eHive Workshop

part 1. How to run and tune pipelines

Leo Gordon

Pipeline and Analysis

- Pipeline is a distributed computation process.

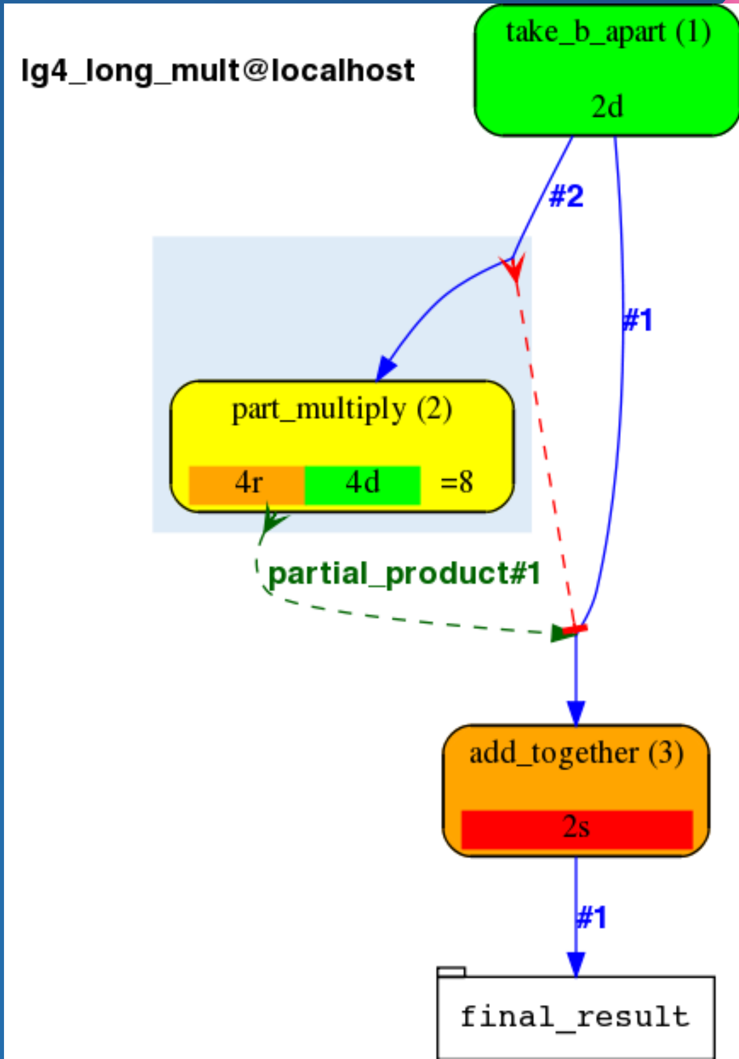
  Parts of the computation are connected to other parts via dependency rules.

- eHive system both sets up these computation processes and runs them.

- Analyses as "big steps" or “stages” of pipelines. *Some may be done, some in progress, some have not started yet*.

