Open Science Grid User School 2016

Project Report:

Finding Minimal Cuts For Volume Downsizing

Student: Dachao Sun

Clemson University, Clemson, SC

dachaos@g.clemson.edu

July 24–29, 2016

Madison, WI

Contents

1 Introduction: What I Have Learned **1**

1.1 Summary . . . 1

1.2 Eg. 1: Run the same program multiple times . . . 1

1.3 Ex. 2: “DAG”: multiple runs with different arguments . . . 4

2 Project description **7**

2.1 Volume retargeting: max-flow/min-cut problem with n^3 nodes . . . 7

2.2 Algorithmic/computational challenges . . . 8

2.3 Simple HTC workflow: the computational plan . . . 9

2.3.1 Why HTC? . . . 9

2.3.2 Approach . . . 9

2.3.3 Resources . . . 10

2.3.4 Unsolved issues . . . 10

3 Expected Results (from local machine) **10**

4 Conclusion: Answers To Questions On Website **12**

Acknowledgement **13**

Appendix: Some Notes (at the User School) **15**

The Unix shell . . . 15

HTCondor . . . 15

Basic job submission (HTCondor) . . . 15

1 Introduction: What I Have Learned

Through the one-week user school of the Open Science Grid we, both trivial OSG users and aspiring scientists, learned a lot in- and out-side the classroom. There are numerous examples covered over the five-day time span, but here in this chapter I'll try to do a high-level summary on the contents that I took away the most. Section 2 describes my project (a fairly-good fit for HTC) and how I'm gonna plan on using the OSG platform.

1.1 Summary

The Open Science Grid (abbreviated as "OSG" from this point) is effectively aimed at High-Throughput Computing (HTC), as no GPU programming is involved but keyed on **how to submit jobs**:

- scheduling multiple jobs,
- handling large input/output files, and
- using the premium platform *HTCondor* to monitor the job queue.

Note that for each of the above three aspects, there are definitely more details included in the user school/manuals. The following three sections include examples that may benefit the configuration of my project, which is introduced in section 2.

1.2 Eg. 1: Run the same program multiple times

(Source: `https://twiki.opensciencegrid.org/bin/view/Education/UserSchool16Mon24QueueN`)

To start with a relatively trivial (but practical) example, here is a C++ program to estimate the value of π:

```
//circlepi.cpp
#include <cstdio>
#include <cstdlib>
#include <sys/time.h>

int main(int argc, char *argv[])
{
   struct timeval my_timeval;
```

```
    int iterations = 0;
    int inside_circle = 0;
    int i;
    double x, y, pi_estimate;
    //..
    pi_estimate = 4.0 * ((double) inside_circle / (double) iterations);
    printf("%d iterations, %d inside; pi = %f\n",
    iterations, inside_circle, pi_estimate);
    return 0;
}
```

The `Makefile` for compiling it as a simple example:

```
CC = g++
LDFLAGS = -O3
CFLAGS  =
PROJECT = circlepi
OBJECTS = ${PROJECT}.o

all: clean ${PROJECT}

${PROJECT}: ${OBJECTS}
   ${CC} -o ${PROJECT} ${OBJECTS} ${LDFLAGS}

%.o: %.cpp
   ${CC} -c $<

clean:
   rm -f core.* *.o *~ *.swp ${PROJECT}
cleanextra:
   rm -f core.* *.o *~ *.swp
```

The “`cleanextra`” is nothing more than deleting everything other than the program itself. To plan ahead there is possibly a chance when the user’s project code cannot be compiled on OSG Connect/other entry point only using `g++`. For those cases the user must compile it their own machine, and bring everything to the entry machine as needed.

Then the most important part: **the job submit file**. It’s a plain text with certain predefined keywords (variables) to be filled in:

```
/* circlepie.sub
 * (can be any filename you like) */
universe   = vanilla  //almost every program uses vanilla
executable = circlepi //the executable file name
```

```
arguments  = 1000000  //input argument(s)

//Use $(Cluster) and $(Process) variables
//to distinguish output files, and they are
//effectively computed by the corresponding
//processes respectively.
output = out-$(Cluster)-$(Process)

should_transfer_files = YES
when_to_transfer_output = ON_EXIT

//Enqueue 3 jobs (with exactly the same arguments), which
//means running it three times on different machines
queue 3
```

Running the job

- Compile it with `Make`
- Test it with `./circlepie 1000000`
- Submit the well-written job file with `condor_submit circle.sub`

Waiting for the job

Use `condor_q [user_name]` to check all the job you sent "recently".

