eHive Workshop

part 2: How to run, troubleshoot and tune pipelines

Instructors:
Leo Gordon and Brandon Walts

Obtaining and installing eHive code

(for reference - this has already been done in your virtual machine)

```
$ export ENSEMBL_REPO_ROOT_DIR=$HOME/ensembl-api-folder
$ cd $ENSEMBL_REPO_ROOT_DIR        # assuming it exists; mkdir & cd otherwise

$ git clone https://github.com/Ensembl/ensembl-hive.git
$ cd ensembl-hive

# one way to install the dependencies listed in cpanfile. consult your system
# administrator for the best way to install Perl modules on your system.
$ cpanm -L $HOME/my_userspace_perl_libraries --installdeps .
```

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

export ENSEMBL_REPO_ROOT_DIR=$HOME/ensembl-api-folder   # specific to your setup!

# not necessary for just running, but makes development easier:
export PERL5LIB=${PERL5LIB}:$ENSEMBL_REPO_ROOT_DIR/ensembl-hive/modules

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

eHive support

Documentation at https://rawgit.com/Ensembl/ensembl-hive/HEAD/docs/index.html
A new user manual is scheduled for later this year.

email list: **ehive-users@ebi.ac.uk**

A couple of quick setup steps for your virtual machine

Here are a few things you should do now to get your virtual machine prepared for this course.
Lines starting with # are comments to explain the purpose
Lines starting with $ should be typed at the command prompt (type everything after the $)
Lines starting with > should be typed at the database prompt (type everything after the >)

```
# patch guiHive
$ cd $ENSEMBL_REPO_ROOT_DIR/guiHive
$ ./guihive-deploy.sh
# press return at the warning "./versions and/or ./ensembl-hive already exist"

# add a utility called 'tree' - useful for viewing directory structures
$ sudo apt-get -y install tree

# get the latest commits from git
$ cd $ENSEMBL_REPO_ROOT_DIR/ensembl-hive
$ git pull
```

A couple of quick setup steps for your virtual machine

Here are a few things you should do now to get your virtual machine prepared for this course.
Lines starting with # are comments to explain the purpose
Lines starting with $ should be typed at the command prompt (type everything after the $)
Lines starting with > should be typed at the database prompt (type everything after the >)

```
# create a directory to keep our work organized
$ cd $HOME
$ mkdir ehive-course
$ cd ehive-course

# download an example dataset to use
$ wget http://www.ebi.ac.uk/~bwalts/workshop_scripts/example_input_2.fa
# download solutions for some of the exercises
$ wget http://www.ebi.ac.uk/~bwalts/workshop_scripts/solutions3.tgz
$ wget http://www.ebi.ac.uk/~bwalts/workshop_scripts/solutions4.tgz
```

Directory structure of eHive code

`$ENSEMBL_REPO_ROOT_DIR/ensembl-hive` contains:

```
.
├── docs                                documentation
├── modules
│   └── Bio
│       └── EnsEMBL
│           └── Hive                   code that runs eHive
│               ├── DBSQL              code for eHive database connections
│               ├── Examples           eHive pipeline example code
│               ├── Meadow             code eHive uses to run on compute farms
│               ├── PipeConfig         base code for PipeConfig files
│               ├── RunnableDB         utility runnables
│               ├── Scripts            code for eHive scripts
│               └── Utils              eHive utility modules
├── scripts                            scripts for running eHive pipelnes
├── sql                                sql to build hive databases
├── t                                  automated tests
└── wrappers                           interface for Python and Java
```

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.

- Your VM has a MySQL server installed and running:

```
• Driver:       MySQL
• Host:         localhost
• Port:         3306 (default)
• Usernames:    ensro (read-only) and ensrw (read-write)
• Password for ensrw:   ensrw_password
```

- Test your connection to the server:

```
$ db_cmd.pl -url mysql://ensrw:ensrw_password@localhost/
```

- CREATE DATABASE / DROP DATABASE

```
> CREATE DATABASE db_create_drop_test;
> DROP DATABASE db_create_drop_test;
> quit;
```

