Data Intensive Computing – Handout 3

Training Cluster

You can login to the training cluster *dlrc* via machine `ufallab.ms.mff.cuni.cz` and port 11422, i.e., using `ssh -p 11422 ufallab.ms.mff.cuni.cz`.

All machines in the cluster use same architecture and share `/dlrc_share` directory.

The cluster consists of master `dlrc` and 5 nodes `dlrc-node1` to `dlrc-node5`. Each node has 2 cores and 4GB ram, but four SGE jobs are allowed to run simultaneously (only because otherwise there would be too little slots; the slot number may be even increased later).

Splitting Large Files on Lines

To help you with splitting large files on lines, you can use supplied `split_file.py` script from `/dlrc_share/data/wiki/split/` directory. When executed as

```
split_file.py input_file start_offset length
```

only lines in the file part starting at offset `start_offset` and `length` are printed (allowing the last line to end outside the file part).

The `split_file.py` and also `split_file.pm` can be used also as modules in your Python and Perl scripts. In that case, you can use the method

```
next_file(file, start_offset, length)
```

which sequentially returns lines in the file part (including newline characters), returning empty string at the end.

Exercises for using Sun Grid Engine

Wikipedia Data

In the `/dlrc_share/data/wiki/` there are following data:

- `cs-text/cswiki.txt`: Czech Wikipedia data (Sep 2009), file size 195MB, 124k articles.
- `en-text/enwiki.txt`: English Wikipedia data (Sep 2009), File size 4.9GB, 2.9M articles.

Both files are encoded in UTF-8 and contain one particle per line. Article name is separated by `\t` character from the article content.

Example Solution of a Simple Distributed Task

Consider simple example of producing sorted list of article names. The example solution consists of two sources.

The source `articles.sh` is the main driver for distributing work and collecting results:

```
#!/bin/bash

set -e

# Parse arguments
[ "$#" -ge 3 ] || { echo Usage: $0 input outdir tasks \[conc_tasks\] >&2;exit 1;}
input_file="$1"
output_dir="$2"
tasks="$3"
conc_tasks="$4"

# Check that input file exists and get its size
[ -f "$input_file" ] || { echo File $input_file does not exist >&2; exit 1; }
input_size=`stat -c%s "$input_file"`

# Check that output dir does not exist and create it
[ -d "$output_dir" ] && { echo Directory $output_dir already exists >&2; exit 1;}
mkdir -p "$output_dir"

# Compute file split sizes
split_size=$((1 + ($input_size / $tasks)))

# Run distributed computations
qsub -cwd -b y -sync y -V -t 1-"$tasks" ${conc_tasks:+-tc $concurrent_tasks} \
 ./articles_distributed.sh "$input_file" "$split_size" "$output_dir"/articles.txt

# Merge all results
sort -m `seq -f "$output_dir/articles.txt.%g" 1 "$tasks"` \
  > "$output_dir"/articles.txt
```

The source `articles_distributed.sh` is the helper script executed distributively on the nodes:

```
#!/bin/bash

set -e

# Parse arguments
[ "$#" -ge 3 ] || { echo Usage: $0 input split_length output_file >&2; exit 1;}
input_file="$1"
split_length="$2"
output_file="$3"

# Parse SGE_TASK_ID and compute file offset
[ -n "$SGE_TASK_ID" ] || { echo Variable SGE_TASK_ID is not set >&2; exit 1; }
task="$SGE_TASK_ID"
input_offset=$((($task - 1) * $split_length))
output_file="$output_file.$task"

# Run computation outputting to temporary file
tmp_file="`mktemp`"
trap "rm -f \"$tmp_file\"" EXIT

/dlrc_share/data/wiki/split/split_file.py "$input_file" "$input_offset" \
  "$split_length" | cut -f1 | sort > "$tmp_file"

# On success move temporary file to output
mv "$tmp_file" "$output_file"
```

Tasks

Solve the following tasks. Solution for each task is a source code processing the Wikipedia source data and producing required results, while utilizing distributed computation. The solution does not need to recover when one of the computation fails, but it should fail as a whole.

Task	*Points*	*Description*
`unique_words`	2	Create a list of unique words used in the articles. Convert them to lowercase to ignore case. Because the article data is not tokenized, use provided `/dlrc_share/data/wiki/tokenizer/{cs,en}_tokenizer`, which reads untokenized UTF-8 text from standard input and produces tokenized UTF-8 text on standard output. It preserves line breaks and separates tokens on each line by exactly one space.
`inverted_index`	3	Compute inverted index – for every lowercased word from the articles, compute ascending *(article id, ascending positions of occurrences as word indices)* pairs. In order to do so, number the articles using consecutive integers and produce also a list of articles representing this mapping (the article on line i is the article with id i). The output should be a file with list of articles ordered by article id, and a file with one word on a line in the following format: `word \t article_id \t space separated occurrences...` Once again use provided tokenizer.
`wordsim_index`	3	In order to implement word similarity search, compute for each lemma with at least three occurrences all *contexts* in which it occurs, including their number of occurrences. List the contexts in ascending order. Given N (either 1, 2, 3 or 4), the *context* of a lemma occurrence is N lemmas preceding this occurrence and N lemmas following this occurrence (ignore sentence boundaries, use empty words when article boundaries are reached). To compute the lemmas for a given article, use provided `/dlrc_share/data/wiki/lemmatizer/{cs,en}_lemmatizer`, which works just like the tokenizer – it reads untokenized UTF-8 text from standard input and produces tokenized and lemmatized UTF-8 text on standard output, each lemma separated by exactly one space. The output should be a file with one lemma on a line in the following format: `lemma \t context \t counts...`
`wordsim_find`	2	Let S be given natural number. Using the index created in `wordsim_index`, find for each lemma S most similar lemmas. The similarity of two lemmas is computed using *cosine similarity* as $\frac{C_A \cdot C_B}{\lvert C_A\rvert \cdot \lvert C_B\rvert}$, where C_L is a vector of occurrences of lemma L contexts. The output shold be a file with one lemma on a line in the following format: `lemma \t most similar lemma \t cosine similarity...`