Getting the results

When they're done, there should be output files trasferred back to this directory. If you see them with `ls` you can try reading the results:

```
$> cat out-25670369-*
1000000 iterations, 785097 inside; pi = 3.140388
1000000 iterations, 785298 inside; pi = 3.141192
1000000 iterations, 784961 inside; pi = 3.139844
```

So that is how it works — a simple example to run the same C++ program on multiple machines.

1.3 Ex. 2: "DAG": multiple runs with different arguments

Often times we want to run the same program (the same executable) but with **different (groups of) input arguments**. Instead of writing multiple submit files for multiple submissions, the HTCondor provides an underlying scheme called **DAGs**, which kinda links multiple jobs sequentially in a directed acyclic graph where each node corresponds to an independent job.

This example uses the "goatbrot" program to generate each of the four pieces/tiles of a *Medelbrot Set*. This program is provided Brian Hall on his Github repository (`github.com/beejjorgensen/goatbrot`).

To test the usage of `goatbrot` on a local machine (laptop), we do:

```
$> goatbrot -i 100000 -o tile_0_0.ppm -c 0,0 -w 3.0 -s 1000,1000
```

, which uses 10 thousand iterations, creating a Mandelbrot set of with 3.0 centered at (0,0). The actual image resolution is set to 1000×1000.

To convert it into JPEG format:

```
$> convert tile_0_0.ppm out.jpg
```

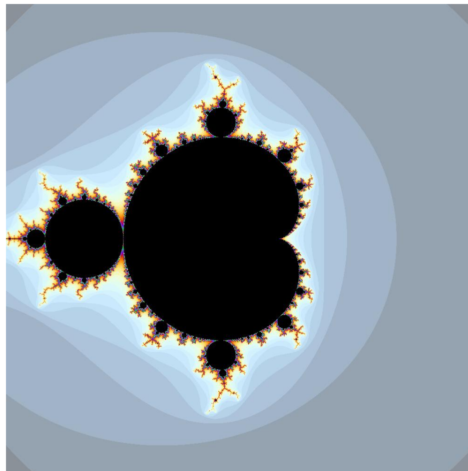


Figure 1: Example output of a Mandelbrot set image (100,000 interations)

Dividing the generation of a Mandelbrot set: to speed the entire thing up, we can break it into multiple invocations, and **stitch** them together. The result stays **correct** when **parallized**, because each sub-task is independent.

```
$>  goatbrot -i 100000 -o tile_0_0.ppm -c -0.75, 0.75 -w 1.5 -s 1000,1000
$>  goatbrot -i 100000 -o tile_0_1.ppm -c  0.75, 0.75 -w 1.5 -s 1000,1000
$>  goatbrot -i 100000 -o tile_1_0.ppm -c -0.75,-0.75 -w 1.5 -s 1000,1000
$>  goatbrot -i 100000 -o tile_1_1.ppm -c  0.75,-0.75 -w 1.5 -s 1000,1000
```

Now we've got four tiles, and stitch them:

```
montage tile_0_0.ppm tile_0_1.ppm
        tile_1_0.ppm tile_1_1.ppm
        -mode Concatenate -tile 2x2 stitched.jpg
```

The neat "`montage`" command is part of the ImageMagic in Linux, and it creates a composite image by combining several separate images.

Submit file with variables

First the submit file would have the arguments set up as "variables", which are to be specified in the DAG file:

```
executable = /usr/local/bin/goatbrot
arguments  = -i 100000 -c $(CENTERX),$(CENTERY)
             -w 1.5 -s 800,800 -o tile_$(TILEY)_$(TILEX).ppm
log        = goatbrot.log
output     = goatbrot.out-$(TILEY)-$(TILEX)
error      = goatbrot.err-$(TILEY)-$(TILEX)

should_transfer_files   = YES
when_to_transfer_output = ONEXIT

request_memory = 1024M
request_disk   = 1024M
request_cpus   = 1

queue
```

Submit the DAG file using `condor_submit_dag` command, which sends the DAG job to the target cluster.