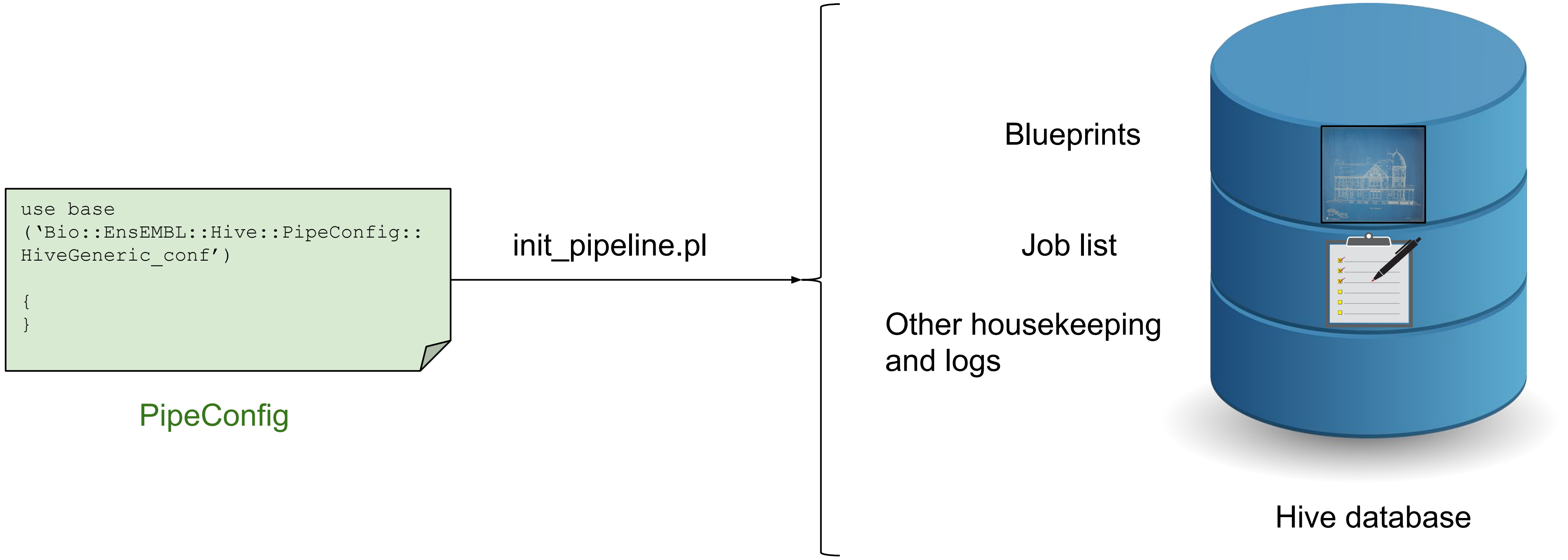
- Database URLs are the main way of representing database connection parameters.This is the main way of representing database connection parameters.
- All Hive tools can use URLs for connecting to databases.

Pipeline database creation

- Try running:

```
$ init_pipeline.pl Bio::EnsEMBL::Hive::Examples::GC::PipeConfig::GCPct_conf \
    -pipeline_url mysql://ensrw:ensrw_password@localhost/gcpct_hive_db
```

After creating the pipeline it should print the list of suggested commands



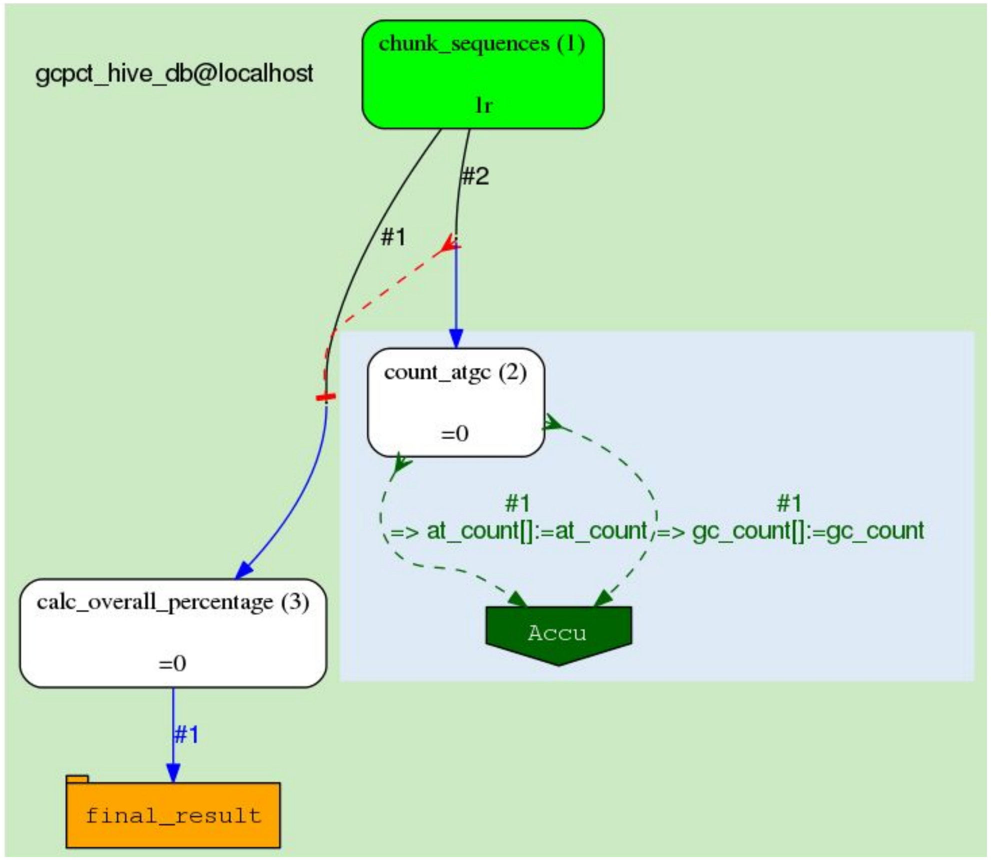
```
use base
('Bio::EnsEMBL::Hive::PipeConfig::
HiveGeneric_conf')

{
}
```

