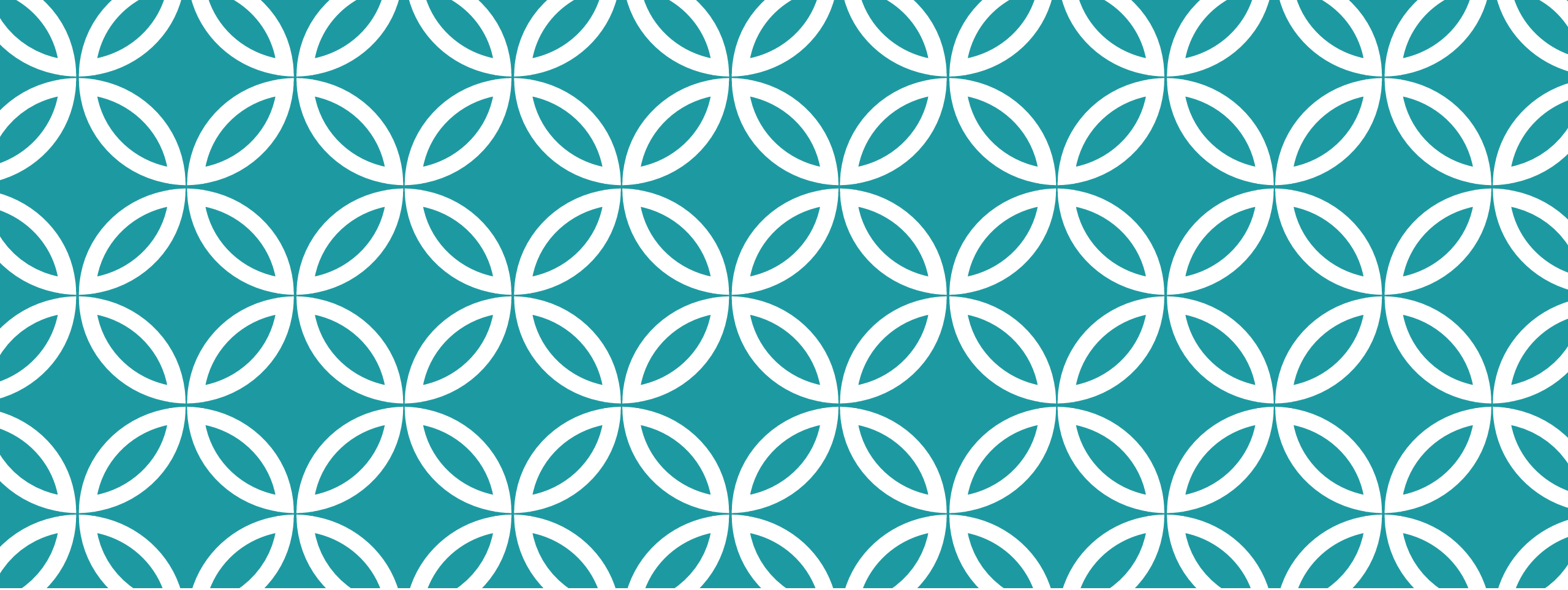
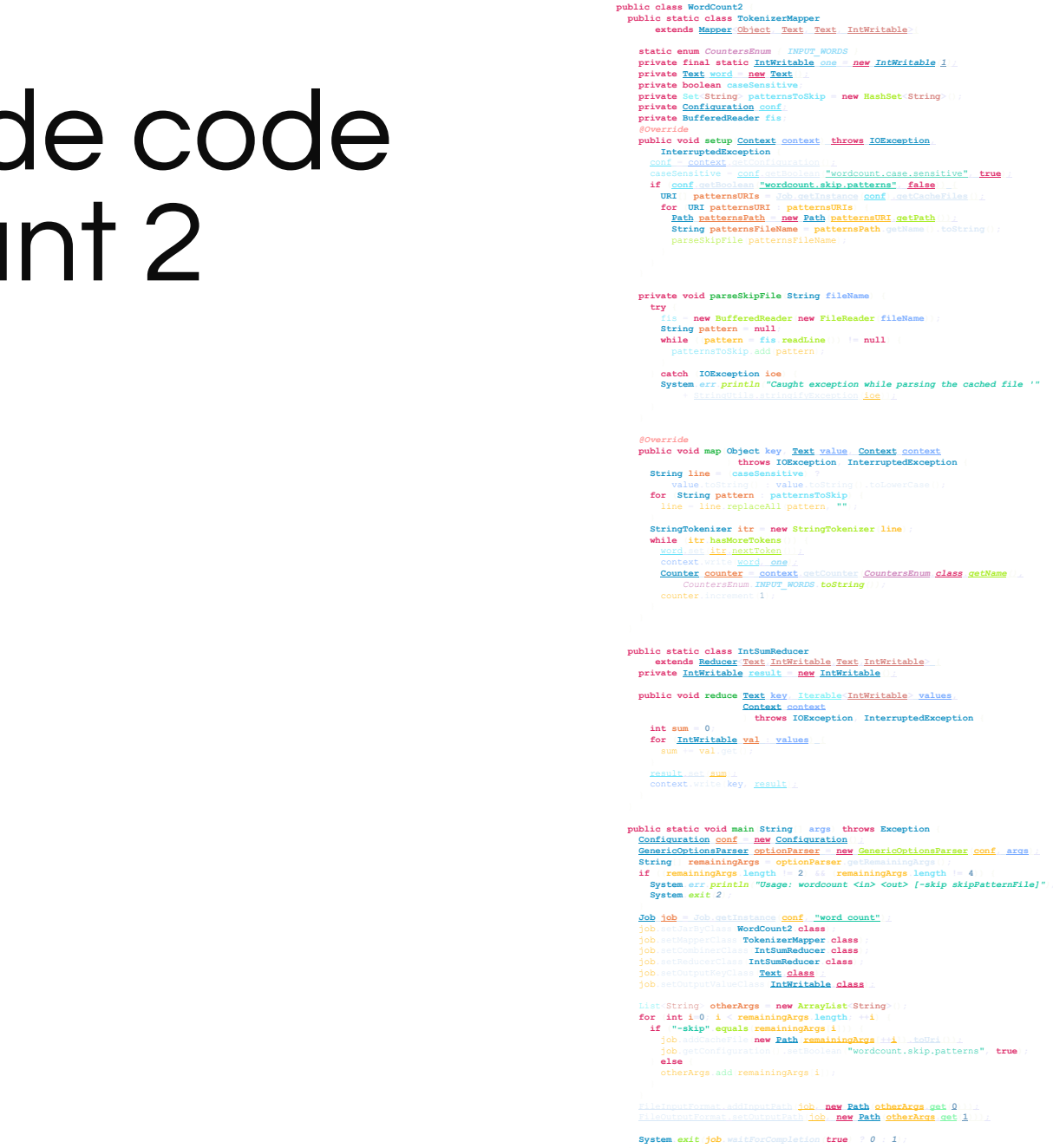


