Common failures to be avoided when we analyze Wikipedia public data

Felipe Ortega
GsyC/Libresoft

Wikimania 2010, Gdańsk, Poland.

#1 Beware of special types of pages

- Official page count includes **articles** with just **one link**.
  - You must consider if you need to filter out **disambiguation** pages.
- Pay attention to **redirects**.
  - Sometimes people wonder how the number of pages in main namespace in the dump is so high.
- Break down evolution trends by **namespace**.
  - Articles are very different from other pages.
  - Explore % of already existing talk pages.
  - Connections from user pages.

#2 Plan your hardware carefully

- There are some general rules.
  - Parallelize as much as possible.
  - Buy more memory before buying more disk...
  - But take a look at your disk requirements.
    - It's very different when you can work on decompressed data, on the fly.
  - Hardware RAID is not always the best solution.
    - RAID 10 in Linux can perform decently in many average studies.

3# Know your engines (DBs)

- Correct configuration of DB engine is crucial.
  - You'll always fall short with standard configs.
  - Fine tune parameters according to your hardware.
  - Exploit memory as much as possible.
    - E.g. MEMORY engine in MySQL.
  - Avoid unnecessary backup...
  - But be sure that you have copies of relevant info elseware!
  - Think about your process:
    - Read only vs. read-write.

4# Organize your code

- Using a SCM is a must.
  - SVN, GIT.
- Upload your code to public repository.
  - BerliOS, SourceForge, GitHub...
- Document your code...
  - ...if you ever aspire to get interest from other developers.
- Use consistent version numbers.
- Test, test, test...
  - Include sanity checks and “toy tests”.

#5 Use the right "spell"

- **Target data** is well defined:
  - XML
  - Big portions of plain text
  - Inter-wiki links and outlinks.
- Some alternatives
  - CelementTree (high-speed parsing)
  - Python (modules/short scripts) or Java (big projects).
  - Perl (regexps).
  - Sed & awk

#6 Avoid reinventing the wheel

- Consider to develop only if:
  - No available solution fits your needs.
    - Or you can only find proprietary/evaluation sofwtare.
  - Performance of other solutions is really bad
- Example: **pywikipediabot**
  - Simple library to query Wikipedia API.
  - Solves many simple needs of researchers/programmers.

#7 Automate everything

- Huge data repositories.
- Even small samples are excessively time consuming if processed by hand.
- You will start to concat individual processes.
- You will save time for later executions.
- Your study will be **reproducible**.
  - Updating results after several months becomes no-brainer solution.

#8 Extreme case of Murphy's Law

- Always expect the **worst possible** case.
  - Many caveats in each implementation.
  - Countless particular cases.
  - It's not OK with just the “average solution”.
    - Standard algorithms may take much more than expected to finish the job.

#9 Not many graphical interfaces

- Some good reasons for that
  - Difficult to automate
  - Hard to display dynamic results in real-time.
  - Almost impossible to compute all results in a reasonable time frame for *huge* data collections (e.g. English Wikipedia).
- To the best of my knowledge, there are very few tools with graphical interfaces out there.
- Is there a real need for that??

#10 Communication channels

- Wikimedia-research-l
  - Mailing list about research on Wikimedia projects.
  - http://meta.wikimedia.org/wiki/Research
  - http://meta.wikimedia.org/wiki/Wikimedia_Research_Network
  - http://acawiki.org/Home
- Final comments
  - Need for consolidated info point, once for all