# ADAPTIVE CO-SCHEDULER FOR HIGHLY DYNAMIC RESOURCE

by

Kartik Vedalaveni

A THESIS

Presented to the Faculty of

The Graduate College at the University Of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master Of Science

Major: Computer Science

Under the Supervision of Dr. David Swanson

Lincoln, Nebraska

May, 2013

# ADAPTIVE CO-SCHEDULER FOR HIGHLY DYNAMIC RESOURCE

Kartik Vedalaveni, M.S

University Of Nebraska, 2013

Adviser: Dr. David Swanson

Schedulers come with a plethora of features and options for customization that fulfill myriad goals of clusters and data centers . Most state of the art schedulers do not take into account load on resources like RAM, Disk I/O or Network I/O for scheduling purposes [8]. This is as much true for the local schedulers as much it is for the grid. On the grid there could be different schedulers on different sites and we cannot rely upon features of one kind of scheduler to characterize the nature and features of scheduling. There are many kinds of scientific applications that are run on the clusters and grid. Some might make use of Disk I/O largely and some might need large amounts of RAM[11]. Unchecked use of these resource will give rise to several issues on the cluster.

One such issue is when resources like RAM, Disk I/O or Network I/O is used in an unchecked manner and performance degradation that occurs as a result of it. Further scheduling jobs that claim the degraded resources could overwhelm the resource to an extent that the resource will finally stop responding or crash the system[11]. Often there is a need to extend these schedulers to solve situations arising from these new use cases either by writing a plugin for existing schedulers or by Co-Scheduling.

With an increase in the number of entities concurrently using the resource, there is a need to monitor and schedule concurrent and unmanaged access to any given resource to prevent degradation.These issues that we encounter in real life at Holland Computing Center [11] are the basis and motivation for tackling this problem and develop an adaptive approach for scheduling that is aware of multi-resource degradation detection,

load-balancing across multiple sites and with a goal to run clusters at high efficiency and share resources fluidly.

## ACKNOWLEDGMENTS

First and the foremost, I would like to thank my advisor Dr. David Swanson. Throughout my research and writing he's been a constant source of support for me. His guidance and teachings have helped not only to shape my thesis but also all walks of my life. I thank David for identifying this interesting research problem and bringing it to me. His hard work has been constant source of inspiration for me. I'd like to thank Derek Weitzel for working with me to solve the issues with grid submissions and for helping me with his technical expertise.

I'd like to take this opportunity to thank the entire team at Holland Computing Center for providing me with the required resources, support and advise even after I broke their systems. Its been a wonderful experience working with these folks.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Grid Computing as defined by Ian Foster in his paper The Anatomy Of the Grid [6] states that *Grid computing is about controlled sharing of resources with resource owners enforcing policies on the owned resources.* The resources come in the form of hardware and software that allows us to submit jobs, run the jobs and monitor the jobs on the grid. Universities usually have multiple clusters across their campus and these are usually owned by different departments but stand united under the banner of the university. Campus grids can be thought of as mini grids in which jobs are spanned across multiple clusters based on the need of the user and available computing infrastructure across multiple clusters within the computing resources across the university and then overflowing to the national grid infrastructure[16].

Modern schedulers used in clusters provide numerable features for policy making, resource management and scheduling, the problem of cluster performance degradation that occurs when either of the resource is throttled is a problem that hasn't been addressed. The problem of performance degradation when many jobs are scheduled on a single system are either based on processor equivalence or based on the number of processor slots. Some of these schedulers like maui [8] are smart enough to take into account contention of other resources like RAM but ultimately convert the 2D vector values of CPU and RAM into single scalar value which drop to hard-coding the value or presenting these resources in a ratio which makes us question effectiveness of such scheduling mechanism.

At Holland Computing Center present in University Of Nebraska-Lincoln, we're tackling this issue of cluster degradation caused by over exploitation of one or more resource. We've come up with a solution by adaptively scheduling such highly dynamic resource on the grid and across multiple sites by adaptively scaling with respect to the quickness of a given cluster. We're scheduling based on least turnaround time of the sites which would help increase the throughput of the overall workload for the Co-Scheduler, which also is a good load-balancer in itself.

Existing schedulers depend on the availability of resources and frequent polling of it to determine the slots of scheduling. It should be noted that state of the art schedulers like maui/torque, slurm, condor take into account only CPU as a resource and the resources like RAM, Disk I/O or Network I/O are either ignored or their resource equivalent is converted into a scalar values which isn't an effective way of tackling the multiple resource scheduling problem as this might result in scheduling excessive jobs on single machine. In some schedulers for example it'll be pre-assigned on every node that 1 CPU-CORE will have 2 GB of memory this kind of pre-assignment seems like its addressing the resource degradation of RAM but suffers from the above mentioned problems. We take a turnaround time approach to measure degradation, we define a resource to be degraded if and when we submit jobs that results in increasing the turnaround time by 25% . It must be noted that there isn't explicit measurement of status of individual resources like RAM, Disk I/O, Network I/O, there are no resource managers keeping track of these resource allocations. The Co-Scheduler is driven by the fact that turnaround time increases when concurrent jobs accessing these resources reaches a threshold value which in-turn causes degradation and this is the basis for our scheduling.