Peeking into the pipeline

- Non-interactive way - type the following

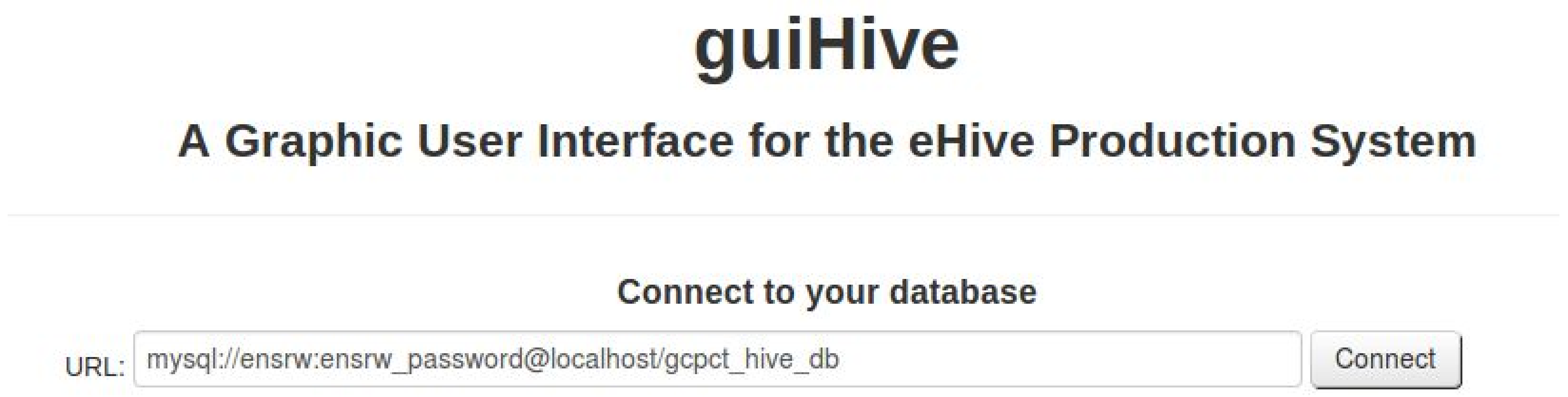
```
$ generate_graph.pl \
    -url mysql://ensro@localhost/gcpct_hive_db \
    -out gcpct_diagram.png
$ xdg-open gcpct_diagram.png
```

and see the initial graph with one initial job in READY state.



Peeking into the pipeline

- Interactive way:
    - Double click on "Start guiHive server"
        - You won't see anything happen, but it should run the server start script
    - Open Firefox (or another web browser)
    - http://127.0.0.1:8080/
    - paste in the pipeline URL



