebenfalls eine maximale Länge von msg_len haben und beinhaltet den verarbeiteten output als String. Alle ausgetauschten Nachrichten zwischen Distributor und Worker müssen valide C Strings sein (**Nullterminiert**).

type	command
map	map auf payload ausführen
red	reduce auf payload ausführen
rip	worker herunterfahren

Tabelle 1: Unterstützte Nachrichtentypen

2.6. Word Count Output

Nachdem map und reduce auf den Workern ausgeführt wurden, sammelt der Distributor alle vorhandenen Wörter und addiert deren Häufigkeiten. Die Wörter werden anschließend in absteigender Frequenz sortiert und im Falle von gleicher Frequenz alphabetisch. Zum Schluss werden alle Wörter im csv Format mit dem Header `word,frequency` auf stdout ausgegeben. Ein beispielhafter Programmablauf ist in Abschnitt 2.7 gegeben.

2.7. Beispiel Programmablauf

Der folgende Text wird als Beispiel verwendet: „The example text is the best example.“. Es wird nur ein Worker genutzt. Als erstes wird map und anschließend reduce ausgeführt. Danach, sammelt der Distributor alle Wörter und addiert/sortiert sie entsprechend (siehe Abschnitt 2.6). Als letztes wird die Liste der Wörter mit deren Häufigkeit im csv Format auf stdout ausgegeben und der Worker heruntergefahren. Im Folgenden findet sich auch der Nachrichtenaustausch sowie die Ausgabe des Distributors.

1. distributor → worker
mapThe example text is the best example.
2. worker → distributor
the11example11text1is1best1
3. distributor → worker
redthe11example11text1is1best1
4. worker → distributor
the2example2text1is1best1
5. distributor output:
word,frequency
example,2
the,2
best,1
is,1
text,1
6. distributor → worker
rip

7. worker → distributor
rip

2.8. Tips ZeroMQ

2.8.1. ZeroMQ einbinden

Um die ZeroMQ Bibliothek korrekt zu finden und einzubinden nutzen Sie folgenden Befehl:

```
1 | find_library(ZeroMQ zmq REQUIRED)
```

Listing 5: ZeroMQ einbinden

Diese kann dann über „zmq“ eingebunden werden.

2.8.2. zmq_context

Es wird empfohlen im Programmaufruf vom Distributor und Worker, jeweils genau einen zmq_context zu erstellen. Falls Sie mehrere erstellen kann dies zu Problemen beim vollständigen Aufräumen führen.

2.8.3. Request-Reply

Ein versendeter Request muss immer von einem Reply gefolgt werden, welcher auch erfolgreich empfangen werden muss.

3. Tests

Die im Folgenden beschriebenen Tests dienen sowohl als Leitfaden zur Implementierung, als auch zur Bewertung. Beachten Sie hierzu insbesondere auch die Hinweise in Anhang B. **Bei Problemen mit Pytest nutzen Sie bitte die EECS-Rechner um Ihren Code zu testen.** Damit können Sie Versionsfehler usw. vorbeugen.

Jeder einzelne Test kann als Teilaufgabe verstanden werden. In Summe führen die Tests zu einer vollständigen Implementierung der Aufgabe. Teilweise werden spätere Tests den vorigen widersprechen, da sich die Aufgabenstellung im Verlauf weiter konkretisiert. Die Tests sind daher so geschrieben, dass diese auch die weitere Entwicklung Ihrer Lösung als korrekt akzeptieren.

3.1. Worker erstellt und variabel (1 Punkt)

Es wird getestet, ob alle Worker gestartet wurden und eine Anfrage erhalten können. Die Anfrage, die an alle Worker geschickt wird, ist ein leerer Payload auf dem map ausgeführt werden soll. Es sollte ein leerer string von jedem Worker zurückkommen.

! Alle Worker sollten antworten. Es werden 1, 2, 4 und 8 Worker getestet.

3.2. Distributor Nachrichtengröße (1 Punkt)

In diesem Test wird der Distributor gestartet und bekommt einen zufälligen Text als Eingabe. Es werden alle ausgehenden Nachrichten vom Distributor auf Einhalten der maximalen Nachrichtengröße überprüft.

3.3. Simple Word Count (1 Punkt)

Als nächstes wird die Ausgabe bei einer zufälligen simplen Texteingabe überprüft. Der Eingabetext enthält nur Wörter, Leerzeichen, Kommas und Punkte.

