Tools to Execute an Ensemble of Serial Jobs on a Cray

Abhinav Thota, Scott Michael,
Sen Xu, Tom Doak, Robert Henschel
Indiana University

May 9, 2013

Outline

- The Biology Application
- What are ensemble serial jobs?
- Why use Cray machines?
- A survey of solutions
    - BigJob
    - PCP
    - aprun
- Experiments and Analysis
- Conclusion

mlRho – A Biology Application

- mlRho software: a serial program that estimates mutation, recombination and sequencing error rates
- SciAPT assisting Biologists at IU
- Data-parallel, hundreds to thousands of iterations can be performed independently
- I/O is the limiting factor
- Open source, available here: https://github.com/CIPaGES/mlrho
- Biologists came up with a plan to do the analyses on tens of genomes
- A total need of ~6 millions compute hours

XSEDE

- XSEDE is the place to go if you are looking for millions of compute hours
- Need a proposal, with performance and scalability numbers
- Did not have an easy way to do this on Kraken
- Did our experiments on Ranger and Stampede
    - Using BigJob, and
    - An MPI wrapper
- But found a couple of solutions on how to do this on a Cray
- Put some more effort into this given that IU has its own Cray

What is an ensemble of serial jobs?

- A set of serial jobs that are ready to go
  - The ensemble of serial jobs are not sequential
  - They are all ready to go at the same time
- Usually independent, data-parallel, parameter-sweep type applications
- When do we need specific tools?
  - Total workload: 100 serial jobs? 1000? 10,000?
  - 100,000 compute hours? 1 Million?
  - Concurrency
  - A few serial jobs at a time is a separate discussion

Traditional v. Non-traditional

- Traditional fields: physics, math, astronomy, chemistry, etc.
    - Highly parallel, MPI applications
- Non-traditional fields: biology, bioinformatics, finance, geology, psychology, etc.
    - Serial, non-scalable, analytical, text-processing
- The diversity of users and applications is increasing as computing becomes cheaper

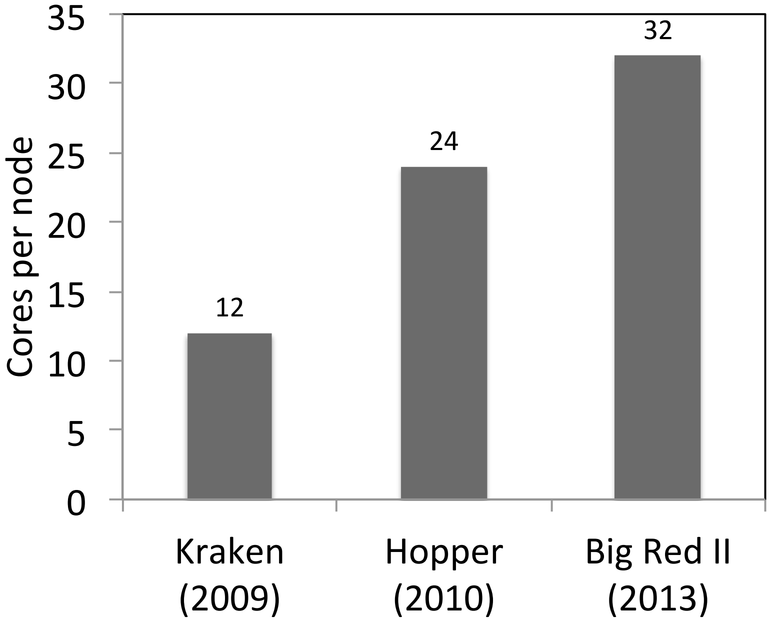
Why do we want to run these jobs on a Cray?

- General Purpose machines
    - Hopper
    - Kraken
    - Big Red II
- Users will have to use compute hours where they can find them
- Cray machines are becoming more and more common

Image Sources: http://www.nics.tennessee.edu/computing-resources/kraken, http://www.nersc.gov/users/computational-systems/hopper/, http://newsinfo.iu.edu/pub/libs/images/usr/15356_h.jpg

Why not let the users figure this out?