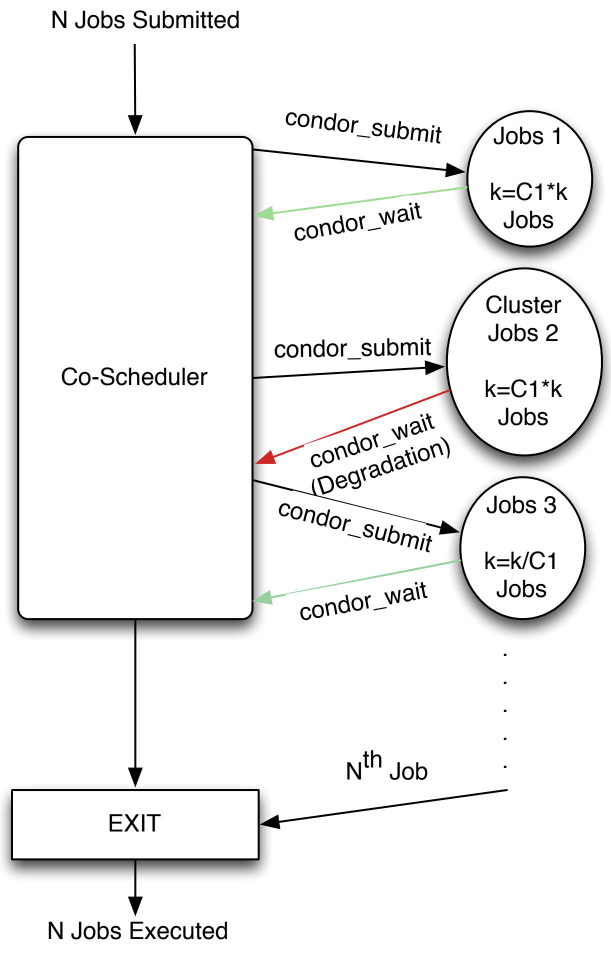

Figure 1.1: Degradation occurrence: Co-Scheduling

One aspect to Co-Scheduler is degradation detection and management by adapting to the throttling of a resource, another aspect is to efficiently distribute jobs across multiple sites which increases the throughput of the workload . The Co-Scheduler adapts to degradation by backing off submission rates and waiting for random period of time for the next incremental submission and efficiently distributes the jobs and load-balances across multiple sites, thus submitting more jobs to a site with lesser turnaround time and also ensuring to submit lesser jobs to the sites with larger turnaround time thus efficiently load-balancing across multiple sites on a grid and improving throughput and because the loads may vary dynamically. The capacity of jobs the cluster can efficiently run without degradation needs to be detected again and again after a random period of time.

In the figure 1.1, if C1 is called job propagation factor and k is initial number of jobs submitted then we can see that on each iteration we're adding $k = C1 * k$ jobs and on the next iteration if we have a degraded system then we submit $k = k/C1$ amount of jobs.

Finally, since the grid environment is a heterogeneous environment, it's absolutely necessary to use API that are available on all the systems across the grid. To implement such a Co-Scheduler we limit ourselves to condor based clusters and utilize libcondorapi. The result is a threaded Co-Scheduler that submits to multiple sites concurrently and is aware of multiple-resource degradation and loadbalances efficiently across multiple sites of the grid.

Please note that all the references to the Co-Scheduler in this thesis refers to the adaptive Co-Scheduler designed by us.

# Chapter 2

# Background

## 2.1 High-Throughput Computing

High Throughput Computing, HTC is defined as a computing environment that delivers large amounts of computational power over a long period of time. The important factor being over a long period of time which differentiates HTC from HPC which focuses on getting large amount of work done in small amount of time. The workloads that run on condor system doesn't have an objective of how fast the job can be completed but how many times can the job be run in the next few months.In another definition of HTC, European Grid Infrastructure defines HTC as a computing paradigm that focuses on the efficient execution of large number of loosely coupled tasks. [1]

## 2.2 HTCondor

HTCondor is a distributed system developed by HTCondor team at the University of Wisconsin-Madison. It provides High-Throughput Computing environment to sites that foster research computing and enables sites to share computing resources when computers are idle at a given site. HTCondor system includes a batch queuing system,scheduling policy, priority scheme, and resource classifications for a pool of computers, which is mainly used for compute-intensive jobs. HTCondor runs on both UNIX and windows based workstations that are all connected by a network. Although there are other batch schedulers out there for dedicated machines. The power of condor comes from the fact that the amount of compute power represented by sum total of all the non-dedicated desktop workstations sitting on people's desks is sometimes far greater than the compute power of dedicated central resource. There are many unique tools and capabilities in HTCondor which make utilizing resources from non-dedicated systems effective. These capabilities include process checkpoint and migration, remote system calls and ClassAds. HTCondor suit also includes a powerful resource manager with an efficient match-making mechanism that is implemented via ClassAds, which makes HTCondor lucid when compared with other compute schedulers[1].

