Smartsubmit

A Frontend to the TAS Babymaking Backdoor

Outline

- What is Smartsubmit, and why would I use it?
- What sort of speed up can we expect?
- Version 0.1 usage, the current system, a basic example.
- A more interesting example – StopAnalysis babymaking.
- Challenges with publishing the product today.
- Conclusions: What do we want out of Version 1.0?

What is Smartsubmit?

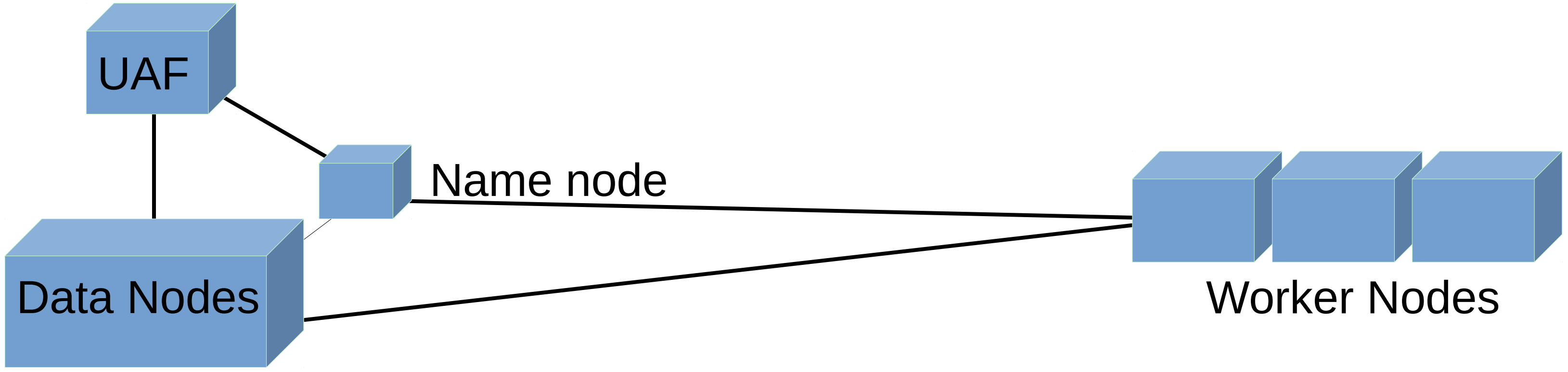
- To the user, Smartsubmit is a cli which allows you to:

  1. load files onto machines in our cluster
  2. submit jobs that run over those files

Advantages over standard batch submission:

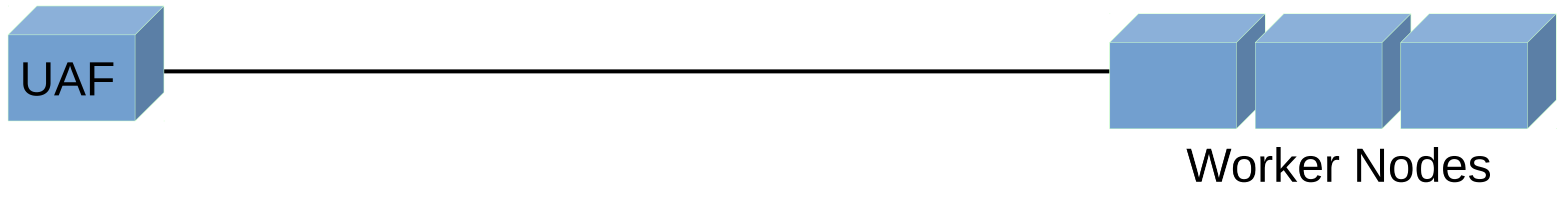
- Maximal I/O parallelization for babymaking without network bottleneck
- Always open condor slots
- Easy to implement within your current system, it should only add a small amount of code to your base.

