System for Running Systems (SRS)

Adam Janin

ICSI Lunch Talk - Sept 24, 2013

(Minor updates from the Feb 19 talk)

SRS at ICSI

- Installed under `/u/drspeech/projects/swordfish`
- See `doc/STARTING.pdf` for Chuck Wooters guide to getting started with Swordfish.
- See `doc/environment.txt` for how to set up your environment to use SRS.
- See `doc/tutorials/README` for tutorials.
- See `doc/rationales` for some informal justifications for why SRS works the way it does.

Okay, now everybody can leave. :-)

Overview

- A full speech/audio system can have:
  - 100s of steps
  - 1000s of configuration options
- Conflicting goals:
  - Fast and easy to run
  - Support for debugging
  - Exploit compute/storage infrastructure
  - Easy to deliver
  - Independent development of components

Steps: Compute features, normalize, reformat, etc.
Configs: Sampling rate, size of context window, audio file extension, where to find models.
Our solution heavily inspired by SRI metadb and go.run-system.
Start with config, since other parts depend on it.

Config Files

- Plain text files with variables and values
- Simple macro facilities
- Ability to chain config files
- Documentation strings associated with variables
- Tracing and debugging

We want low barrier of entry so that researchers will use it. No XML, no complicated syntax, easy to understand. Replace hard-coded values. Easy to change to experiment. Simple command line tool, plus python and matlab interfaces to read values and doc from config files.
Chains can be long. Where was variable defined? Doc string from there.

Example Config File

```
language_pack BABEL_BP_101

# Speech to speech transition probability
ss_prob 0.99

# Input feature file
input_features chan1.pfile

SDEFINE root printenv SWORDFISH_ROOT

norm_file $root/models/$language_pack/mlp.norms

INCLUDE $root/config/mlp.config
```


Example of speech/nonspeech mlp. Simple variable/value. Change one thing to change which LP.
Comments associated with variables. "Shell define" calls a shell command.
Values can refer to other variables (normal or SDEFINEd). Can chain config files. FIRST occurrence defines value.

srs-config

```
> srs-config -c example.config norm_file
/u/drspeech/projects/swordfish/models/
BABEL_BP_101/mlp.norms
> srs-config -doc -c example.config ss_prob
(Line 4 of config file example.config)
Speech to speech transition probability
```

```
language_pack BABEL_BP_101

# Speech to speech transition probability
ss_prob 0.99

# Input feature file
input_features chan1.pfile

SDEFINE root printenv SWORDFISH_ROOT

norm_file $root/models/$language_pack/mlp.norms

INCLUDE $root/config/mlp.config
```


Additional arguments for debugging with verbose output.

srs-config

```
> srs-config -c example.config -dumpsh CONFIG_
CONFIG_root='/u/drspeech/projects/swordfish'
CONFIG_language_pack='BABEL_BP_101'
CONFIG_input_features='chan1.pfile'
CONFIG_norm_file='/u/drspeech/projects/swordfish/
models/BABEL_BP_101/mlp.norms'
CONFIG_ss_prob='0.99'
```

```
> eval `srs-config -c example.config -dumpsh CONFIG_`
> echo $CONFIG_ss_prob
0.99
```


Dump options if you're using a lot of vars in a script. Similar for matlab or csh. Handles quoting (I think :-)

Native Python Binding

```
>>> c = srs.Config('example.config')
>>> c.vars.ss_prob
0.99
>>> c.vars.pause_duration
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: _VarEntry instance has no attribute
'pause_duration'

>>> 'pause_duration' in c
False
```


Quick taste of python bindings. Lots of other ways to do things. See tutorial.

SRS Templates

- Organized into “steps”.
- Each step is either an executable or a directory.
- Executable steps read an input config file, do some work, and write an output config file.
- Directory steps are called recursively.

Executable step might wrap legacy code. Any executable (shell script, C program, java, etc). By reading/writing config, independent as possible from other steps.
A directory step might be “feature computation”. It would have format conversion, normalization, and features comp as substeps.
Whole system is just a top-level directory step.

Example Template

```
> ls mlpfst_run_file

README              step010.reformat*    step040.fst*
default.config      step020.features/
makerttm.py*        step030.runnet*
```


Example of speech/nonspeech using an mlp. Just a plain directory somewhere.
Only names starting with “step” are steps. Executed in alpha order. Everything else is up to writer. Executable steps, directory steps.

Example Step

```
#!/bin/bash

inconfig=$1
outconfig=$2

audiodir=`srs-config -c $inconfig audio_directory`

find $audiodir -name '*.wav' > include.list

echo audio_list include.list > $outconfig
echo INCLUDE $inconfig >> $outconfig
```


Really simple example. No error checking.
First arg is input config. Second arg is output config. ALL I/O through configs.
The tool to call templates “srs-go” handles naming of config files.

srs-go

- The "srs-go" tool coordinates running of a template.

  ```
  srs-go -template mlpfst_file -dir exp01
  language_pack BABEL_BP_101 ss_prob 0.5
  ```

- Recursively copies `mlpfst_file` directory to `exp01`, thereby archiving the actual run.
- Creates a config file from extra arguments on the command line, allowing easy experimentation.
- Logs all output from steps to help with debugging.

exp01 is now independent of template. We know EXACTLY how it was run. srs-go stores command line, machine it was run on, timestamps, etc. It handles naming and storing of input/output config files for each step.

srs-go

- Stops if a step fails.
- Future runs of `exp01` will only execute failed step onward.
- Can be restarted from any step.
- Can be stopped at any step.

```
srs-go -dir exp01 -from
step020.features -to
step040.runnet -restart
```

Disks and srs-go

- The srs-go tool creates links automatically under the experiment directory:
  - `scratch`
  - `scratch_ttmp`
  - `scratch_local_ttmp`
- Steps may use these however they wish.
- srs-go supports cleaning options.

Encourage “correct” use of disk.
Links to physical directories that srs-go creates automatically.
Clean entire experiment or just data. Complex since may be on another machine.

srs-go-parallel

- Execute a step in parallel
- Called almost exactly the same as srs-go:

```
srs-go-parallel -template mlpfst_file
-dir exp01 -pvar language_pack lang.txt
ss_prob 0.5
```

- Config variables specified with `-pvar` are read from a file, one line per job.
- Options for controlling batch size, number of parallel jobs, clean up, retries, etc.

Lots of ways to parallelize. A step can be parallel explicitly (e.g. NN training using CUDA). Common pattern is to parallelize over a list (e.g. of input files).
Common: a step itself calls srs-go-parallel
Assume ICSI's cluster and uses run-command under the hood.

srs-go-parallel-litestep

- `srs-go-parallel` calls `srs-go` for each job and is therefore pretty "heavy weight".
- If your step is simple, `srs-go-parallel-litestep` is a lighter weight version of `srs-go-parallel`.

Miscellany

- `srs-arch` returns an architecture-specific string (e.g. `x86_64-linux`). Useful for constructing paths.
- `srs-clean-orphans` searches for data directories that appear to not be associated with experiments and deletes them.