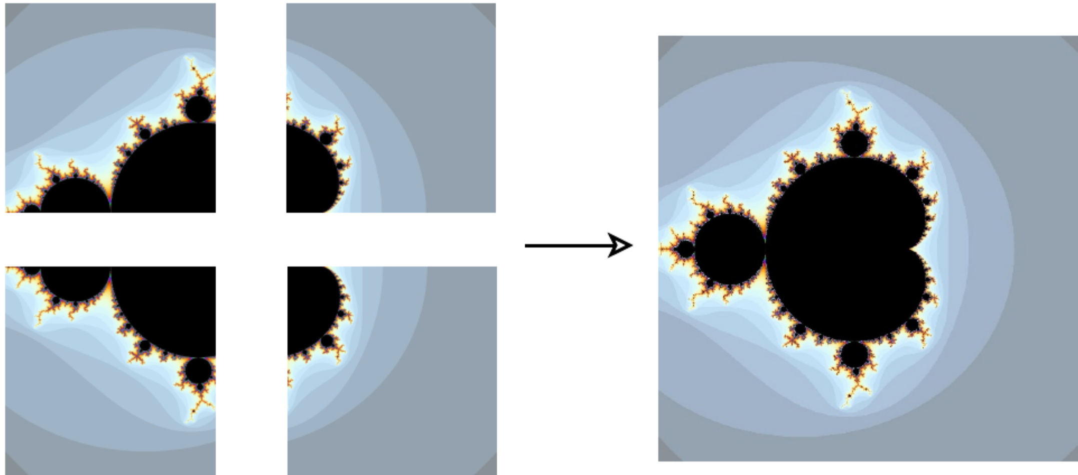


Figure 2: Stitching four tiles using `montage` program

Sequential 2-level DAG:

We can also incoporate the stitching part into the whole task, which will run at the "2nd level" after each of all four jobs is finished. We create a parent-child relationship to achieve this:

```
JOB g1 goatbrot1.sub
JOB g2 goatbrot2.sub
JOB g3 goatbrot3.sub
JOB g4 goatbrot4.sub
JOB montage montage.sub
PARENT g1 g2 g3 g4 CHILD montage
```

Such that all the goatbrot commands run simultaneously and will complete **before** running the montage job.

2 Project description

My plan for project making usage of OSG resources is effectively the computational phase of my Master thesis "*volumetric seam carving*" (`people.cs.clemson.edu/~dachaos/research/`), in which the goal is to reduce the size of volumetric data, namely 3D images such as CT scan of human organs, in a "content-aware" [1] manner.

2.1 Volume retargeting: max-flow/min-cut problem with n^3 nodes

The algorithm/operator we use to find the best 2D "plane" to remove from a 3D volume is a derived version for video [2] of the original *Seam Carving* [1] which works on 2D images.

Effectively, what this project does is the following:

1. Take as the input a 3D volume of size `Width` by `Height` by `Depth`, and denote the size of each dimension as n for simplicity;
2. Construct a (connected) graph from this volume, where each node $\longleftrightarrow$ a "voxel" in the volume. Nodes are connected via bi-directional arcs, where the weights are specified in a smart way in [2] to exactly represent an optimization problem.
3. Compute the ***minimum cut*** (a set of arcs) of the above graph;
4. Pick the left end-points for arcs in the minimum cut. These end-points (voxels) form the 2D seam.
5. Remove the 2D seam from volume (and pinch the two separated parts to fill the "gap").