## 2.3 Open Science Grid

Open Science Grid(OSG), provides service and support for resource providers and scientific institutions using a distributed fabric of high throughout computational services. OSG was created to facilitate data analysis from the Large Hadron Collider[12]. OSG doesn't own resources but provides software and services to users and enables opportunistic usage and sharing of resources among resource providers. The main goal of OSG is to advance science through open distributed computing. The OSG provides multi-disciplinary partnership to federate local, regional, community and national cyber-infrastructures to meet the needs of research and academic communities at all scales.

OSG provides resources and directions to Virtual Organizations(VO's) for the purposes of LHC experiments and HTC in general.

Building a OSG site requires listing background and careful planning. The major components of a OSG site includes a Storage Element and Compute Element.

Storage elements (SE) manage physical systems, disk caches and hierarchical mass storage systems, its an interface for grid jobs to underlying storage Storage Resource Management protocol and Globus Grid FTP protocol and others, A storage element requires an underlying storage system like hadoop, xrootd and a GridFTP server and an SRM interface.

A Compute Element(CE) allows grid users to run jobs on your site. It provides a bunch of services when run on the gatekeeper. The basic components include the GRAM and GridFTP on the same CE host to successfully enable file transfer mechanisms of Condor-G.

## 2.4 Grid Computing

As the cost of computers and network is decreasing, we're moving towards a paradigm shift in computing with the clustering of geographically distributed resource we're into the age of Grid Computing, the goal of which is to provide a service oriented infrastructure with institutions like Grid community and Global Grid Forum constantly trying to invest effort in making Grid a platform with standard protocols, seamless and secure discovery and access to infrastructure and interactions among the resources and services.

With the advent of parallel programming and distributed systems it became obvious that tightly couple computers can be used for computing purpose and this network of workstations gave rise to the notion of distributed computing. The Globus project began in 1996 and argonne national laboratory was responsible for process and middleware communication system called Nexus which provides remote service requests across heterogeneous machines and the goal of Globus was to build a global Nexus that would provide support for resource discovery, data access, authentication, authorization.

At this point Grid replaced the use of metacomputer and *researchers from coolaboration of universities termed Grid or Power Grid as integrated resources with integration of many computational visualization and information resources into a coherent infrastructure.*

The following are the goals and visions of Grid Computing:

**Seamless Aggregation of Resources and Services** Aggregation involves three things,

1. Aggregation of geographically distributed resources
2. Aggregation of capacity
3. Aggregation of capability

The key factors to enable such aggregation includes protocols and mechanisms to secure discovery, access to and aggregation of resources for the realization of virtual organizations and applications that can exploit such environment.

**Ubiquitous Service-Oriented Architecture** It is the ability of grid environment to do secure and scalable resource and data discovery along with scheduling and management based on wide variety of application domains and styles of computing.

**Autonomic Behaviors** The dynamism, heterogeneity and complexity of the grid has made lot of researchers rethink their systems. This new trend aims at system configuration and maintenance of the grid with least human effort and has led to numerous projects like autonomic grids, cognitive grids and semantic grids.

## 2.5 Globus

The Globus project is intended to accelerate meta-computing that build on distributed and parallel software technologies. The term meta-computing refers to a networked virtual supercomputer, constructed dynamically from geographically distributed resources linked by high speed networks [5]. Scheduling and managing such large group of heterogeneous resources is challenging and daunting task on the grid. The Globus toolkit provides framework for managing and scheduling of grid resources across heterogeneous environments.

The Globus toolkit consists of a set of modules, each module provides an interface for the provision of implementation of low level functionalities of the mechanisms of the toolkit:

- Resource location and allocation
- Communications
- Unified resource information service
- Authentication interface
- Process creation
- Data Access

We delegate our identification on the grid. *voms-proxy-init* is used to generate voms proxy at the submit host, we can specify `--voms` as our virtual organization, hcc:/hcc in this case and `--hours` as the number of hours the proxy would be active/valid, it must be ensured that the proxy period is approximately greater than the length of period of runtime of jobs for successful running of all the jobs.

HTCondor-G is the grid environment, when jobs are sent across grid universe using Globus software. Globus toolkit provides support for building grid systems. Submitting, managing and executing jobs have same capabilities in both HTCondor and HTCondor-G world. Globus provides fault tolerant features to HTCondor-G jobs. GRAM is Grid Resource Allocation and Management protocol, supports remote submission of computational request.

gt2 is an initial GRAM protocol which is used in Globus Toolkit version 1 and 2. gt2 is also referred to as the pre-web services GRAM or GRAM2.

gt5 is the latest GRAM protocol, which is an extension of GRAM2 and is intended to be more scalable and robust, referred to as GRAM5.