Exemple WordCount 2 Avancé

Analyse de code Wordcount 2



Hadoop Input Format

InputFormat décrit les spécifications des entrées d'une Job MapReduce.
La classe InputFormat à pour role de:
1.Vérifier les entrées du Job.
2.Découper les fichiers en entrée en InputSplit, chacun de ces input split est ensuite envoyé au Mappeur.
3.Procurer une fabrique vers une implémentation de RecordReader utilisée pour décomposer les InputSplit en message <key, Value> pour les mappeurs.

Le comportement par défaut des InputFormat sur les fichiers, sous-classe de FileInputFormat, est de découper les fichieis en fonction de leur taille en octets. Cependant la taille des blocs du system de fichier est utilisée comme borne supérieure (HDFS bloc). On peut fixer une borne inférieure via la configuration mapreduce.input.fileinputformat.split.minsize.

Les splits basés sur la taille du fichier ne sont pas suffisant pour de nombreuses applications. Il est donc parfois nécessaire de ré-implementer un RecordReader, qui s'occupe de découper en bloc intelligible par l'application les InputSplits.

TextInputFormat est l'InputFormat par défaut. Si il est utilisé, Hadoop détecte automatiquement les fichiers compressés et les décompresse en utilisant le codec approprié. Cependant, quand on utilise des fichiers compressé, le fichier ne peut pas être découpé en InputSplit et il est traité par un seul mapper.

Hadoop InputSplit

InputSplit représente les sous-ensemble de données qui seront traitées par unique Mapper.
L'InputSplit donne une vue binaire des données, c'est au RecordRearder de donner une vue clé,valuer au données.

FileSplit est le l'InputSplit par défaut. Il positionne la variable mapreduce.map.input.file sur le path du fichiers découpé.

Hadoop RecordReader

RecordReader lit des <key, value> à partir de l'InputSplit.