- Analyses as “containers” or “templates” for Jobs

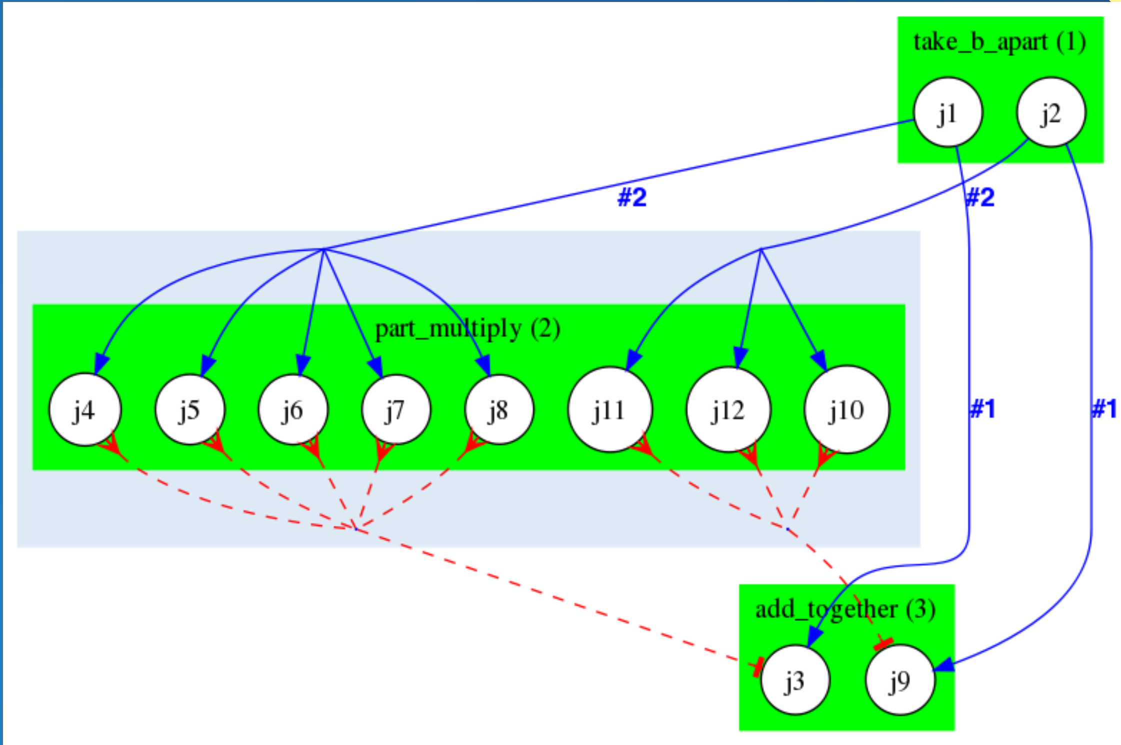
- What are Jobs?



Jobs and Dataflow

- Jobs are individual units of computation that can be run.
- A Job is defined by individual parameters and the Analysis it belongs to.
- Jobs share whatever they inherit from their Analysis.

- Dataflow - a universal messaging mechanism central to Hive.
- Jobs send out parameterized Dataflow Events
- Depending on Dataflow Rules (shared between Jobs of the same Analysis) these Events may:
  - do nothing (default)
  - create new Jobs
  - store data in tables
  - send data to other Jobs
  - establish complex Job dependency patterns



Obtaining and installing eHive code

[http://www.ensembl.org/info/docs/eHive/installation.html]

Currently main dependency is EnsEMBL Core API, but strangely not BioPerl

```
$ export ENSEMBL_CVS_ROOT_DIR="$HOME/ensembl_main"
$ cd $ENSEMBL_CVS_ROOT_DIR        # assuming it exists, mkdir & cd otherwise

$ cvs -d :pserver:cvsuser@cvs.sanger.ac.uk:/cvsroot/ensembl login
Logging in to :pserver:cvsuser@cvs.sanger.ac.uk:2401/cvsroot/ensembl
CVS password: CVSUSER

$ cvs -d :pserver:cvsuser@cvs.sanger.ac.uk:/cvsroot/ensembl checkout ensembl
$ cvs -d :pserver:cvsuser@cvs.sanger.ac.uk:/cvsroot/ensembl checkout ensembl-
hive
```

```
#----------------------[~/.bash_profile]--------------------------------

export PERL5LIB=${PERL5LIB}:${ENSEMBL_CVS_ROOT_DIR}/ensembl/modules

    # no longer necessary for just running, but easier for development:
export PERL5LIB=${PERL5LIB}:${ENSEMBL_CVS_ROOT_DIR}/ensembl-hive/modules

    # simply convenient; all Hive scripts live here:
export PATH=$PATH:$ENSEMBL_CVS_ROOT_DIR/ensembl-hive/scripts
```

A quick test of your code/environment setup

- `$ db_cmd.pl -url mysql://anonymous@ensembldb.ensembl.org/`

  (should connect the mysql client to the public ensembl server)

- `$ perldoc Bio::EnsEMBL::Hive::PipeConfig::LongMult_conf`

  (should show the POD of the pipeline)