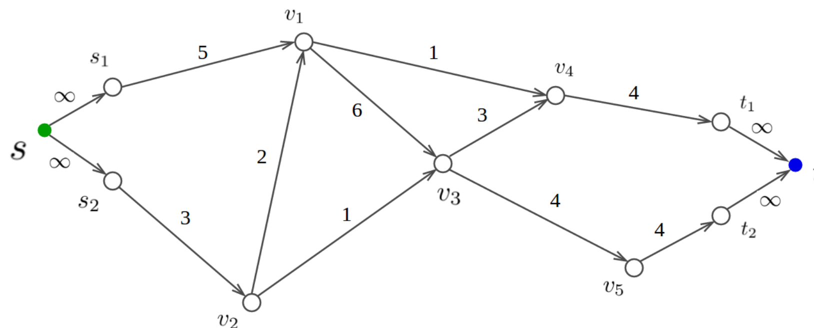


Figure 3: Graph/network with one source and one sink

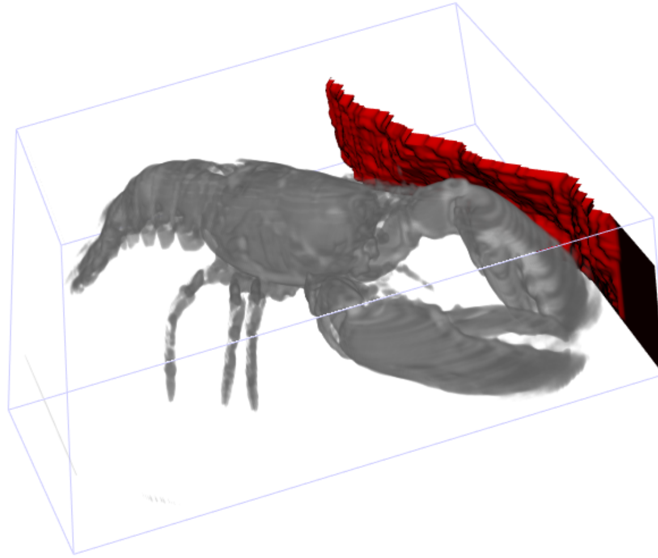


Figure 4: 2D seam computed from the "lobster" dataset

2.2 Algorithmic/computational challenges

The algorithmic challenge is mainly at the part of **solving the max-flow/min-cut for a graph with n^3 nodes and far more arcs**, where n goes from 10^2 to 10^3 for the datasets I plan to test. Because the max-flow/min-cut algorithm is mostly a "global" algorithm that is presumably applied at once, we need to do it on one CPU. There should be some parallel (GPU programming) implementation of this algorithm available, but concerning accuracy and the focus of this project (visual quality), I'm still sticking on the algorithm and code written by Kolmogorov along with his 2004 publication [3].

Memory usage could be the other challenge, basically:

- Buffers to store original input data (10–100 MB) and intermediate computed primitives (energy function of the volume, graph object constructed from the volume etc.).
- One single input data itself is not extremely big to handle, but along the way of computation the demand for RAM memory increases.
- When trying to run experiments across many datasets simultaneously (each is expected to take a relatively long time), one computer cannot support it efficiently.

2.3 Simple HTC workflow: the computational plan

2.3.1 Why HTC?

The High Throughput Computing (HTC) environments are those that **deliver large amounts of processing capacity over some (relatively large) period of time** [4]. Associated tasks focus less on floating point operations per second (FLOPS) that dictates the level of "real-time" computation, which is the core concern of High Performance Computing (HTC).

This project can be formulated as an HTC task in the case if the algorithm is not parallelized, and there are a certain amount of input datasets to be processed. My current plan in this project is to **prioritize quality/correctness**, and **build a good workflow to run experiment with on a sequence of datasets one after another**. In other words, waiting time is tolerable as long as the program is effective.

Therefore, the jobs will be structured as **many single, independent jobs, each with an associated input dataset**. They can be executed either in a queue on one machine, or in separate machines.

2.3.2 Approach

Plan A: On my personal machine (laptop), compile the non-UI program exhaustedly to generate an independent binary executable. Then bring this executable + data sets to OSG Connect's log-in node, and send jobs out.

Reason: obviously, this is the neatest and the most "portable" case.

Plan B: On OSG Connect's log-in node, install necessary libraries (higher version of *CMake*, and the *Visualization Toolkit*); compile and test the program on log-in node, before sending jobs out, each to a remote node respectively.

Reason: it's better to keep a library on the log-in node, and test/modify the code there. The downside is that it takes some disk space (1–2 GB) to install libraries under the `$HOME` directory.

My plan A has failed because it's messier to compile the C++ classes (and super-classes) altogether than I thought. Plan B is half-way and the problem right now is the job can't access the library installed, when it is submitted to a remote OSG node.

2.3.3 Resources

I plan to request 1GB memory, 1GB disk space and 1 CPU, for each individual job. The generic submit file is the following:

```
universe   = vanilla
executable = ./gcut3D

should_transfer_files = YES
transfer_input_files = $(IMAGE_FILE)
when_to_transfer_output = ONEXIT

arguments = $(IMAGE_FILE)
output = out-$(Cluster)-$(Process)
error  = err-$(Cluster)-$(Process)
log    = log-$(Cluster)-$(Process)

request_memory = 1024M
request_disk   = 1024M
request_cpus   = 1
requirements = (OpSys == "LINUX")

queue
```

Each input data set (there are tens of them) corresponds to a `.sub` job file where the `$(IMAGE_FILE)` variable is substituted. All these jobs are threaded by a `.dag` DAG file and submitted via `condor_submit_dag`. There is an output file (the "shrinked" volume written out at the end of the program) which is transferred on-exit.

