University of Reading

Advanced Computing

Coursework Pseudo-code Example Structure

Concepts

University of Reading

- Input Data Mode
  - Streamed or Block data
- Data structures
  - Raw Data Handling
  - Structured Data Format
- Error Handling Strategy
  - Detection/Correction
  - What Process is relevant, and at which stage(s)
- Apache Hadoop (Chained) Jobs
  - Mapper/(Combiner)/Reducer
  - More than one set of Constructs ?
  - Raw/Structured Data Transform to format appropriate to current requirement
- A (very high-level) pseudo-code partial example

Concepts: Input Data Mode (1)

University of Reading

- The data-sets (*.csv) are available as discrete input files of known size - Simple
  - Read them in
  - Process them for simple format errors
  - Apply Functional Transformation
  - Produce output
  - The end

- In the real world this is also quite possible
  - Many systems are dynamic (real-time)
  - These data-sets will be in constant flux
  - They could benefit from by imposing some ’Arbitrary Block format’ on them
  - There is no fixed rule about how they should be manipulated
  - For this example it is (largely) dependent on your design intentions

Concepts: Input Data Mode (2)

University of Reading

- Streamed (Piped) data
  - In the real-world this could be asynchronous – (i.e. users don't click on websites to a fixed timetable)
  - There can be synchronous as well – any process that produces an expected output (i.e. Watchdog)
  - Your code design can use this model – you can control the input stream record by record
  - Read in a line (this continues until ***some decisive factor*** causes it to stop – in this case the **EOF**)
    - Send single record to the **Mapper** function
    - Collate the results (Task 1 – in a vector/array, Task 2 - Hadoop via the Framework/HDFS)
    - **Combine** the results – to reduce Network traffic (***this is an optional phase***)
    - **Reducer**
    - Formatted Output
- Considerations
  - Reading in data line-by-line is quite inefficient (not a problem here as the data-sets are text files)
  - Collate the raw text input into a fixed, known size 'Block Data' format using a buffering technique
  - For systems with truly asynchronous event – process each 'Data Request' discretely, collation takes time

Concepts: Input Data Mode (3)

University of Reading

- Block data
- A set of data records in a fixed ‘Block’ size provides several potential advantages
  - They can be processed as a single entity – often more efficient
  - This allows for inter-record ‘dependencies’ to be identified
  - These can be requirements i.e.
    - (A & B & C) All data must be present
    - (A | B | C) Any data record requirement will suffice
    - (A & (B | C)) A and B or C
    - (A & !(B | C)) Only A
- Enables/supports Error Detection and Correction

Concepts: Data structures (1)

University of Reading

- This will depend on:
  - The data input type
  - Your requirements at any given point in the Run-time functional transformation/processing
  - Design them to support your processing/design requirements
  - A good design will make writing the code so much easier (And, the opposite will also be (very) apparent)

- Question: Have you ever thought ‘The data format is not quite right for my current needs’ ?

- Your answer is Yes - What do you do to compensate
  - Code development is an iterative process
  - You may go back and reconsider your new requirements in view of your new knowledge
  - Modify the data structures to allow for the new run-time requirements
  - This may make the Data structure unwieldy for initial purposes
  - Do you add a secondary level of Data structure instead
- It is a **judgement call**, one you will have to make many times

- Your answer is no - You are magnificent (and unique) in your coding ability……

Concepts: Data structures (2)

University of Reading

- Raw Data Handling
  - The Input stage
  - Based on the decision/system requirement regarding streamed/Block data
  - Read the data into a suitable vector/array of ‘Memory’ (could be string/char array/byte array etc)
  - Optional: convert the Raw data into some inter-mediate format

- Structured Data Format
  - Generally second/subsequent stages (although it can also be the input stage)
  - Design the format (class structure) to be flexible to suit current Run-time processing requirements
  - Ideally, it will also (relatively) easily support latter Run-time processing requirements
  - In an efficient manner - only store/extract what is required
    - Memory is limited
    - Processing is costly (in time and therefore money)

- Reminder: Iterative process – be prepared to modify as the development proceeds

Concepts: Error Handling Strategy (1)

University of Reading

- Detection/Correction
  - At any stage in the Run-time processing (Input/Inter-mediate, and Output)

- Syntactic Errors:
  - Data Format
  - Records – Empty (NULL) Fields/Number of Fields incorrect

- Semantic Errors:
  - Record is formatted correctly – but some data is meaningless in the context

- The various data records have a published format
  - As the records are created they should be tested against this published format
  - At some point in the Run-time processing…..