- `$ tree -Ad $ENSEMBL_CVS_ROOT_DIR/ensembl-hive`

  (should show the content of the next slide)

Directory structure of eHive code

$ENSEMBL_CVS_ROOT_DIR/ensembl-hive contains:

```
.
|-- ./docs
|   `-- ./docs/presentations
|-- ./modules
|   `-- ./modules/Bio
|       `-- ./modules/Bio/EnsEMBL
|           `-- ./modules/Bio/EnsEMBL/Hive
|               |-- ./modules/Bio/EnsEMBL/Hive/DBSQL
|               |-- ./modules/Bio/EnsEMBL/Hive/Meadow
|               |-- ./modules/Bio/EnsEMBL/Hive/PipeConfig
|               |-- ./modules/Bio/EnsEMBL/Hive/RunnableDB
|               |   `-- ./modules/Bio/EnsEMBL/Hive/RunnableDB/LongMult
|               `-- ./modules/Bio/EnsEMBL/Hive/Utils
|-- ./scripts
`-- ./sql
```

MySQL server setup for the workshop

A Hive pipeline is centred around a database.
To be able to create and/or run the pipeline you will need a running server and know the connection parameters including the password.

- In this workshop we will be using the following connection parameters:

```
Driver:      MySQL (supported alternatives: SQLite and PostgreSQL)
Host:        mysql-ehive-training.ebi.ac.uk
Port:        4427
Username:    train02..train19
Password:    psw02..psw19
```

- Number off, please :)

- Test your connection to the server:

```
$ db_cmd.pl -url mysql://trainXX:pswXX@mysql-ehive-training.ebi.ac.uk:4427/
```

- `CREATE DATABASE / DROP DATABASE`
  Make sure you prefix database names with your username

- You will soon notice the ubiquity of database URLs.
  This is the main way of representing database connection parameters.
  Virtually all Hive tools use URLs for connecting to databases.

Pipeline database creation

- Try running:

```
$ init_pipeline.pl Bio::EnsEMBL::Hive::PipeConfig::LongMult_conf \
    -host mysql-ehive-training -port 4427 -user trainXX -dbowner trainXX
```

- Missing parameters? A pipeline typically has several parameters that can be supplied from ***init_pipeline.pl*** command line. Some are optional, some are not. They will affect the database creation and will "stay" in the pipeline in one form or another.

- Supply the password and run ***init_pipeline.pl*** successfully.
After creating the pipeline it should print the list of suggested commands which is a good reference point of what to do next.

- If you tend to use the same username and password all the time, you can set them in $EHIVE_USER and $EHIVE_PASS environment variables, but make sure this is done securely (either .bash_profile or stuff that you source from there is not readable by the world).

Peeking into the pipeline

- Non-interactive way - run
  generate_graph.pl -url <URL> -out diagram.png
  and see the initial graph with only initial jobs in READY state.

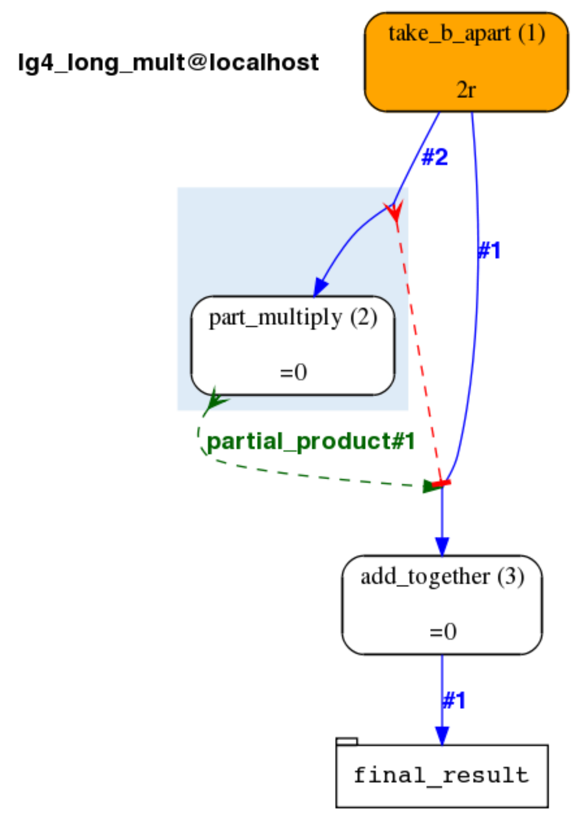
- Interactive way - open
  - http://guihive.ebi.ac.uk:8080/
  - http://guihive.sanger.ac.uk:8080/

  and paste in the pipeline URL.