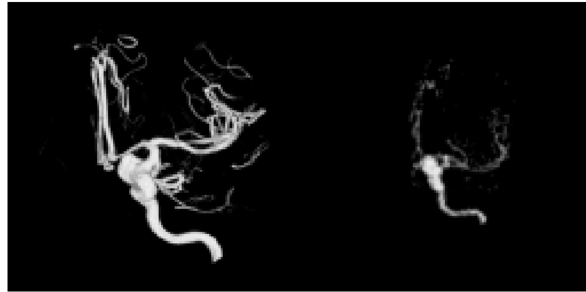
2.3.4 Unsolved issues

How to get the sent job (program) access the "shared library" (the `LD_LIBRARY_PATH` environment variable) that is installed/available on the log-in node.

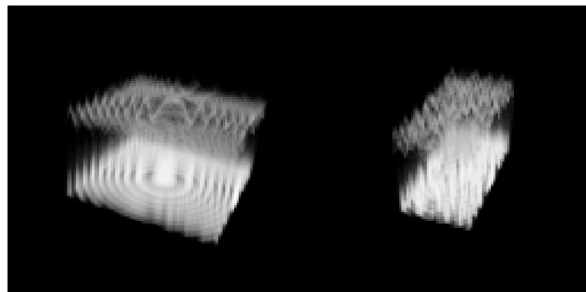
3 Expected Results (from local machine)

Here are several (expected) results of "retargeted" volumes, with their width reduced by half. The algorithm (seam carving operator) doesn't work very well at this point as some topological (i.e. "connectedness") features are lost after the x-dimension is reduced to half of the original.

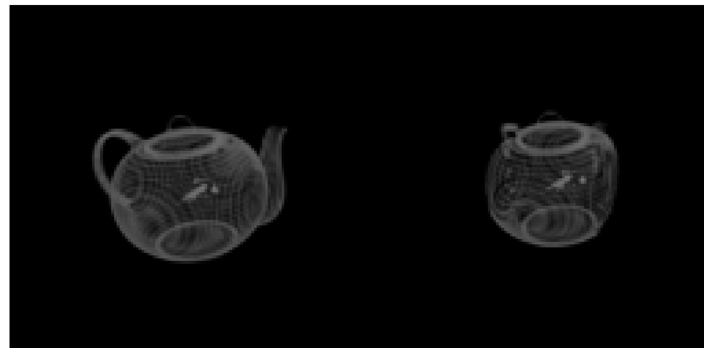
They are created from my local machine, not on OSG.



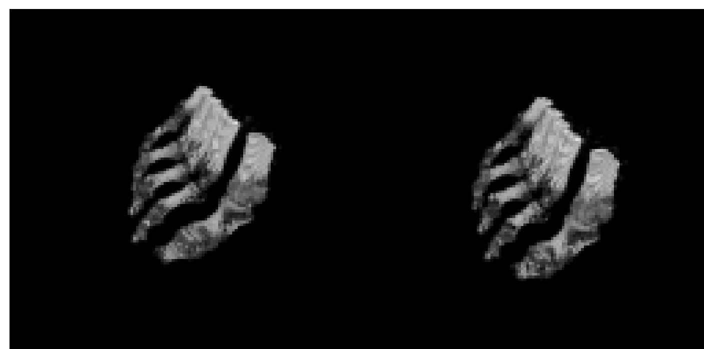
‘aneurism’ dataset



‘marschnerlobb’ dataset



‘BostonTeapot’ dataset



‘foot’ dataset

4 Conclusion: Answers To Questions On Website

What local resources do you have access to?

A laptop, and a desktop at lab which has much stronger GPU/CPU/RAM ability.

Would you use just local resources or do you need remote resources, too?

For testing the algorithm and running visualization interface, no; for generating outputs from numerous datasets, yes.

How would you turn your project into actual jobs?

It is the computation part of my MS thesis project, if that can be what "job" refers to.

What are the resource needs of the jobs themselves?

Sufficient amount of RAM to store buffers while the program is executed (takes a long time for one single dataset).

What sort of workflow, if any, would you use? Are there manual steps in your overall workflow? Could they be automated (e.g., with DAGMan)?

Manually "write" a `.dag` file is necessary, where input filenames are included.