Génarellement, le RecordReader convertit les données de l'InputSplit pour qu'ells puissant être traitées par les mapper.

La RecordReader a donc la responsabilité des trouver redécouper les InputSplit en bloc keys and values.

Hadoop Output Format

OutputFormat décrit les sorties d'un job MapReduce.
Son role dans l'application est de :
1.Vérifier la validitté de la sortie. Par exemple que le fichier n'existe pas.
2.Fournir un une implementation de RecordWriter utilisée pour écrire les <key,value> dans la sortie. Les sorties sont stockées dans le FileSystem.

Le TextOutputFormat est utilisé par défaut comme OutputFormat.

Hadoop Output Commiter

OutputCommitter
OutputCommitter describes the commit of task output for a MapReduce job.
The MapReduce framework relies on the OutputCommitter of the job to:
1.Setup the job during initialization. For example, create the temporary output directory for the job during the initialization of the job. Job setup is done by a separate task when the job is in PREP state and after initializing tasks. Once the setup task completes, the job will be moved to RUNNING state.
2.Cleanup the job after the job completion. For example, remove the temporary output directory after the job completion. Job cleanup is done by a separate task at the end of the job. Job is declared SUCCEDED/FAILED/KILLED after the cleanup task completes.
3.Setup the task temporary output. Task setup is done as part of the same task, during task initialization.
4.Check whether a task needs a commit. This is to avoid the commit procedure if a task does not need commit.
5.Commit of the task output. Once task is done, the task will commit it's output if required.
6.Discard the task commit. If the task has been failed/killed, the output will be cleaned-up. If task could not cleanup (in exception block), a separate task will be launched with same attempt-id to do the cleanup.

FileOutputCommitter is the default OutputCommitter. Job setup/cleanup tasks occupy map or reduce containers, whichever is available on the NodeManager. And JobCleanup task, TaskCleanup tasks and JobSetup task have the highest priority, and in that order.

Hadoop Output

Task Side-Effect Files
In some applications, component tasks need to create and/or write to side-files, which differ from the actual job-output files.
In such cases there could be issues with two instances of the same Mapper or Reducer running simultaneously (for example, speculative tasks) trying to open and/or write to the same file (path) on the FileSystem. Hence the application-writer will have to pick unique names per task-attempt (using the attemptid, say attempt_200709221812_0001_m_000000_0), not just per task.
To avoid these issues the MapReduce framework, when the OutputCommitter is FileOutputCommitter, maintains a special ${mapreduce.output.fileoutputformat.outputdir}/_temporary/_${taskid} sub-directory accessible via ${mapreduce.task.output.dir} for each task-attempt on the FileSystem where the output of the task-attempt is stored. On successful completion of the task-attempt, the files in the${mapreduce.output.fileoutputformat.outputdir}/_temporary/_${taskid} (only) are *promoted* to ${mapreduce.output.fileoutputformat.outputdir}. Of course, the framework discards the sub-directory of unsuccessful task-attempts. This process is completely transparent to the application.
The application-writer can take advantage of this feature by creating any side-files required in ${mapreduce.task.output.dir} during execution of a task via FileOutputFormat.getWorkOutputPath(Conext), and the framework will promote them similarly for succesful task-attempts, thus eliminating the need to pick unique paths per task-attempt.
Note: The value of ${mapreduce.task.output.dir} during execution of a particular task-attempt is actually ${mapreduce.output.fileoutputformat.outputdir}/_temporary/_{$taskid}, and this value is set by the MapReduce framework. So, just create any side-files in the path returned by FileOutputFormat.getWorkOutputPath(Conext) from MapReduce task to take advantage of this feature.
The entire discussion holds true for maps of jobs with reducer=NONE (i.e. 0 reduces) since output of the map, in that case, goes directly to HDFS.

Hadoop RecordWriter

RecordWriter écrit les paires de <key, value> dans un fichier de sortie.

Le recordwrite à la responsabilité d'écrire les résultats du Job sur le système de fichier.

Exemple de formats existants

SequenceFileInputFormat / SequenceFileOutPutFormat:

- Format binaire de stockage des <clé,valeur> beaucoup plus performant que le format texte. A privilégier pour stocker vos données dans HDFS.

TableOutputFormat / TableInputFormat:

- Format d'entrée sortie pour lire et écrire dans la base de données NoSQL HBASE fournit disponible dans l'écosystème Hadoop.

DBInputFormat/DBOuputFormat

- Format d'entrée sortie vers des bases de données SQL