Experience and Practice of Batch Scheduling on Leadership Supercomputers at Argonne

William Allcock[1], Paul Rich[1], Yuping Fan[2], and Zhiling Lan[2]

[1] Argonne National Laboratory, Argonne, IL,USA
allcock@anl.gov, richp@anl.gov
[2] Illinois Institute of Technology, Chicago, IL, USA
yfan22@hawk.iit.edu, lan@iit.edu

Abstract. The mission of the DOE Argonne Leadership Computing Facility (ALCF) is to accelerate major scientific discoveries and engineering breakthroughs for humanity by designing and providing world-leading computing facilities in partnership with the computational science community. The ALCF operates supercomputers that are generally amongst the Top 5 fastest machines in the world. Specifically, ALCF is looking for the science that is either too big to run anywhere else, or it would take so long as to be impractical (i.e., "capability jobs"). At ALCF, batch scheduling plays a critical role for achieving a set of site goals within a set of constraints. While system utilization is an important goal at ALCF, its largest mission constraint is to enable extreme scale parallel jobs to take precedence. In this paper, we will describe the specific scheduling goals and constraints, analyze the workload traces collected in 2013-2017 from the 48-rack petascale supercomputer Mira, and discuss the upcoming scheduling challenges at ALCF.

1 Introduction

Argonne National Laboratory [1] is a U.S. Department of Energy (DOE) general research laboratory. Argonne performs research in a broad range of disciplines and operates several user facilities [2] that provide access to resources that are generally too large and expensive for universities or commercial companies to operate. One of those facilities is the Argonne Leadership Computing Facility [3] (ALCF). The ALCF fields supercomputers that are generally amongst the top 5 fastest machines in the world, as rated by the Top500 [4] site. From the ALCF web site:

> The Argonne Leadership Computing Facilitys (ALCF) mission is to **accelerate major scientific discoveries and engineering breakthroughs** for humanity by designing and providing world-leading computing facilities in partnership with the computational science community.

Unlike most divisions at Argonne, the ALCF's primary focus is not on **doing** research, though we do some, but is on **enabling** research by operating a supercomputer facility for researchers around the world. Specifically, our mission is

to enable the solution to the largest computational challenges, what we refer to as "capability jobs". We are looking for the science that is either too big to run anywhere else, or it would take so long as to be impractical. Access to ALCF resources is available via three different programs:

- Innovative and Novel Computational Impact on Theory and Experiment (INCITE) Program [6] (60% of the core-hours)
- Advanced Scientific Computing Research (ASCR) Leadership Computing Challenge (ALCC) Program [7] (30% of the core-hours)
- The Directors Discretionary (DD) program [8] (10% of the core-hours)

Note that these percentages are an allocation time constraint, but not a scheduling constraint. While they do generally end up split approximately along those lines simply because of the allocations, the scheduler does not track usage by program, nor does it use that to influence scheduling decisions.

The rest of this paper is organized as follows. We start by introducing Mira and Cobalt at Argonne in Section 2. Section 3 describes the goals and constraints on scheduling at the ALCF. Section 4 describes the current scheduling algorithm used at ALCF. Section 5 presents our analysis on Mira log and our key observations. Section 6 describes the upcoming challenges at Argonne. Finally, we conclude the paper in Section 7.

2 Mira: The 48-Rack Blue Gene/Q

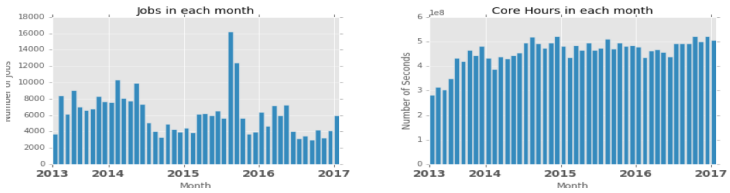
Mira is a is a 10-petaflops IBM Blue Gene/Q system [9] installed at Argonne National Laboratory. It is a 48-rack system, equipped with 786,432 cores and 768 terabytes of memory. Every 16 cores form a node and 512 nodes in a 4x4x4x4x2 structure are grouped into a midplane. Each rack consists of two midplanes and the system has 48 racks. Mira was ranked ninth in the Top500 list in November 2017 [4]. It features 5D torus interconnection, which reduces the average number of hops and latency between compute nodes and thus achieves high efficient computation and saves the energy for transporting data across long distance. The smallest partition size on Mira is 512 nodes. Jobs smaller than the minimum partition run exclusively on one partition.