! Es werden 1, 2, 4 und 8 Worker getestet.

3.4. Complex Word Count (1 Punkt)

Es wird erneut der Output des Distributors auf Richtigkeit überprüft, mit einem längeren, zufälligen Text und allen möglichen Satzzeichen.

! Es werden 2, 4 und 8 Worker getestet.

3.5. Bücher Test (2 Punkte)

In diesem Test werden die Wörter aus zwei zufällig ausgewählten Büchern gezählt. Insgesamt stehen fünf Bücher zur Auswahl und lauten wie folgt:

- Romeo and Juliet
- Moby Dick; Or, The Whale
- Pride and Prejudice
- Frankenstein; Or, The Modern Prometheus
- Middlemarch

! Es werden 2, 4 und 8 Worker getestet.

3.6. Protokollimplementierung (1 Punkt)

Als Erstes wird in diesem Test der Distributor mit einem Python Distributor ersetzt und es wird überprüft, ob die Nachrichten das richtige Format haben und das Protokoll korrekt implementiert wurde. Danach wird der Worker durch einen Python Worker ersetzt und die Protokollkorrektheit wird erneut überprüft.

! Es werden 2, 4 und 8 Worker getestet.

3.7. Worker Load Distribution (1 Punkt)

In diesem Test wird überprüft, wie die Arbeit auf die Worker aufgeteilt wird. Gezählt wird, wie viele Anfragen jeder Worker vom Distributor bekommt und es wird die Standardabweichung berechnet welche ≤ 2 sein muss.

! Es werden 2, 4 und 8 Worker getestet.

3.8. Memory Leaks (2 Punkte)

Als Letztes wird überprüft, ob memory leaks beim Ausführen des Distributors oder Workers existieren. Da die Worker den *rip* Befehl erhalten, sollten diese sauber herunterfahren und den Speicher aufräumen. Der folgenden valgrind Befehl wird zum testen genutzt:

```
5 | valgrind --tool=memcheck --xml=yes --xml-file=<output_file> --gen-suppressions=all  
    --leak-check=full --leak-resolution=med --track-origins=yes --trace-children=  
    yes --vgdb=no --fair-sched=yes <distributor/worker exec> <program args>
```

Listing 6: valgrind Aufruf

! Es werden 2, 4 und 8 Worker getestet.

4. Test Debugging

Um den Output des Distributors besser zu debuggen, kann dem `test_praxis3.py` Script der Parameter `--debug_test` übergeben werden. Nun wird die Ausgabe von Ihrem Distributor und die erwartete Ausgabe des Tests im `debug` Ordner abgespeichert. Es werden jeweils zwei Ausgabedateien pro Testfall erzeugt. Sie können über ein diff Programm, wie z.B. `vimdiff`, die Ausgaben miteinander vergleichen. Zusätzlich, werden alle Testdateien gespeichert und können zum Testen genutzt werden. Ebenfalls werden die valgrind output Dateien beibehalten. Normalerweise werden Test- und valgrind Dateien beim Test-Cleanup wieder gelöscht.

A. Abgabeformalitäten

Die Aufgaben können in Gruppen aus **ein bis drei Mitgliedern** bearbeitet (und abgegeben) werden. Dazu wählen Sie jeweils in der **ersten Woche** der Bearbeitungszeit (siehe oben) eine (neue) Abgabegruppe in dem entsprechenden Gruppenwahlmodul auf ISIS. Wenn Sie alleine arbeiten, müssen Sie trotzdem eine Einzelgruppe wählen. Benutzen Sie dazu das entsprechende, weitere Gruppenwahlmodul auf ISIS. Die Wahl einer Einzelabgabegruppe ist bis zur Abgabe möglich, beginnt aber erst, nachdem die Gruppenwahl für Mehrpersonengruppen nach einer Woche geendet ist. Die getätigte Gruppenwahl gilt jeweils für den gesamten Bearbeitungszeitraum eines Praxis-Zettel.

Ab der jeweils zweiten Woche der Bearbeitungszeit können Sie Ihre Lösung abgeben. Ab diesem Zeitpunkt kann die Gruppenwahl **nicht mehr** verändert werden! Sollten Sie zu diesem Zeitpunkt keine Abgabegruppe gewählt haben, können Sie diesen Praxis-Zettel **leider nur noch allein bearbeiten!** Die Gruppenabgabe muss von einem der Gruppenmitglieder in ISIS hochgeladen werden und gilt dann für die gesamte Gruppe. Ohne eine Abgabe auf ISIS erhalten Sie keine Punkte!