# Chapter 3

# Related Work

## 3.1 Comparison of existing mechanisms

There might be situations where we can have multiple condor pools and some of the pools might have many idle slots that are available for utilization. To efficiently utilize the resources across pools condor provides mechanisms like condor flocking and condor job router. These mechanisms provide similar functionality to the Co-Scheduler that I designed here. In the following sections, we compare and evaluate these mechanisms with the design and functionality of Co-Scheduler.

### 3.1.1 Condor Flocking

Flocking refers to a mechanism where jobs that cannot run in its own condor pool due to lack of resources runs in another condor pool where resources are available. Condor flocking enables load sharing between pools of computers. As pointed out by Campus Grids Thesis[16] flocking helps balance large workflows across different pools but not necessarily the jobs across these pools because of the scavenging and greedy nature of the condor scheduler.

To accomplish these goals condor uses multiple components, `condor_schedd` advertises that it has idle jobs to the remote `condor_collector` and during the next phase of negotiation, if it's found that there are computers available then they are allotted to the jobs in the matchmaking phase and the jobs then run on the remote pools. It so appears and the local job queue maintains as if the jobs are running locally.

Although Condor flocking has workflow balancing features across multiple condor pools it isn't aware of the slower and faster sites/pools. The adaptive Co-Scheduler keeps tabs on turnaround time and is aware of which site is faster or slower.If we look at the idea that scavenging idle resources increases throughput, it does but If we look at the case where jobs are greater than the available slots across all the pools, condor flocking stops working here as deeper understanding of sites would be required here to push more jobs at faster sites and importantly push less jobs at slower sites, its role is to scavenge idle computing slots across multiple pools.

Another aspect where condor flocking isn't designed to perform is when jobs are contending for the same resource (CPU, RAM, Disk I/O & Network I/O). Even though we'll be having idle slots on remote pools, condor flocking might not be able to use those slots as they'd be degraded because of the contention. In this case again Co-Scheduler comes in handy. Based on the turnaround time Co-Scheduler waits for random amount of time for degradation to clear and doesn't put excessive job load on the degraded resource. It diverts the jobs elsewhere to perhaps another site, which in-turn increases the throughput.

To conclude, we can say that condor flocking provides features of balancing large workflows and doesn't include features that would detect degradation in the cluster and also doesn't keep track of information of faster and slower sites, which might be exploited to increase the overall throughput of the system.

### 3.1.2 Condor Job Router

Condor manual defines the functions of job router to be the following:

> The Condor Job Router is an add-on to the condor_schedd that transforms jobs from one type into another according to a configurable policy[1]. This process of transforming the jobs is called job routing.

Condor Job Router can transform vanilla universe jobs to grid universe jobs and as it submits to multiple sites, the rate at which it starts submitting equals rate at which the sites execute them thus providing platform to balance large workflows across multiple grid sites and replenishing the jobs at faster sites once they get done. Job router sends more jobs to a site if the jobs submitted are not idle and stops submitting jobs if the submitted jobs sit idle on the remote cluster and Job router is not aware about which site is faster of slower.