guiHive

A Graphic User Interface for the eHive Production System

Connect to your database

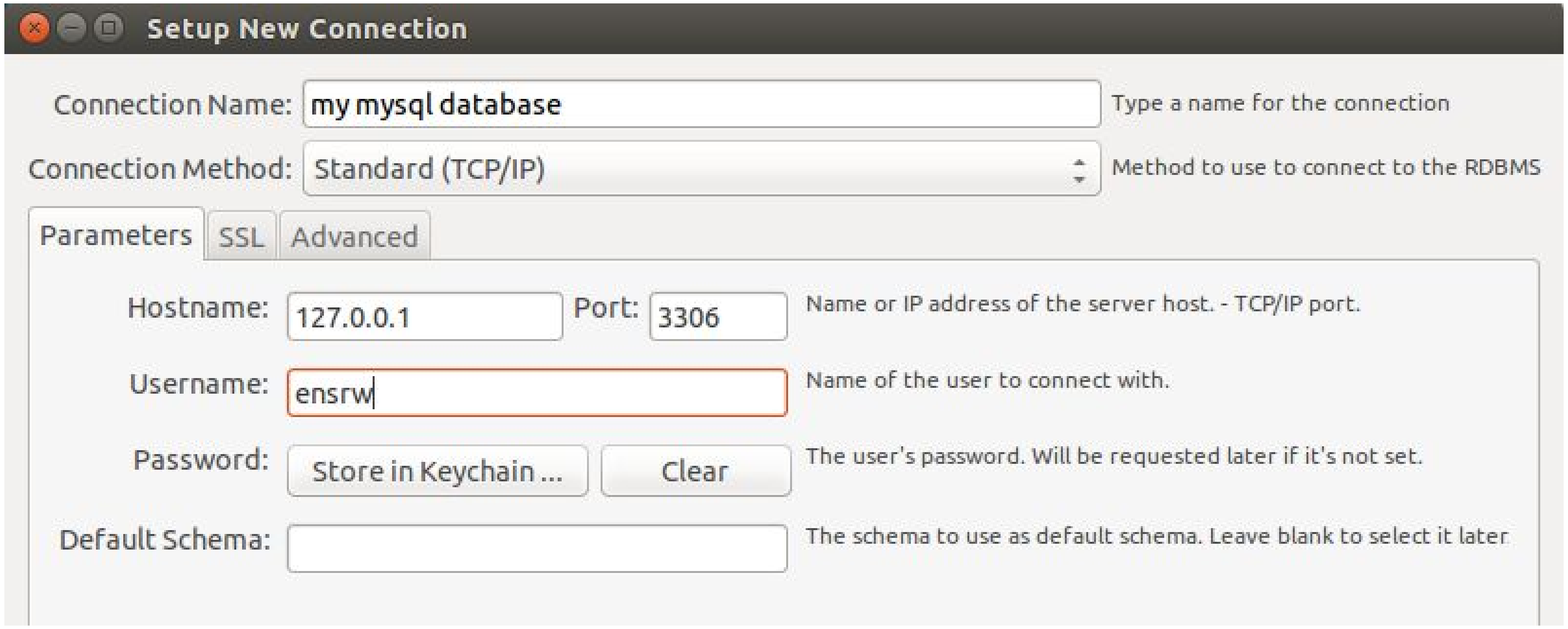
URL: mysql://ensrw:ensrw_password@localhost/gcpct_hive_db

Peeking into the pipeline

- Power-user way:
  - Connect to the database

Using a graphical interface to the database, type:

```
$ mysql-workbench
```



Setup New Connection

Connection Name:	my mysql database	Type a name for the connection
Connection Method:	Standard (TCP/IP)	Method to use to connect to the RDBMS

Hostname:	127.0.0.1 Port: 3306	Name or IP address of the server host. - TCP/IP port.
Username:	ensrw	Name of the user to connect with.
Password:	Store in Keychain ... Clear	The user's password. Will be requested later if it's not set.
Default Schema:		The schema to use as default schema. Leave blank to select it later

Or, to use the mysql command line, type

```
$ db_cmd.pl -url mysql://ensrw:ensrw_password@localhost/gcpct_hive_db
```

Peeking into the pipeline

- Some interesting tables to look at:
    - analysis_base
    - job
    - log_message
    - analysis_stats

Workers : independent agents running the pipeline

- Review: 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.

- By default a Worker “specializes” into an analysis and then runs multiple Jobs for about an hour.
- How do workers get started?
  - One way is to explicitly start them with runWorker.pl

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

- Restrictions can be placed on workers

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

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

$ runWorker.pl -url <URL> -can_respecialize 0 #(default)
```

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

Exercise: Running individual Workers

Let's examine the hive database while we run some workers.
Run the selects in MySQL Workbench, run the runWorker.pl commands on the command line

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

```
    $ runWorker.pl -url mysql://ensrw:ensrw_password@localhost/gcpct_hive_db
```

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

```
    $ runWorker.pl -url mysql://ensrw:ensrw_password@localhost/gcpct_hive_db
```

```
> SELECT * FROM worker;        -- worker#2 specialized into analysis#2 and did 3
jobs
> SELECT * FROM job;           -- performed sequence chunking
> SELECT * FROM final_result;  -- no results yet
> SELECT * FROM accu;          -- accumulators contain data for analysis#3
```

```
    $ runWorker.pl -url mysql://ensrw:ensrw_password@localhost/gcpct_hive_db
```

```
> 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
```

Creating extra tasks by seeding

- Jobs are created:
  - by init_pipeline.pl when the Pipeline database is initialized
    - The initial job(s) have to be explicitly set up in the PipeConfig
  - dynamically by the pipeline itself - workers creating new jobs as they run or finish with a job. (More precisely, this is through activating "dataflow rules" - which we will cover later)
  - by running seed_pipeline.pl script at any moment

- Examples of the seed_pipeline.pl script's syntax

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

$ seed_pipeline.pl -url <URL> -logic_name chunk_sequences \
    -input_id '{"inputfile" => "example_input_2.fa"}'
```

- Not all pipelines are designed to be dynamically seeded, but some are.