A Typical Babymaking Procedure



1. A user submits a job to the worker nodes via condor.
2. Condor finds a machine with an open “condor slot” and assigns it the job.
3. The job references a file stored on hadoop, the worker asks the “name node” where the file is stored.
4. The “name node” looks up the location of the blocks of the file and returns that info to the worker which requests blocks from the “data nodes.”
5. The blocks travel from the data nodes to the worker and is bottlenecked by a 1Gbps network connection into the worker.
6. The worker does analysis on the data.
7. Return to step 3 until all data has been read.
8. Some output is returned to the UAF, stored on Hadoop, etc…

Babymaking with Smartsubmit



0. Files are added to the SS database, babymaking executables are loaded onto network drives.

1. A user submits a job to Smartsubmit.

2. Smartsubmit determines the location of the full files and submits a condor job for each hard disk housing the files.

3. The jobs land on the machine holding the file with low latency.

4. The job reads the files from the local disk, which is not bottlenecked by the network.

5. Data is processed

6. Some output is returned to the UAF, stored on Hadoop, etc…

UC San Diego

An Example: Word Counting

Loading in a sample from Hadoop:

Single file:
```
ss_ctrl --absorb_sample -f /path/to/single_file -s sample_name
```

Full directory:
```
ss_ctrl --absorb_sample -d /path/to/sample_dir -s sample_name
```

The executable: (placed on network drive)

```
#!/bin/bash

whoami

for f in $@
do
        wc -w $f
done
```

The call

```
ss_ctrl --run_job -e /path/to/exe -s sample_name
```

UC San Diego

Another Example: Stop Babies

The executable: (placed on network drive)

```
dpath=
if [[ $# -gt 0 ]]
Then
        dpath=`dirname $1`'/'
        <Make directory for output>
        <Construct auxiliary samples data file>
        <setup environment variables>
        mkdir json_files
        ln -s $sabin/json_files/* ./json_files/
        ln -s $sabin/jecfiles ./jecfiles
        <net_drive>/runBabyMaker smartsubmit -1 -1 . aux_samples.dat
        mv smartsubmit.root baby1.root
        <cleanup>
else
        echo "No files given"
fi
```

The call

```
ss_ctrl --run_job -e /path/to/exe -s sample_name
```

Benchmark Results

Dataset: ~150 GB of data

/tZq_ll_4f_13TeV-amcatnlo-pythia8_TuneCUETP8M1/RunIISpring15MiniAODv2-74X_mcRun2_asymptotic_v2-v1/MINIAODSIM

	Smartsubmit			Regular Batch		
Run #	Total Run Time	Time In Que	Time Running	Total Run Time	Time In Que	Time Running
1	27:06	1:11	25:54	~45		
2	29:06	2:16	26:50			
3	30:13	2:10	28:02			
4						
5						
6						
7						
8						
9						
10						
Average:						

Challenges

- What to do with disk space?
  - When an Ntuple is loaded into smartsubmit, it has an infinite lifetime. How should we delete files that have been there for too long?
  - Should users be able to delete other users' files?
- How do we deal with catastrophic failures?
  - Should we reload in all files by hand or setup a helper?
  - What happens if a disk goes bad? How will S.S. know?
- Odd instabilities, perhaps brought on by SQLite.
  - Should we replace our database?
  - How do we deal with Segmentation Faults, how can we guarantee a job is run until it finishes?
  - Why does smartsubmit go down at random when no code has been shown to be unstable?
- Infinite Loops on jobs?
  - How should we handle users submitting a script with an infinite loop?

Conclusions

Smartsubmit is ready for testing and general use under monitored conditions to catch any lurking bugs.

It still needs to be decided whether time should go in to changing the database n order to increase stability. An alternative would be to simply allow the program to fail and set up a script restarts the server whenever it crashes.

A system should be put in place for dealing with disk corruptions or files being removed without Smartsubmit being notified.

A system should be put in place to manage the data without heavy user intervention (e.g. monitoring available disk space, removing old files that haven't been used when the disks are full,)

Under the current circumstances, we are getting an approximately twice the speed from Smartsubmit as we are from conventional batch submission.