Mira logs record scheduling events of batch jobs on Mira including requested and utilized resources (i.e. requested time, used time, requested nodes count, used nodes count), timestamps of major scheduling events (i.e. submit time, start time, end time), and job execution environment (i.e. machine partitions). Table 1 shows the basic information about the Mira logs. There are 554 projects and 1054 users submitted jobs on Mira. 283,385 jobs were submitted during this period and approximately 70% of jobs exited system normally. Figure 1 shows the number of job submitted and core hours used per month.

The information in logs are useful for data analysis and new scheduler evaluation. Data analysis help administrators learn about machines' status and identify problems in a system. To evaluate a new scheduling policy, logs can be used directly to generate the input workload for trace-based simulation. In simulations, jobs arrive according to timestamps in a trace. In order to schedule jobs, simulators require each job providing requested time and node.

Table 1. Basic Information of the job log

Period	April 9, 2013 - January 31, 2017
Total number of jobs	283,385
Total number of projects	554
Total number of users	1,053
Percentage of jobs exit system normally	70.464915221342%



(a) Number of job submitted (b) Core hours used

Fig. 1. Number of jobs submitted and total core hours used in each month.

3 ALCF Goals and Constraints

Schedulers and scheduling policies exist to achieve a set of goals within a set of constraints. Conceptually, that seems obvious, but enumerating them is not necessarily straightforward, accomplishing them is difficult, and because of the goal to enable capability jobs, the ALCFs are somewhat different than most.

The most obvious constraints are the metrics a facility is required to report on to its department, funding agency, etc. For the ALCF, the scheduling related items are:

- Total core hours delivered: This is based on theoretical hours (number of cores in the machine multiplied by the hours in a year) multiplied by factors for availability and utilization.
- Core hours delivered to the INCITE program as described above.
- Capability hours: A job is considered capability if it uses $\geq 20\%$ of the nodes in the machine for a single scheduler job. Each year a minimum percentage of our jobs that have to be capability is agreed upon with our sponsors.
- Utilization: Utilization is reported, but there is no set minimum. Where many facilities will have utilization as the primary metric, the ALCF sacrifices some utilization in order to prioritize capability jobs.

Until recently, ALCF has been fielding IBM Blue Gene [9] machines, though the most recent machine is a Cray XC40 machine based on the Intel Knights Landing many core processors. More than the processor, one could argue that what sets high-end supercomputers apart are their networks. They generally have proprietary, low latency, high bandwidth networks. The Blue Gene series runs an IBM proprietary torus network. The Blue Gene/P series had a 3D torus, the Blue Gene/Q series, including the ALCF's current production machine named Mira, has a 5D torus, and the new Cray has a dragonfly network topology.

This leads to our first constraint: The architecture of the machine itself. The Blue Gene is a very unique machine. Like all things, it has its advantages and disadvantages. The Blue Gene torus network is dynamically programmable enabling every job to have its own dedicated, electrically isolated network. This means practically zero performance jitter during communications, and while not a significant concern on an open science machine, zero chance for packet snooping for the security conscious. However, it does put constraints on node allocations. One of those constraints is that you cannot pick any random set of nodes for a given job. The nodes must be allocated in "partitions" which are controlled by the torus cabling. This generally means contiguous nodes, though 4K partitions can be built as 2K nodes, skip 2K nodes, and then the other 2K nodes. Nor can you allocate a single node. The Blue Gene has the concept of a pset, which is a set of compute nodes, and it its associated I/O node(s). On Mira, the ALCF's BG/Q, due to the I/O node configuration the pset size is 128 nodes, which is physically the smallest allocation possible. Additionally, due to the ALCF mission to run large jobs, by policy, as well as for resource isolation considerations, nothing smaller than a midplane, which is a contiguous group of 512 nodes, is allowed. See Figure 2.