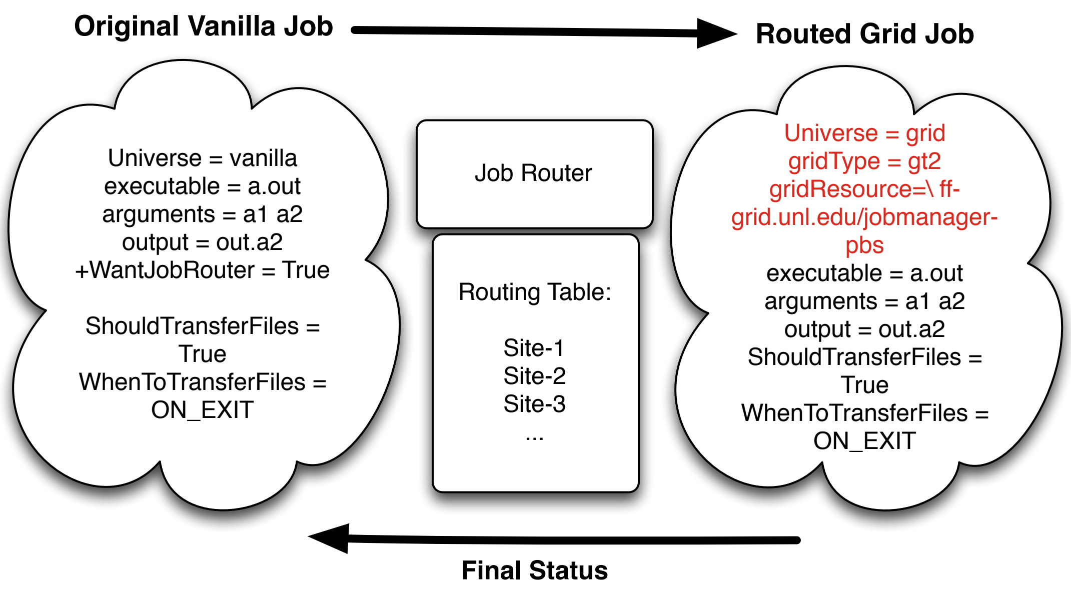

Figure 3.1: JobRouter: Transformation of Jobs

A job is transformed to the grid universe by making a copy of the original job ClassAd, modifying some attributes of the job , this copy is called the routed copy and this routed copy shows up in the job queue with a new job id[1].

Condor job router utilizes routing table which contains the listings of sites the job must be submitted to and the name of the grid resource declared and processed during condor config file which is defined by the new ClassAds.

```
 # Now we define each of the routes to send jobs on
JOB_ROUTER_ENTRIES = \
   [ GridResource = "gt5 ff-grid.unl.edu/jobmanager-pbs"; \
     name = "Firefly"; \
   ] \
   [ GridResource = "gt5 tusker-gw1.unl.edu/jobmanager-pbs"; \
     name = "Tusker"; \
   ] \
   [ GridResource = "gt5 pf-grid.unl.edu/jobmanager-condor"; \
     name = "Prairiefire"; \
   ]\
```

Condor Job Router seems to be a step up from the Condor Flocking in terms of scavenging resources and sending the extra jobs to another condor pool. Condor Job Router also maintains the rate of the jobs on submit hosts equal to that of remote clusters But the job router does not keep track of how fast each condor cluster is. Thus condor job router does not optimize the displacement of jobs to slower cluster. Since it maintains the rate of submission of jobs, we can be sure that if a job is complete at a faster site, its immediately replaced by the next one. The same is true for a job at slower site too which might result in increasing the degradation if there exists some, for example if the job router kept track of slower sites, it can send less jobs to slower sites and thereby increasing the overall throughput of the given workflow.

A preliminary examination shows that condor job router does seem to have degradation detection features, it does not submit to a pool that already has idle jobs but a close examination reveals that even though it submits to a pool with non-idle jobs it isn't aware of the fact that the jobs in the queue would have undergone degradation due to contention on same resource and thus isn't a viable mechanism for degradation detection.

# Chapter 4

# Adaptive Co-Scheduler For Highly Dynamic Resources

## 4.1 Introduction

Due to the heterogeneous nature of the grid there could be problems with resource sharing on the grid. I came up with a solution that primarily solves the problem of degradation due to contention among jobs for any given resource like RAM,I/O and network. The contention among jobs sometimes give rise to degradation depending upon the availability of the resource that results in reduced performance of the cluster or crashes the cluster altogether. Efficient and high throughput distribution of jobs across multiple cluster is another challenging problem that we have solved in this particular Co-Scheduler Solution.

## 4.2 Degradation detection and Capacity based scheduling

In the first part of the solution we target the basic problem of degradation to give an example, suppose we have a file server that can serve 100 MBps of data which is available for concurrent access and user has submitted 200 jobs each consuming 10 MBps of data I/O. If 100 CPU slots are available, a typical state of the art Scheduler schedules these 200 jobs without actually looking at the I/O load. This results in a degraded system, overtime if more jobs are Scheduled to utilize this file server. This file serve might even crash. We would need a degradation handling mechanism that would adaptively scale with the load and exponentially backoff during high contention period thus we need an intervention in the form of Co-Scheduler that intelligently handles degradation. We start the capacity algorithm by submitting jobs to the cluster and measure the turnaround time of each iteration if we find that the current turnaround time of an iteration is 25% greater than the previous iteration we term it as a degradation. We now try to find the best capacity of jobs between the current and the previous iteration by exponentially backing-off between these two iteration. We keep finding the optimal capacity dynamically for each iteration as the contention may change and optimal capacity keeps varying. Once the optimal capacity is found then it is retained for job submission for random amount of time limited to maximum of ten iterations of jobs thus solving the problem of performance degradation.