Concepts: Error Handling Strategy (2)

University of Reading

- What Process is relevant, and at which stage(s)
- This will depend on many considerations:
  - The requirements in any technical problem domain description you <may> have
  - The processing you consider necessary in the context of the current Run-time process
  - It will be subject to limitations imposed by the **Data-set Itself**
  - The **Data-set** may be incomplete
    - This could limit the type and amount of 'Global' Error Detection/Correction you can apply
    - Applying inappropriate Error Detection/Correction at this stage could lead to incorrect results
  - The **Data-set** may become more, or totally, complete at a later stage in the Run-time process
    - This may reveal more meta-data to ***'inform'*** (i.e. modify) the Error Detection/Correction strategy
    - This will enable a more appropriate part of the Error Detection/Correction strategy to be applied
- Considerations:
  - Another iterative part of the design process
  - It will applicable throughout the Run-time processing
  - However, It must be applied correctly to avoid generating another set of problems

Concepts: Apache Hadoop (1)

University of Reading

- (Chained) Jobs – when to consider using this technique
  - Remember this : '**How long is a piece of string**' ?
  - In other words: There is no single answer to all problem design domains
  - This is the strategy/consideration that is to be at the heart of your system design
  - The input Data-set files must have come from somewhere – they did not just appear from thin air !
  - They were 'constructed' by some means exterior to your **'Problem Domain'**- via a 'Chain'
  - **'Chained Jobs'** has specific meaning in Hadoop
  - **But it is just an example** of how such computer processing/systems are often inter-connected
  - Each 'job' is an inter-mediate stage in the overall processing transformation from input to output

- The Problem Domain has a number of objectives
  - These can be considered as inter-mediate stages in the overall processing of **any** system
  - They can also be considered as **'Jobs'** in the Hadoop paradigm
  - The overall design of this is another **judgement call** – one for you to determine

Concepts: Apache Hadoop (2)

University of Reading

- An Objective can be considered as a Hadoop ‘Job’
  - Design the Data structures appropriately to support the current Run-time requirement
  - At some point the Data structure format/contents may match what is required to answer an objective
  - At this point the **‘Job’** ***could*** be dispatched to **Hadoop**
  - If memory is sufficient (and there is no dependency on the previous **Hadoop ‘Job’**) it can be delayed
  - Some or all the objectives can be combined in a single **Hadoop ‘Job’** if required
    - This is perfectly valid (although it will not always be possible)
    - May be more efficient to do this
    - Another design decision

- Mapper/(Combiner)/Reducer
  - In any combination to the suit the Problem Domain
  - More than one set of Constructs ?
  - Raw/Structured Data Transform to format appropriate to current requirement

Pseudo-code: Example - A single Job

University of Reading

- Phase: Initialise
- Phase: Read/Parse Input Data
- Phase: Mapper
- Phase: Combiner (optional)
- Phase: Shuffle/Sort (optional)
- Phase: Reducer
- Overall: Apply Objective(s) as Hadoop ‘Jobs’
  - Determine the number of flights from each airport, include a list of any airports not used.
  - Create a list of flights based on the Flight id, this output should include the passenger Id, relevant IATA/FAA codes, the departure time, the arrival time (times to be converted to HH:MM:SS format), and the flight times.
  - Calculate the number of passengers on each flight.
  - Calculate the line-of-sight (nautical) miles for each flight and the total travelled by each passenger.
- Phase: Output Results
- Phase: End Processing – Clean Up

Pseudo-code: Initialise

University of Reading

```
SET: EXIT CLEAN-UP FUNCTION (Java: addShutdownHook() - not recommended OR required)
CALL: PARSE COMMAND LINE ARGUMENTS
      IF TEST: COMMAND LINE ARGUMENT ERROR
      EXIT: PRINT ERROR MESSAGE [FILE <PATH/NAME> NULL]
      ENDIF
END CALL
IF TEST: INPUT DATA-SET FILENAME(S) NULL
      EXIT: PRINT ERROR MESSAGE [FILE <PATH/NAME> NULL]
END IF
IF TEST: INPUT DATA-SET FILENAME(S) NOT EXIST
       EXIT: PRINT ERROR MESSAGE [FILE <PATH/NAME> NOT EXIST]
END IF
```

Pseudo-code: Read/Parse Input Data

University of Reading

```
OPEN: FILE <PATH/NAME>
IF TEST: FILE NOT OPEN
    EXIT: PRINT ERROR MESSAGE [FILE <PATH/NAME> CANNOT OPEN]
ENDIF
CALL: LOAD INPUT-DATA-SET FILES
    LOOP:
    READ LINE
    IF LINE NULL: SKIP
    ADD LINE TO INPUT-DATA-STRUCTURE
    END LOOP
    IF TEST: INPUT-DATA-STRUCTURE EMPTY
    EXIT: PRINT ERROR MESSAGE [FILE <PATH/NAME> EMPTY]
    END IF
END CALL
```