Creating extra tasks by seeding

- Let's try seeding a pipeline

- Connect to gcpct_hive_db in mysql-workbench and
  - look at the chunk_sequence analysis, and the seeded chunk_sequence job.

```
> SELECT * FROM analysis_base WHERE logic_name = "chunk_sequences"
> SELECT * FROM jobs WHERE analysis_id = [see previous query, probably 1] ;
```

- Note one chunk_sequences job exists. It's input_id is empty, so its parameters default to the analysis parameters. In analysis_base, the parameters hash includes

```
 "inputfile" => "/home/ensembl/ensembl-api-folder-87/ensembl-hive/t/input_fasta.fa"
```

- Now, seed the pipeline with another job

```
$ seed_pipeline.pl -url mysql://ensrw:ensrw_password@localhost/gcpct_hive_db \
    -logic_name chunk_sequences \
    -input_id '{"inputfile" => "example_input_2.fa"}'
```

- Look at the jobs table after seeding
- Question, what would happen if you did a runWorker using this database now?

Priming the pump: seeding and parameters...

```
$ seed_pipeline.pl -url mysql://ensrw:ensrw_password@localhost/gcpct_hive_db \
    -logic_name chunk_sequences \
    -input_id '{"inputfile" => "example_input_2.fa"}'
```

- We're passing something to the pipeline...
  - Data gets passed around the pipeline as parameters
    - These are analogous to variables in a typical programming language
  - Parameters are assigned values many ways:
    - Jobs fill them in based on the work they do
    - You can fill them in when you seed a job using -input_id
- Why is the input parameters called what they are?
  - Because the person who wrote the Runnable decided to call it that
- How do we know what input parameters to pass?
  - Read the Runnable's documentation/code depending on whether whoever wrote it is a nice/not nice person.

Parameters: order of precedence

- Parameters can be assigned values in many places. The rule of thumb for determining what value a parameter will have is to look at how "local" the parameter is to the job.
- The order of precedence (from highest to lowest -- i.e. if no value is set at the higher precedence level, eHive defaults to a value from the next lower level of precedence):
  - Any value set in a job's input_id has the highest precedence. This means parameter values
    - Passed to the job from a previous job (directly or via an accumulator)
    - Set in the -input_id hash of a seed_pipeline.pl command
    - Set in the input_id section of a PipeConfig file
  - Analysis-wide parameters
  - Pipeline-wide parameters
  - Defaults set in the Runnable's code are the lowest precedence -- these will be used if a value is not set at any of the other levels.

Using tweak_pipeline.pl to make adjustments

- You can adjust a pipeline using tweak_pipeline.pl
  - Change parameters
  - Update resource classes (covered later in the course)
  - Update flow control (change the blueprints)
  - Update analysis_capacity
- Syntax:

```
# set a pipeline-wide parameter called 'take_time' to 20
$ tweak_pipeline.pl -url <URL> -SET 'pipeline.param[take_time]=20'

# show the current value for pipeline-wide parameter called 'take_time'
$ tweak_pipeline.pl -url <URL> -SHOW 'pipeline.param[take_time]'

# delete a parameter entirely
$ tweak_pipeline.pl -url <URL> -DELETE 'pipeline.param[foo]'

# set a parameter for a particular analysis
$ tweak_pipeline.pl -url <URL> \
  -SET 'analysis[chunk_sequences].param[inputfile]=newfile.fa'
```

Using tweak_pipeline.pl to make adjustments

- Exercise: Let's use tweak_pipeline.pl to change the input file for our %GC pipeline

- First, re-initialize the database. Note "-hive_force_init 1" This clears out the database and starts over fresh

```
$ init_pipeline.pl Bio::EnsEMBL::Hive::Examples::GC::PipeConfig::GCPct_conf \
        -pipeline_url mysql://ensrw:ensrw_password@localhost/gcpct_hive_db \
        -hive_force_init 1
```

- Look at the tables job and analysis_base, noting where inputfile is defined

```
> SELECT * FROM job;
> SELECT * FROM analysis_base;
```

- Now, use tweak_pipeline.pl to change the value for the inputfile parameter

```
$ tweak_pipeline.pl -url mysql://ensrw:ensrw_password@localhost/gcpct_hive_db \
  -SET 'analysis[chunk_sequences].param[inputfile]=example_input_2.fa'
```