How much data do you need to move around? Which type of data situation do you have? What is your plan for data management?

Input datasets totalling only 100+ MB. Management plan is simply transferring each data set along with the job.

Do you think your project is better suited for HTC or HPC? Why?

If the max-flow/min-cut is not parallelized through GPU programming with a correctly-defined algorithm, then it's suitable for distributed computing: let multiple CPUs run the multiple jobs respectively. It is potentially HPC-able.

It is not very HTC-like at the moment because every data set (a 3D image) itself isn't super large. However, the workflow as a whole involves a sequence of input images as well as a sequence of output images.

What security or privacy concerns do you have with your project? Do you need to do anything special regarding security?

Generally it's good. No special security need.

How would your science be transformed by increasing the amount of computation you can use?

Before the algorithm is improved (more efficient / parallelized), it would only make a difference in that more results can be generated in a short period of time (a few days) and **all jobs don't have to wait in a queue in the same/few computer(s)**.

Acknowledgement

First I want to thank Dr. Joshua Levine, whose recommendation letter earlier in March was likely why I got funded to join other students at the School in July.

School coordinators and instructors: From the very beginning to after-school assignment/coordination, every single procedure was very well planned and takend care of, giving me the impression of a "team" of people hosting the event to welcome (also a team of) students/researchers. The level of dedication and quality of work sufficiently represent the organization of OSG and the CS Department at UW-Madison (plus, the great dinning service of the University!).

Fellow graduate students: The most important part aside from learning the HTC knowledge is joining and being with the group of great graduate students / aspiring researchers during the five days at the School. On one hand, we communicate, help and have fun with each other during the study — especially knowing the young faces among us gave me the familiarity of the peer people in the context of sciences. On the other hand, as already being a major of computer science, I was within and along with this group of actual scientists in their multiple displines. It helped me, subtly, realize where my position is and figure out how am I going to advance the path.

It is a very great experience. In particular I was stunned by the depth of academics at the University of Wisconsin-Madison. All the instructors, students and people I met at this beautiful city have, with no doubt, demonstrated the power and determination of sciences.

References

[1] Avidan, S.; Shamir, A. *Seam Carving for Content-Aware Image Resizing.* ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2007

[2] Rubinstein, M.; Shamir, A.; Avidan, S. *Improved Seam Carving for Video Retargeting.* ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2008

[3] Boykov, Y.; Kolmogorov, V. *An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision.* EEE Transactions on PAMI, Vol. 26, No. 9, pp. 1124-1137. September 2004.

[4] HTCondor *High Throughput Computing (HTC).* `http://research.cs.wisc.edu/htcondor/htc.html`

Appendix: Some Notes (at the User School)

The Unix shell

(July 24)

Read-evaluate-print loop, REPL: when the user types a command and then presses the Enter (or Return) key, the computer reads it, executes it, and prints its output. The user then types another command, and so on until the user logs off.

It sounds like the user sends commands directly to the computer, and so as the other way around. What actually happens is there is a **program in between, called the command shell**. "Shell" is for its enclosing the OS to make it simpler to interact with.

The most popular Unix Shell: **BASH**, for **B**ourne **A**gain **SH**ell which it's derived from a shell written by Stephen Bourne.

HTCondor

Why, and what you can change your science through lots of computing resources?

What are the barriers in your work, because of the lack of computing?

We **parallelize** the serial tasks.

- **High-Performance Computing** is more inter-dependent sub-tasks.

- **High-Throughput 64Computing**: independent tasks (really the world of "super-computers"). In other words, in HTC, you chop it up to solve the highly indenpendent tasks.

No special programming; **number of concurrently running jobs** is the concern.

- Is your problem **"HTC-able"**?

- Which available resources are best for you?

Submit tasks to a **queue**. HTC Condor is a tool for this.

Job is a compute program or a run of it.

Machine is a physical computer; may have multiple processors; and note that:

$$\text{one processor} \longleftrightarrow \text{multiple cors.}$$

Queue is more than waiting in line.

Basic job submission (HTCondor)

`queue` is the keyword of running a job.

`condor_submit` — **creates a cluster**.

`condor_q` to view jobs (lists all of your jobs waiting/running)

`condor_rm` remove a job.

`condor_status` view slots

In HTC (and probalby any other unknown platform), a typical configuration is to have your code more "blackbox-like", namely **being able to execute and only taking in input arguments**.