- Running serial jobs on a Cray is not a difficult task
- There is a good chance that users will just submit serial jobs
- Without shared node scheduling, 90% of a compute node is unutilized
- Processor parallelism has been growing
    - So has cores per compute node
- Need tools and queue policies to avoid this



A survey of solutions

- Shared node computing using Cluster Compatibility mode (CCM)
    - Service needs to be provided by the admins
- Using regular batch job submission
    - with or without packing all the cores on a node
    - aprun and back grounding the jobs
- PCP – parallel command processor
- BigJob – SAGA based pilot-job tool
- Swift – a parallel scripting language

Adapt to Queue Policies

- Why don’t we stick to aprun?
    - The users can be told to pack enough jobs per node
    - Or the users will waste 90% of their allocation
- But, even with this solution, the user is required to submit many single node requests to the scheduler
- Many centers with large Cray machines prioritize or prefer large jobs
    - Scheduler policies, discounts, etc.
- Need a tool that allows one to bundle as many serial jobs as needed in to one large job of a size that makes it appropriate for a particular machine
- Should make it possible to use multiple compute nodes and all the cores on an individual node
- **Allows users to adapt to the available machines and their policies**

BigJob, PCP and aprun

- Out of all the tools we found, we chose these three
- Reason for that:
  - Previous experience with BigJob, can be used as a container job
  - PCP is easy to use, can be used to bundle jobs
  - aprun is the default choice, used as a baseline
- PCP is a really simple tool, thank NICS user support group for referring PCP
- BigJob is more sophisticated
  - Gives the users more options and control over workflow
  - But adds complexity

BigJob

- Developed and supported by the RADICAL Lab at Rutgers University
- Not necessary to understand the architecture
  - There's an API.
  - A python script defines the workload
  - Asks the scheduler for nodes
  - Runs the jobs
  - Need to understand more if you are running jobs across multiple machines
- Can be used:
  - as a container job,
  - to distribute jobs to multiple resources
  - to coordinate the launch and interaction of jobs within the container
  - And to design lot of other exotic workflows
- We use it as a simple container job

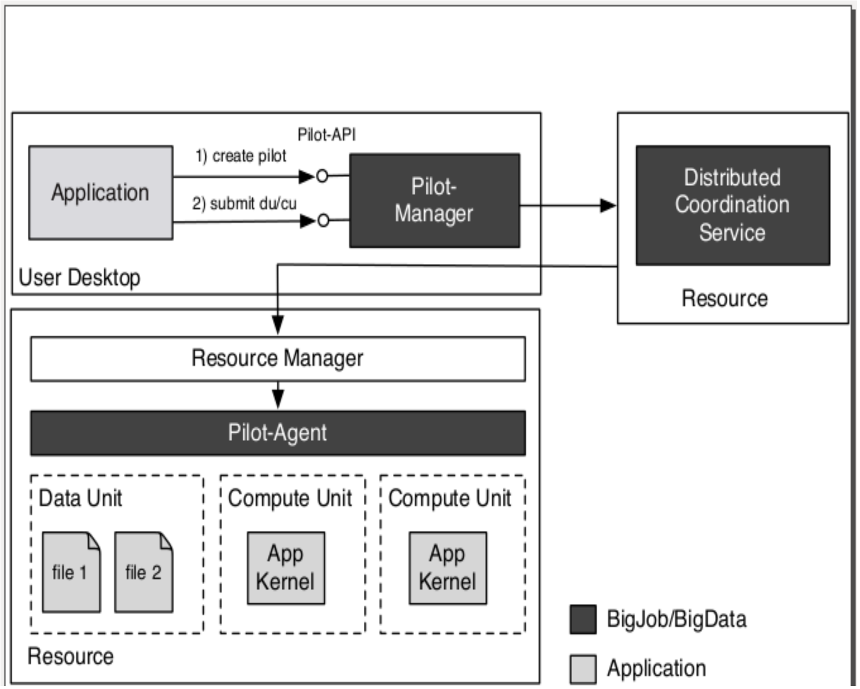


Image source: https://github.com/saga-project/BigJob/wiki/BigJob-Architecture

More about BigJob

- Available for download from the Python Package Index
- API preview:
    - Job request submitted to the scheduler

```
"service_url": "xt5torque+gsissh://kraken.nics.xsede.org",
"number_of_processes": 960,
"allocation": "TG-123456",
"queue": "debug",
"working_directory": "/work/user/",
"walltime":120, #minutes
```