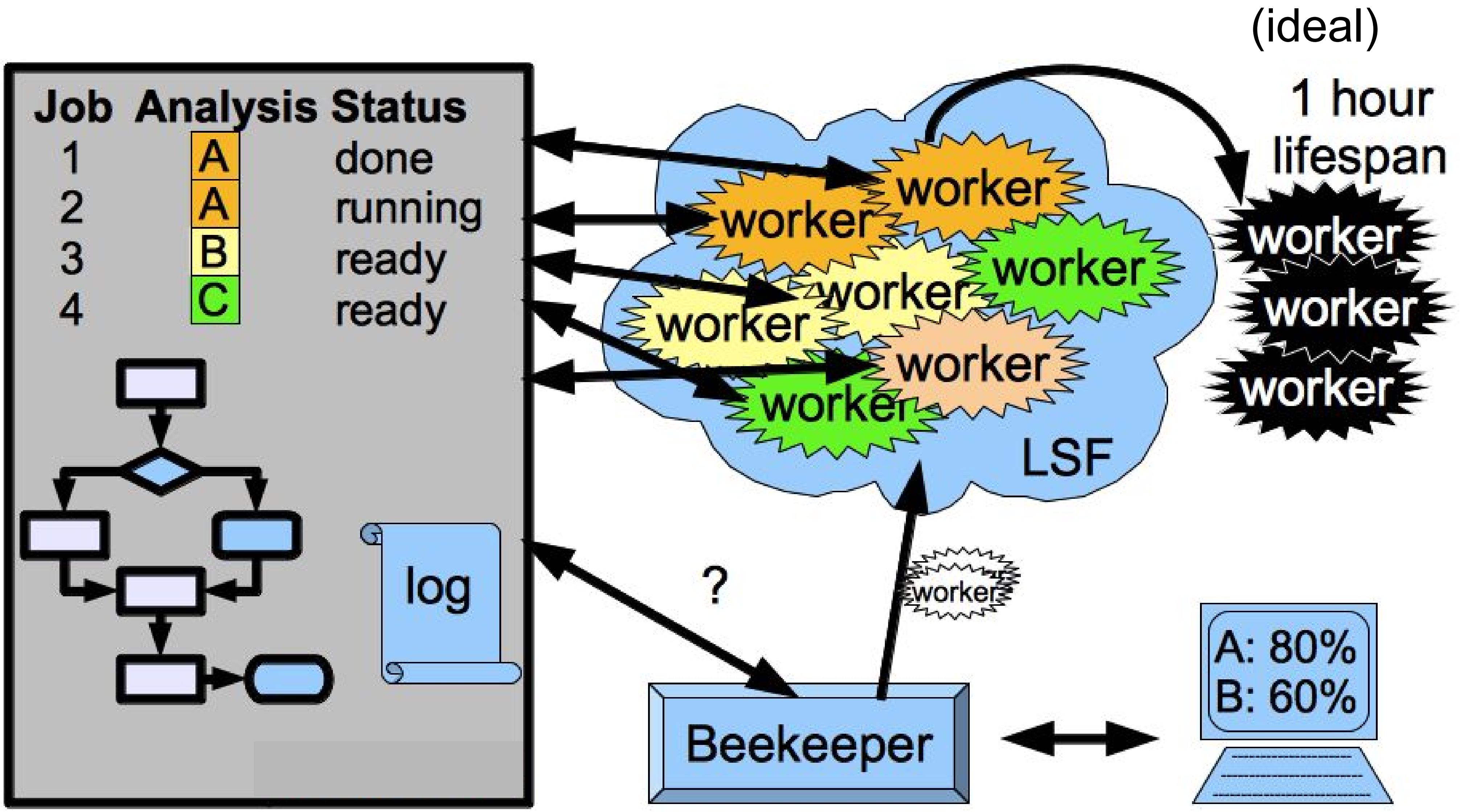
- Look at the new value for "inputfile" in analysis_base. Note, nothing changed in the input_id column of the job table for job 1

```
> SELECT * FROM job;                 # nothing changed?
> SELECT * FROM analysis_base;
```

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.

- Let's re-initialize our database (note -hive_force_init 1):

```
$ init_pipeline.pl Bio::EnsEMBL::Hive::Examples::GC::PipeConfig::GCPct_conf \
    -pipeline_url mysql://ensrw:ensrw_password@localhost/gcpct_hive_db \
     -hive_force_init 1
```

- and start the Beekeeper:

```
$ beekeeper.pl -url mysql://ensrw:ensrw_password@localhost/gcpct_hive_db -loop \
   -sleep 0.2
```

•-loop sets the beekeeper into looping mode
•-sleep parameter controls interval between beekeeper loops, in minutes. Default is 1 minute

  - affects output to screen
  - affects worker creation - normally max of 50 per loop.

Some practical aspects of running the Beekeeper

- `$ screen` or `tmux`

(note, not installed on your VM)
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.

- *guiHive*

guiHive is the most common way to monitor a running beekeeper. It gives a good overview of the status of a running pipeline, and has tools to make adjustments.