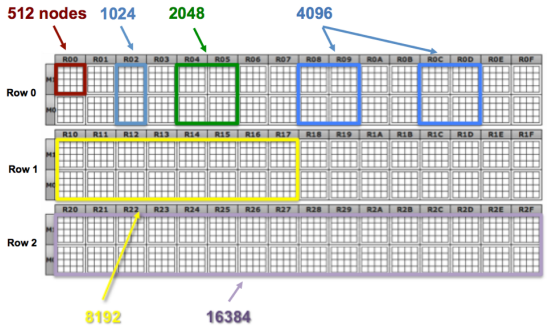


Fig. 2. Partitions on Mira.

These constraints simplify the scheduling. They effectively reduce a 49,152 node BG/Q to a "96 allocation unit" system, where each "allocation unit" is a 512 node midplane. It also makes fragmentation much less of an issue. Due to the torus cabling design, only specific arrangements of midplanes are allowed to from larger partitions. Additionally, due to the network partitioning and isolation scheme, links are allocated to one and only one active partion at a time. To deal with this, the torus connections are considered a first class resource that needs to be scheduled as well. To schedule a job, you not only needed the number of nodes, but those nodes needed to have the appropriate torus network connections.

Another hardware constraint that was different between the BG/P and the BG/Q was where the network hardware was at and how it could be managed. On the BG/P there is a separate link card that has all of the midplane to midplane network connectivity on it. The implication of this is that if a node failed, a node board (containing 32 nodes) could be taken offline to replace the node,

and then brought back into production without affecting the network traffic of any other job. However, on the BG/Q, the torus optics are on the node boards and it has the concept of "pass through" such that a job not using the compute nodes on that particular node board could be using the network optics. This makes maintenance much more impactful. On the BG/P, when a node dies, the job dies, and the system automatically takes that node board offline. The scheduler would pick that up and automatically not schedule jobs on it, and the admin could immediately go replace the node if he chose. Similarly, on the BG/Q the node board is taken offline and the scheduler won't schedule on it, but the admin must wait for any job using pass through to finish before they can change a node. This lead to the addition of an extension to reservation support, such that you could choose to reserve only the compute nodes, or the compute nodes and the optics links. This avoids scheduling any future jobs that use the pass through links which could, theoretically, infinitely delay replacement of the compute nodes.

A second constraint came from the selection of projects that ran on our machines. We were supporting computer science projects doing operating systems research (Plan9 [10] and ZeptoOS [11]). This required us to be able to boot the Blue Gene into an alternate operating system. As an alternative OS could involve an IO node image as well [25], resource isolation to the IO node is also required.

These two constraints, the Blue Gene hardware and the need to boot alternate operating systems, drove the selection of scheduler software. ALCF uses an Argonne written and maintained scheduler called Cobalt. It's origins were from a DOE program called the "Scalable Systems Software Project" in the late 1990s, which later went on to become the SciDAC Scalable Systems Software ISIC [13]. When the ALCF was formed, the only scheduler that supported the Blue Gene was IBMs LoadLeveler, but it could not support alternate OSes. As a result, the ALCF decided to modify Cobalt [12] to support those two requirements.

3.1 Policy and Mission Constraints

For many facilities, utilization is the number one goal. While the ALCF considers utilization, its larger mission constraint of enabling extreme scale parallel jobs takes precedence. The ALCF prioritizes jobs that take up a significant fraction of the machine, up to and including jobs that consume the entire machine. In facilities that optimize purely for utilization, large jobs can be blocked for significant periods of time because smaller jobs monopolize the resources. In theory, if utilization were the only metric, a full machine job would never run unless it was the only job in the queue. Fair share algorithms help ameliorate "large job starvation" delays, but given that running the larger jobs is our primary mission, we needed a better way.