## 4.3 Multi-site load distribution based scheduling

The other problem that we tackled was that of efficient distribution of work load to multiple sites of the grid. Some of these sites could be error prone, some of them could be faster and some more can be slower. We need to take an approach that improves the overall throughput of the workload, thus we keep track of average turnaround time

```
C1 -> Job propagation constant
multiSite -> list storing turnaround times of multiple sites.
low ->  min(multiSite)
high -> max(multiSite)
avg -> average(multiSite)
```

$$f(C1) = \begin{cases} C1 * 4 & multiSite_i = low \\ C1/2 & multiSite_i = high \\ C1 * 2 & multiSite < avg \end{cases}$$

Figure 4.1: Multi-site Job distribution function

for each batch of jobs submitted to the cluster. We take average of turnaround time of the batch of job that is submitted across multiple cluster we group turnaround times of these batches of jobs based on the value of the turnaround time to be greater than, lesser than or equal to the average turnaround time of all batch of jobs. This gives us a classification of sites that run faster, we exploit this information in our scheduler. To implement multi-site jobs submission and load balancing feature we start submitting one job to each cluster and measure the turnaround time once the job returns. Now we have the information of turnaround time of a particular resource of all the clusters. Continuing to submit jobs concurrently in a similar manner would result in the completion of the workload in a non efficient way. Out of the list of all the turnaround time of different sites we take the sites that have lower than the average turnaround time of all batches of jobs and increase the job propagation factor for these jobs. This in-turn increases the number of jobs submitted to that site and helps us efficiently distribute workload to faster clusters than the slower clusters ( submitting to faster site would enable us to submit more number of jobs to the particular site. As the turnaround time is lower we would finish jobs quickly and make room for more number of jobs if we submit less to slower sites that can in turn help us improve the throughput to a greater extent by enabling us to stop further degradation at those sites and help us submit more to faster sites . ) Error handling mechanism is gracefully handled by condor and by this approach we have an efficient system with increased throughput and with proper distribution of load across multiple clusters.

The co-scheduler requires the presence of a condor installation and libcondorapi. It is written in the C++ language and has extensively made use of pthreads for synchronization and multithreading. The co-schedule has two components to it,the first one is capacity detection algorithm which is found by measuring degradation and other aspect to it is multi-site workload distribution across grid .The following paragraphs detail the engineering aspect of the design. The program begins by taking number of jobs, sites information and submit script information as its input. These number of jobs are the ones submitted across multiple sites. The next input file contains information on the list of sites that can be used for load balancing in the GridResource format of the condor ClassAds API, like *tusker-gw1.unl.edu/jobmanager-pbs*. It is assumed that all the sites listed in the sites input file are working and do not have any misconfiguration issues. The third and the final option to the scheduler is the submit description file. All the job ClassAd information and requirements can be written in this section and all of these will be applicable on the grid when a particular job is scheduled. The two main important attributes required in the submit description file is *universe* which needs to be grid, all the time as we are submitting to grid sites and the other most important thing is the *grid-proxy*. As we know that `condor_wait` [1]can be used to wait for a certain job or number of jobs to complete, it watches the log file and sees if the completion entry of the job is made in the log file that is generated by the *log* command in the condor submit description file. There are two options we can specify to `condor_wait` one is -num, number-of-jobs that waits till the number-of-jobs are completed and the other option is to specify -wait, seconds that waits for seconds amount of time, if nothing is specified `condor_wait` waits indefinitely till the job(s) are complete.

## 4.4 Implementation of Capacity based scheduling

This algorithm detects degradation and once we detect degradation we find the optimal capacity:

**Algorithm 1** Algorithm for determining optimal capacity by detecting degradation

```
$c2 \leftarrow 1.25$ {c2: Degradation factor}
while true do
  if T2 < c2*T1 then
    $jobSubmission(k)$
    $T1 = (T1 + T2)/2$
  end if
  if T2 > c2*T1 then
    $degradation\_high \leftarrow k$
    $degradation\_low \leftarrow k/2$
    $optimalCapacity(degradation\_high, degradation\_low)$
  end if
end while
```

**Algorithm 2** Algorithm for determining optimal capacity by detecting degradation

```
$mid \leftarrow (high + low)/2$
$k \leftarrow mid$
$jobSubmission(k)$
if T2 < c2 * T1 then
  $T1 \leftarrow (T1 + T2)/2$
  optimalCapacity(mid,high);
end if
if T2 > c2 * T1 then
  optimalCapacity(low,mid);
end if
return mid
```

Diagram:

## 4.5 Implementation of Multi-site load distribution based scheduling

Multi-site load distribution based scheduling uses the turnaround time at each site for sending future jobs, its a threaded algorithm, the following is extracted function per thread:

**Algorithm 3** Algorithm for distribution of workflow load across multiple sites on the grid

```
$c1 \leftarrow 2$ {c1: Job propagation factor}
$high \leftarrow 0$
$low \leftarrow 0$
$average \leftarrow 0$
{multiSite is a list storing turnaround times of multiple sites.}
$low = min(multiSite)$
$high = max(multiSite)$
average = $\sum_i multiSite_i$ / Sizeof(multiSite)
if turnaroundTime_Thread == low then
    $c1 \leftarrow c1 * 4$
end if
if turnaroundTime_Thread == high then
    $c1 \leftarrow c1/2?(c1/2 : 1)$
end if
if turnaroundTime_Thread < average then
    $c1 \leftarrow c1 * 2$
end if
```