- Power-user way - run
  db_cmd.pl -url <URL>
  and look into the tables.

Hive database schema diagram:

$ENSEMBL_CVS_ROOT_DIR/
ensembl-hive/docs/hive_schema.png



Workers : independent agents running the pipeline

- The actual running of the pipeline's Jobs is done by Workers.
  They are independent agents (processes on the farm) that are synchronized only through the Hive database that acts as a blackboard.

- Workers are the only "doers" in the whole Hive system,
  and until some Workers are created no computation is performed.

- By default a Worker “specializes” into an analysis and then runs multiple Jobs for about an hour. You can impose various constraints.

```
$ runWorker.pl -url <URL>  # automatic specialization

$ runWorker.pl -url <URL> -logic_name <analysis_name>

$ runWorker.pl -url <URL> -job_id <job_id>
```

- Workers can be run locally or submitted to the farm.
  A typical run of a pipeline is a combination of both.

Let’s peek into the database and run some Workers:

```
$ db_cmd.pl -url <URL>
```

Running individual Workers (2 terminals)

```
SELECT * FROM worker;         -- no workers yet
SELECT * FROM job;            -- initial tasks to be performed
```

```
    $ runWorker.pl -url <URL>
```

```
SELECT * FROM worker;         -- worker#1 specialized into analysis#1
SELECT * FROM job;            -- analysis#1 is a factory, new jobs created
SELECT * FROM accu;           -- accus should be empty
```

```
    $ runWorker.pl -url <URL>
```

```
SELECT * FROM worker;         -- worker#2 specialized into analysis#2
SELECT * FROM job;            -- perform partial multiplications
SELECT * FROM final_result; -- no results yet
SELECT * FROM accu;           -- accus contain data for analysis#3
```

```
    $ runWorker.pl -url <URL>
```

```
SELECT * FROM worker;         -- worker#3 specialized into analysis#3
SELECT * FROM job;            -- perform addition, dataflow into final_result
SELECT * FROM final_result; -- see the results
```

Note they were all local Workers. Re-create the pipeline (we shall run it on the farm) :

```
    $ init_pipeline.pl LongMult_conf.pm -take_time 10 -hive_force_init 1
```

Creating extra tasks by seeding.

- Jobs are created:
  - by init_pipeline.pl when the Pipeline database is initialized (pipeline’s main entry point(s))
  - by running seed_pipeline.pl script at any moment (high-level calls to the whole pipeline's logic)
  - dynamically by the pipeline itself when the Dataflow Rules are activated (functions calling other functions)

```
$ seed_pipeline.pl -url <URL>

$ seed_pipeline.pl -url <URL> -analysis_id 1 \
  -input_id '{"a_multiplier" => "96966905521","b_multiplier" => 327358788}'

$ seed_pipeline.pl -url <URL> -logic_name take_b_apart \
  -input_id '{"a_multiplier" => "96966905521","b_multiplier" => 327358788}'
```

- Not all pipelines are designed to be dynamically seeded, but some are. Watch out for control rules - solid red “inhibitor” arcs on diagrams.