- `> 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) and releasing their jobs. This is useful when re-starting a beekeeper after fixing an error. A running beekeeper takes care of this automatically while it is running

Troubleshooting : 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

- Things to watch out for:
  - failing analyses (obvious, ***`beekeeper.pl`*** will probably stop)
  - failing individual jobs (check guiHive, progress or ***`beekeeper.pl`*** output)
  - failing attempts (when retry_count>0). Check msg for clues to what's causing transient errors.
  - 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 (using tweak_pipeline.pl) - in eHive versions after 2.4 you can also exclude an analysis by setting is_excluded to 1
  - ***`beekeeper.pl -dead`*** and ***`beekeeper.pl -sync`***
  - to have a more up-to-date picture of what happened; note the time of running ***`-dead`***

Troubleshooting : process

- Which Jobs/Analyses were affected?
  - See guiHive, or

```
> SELECT * FROM progress;
```

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

```
> SELECT * FROM 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

Troubleshooting : success

- Check for things that might need cleanup
  - (files, entries in database tables, logs, etc)

- Restarting failed jobs :
  - can do this via guiHive (in the job table - click on the job state to get a menu)
  - can also do this via command line. This can be useful if you have many analyses to reset, because you can use "%" as a wildcard

```
$ beekeeper.pl -url … -analyses_pattern <failed_analysis> -reset_failed_jobs
# example
$ beekeeper.pl -url … -analyses_pattern blast% -reset_failed_jobs
```

- Unlock the analyses that you have stopped by changing analysis_capacity
  - Do this with guiHive or tweak_pipeline.pl

```
$ tweak_pipeline.pl -url … -SET \
  'analysis[count_atgc].analysis_capacity=4'
```

- run `beekeeper.pl -loop` again

Troubleshooting : exercise

- Make a copy of example_input_2.fa and call it example_input_3.fa
- Edit example_input_3.fa to remove the first FASTA header (the line starting with '>')
- Re-initialize the GC percent pipeline (remember -hive_force_init 1), adding a tweak to change the 'input_file' parameter of the first analysis:

```
$ init_pipeline.pl Bio::EnsEMBL::Hive::Examples::GC::PipeConfig::GCPct_conf \
        -pipeline_url mysql://ensrw:ensrw_password@localhost/gcpct_hive_db \
        -hive_force_init 1 \
        -SET 'analysis[chunk_sequences].param[inputfile]=example_input_3.fa'
```

- Run the pipeline with the beekeeper
- Note the failure. Look at the msg table to identify what went wrong

```
> SELECT * FROM msg
```

Troubleshooting : exercise (continued)

- Fix example_input_3.fa so that it is properly formatted again
- Reset the failed jobs. Reminder, you use beekeeper.pl, choose the analysis with the -analyses_pattern option, and the reset command name is -reset_failed_jobs
- Re-run the beekeeper loop.

Tuning the pipeline

- Changing non-structural parameters of the pipeline/Analyses to make things run more efficiently.

- You can make experimental changes in the database with -tweak_pipeline.
- Once satisfied, make them permanent by changing the PipeConfig file.
- 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)

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

Tuning the pipeline: analysis_capacity and batch_size

- analysis_capacity - maximum number of workers that can be running for a particular analysis
- batch_size - number of jobs a worker claims at one time
  - There is overhead (time + database access) each time a worker deals with a job.
  - For short jobs, this overhead can add up if a worker has to claim one job at a time
  - Even for long jobs, the overhead can add up if there are many thousands of jobs (multiple workers trying to update the database at the same time)
- Consequence - analysis_capacity and batch_size tend to move in parallel when tuning a hive pipeline
  - Higher analysis_capacity may require raising batch size to reduce the number of workers trying to update the database at the same time

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
    - 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":

```
'-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.

Resource usage 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 the GCPct pipeline?