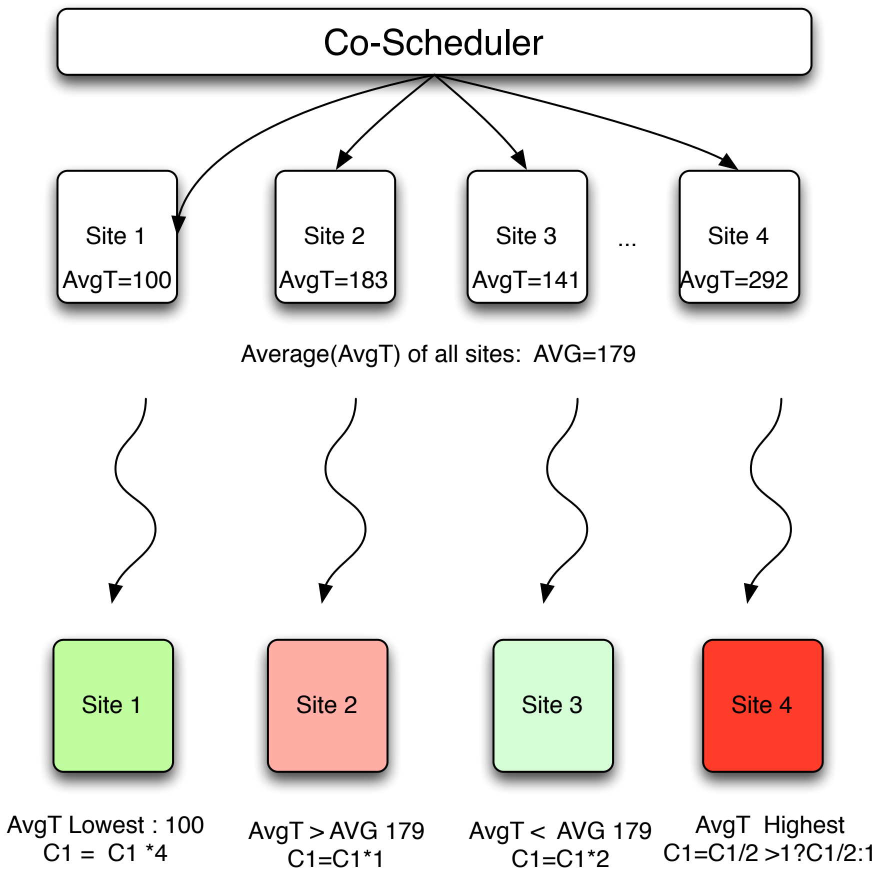

Figure 4.2: Multi-site scheduling algorithm overview, classification of sites into slower and faster

Diagram:

## 4.6 Programming APIs

### 4.6.1 Condor Log Reader and User API

Condor provides Job Log Reader API[1] that polls logs for job events by giving us API access to the events and outcomes. The following is the constructor for initializing a ReadUserLog object.

Constructor: ReadUserLog reader(fp,false,false);

```
ULogEventOutcome (defined in condor_event.h):


Status events for job detection:


ULOG_OK: Event is valid

ULOG_NO_EVENT: No event occurred (like EOF)

ULOG_RD_ERROR: Error reading log file

ULOG_MISSED_EVENT: Missed event

ULOG_UNK_ERROR: Unknown Error
```

All the entries in the log file end with .... All the job log entries as named as events and these events could range form being `ULOG_OK` where the event has taken place and is valid to `ULOG_UNK_ERROR` where an error has taken place.

The following pseudo-code is an extract from the logReader function that first detects all the valid events and then based on the data-structure of the event object detects if the event kind is `ULOG_EXECUTE` meaning the job has begun executing via eventNumber data member. Finally we detect the `ULOG_JOB_TERMINATED` event where job has successfully terminated. We also cast more general event object into JobTerminatedEvent to access data members of the JobTerminatedEvent.

Job Submission Pseudo-Code:

```
void logReader(string hostFile, args *data, int nSites) {
        FILE *fp;
        ReadUserLog reader(fp,false,false);
        ULogEvent *event = NULL;
             while(reader.readEvent(event)==ULOG_OK)    {
```

```
            if((*event).eventNumber==ULOG_EXECUTE ) {
            //Cast into Execute Event
                ExecuteEvent *exec
                = static_cast<ExecuteEvent*>(event);
                //condor_wait -num K, where K is the
                //amount of jobs completed till the wait.
                    char tmp[100];
                    sprintf(tmp,"condor_wait -num
                    \%d \%s",count,hostFile.c_str());
            }
            if((*event).eventNumber==ULOG_JOB_TERMINATED)   {
            //Cast into Job Terminated Event
                JobTerminatedEvent *term
                = static_cast<JobTerminatedEvent*>(event);

                if(term->normal)    {
                    //on Normal termination,
                    //works only for local jobs,
                    //find the CPU time of local jobs
                }
            }
        }
}
```

