# MG4J: Managing Gigabytes for Java

# Introduction

Adriano Fazzone

DIAG Department
Sapienza University of Rome

#### **Contact information**

#### Adriano Fazzone

Dipartimento di Ingegneria Informatica, Automatica e Gestionale "A. Ruberti" (**DIAG**)

Via Ariosto 25, Rome, Italy

Room A220 (second floor)

Email: fazzone@diag.uniroma1.it

Home page: www.diag.uniroma1.it/~fazzone

### **Managing Gigabytes for Java**

#### Schedule:

Introduction to MG4J framework.

 Exercise: try to set up a search engine on particular collections of documents.

# MG4J: introduction (1)

- Free full-text search engine for large document collections
- Written in Java
- Developed by the Department of Computer Science (University of Milan)
- MG4J home: http://mg4j.di.unimi.it/
- Documentation: <a href="http://mg4j.di.unimi.it/docs-big/">http://mg4j.di.unimi.it/docs-big/</a>
- Manual: <a href="http://mg4j.di.unimi.it/man-big/manual.pdf">http://mg4j.di.unimi.it/man-big/manual.pdf</a>

# MG4J: introduction (2)

 MG4J can be used for indexing and querying a large collection of documents

 INPUT: set of documents with the <u>same</u> number and type of fields

OUTPUT: essentially an inverted index

# MG4J: introduction (3)

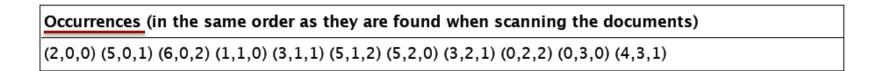
To create and to query an inverted index with MG4J we have to complete these 3 steps:

- 1. Create a collection of documents.
- 2. Create an inverted index on the created collection.
- 3. Query the created inverted.

# **Example of index (1)**

Document pointer	Document
О	I love you
1	God is love
2	Love is blind
3	Blind justice

Term index	Term
0	blind
1	god
2	i
3	is
4	justice
5	love
6	you



### **Example of index (2)**

- An occurrence is a group of three numbers (t,d,p)
- (t,d,p) means that the term whose index is t appears in document d at position p
- Inverted lists can be obtained by re-sorting the occurrences in increasing term order, so that occurrences relative to the same term appear consecutively

Term	Occurrences
0 (blind)	(0,2,2) (0,3,0)
1 (god)	(1,1,0)
2 (i)	(2,0,0)
3 (is)	(3,1,1) (3,2,1)
4 (justice)	(4,3,1)
5 (love)	(5,0,1) (5,1,2) (5,2,0)
6 (you)	(6,0,2)

# **Building batches**

- MG4J scans the whole document collection producing batches
- Batches are <u>sub-indices limited to a subset of</u>
   <u>documents</u>, and they are created each time the number
   of indexed documents reaches a user-provided
   threshold, or when the available memory is too little
- Once the batches are created, they are combined in a single index

# Time and space requirements

- Scanning phase is time and space consuming
- MG4J is able to work with little memory, but more memory allows to create larger batches which can be merged quickly:)
- The indexer produces a number of subindexes proportional to the number of occurrences
- But combining subindexes is a resource consuming activity.
- Good strategy: Try to create batches <u>as large as</u> <u>possible</u> and check the logs to avoid to work with an <u>almost full heap</u> (this slows down Java)

# **Querying the index**

Once the index is built we can query it using a web server

MG4J allows to use <u>command line</u> and the <u>web browser</u>

# Sophisticated query: scorer

- MG4J provides very sophisticated query tuning
- To use this features, <u>we must</u> use the command line interface. All settings are then used for subsequent queries (sticky behavior)
- MG4J allows to choose the scorer for the ranking of the results of a query
- Scorers assign a score to the document, depending on some criterion (e.g. the frequency of query terms in the document)
- Documents are ordered using the scores, so that the most relevant documents appear at the beginning of the result list

# **Performance (1)**

- MG4J provides a great flexibility for the index construction, and all the choices have a significant impact on performance
- If we have enough memory, building large batches is a good trick
- We can choose several different codes for the components of the index.

# Performance (2)

- Another trick is <u>discarding what is not necessary</u>: pointers, counts, positions, etc.
   For example, if we use *TF/IDF* scoring, we do not need to store positions in the index (use -c POSITIONS:none)
- Indexes contain skipping structures to skip index entries.
   However, skipping structures introduce a slight
   overhead when scanning sequentially a list (use --no-skips option to disable skipping structure)

# Performance (3)

- The index can be:
  - read from disk
  - memory mapped
  - directly loaded into main memory
- The three solutions work with increasing speed and increased main memory usage
- The default is to read the index from disk
- We can add suitable options to the index URI (e.g. mapped=1 or inmemory=1) to use the other solutions

# **Partitioning**

- Partitioning an index means <u>dividing an index</u> using some criterion
- Partitioning can be:
  - Documental (splitting using documents)
  - Lexical (splitting using terms)
  - Personalized
- Improving performance: we can partition our index, and once we have several subindexes, we can decide which ones will be loaded into main memory, which ones will be mapped, etc.

### **End of MG4J Presentation.**

Let's move now to mg4j-exercise.pdf;)