Es werden nur mit CMake via `make -C build package_source` erstellte Abgabearchive im `.tar.gz`-Format akzeptiert. Beachten Sie, dass eine falsche Dateieindung nicht einfach umbenennen können. **Wir empfehlen dringend**, ihr so erstelltes Archiv einmal zu entpacken, und die Tests auszuführen. So können Sie viele Fehler mit ihrer Projekt-konfiguration vor der Abgabe erkennen und vermeiden.

Ihre Abgaben können ausschließlich auf ISIS und bis zur entsprechenden *Abgabefrist* abgegeben werden. Sollten Sie technische Probleme bei der Abgabe haben, informieren Sie uns darüber unverzüglich und lassen Sie uns zur Sicherheit ein Archiv Ihrer Abgabe per Mail zukommen. Beachten Sie bei der Abgabe, dass die **Abgabefrist fix ist** und es **keine Ausnahmen für zu späte Abgaben oder Abgaben via E-Mail** gibt. Planen Sie einen angemessenen Puffer zur Frist hin ein. In Krankheitsfällen kann die Bearbeitungszeit angepasst werden, sofern sie uns baldmöglichst ein Attest zusenden.

B. Tests und Bewertung

Die einzelnen Tests finden Sie jeweils in der Vorgabe als `test/test_praxisX.py`. Diese können mit `pytest` ausgeführt werden:

```
1 | python3 -m pytest test # Alle tests ausführen
2 | python3 -m pytest test/test_praxisX.py # Nur die Tests für einen bestimmten Zettel
3 | python3 -m pytest test/test_praxis1.py -k test_listen # Limit auf einen bestimmten
   | Test
```

Sollte `pytest` nicht auf Ihrem System installiert sein, können Sie dies vermutlich mit dem Paketmanager, beispielsweise `apt`, oder aber `pip`, installieren. Analog müssen Sie zusätzlich auch das Paket `pytest-timout` installieren, damit Sie die Tests ausführen können. Alle Abhängigkeiten finden Sie auch in der Datei `requirements.txt`.

Ihre Abgaben werden anhand der Tests der Aufgabenstellung automatisiert bewertet. Beachten Sie, dass Ihre Implementierung sich nicht auf die verwendeten Werte (Node

IDs, Ports, URIs, ...) verlassen sollte, diese können zur Bewertung abweichen. Darüber hinaus sollten Sie die Tests nicht verändern, um sicherzustellen, dass die Semantik nicht unbeabsichtigterweise verändert wird. Eine Ausnahme hierfür sind natürlich Updates der Tests, die wir gegebenenfalls ankündigen, um eventuelle Fehler zu auszubessern.

Wir stellen die folgenden Erwartungen an Ihre Abgaben:

- Ihre Abgabe muss ein CMake Projekt sein.
- Ihre Abgabe muss eine `CMakeLists.txt` enthalten.
- Ihr Projekt muss ein Target entsprechend der oben genannten Definition haben (z.B. `hello_world`) mit dem selben Dateinamen (z.B. `hello_world`) (case-sensitive) erstellen.
- Ihre Abgabe muss interne CMake Variablen, insbesondere `CMAKE_BINARY_DIR` und `CMAKE_CURRENT_BINARY_DIR` unverändert lassen.
- Ihr Programm muss auf den EECS Poolrechnern² korrekt funktionieren.
- Ihre Abgabe muss mit CPack (siehe oben) erstellt werden.
- Ihr Programm muss die Tests vom jeweils aktuellen Zettel bestehen, nicht auch vorherige.
- Wenn sich Ihr Program nicht deterministisch verhält, wird auch die Bewertung nicht deterministisch sein.

Um diese Anforderungen zu überprüfen, sollten Sie:

- das in der Vorgabe enthaltene `test/check_submission.sh`-Script verwenden:

```
1 | ./test/check_submission.sh praxisX
```

- Ihre Abgabe auf den Testsystemen testen.

Fehler, die hierbei auftreten, werden dazu führen, dass auch die Bewertung fehlschlägt und Ihre Abgabe entsprechend benotet wird.

²Die Bewertung führen wir auf den Ubuntu 20.04 Systemen der EECS durch, welche auch für Sie sowohl vor Ort, als auch via SSH zugänglich sind.