### 4.6.2 Synchronization Co-Scheduler Code

There are critical sections in the code where synchronization becomes absolutely necessary. One such variable is number of jobs. Number of jobs executed across the sites should remain fixed and the value must match the value that has been given as input. In a threaded system where each thread is executing on a different cluster it becomes necessary to define `N`, number of jobs executed or executing as a critical section. Here we define a pthread mutex and lock it for all write accesses to the `N` and serialize the access to `N` and make conditional checks during job submission, not to allow job submissions when `N` is greater than input value. By serializing the access across multiple threads the total jobs executed remains equal to the input number of jobs. The following chunk of code block demonstrates the use of mutexes for serialization of `N` among the threads.

```
pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_lock (&mymutex);
sumOfK+=k;
pthread_mutex_unlock (&mymutex);
```

Figure 4.3: Mutex on Number Of Jobs, sumOfK variable

Another section of code where synchronization becomes important is while distributing the jobs to multiple sites. We're measuring which cluster is faster, in doing so we need information of turnaround time from all the sites before proceeding that means all the threads need to be executing a line of code before proceeding with the further program. Thus the absolute need for synchronization. To handle this problem we make use of conditional variable of pthread. The following chunk of code demonstrates the use of conditional wait and conditional signal variable that clears the block on all the threads waiting based on the given condition.

```
pthread_mutex_t syncMutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t synchronize_cv = PTHREAD_COND_INITIALIZER;

pthread_mutex_lock(&syncMutex);
multiSite.push_back(stats[thread_id].T1);
tid_multiSite.insert(std::pair<int,int>
(data->tid,stats[thread_id].T1)
);

if ( multiSite.size() < numOfSites )
{
  pthread_cond_wait(&synchronize_cv, &syncMutex);
}
else
{
pthread_cond_broadcast(&synchronize_cv);
}
pthread_mutex_unlock(&syncMutex);
```

Figure 4.4: Synchronization of threads after reading turnaround time

We need to have turnaround time of the first job, this enables us to measure how quick each cluster is. To do so, we need to wait for each job to complete on the first submission thereafter the jobs are submitted asynchronously and the average turnaround time is updated in the list. In this pseudocode the threads beginning first add the turnaround time to the list and conditionally wait if the size of the list is less than the number of sites. The last thread comes in and the same condition is voided and executes a conditional broadcast to unblock all the waiting threads and the scheduling proceeds further to asynchronously schedule further .

# Chapter 5

# Evaluation

# Chapter 6

# Conclusion

# Bibliography

[1] Condor manual, section 5.6. 2.1, 2.2, 3.1.2, 3.1.2, 4.3, 4.6.1

[2] Monika Choudhary and Sateesh Kumar Peddoju. Turnaround time based job scheduling algorithm in dynamic grid computing environment. In *Proceedings of the CUBE International Information Technology Conference*, CUBE '12, pages 490–493, New York, NY, USA, 2012. ACM.

[3] David Cieslak Douglas Thain and Nitesh Chawla. Condor log analyzer. http://condorlog.cse.nd.edu, 2009.

[4] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A worldwide flock of Condors: Load sharing among workstation clusters. *Future Generation Computer Systems*, 12(1):53–65, 1996.

[5] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of High Performance Computing Applications*, 11(2):115, 1997. 2.5

[6] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, August 2001. 1

[7] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.

[8] Cluster Resources Inc. Cluster resources :: Products - TORQUE Resource Manager:, Jaunary 2011. http://www.clusterresources.com/pages/products/torque-resource-manager.php. (document), 1

[9] Miron Livny and Rajesh Raman. High-throughput resource management. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[10] LLNL. Slurm reference.

[11] Holland Computing Center. Holland computing center, January 2011. http://hcc.unl.edu/. (document)

[12] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, et al. The Open Science Grid. In *Journal of Physics: Conference Series*, volume 78, page 012057. IOP Publishing, 2007. 2.3

[13] I. Sfiligoi. glideinWMS–a generic pilot-based workload management system. In *Journal of Physics: Conference Series*, volume 119, page 062044. IOP Publishing, 2008.

[14] J.V. Sumanth, D.R. Swanson, and Hong Jiang. Adaptive load balancing for long-range md simulations in a distributed environment. In *Parallel Processing, 2006. ICPP 2006. International Conference on*, pages 135–146, 2006.

[15] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The Condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4):323–356, 2005.

[16] Derek Weitzel. Campus grids: A framework to facilitate resource sharing, 2012. 1, 3.1.1

[17] M. Zvada, D. Benjamin, and I. Sfiligoi. CDF GlideinWMS usage in Grid computing of high energy physics. In *Journal of Physics: Conference Series*, volume 219, page 062031. IOP Publishing, 2010.