Individual mlRho job descriptions:

```
for i in range(0, NUMBER_JOBS):
 compute_unit_description =
    "executable": "/work/user/mlRho",
    "arguments": [" -m "+ str(start) +" -M "+ str(end)+  "
-n profileDb"],
    "number_of_processes": 1,
    "spmd_variation":"single”, #MPI or serial
    "working_directory":"/work/user/",
    "output": "output"+str(i)+".out",
    "error": "error"+str(i)+".err”,
```

Need to know basis…

- The user only needs to know what the software can do for them and how to get the software to do it
- The only interaction the user has with the BigJob software is via the python job submission script
    - which takes in the same details as a batch job submission script.
    - A quick-start guide is available on the BigJob website https://github.com/saga-project/BigJob/wiki/BigJob-Tutorial-Part-3:--Simple-Ensemble-Example
    - Good to be familiar with python

Parallel Command Processor (PCP)

- Original implementation of the tool was produced by the Ohio Supercomputer Center (OSC)
- Ported by the NICS team to work on a Cray specific architecture
- The source code of PCP is available from NICS
- Tested this code on multiple Cray machines, works as expected.
- PCP expects a text file containing a list of commands to be run
- We have used PCP to run hundreds of mlRho jobs concurrently.
- Basic scripting knowledge useful in creating text files with the jobs that need to be executed
- The barrier to entry for using PCP is very low compared to other similar tools

More about PCP

- Build as simple as: “cc pcp.c”
- `“aprun -n 512 ./pcp list.txt”`
- Where list.txt contains the 512 commands to run:

```
mlRho -m 1000 -M 1005 diatom.pro > out_1
mlRho -m 1006 -M 1010 diatom.pro > out_2
.
.
.
mlRho -m 2551 -M 2555 diatom.pro > out_511
mlRho -m 2556 -M 2560 diatom.pro > out_512
```

All the advantages of a container job… but

- No control over job management
- Job/load balancing not available
- For very similar jobs that are independent and have the same running time, both BigJob and PCP work.

	BigJob	PCP
Container Job	X	X
Job Management	X	
Load Balancing	X	
Data Management	X	
API	X	

aprun

- The default choice
- Scripts that each contain as many binary commands as there are cores on a single node
- Cannot runs jobs across multiple nodes
- Without CCM, whether we can run more than one unique job on a single node is questionable
- If there are 1000,000 jobs to run, need to submit ~10,000 separate jobs
- Lot of scripting

The usual batch script

- On Kraken: `aprun -n 1 -d 12 -cc none -a xt run.sh`
    - -n 1 # run on a single node
    - -d 12 # allows the script to access all the cores on a node
    - -cc none # allows each serial process to run on its own core
    - -a xt # required by aprun to run a script instead of a program

- Where run.sh contains:

```
mlrho -m 1 -M 2500 input.pro > data1.out &
mlrho -m 2501 -M 5000 input.pro > data2.out &
.
.
.
mlrho -m 27501 -M 30000 input.pro > data12.out &
wait
```