- Seeding may not make sense for some analyses. Watch out for:
  - ➢ dataflow into the analysis
  - ➢ accumulated dataflow into the analysis

Submitting Workers to the farm manually

- Optionally re-create the pipeline database :

```
$ init_pipeline.pl LongMult_conf.pm -take_time 10 -hive_force_init 1
```

- Run one factory Worker locally (it is special):

```
$ runWorker.pl -url <URL>
```

- Submit some Workers to the farm (run a few of these):

```
$ bsub -o /dev/null -e /dev/null  runWorker.pl -url <URL>
$ bsub -o /dev/null -e /dev/null  runWorker.pl -url <URL>
```

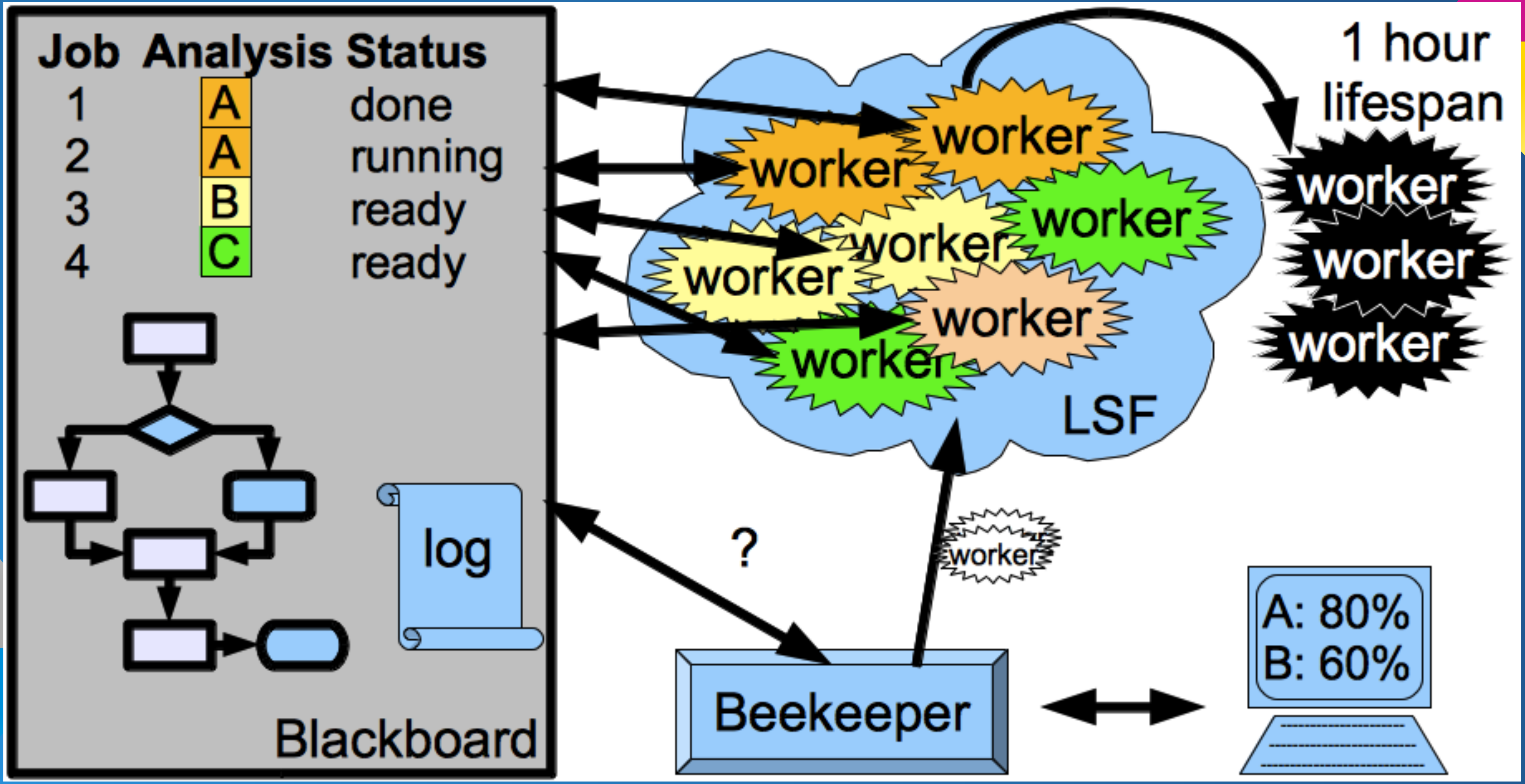
- Check the status of your submission:

```
$ bjobs -w
```

- Any difference between Workers in the worker table?
- Any confusion between Hive Jobs and LSF Jobs?

