1 Introduction

In this assignment you will reuse the components that you built in `PA1, PA2`.

The assignment deals with the building of a higher performance search engine. In general data on search engines reside on multiple machines. Therefore the processing done with `pa1, pa2` usually involve multiple machines. This is the objective of `PA4`. To implement parallel (high performance) search engine components.

2 Deliverables

The relevant files will become available in `homework/pa4` as described in Handout 2. A single executable file will be the result of your compilable source code into the form of a file named `pa4`. The file will be an executable constructed using the LAM MPI library tools. However it should be possible to reuse your own code written in other language(s). In a UNIX environment this can be accomplished through the C function call `system`. More elaborate uses involve `execve` etc.

3 Part 1 : Simple action parallelization (50 points)

The LAM-MPI executable file `pa4` will read the command line and behave as follows.

```
% mpirun -np number  ./pa4 token       some-name
% mpirun -np number  ./pa4 debug-token some-name
% mpirun -np number  ./pa4 stopwords   some-name
% mpirun -np number  ./pa4 invert      some-name
```

The value of `some-name` is that explained in `PA2`, Part 1. Ignoring the MPI-related part of command line arguments, the first argument in the command line involves actions related to `PA1, PA2`. However these actions occur on the same-named file or directory of a `number` of machines (PCs) and that directory might contain different files/subdirectories. For example if `some-name` might contain `alexg` on one machine and say `some-other-login` on some other machine.

The side-effects remain the same.

4 Part 2 : Simple query parallelization (30 points)

```
% mpirun -np number  ./pa4 AND term1 term2 some-name
```

This is `PA2`, Part 3 on multiple machines. One difficulty that might arise however is dealing with words that have different `wordID`s on different machines. The same applies to `docID`s, doclists etc. These are easily fixable; however the difficult or more complicated part might be the communication of this information to the host machine for further printing/processing.

5 Comments

All output will be done by a single machine that names `0` by the MPI-based process numbering scheme. This will be in most cases the local machines which you used to issue the `mpirun` command.