Python and Distributed Computing

James Westover

Overview

- Local Distribution
  - Multithreading
  - The GIL
  - Multiprocessing
- Distribution across the network
  - MPI
  - Other distributed state systems and RPC

Local Distribution

- Local distribution is a first step in parallelizing a program. It is often challenging to determine which sub routines can run independently.
- Shared memory space is easiest to achieve inside a multithreaded application but python's garbage collector which cleans up dead objects depends on the use of a global interpreter lock (GIL)
- Beyond that it is possible to fork new processes to distribute tasks but this tends to be memory intensive as each process has it's own memory space and objects have to be copied between the two or kept in a managed shared memory space referred to as a queue.

Multithreading Hands On

Let's give this a shot

Global Interpreter Lock

- Python has a garbage collect system that relies on locking of all live objects as each line of code is executed even if those lines are being executed on different threads. This intensive garbage collection is how python can remain so low in memory weight in comparison to other virtual machine dependent languages like Java. Memory leaks in python are not common like they are in Java.
- But this has a performance hit in execution speed because the threads not only have to deal with object locks but also on the locking of the global interpreter. This feature isn't present in all of the python implementations most famously absent from the Java based implementation Jython. But there is an alternative in CPython which is to spawn multiple interpreters, the multiprocess method. This gives every PID it's own GIL to play with.

Multiprocess Hands On

Let's see it in action

Which to use when?

- In general it is easier to use multithreading when your different processes will need to share objects and in essence interact with one another.
- Multiprocessing is great for those times where your parallel targets are 'more parallelizable'
- But what happens when I outgrow my box?!?!?

Network Distribution

- In general when a problem becomes too big to handle with a single box you could either get a bigger box or distribute the process over multiple boxes. There are many approaches to this problem and I will cover MPI and RPC using RabbitMQ with a hands on demo

Message Passing Interface (MPI)

- Scientific computing relies on large scale Beowulf clusters which have a single head node which is responsible for distributing jobs to the nodes it has in its Ring. A ring is no more than a cluster and these rings are typically grouped in terms of homogenous hardware to allow compiled programs to distribute easily. But that comes with its own issues including bottlenecks and single points of failure.
- In python this is possible using the mpi4py library. With this library you can take in information from the MPI ring and job scheduler to be able to dynamically scale your application.
- I was planning on a demo of this but the infrastructure can be hard to assemble… another drawback
- But when it works it can be very effective especially for jobs which break up simply and have very little interprocess communication.

Remote Procedure Call and AMQP

- RPC is the basic process of passing objects between processes which could be located on different machines on the network.
- Unlike MPI which depends on a strict network configuration. RPC via AMQP can be very flexible to allow communication between different languages passing objects after a serialization which allows for distribution of work.

Hands on with RabbitMQ for an RPC

Lets hope this works