Automatic scheduling of Workers

- Submitting individual Workers to the farm is tedious.
  It has to be done in time, and the number of Workers has to be right.
  Since we don't want to babysit our pipelines all day/week/month,
  we have a dedicated "scheduler".



Using Beekeeper to schedule Workers

- ***`beekeeper.pl`*** is our main "scheduling" script that drives the execution of the pipeline. Submitting the right number of Workers performing our pipeline is the only task of ***`beekeeper.pl`*** , it does not by itself perform any computation. Nor does it keep any connection to the Workers.

- What happens when we kill the ***`beekeeper.pl`*** ?

Let’s re-initialize our database:

```
$ init_pipeline.pl LongMult_conf.pm … -hive_force_init 1
```

and start the Beekeeper:

```
$ beekeeper.pl -url <URL> -loop
```

Note that Beekeeper can create local Workers even on the farm (configurable).

Some practical aspects of running the Beekeeper

- ***$ screen***
  In real, practical applications we want our Beekeeper session to stay alive for longer periods. But keeping an SSH connection open to a farm head node is not always practical or possible.
  Use “screen” as a disconnectable/reconnectable session manager.

- `SELECT * FROM progress;`
  “progress” is a dynamic view over multiple tables that gives an overview of how many jobs are in which states.
  It gives the freshest data available, but is relatively expensive to run.
  Try to avoid using during times of db congestion.

- ***$ beekeeper.pl -dead***
  “Garbage collection” of Workers that died (MEMLIMIT, RUNLIMIT, etc)

- ***$ beekeeper.pl -sync***
  Keeping analysis_stats counters up to date with “progress”.
  Is run automatically from time to time, but may need to be run manually if statuses of individual Jobs changed.

Debugging : surprise

- Things can go wrong even in a well-tested pipeline, e.g. on a new farm:
    - file systems or individual files have moved or disappeared
    - binaries/libraries need to be recompiled, work differently
    - person running the pipeline may not have right file permissions
    - input data from external collaborators wrongly formatted
    - *somebody* checked in untested code

- Things to watch out for:
    - failing analyses (obvious, ***`beekeeper.pl`*** will probably stop)
    - failing individual jobs (check guiHive, progress or BK output)
    - failing attempts (retry_count>0).
      "Transient errors" and why retry may be useful.
    - nothing running on the farm (although it may be busy)

- Stop the pump?
    - stop the ***`beekeeper.pl`*** process (we know it is safe)
    - stop the failing analyses by setting their analysis_capacity to 0
    - ***`beekeeper.pl -dead`*** and ***`beekeeper.pl -sync`***
      to have a more up-to-date picture of what happened;
        - note the time of running ***`-dead`***

Debugging : process

- Which Jobs/Analyses were affected?

```
SELECT * FROM progress;
```

- Warnings and “last breath” messages from specific Jobs/Attempts :

```
SELECT * FROM msg;
```

Only failures from Perl layer may be captured in `log_message / msg` .

- Run a Worker with just one job (locally or on the farm) :

```
$ runWorker.pl -url <URL> -job_id <failed_job> [ -debug 1 ]
```

(among other things ***-debug 1*** protects the "/tmp directory of the process" from removal)

- Capturing STDOUT/STDERR of Workers (per Job) :

```
$ runWorker.pl -url <URL> -worker_log_dir log_this_worker/
$ runWorker.pl -url <URL> -hive_log_dir log_entire_hive/
$ beekeeper.pl -url <URL> -hive_log_dir log_entire_hive/
```

How to reach specific dump files - reverse the worker_id

Debugging : success

- Clean up after your pet (files, entries in database tables, logs, etc)
- Restarting failed jobs :

```
$ beekeeper.pl -url … -reset_failed_jobs_for_analysis <failed_analysis>
```

- Unlock the analyses that you have stopped (analysis_capacity := N)
- run ***`beekeeper.pl -loop`*** again

Tuning the pipeline

- Changing non-structural parameters of the pipeline (separate Analyses) to make it run more efficiently.