Resource usage estimation - Memory

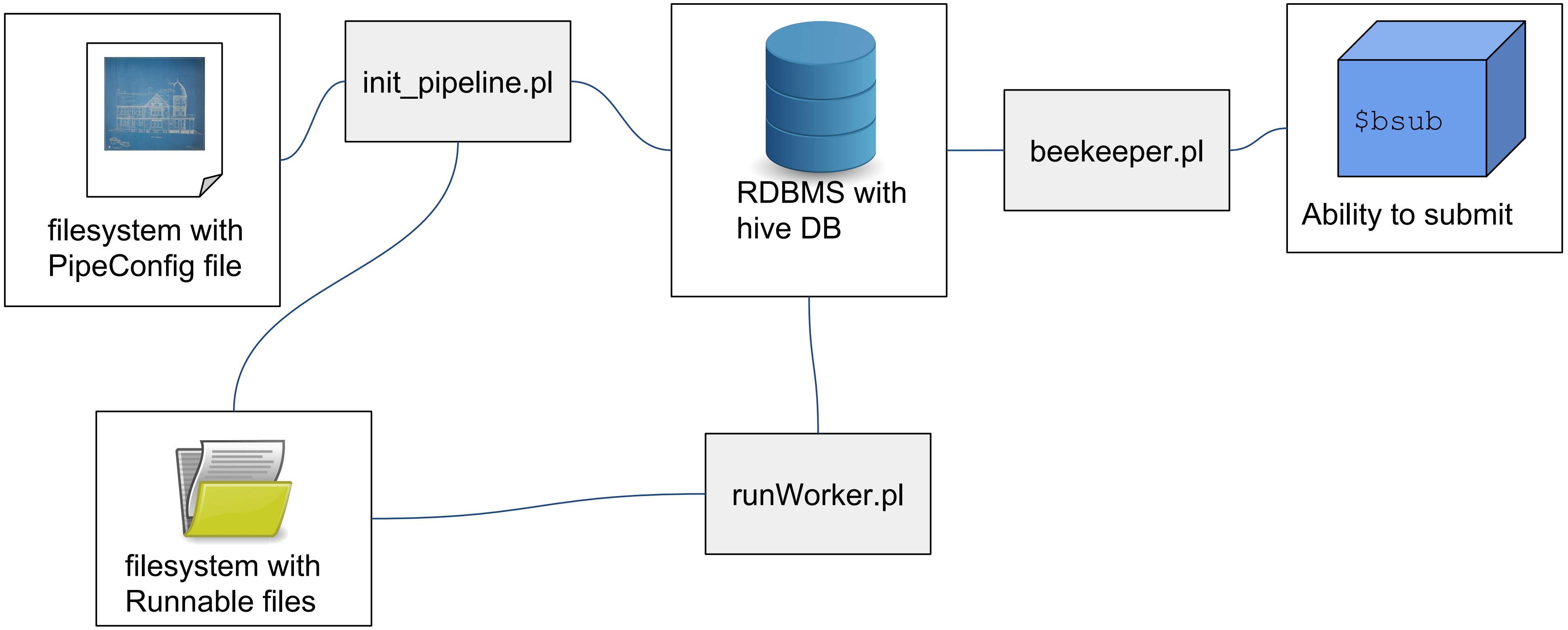
- Memory measurements are available in a view called resource_usage_stats

```
> SELECT * FROM resource_usage_stats
```

- This depends on BSD::Resource being installed on your system (it is listed in eHive's cpanfile as recommend:
- You can run load_resource_usage.pl to fill in this table if data is missing (this can happen on a farm if workers aren't able to record their own resource usage)

System requirements

- Bare minimum:
  - Machine with sqlite where jobs can run
- Ideal:
  - Cluster (LSF, SGE, HTCondor)
  - Head node where lightweight processes are allowed to run
  - RDBMS (MySQL/PostgreSQL) accessible from all head/compute nodes



Questions?

Ensembl Acknowledgements

The Entire Ensembl Team

Bronwen L. Aken[1], Premanand Achuthan[1], Wasiu Akanni[1], M. Ridwan Amode[1], Friederike Bernsdorff[1], Jyothish Bhai[1], Konstantinos Billis[1], Denise Carvalho-Silva[1], Carla Cummins[1], Peter Clapham[2], Laurent Gil[1], Carlos García Girón[1], Leo Gordon[1], Thibaut Hourlier[1], Sarah E. Hunt[1], Sophie H. Janacek[1], Thomas Juettemann[1], Stephen Keenan[1], Matthew R. Laird[1], Ilias Lavidas[1], Thomas Maurel[1], William McLaren[1], Benjamin Moore[1], Daniel N. Murphy[1], Rishi Nag[1], Victoria Newman[1], Michael Nuhn[1], Chuang Kee Ong[1], Anne Parker[1], Mateus Patricio[1], Harpreet Singh Riat[1], Daniel Sheppard[1], Helen Sparrow[1], Kieron Taylor[1], Anja Thormann[1], Alessandro Vullo[1], Brandon Walts[1], Steven P. Wilder[1], Amonida Zadissa[1], Myrto Kostadima[1], Fergal J. Martin[1], Matthieu Muffato[1], Emily Perry[1], Magali Ruffier[1], Daniel M. Staines[1], Stephen J. Trevanion[1], Fiona Cunningham[1], Andrew Yates[1], Daniel R. Zerbino[1] and Paul Flicek[1,2,*]

[1]European Molecular Biology Laboratory, European Bioinformatics Institute, Wellcome Genome Campus, Hinxton, Cambridge CB10 1SD, UK and [2]Wellcome Trust Sanger Institute, Wellcome Genome Campus, Hinxton, Cambridge, CB10 1SA, UK

Funding

wellcome

EMBL

National Human Genome Research Institute

Open Targets

BBSRC bioscience for the future

TRANSFORMING GENETIC MEDICINE INITIATIVE

MedBioinformatics

Co-funded by the European Union