Pseudo-code: Mapper

University of Reading

```
CALL: PARSE INPUT-DATA-STRUCTURE(S)
    LOOP: INPUT-DATA-STRUCTURE(S)
    READ: INPUT-DATA-STRUCTURE ELEMENT
    IF TEST: PARSE ELEMENT ERROR
    CALL: CORRECT INPUT-DATA-STRUCTURE ELEMENT
    IF TEST: CORRECT ELEMENT FAILED
    REMOVE: INPUT-DATA-STRUCTURE ELEMENT
    END IF
    END IF
    END LOOP
    IF TEST: DATA-STRUCTURE EMPTY
    EXIT: PRINT ERROR MESSAGE [INPUT-DATA-STRUCTURE EMPTY]
    END IF
END CALL
```

Pseudo-code: Combiner

University of Reading

- The combiner is a local Reducer function
- Usually located on the same node(s) as the Mapper functions
- Primary role is to limit the data-sets input to the Reducer nodes
- Reducing the amount of network traffic (and therefore latency)
- Useful for processing the data on a more ‘Global’ level
- An intermediary between the Mapper and Reducer Nodes
- Therefore it must have the same input and output types

Pseudo-code: Shuffle/Sort (optional)

University of Reading

- This is optional:
  - The order of the data in this instance is not important
  - It may benefit subsequent processing if the data is ‘key’ sorted (especially if it the data-set large)

```
CALL: SHUFFLE_SORT INPUT-DATA-STRUCTURE(S)
    LOOP: INPUT-DATA-STRUCTURE(S)
        CALL: SORT ALGORITHM
    END LOOP
END CALL
```

Pseudo-code: Reducer

University of Reading

```
CALL: COMBINE INPUT-DATA-STRUCTURE(S)
    LOOP: INPUT-DATA-STRUCTURE(S)
    READ: INPUT-DATA-STRUCTURE ELEMENT
    IF TEST: PARSE ELEMENT DUPLICATION ON <key>
    REMOVE: INPUT-DATA-STRUCTURE ELEMENT
    INCREMENT: ELEMENT DUPLICATION COUNTER
    END IF
    END LOOP
    IF TEST: DATA-STRUCTURE EMPTY
    EXIT: PRINT ERROR MESSAGE [INPUT-DATA-STRUCTURE EMPTY]
    END IF
END CALL
```

Pseudo-code: Apply Objectives

University of Reading

- Phase: Apply Objectives
  - Determine the number of flights from each airport, include a list of any airports not used.
  - Create a list of flights based on the Flight id, this output should include the passenger Id, relevant IATA/FAA codes, the departure time, the arrival time (times to be converted to HH:MM:SS format), and the flight times.
  - Calculate the number of passengers on each flight.
  - Calculate the line-of-sight (nautical) miles for each flight and the total travelled by each passenger.

- These are all standard programming processes – pseudo-code is all yours to determine…..

Pseudo-code: Output Results

University of Reading

- These are also standard programming processes
  - The pseudo-code is all yours to determine…..
  - The format of the output is also yours to determine…..

Pseudo-code: End Processing – Clean Up

University of Reading

```
IF EXIT CLEAN-UP FUNCTION
    CALL: EXIT CLEAN-UP FUNCTION
    EXIT:
END IF
IF TEST: OUTPUT_FILE(S) OPEN
    CLOSE OUTPUT_FILE(S) OPEN
    IF TEST: OUTPUT_FILE(S) CLOSE FAIL
    EXIT: PRINT ERROR MESSAGE [FILE <PATH/NAME> CLOSE FAIL]
    END IF
   REMOVE ANY TEMPORARY FILES FROM STORAGE MEDIA
END IF
```

Pseudo-code: Final thoughts (1)

University of Reading

- C/C++ ‘atexit’ function is not easily replicated in Java
  - **java.lang.Runtime.addShutdownHook(Thread hook)**
  - Some suggestion that this represents a security hole
  - There are some proprietary AtExit() Java classes out there (avoid these)

- The Mapper may seem overly simple in this example….
  - Which it is – they often are

- Read/Parse Input Data
  - This is shown as a function early in the program structure
  - But, it can be performed anywhere in the program run-time where it is appropriate
  - Especially in the case of error detection/correction
  - **And**, it can (and should) be applied to the output data…..

Pseudo-code: Final thoughts (2)

University of Reading

- Does your pseudo-code design actually (and accurately) represents:
  - The input data-set structure
  - The problem domain constraints: i.e.
    - The program must: do **this, that,** and the **other (**and **NOT** some other things**)**
    - Ensure you supply an answer to the question that has been ***actually*** asked !!!!
  - An efficient design (in principle)
  - That the output results are in the right format - and placed in the right location

- Finally, this is just an example of how I might approach the coursework
  - This pseudo-code is instead of the automotive interface software design (no time for that !)
  - It is not meant as a model for your version
  - Structure your code in a way appropriate to your thinking
  - No mention of threads (!) but still a valid option – this is just very high-level pseudo-code
  - A VERY useful process to master – both for the CW/exam – (and for when you are at work)