- Experimental change in the DB vs permanent change in PipeConfig.

- max number of Workers of this Analysis that can run in parallel :
  ```
      -analysis_capacity => <number_of_Workers>
  ```
  (make sure the database backend is not overloaded)

- number of Jobs of this analysis that Workers can claim in one go :
  ```
      -batch_size => <number_of_Jobs>
  ```
  (increase if average job.runtime_msec or analysis_stats.avg_msec_per_job is too low)

- how many times to attempt running a Job of this Analysis :
  ```
      -max_retry_count => <number_of_Attempts>
  ```
  (may increase for analyses flowing to #-1 and failing)

- a reciprocal formula for limiting the number of Workers of the whole Hive :
  ```
      -hive_capacity => <number_of_Workers>
  ```
  (time permitting)

- per-Analysis resources (space+time) :
  ```
      -rc_name => <resource_class_name>
  ```

Resource requirements (Time+Memory)

- If we know how much resources each analysis needs,
  we can tune our resource requirements, be nice to other users
  and even potentially speed things up.

- Resource requirements are very much farm-dependent :
  - “meadow_type” - LSF/SGE/Condor
  - 32bit vs 64bit farms
  - static/dynamic linking
  - specific farm tuning idiosyncrasies (measuring megabytes in kilobytes)

  but within the same farm things are usually tunable.

- Each Analysis has an associated resource_class, which maps to a specific "resource line" :

```
'1Gb'=> { 'LSF' => '-q normal -C0 -M1000 -R"select[mem>1000] rusage[mem=1000]"'
},
```

- resource_class names allow us to separate the logic from implementation;
  you may need to change the implementation for each farm.

- In case of on-Campus farms usually queue (time-limited), memory and number of required database connections can be set.

Resource estimation - Timing

- Time is kept by the Hive internally and doesn't depend on where you run your Workers (LOCAL, LSF, SGE,...)

- timing Jobs:

```
SELECT * FROM job; -- includes 'runtime_msec' field

SELECT MIN(runtime_msec),
       AVG(runtime_msec),
       MAX(runtime_msec)
FROM job GROUP BY analysis_id;
```

- timing whole Analyses
  (from the birth of the first Worker to the death of the last Worker) :

```
CALL time_analysis('%blast%'); -- use patterns creatively
```

- Exercise:
  what was the longest running Analysis of LongMult pipeline?

Resource estimation - Memory

- Memory measurements can only be done on the farm.
  Since we mainly use LSF, our scripts/views are LSF-specific.

- At any moment during or after running the pipeline you can run:
  ```
  $ lsf_report.pl -url <URL>
  ```
  (optionally change the user or constrain the time interval)

- The script parses LSF's log entries that match the pipeline's Workers and loads the memory usage stats into the database.
  It can be refreshed at a later stage.

- Get a nice per-Analysis report by selecting from a view :
  ```
  SELECT * FROM lsf_usage;
  ```
  both memory and swap units are megabytes.

- Exercise: find memory usage stats of LongMult pipeline

Resources : case study

- Run the MemlimitTest pipeline on the farm:

```
$ init_pipeline.pl MemlimitTest_conf.pm <connection parameters>
```

- What happened to some Workers?

- How much memory is actually needed?
  - NB: EBI farm and the mysterious x1000 multiplier

- Exercise:

Fix the ***MemlimitTest_conf.pm*** file so that the pipeline would run to the end.

End of part 1. Acknowledgements

Matthieu Muffato

Current and previous members of Compara team

All users of eHive system for testing, feedback and ideas

Paul Flicek, Steve Searle and the entire Ensembl Team

Funding

wellcome trust

EMBL

National Human Genome Research Institute

BBSRC bioscience for the future

European Commission Framework Programme 7

SEVENTH FRAMEWORK PROGRAMME

Quantomics

From Sequence to Consequence : Tools for the Exploitation of Livestock Genomes

GEN2PHEN

BLUEPRINT epigenome