Experiments on Kraken

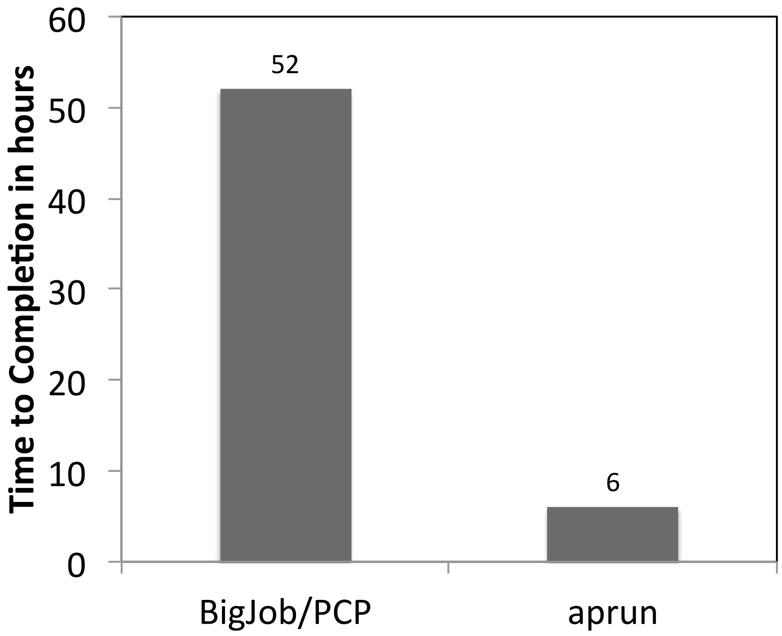
- A trial run to see if:
  - The tools work
  - check whether it is beneficial to bundle serial jobs in general into larger jobs to get better throughput.
- Metric of interest is total time to solution
- Disclaimer: not useful to make broad generalizations, either with respect to Kraken or other large machines, further studies are planned to support more general claims

Workload

- Kraken is a 112,896 core Cray XT5 machine operated by NICS
- A variety of queues are supported
- Same workload with all three tools on Kraken
- We selected a job size of 960 cores, which is 80 compute nodes on Kraken
- One instance of mlRho was run on each core and, in the runs where actual computations were done, ran 250 iterations on a zebra genome.
- Yes, we just collected the queue wait time in all but one experiment
    - Can't waste those SUs!

Experiments

- BigJob and PCP:
  - Single set of experiments will work for both
  - A single job is submitted to the queue in both cases, requesting 80 nodes
- aprun:
  - 80 separate single node job requests submitted to the queue
- Experiments repeated 5 times
- Surprising result:
  - BigJob/PCP type took 52 hours
  - aprun type took 6 hours
  - mlRho runtime is ~4 hours



Analysis

- It appears that aprun was faster than PCP/BigJob
- But need to consider many factors
- While this may be true at 80 nodes, it may not be true at 120 or 200 nodes
- Many machines have queued and run limits:
  - the number of jobs from one user that can be queued at a time
  - The number of jobs from one user that can be running at a time

	Kraken	**Hopper**
Queued Limit	100	16
Run Limit	25	16

- Hopper has a separate throughput queue, where the queued limit is 500 and run limit is 250, but a maximum of only 2 nodes can be requested per job
- The run limit on Kraken is probably not being enforced

Other factors

- Backfilling:
    - the backfilling algorithm attempts find any unused nodes or “holes” in the schedule and fill them with appropriately sized jobs
- While BigJob/PCP jobs were submitted with a gap of multiple days, all 80 of the aprun jobs were submitted simultaneously.
- It is possible that all 80 jobs re-used a single, since these jobs were only collecting waiting time
- We recorded all the node numbers of the compute nodes that our jobs ran on
    - With the exception of one set of runs the number of unique nodes used for the 80 jobs was in the 65-80 range.
    - One set ran on the same five nodes, however this set of runs did not have the smallest overall wait time
    - it had the third longest wait time in the set of five aprun submissions

Conclusion

- Non-traditional applications on the Cray are one the rise
    - Both Cray and non-traditional users are moving towards each other
- Parametric sweeps are not new to the supercomputing field, they are new to Cray supercomputers.
- Previous obstacles to running multiple binaries on the same compute node have now been overcome
- Submitting separate single node job requests to the scheduler is straightforward and easy to implement
- BigJob and PCP are more elegant, offer the ability to submit much larger job requests
    - can be advantageous depending on site specific policies
- **Factors specific to the application, machine, scheduler policies and ease of use determine best tool for the task**

Acknowledgements

We would like to thank the SAGA-BigJob team at Rutgers University for their help with BigJob on Kraken. We would like to thank the user support team at NICS for providing the source code and instructions for using PCP.

This research was enabled by IU's advanced cyberinfrastructure, including the Big Red II supercomputer and the Data Capacitor II storage system, the implementation of which has been supported by the Lilly Endowment through their support for the IU Pervasive Technology Institute and the Indiana Metacyt initiative. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575.