4 Description of the Current Cobalt Scheduling Algorithm

Cobalt does not attempt to construct a time-based schedule [24], nor does ALCF run a "fair share" algorithm. Instead, Cobalt periodically applies a schedul-

ing "utility function" that calculates a priority increment for each job in the queue. The jobs are sorted into priority order and sets of resources, some of which may be in use at that point, are selected to run the highest priority jobs. Cobalt will then begin to "drain" those resources to make room for the jobs, backfilling jobs where it can. This draining obviously hurts utilization, but supports the mission to run large-scale "capability scale" jobs. When the facility first began production operations back in 2008 the utilization was at approximately 70%. Over time, scheduling algorithms have improved, ALCF has educated users and helped scale the computational algorithms, and policy adjustments have been made that have improved the utilization to approximately 90%.

Resource selection/allocation is based on network topology, heuristics based on what resources are in use, and when various resources are scheduled to be available. On the Blue Gene/P platform a static-partitioning scheme was required, using site-specified predefined partitions. On the Blue Gene/Q platform, a dynamic, on the fly, partition construction scheme could be used, or a more traditional static partitioning scheme could be used. The ALCF chose to use a hierarchical static partitioning scheme that utilized some features of the system wiring to its advantage. For instance, because of the unique Blue Gene Torus design, there can be a natural regular hierarchical structure to node partitions. A 32K node partition consists of two 16K node partitions, each of which consist of two 8K node partitions, and so forth. The implication of this is that once a small job, say a 1K node job, is started, it not only consumes those nodes, but it blocks all the larger partitions it is a member of. For that reason, Cobalt will preferentially select partitions that minimize the number of additional larger partitions that are blocked. Due to the constraints imposed by block-exclusive wiring, fragmentation of the torus is costly in terms of the impact on the throughput of larger jobs, which is compounded due to a need to favor workloads involving large jobs.

Generally, all jobs that are not part of user requested reservations are submitted to the default queue. These jobs are then run through a "job router" which applies scheduling policy to place them into one of three queues based on size and requested wall time:

- Prod-capability: Any job that requests $\geq 20\%$ of the nodes is routed to this queue up to the maximum run time of 24 hours. This queue has access to all of the nodes on the machine. Ideally, every job would meet these criteria since these "capability" jobs are the ALCFs primary mission. Capability class core-hours is one of our reportable metrics.
- Prod-short: Any job that requests $< 20\%$ of the nodes and has a requested wall time of ≤ 6 hours is routed to this queue. This queue has access to all of the nodes on the machine. These are not capability jobs, but they are also relatively short so will block larger jobs for a shorter period of time.
- Prod-long: Any job that requests $< 20\%$ of the nodes and has a requested wall time > 6 hours is routed to this queue. This queue only has access to 1/3 of the nodes on the machine. These jobs are small and long and so can block larger jobs for a significant length of time and thus are very disruptive our desired goal of running larger capability jobs. By restricting them to 1/3

of the machine, we guarantee that we have at least one 32K partition, and all of its sub-partitions, available to the prod-capability queue.

There are two corner cases that the Blue Gene architecture brings to this. First, the 20% limit for being considered capability was set when the ALCF had a Blue Gene/P, called Intrepid [14]. Intrepid was a 40K node machine and 20% of that was 8K nodes, which was the size of one of the torus partitions. However, when the ALCF installed Mira, a Blue Gene/Q, it was 48 racks and 20% of that was 9.6K, which was not a supported partition size. To deal with this, a "split" capability metric was developed. A metric for both 8K nodes and 16K nodes was set and the weighted average of the two is 20%. The second corner case deals with jobs that request irregular size partitions. The partition sizes are generally powers of two with the exceptions of 12K (25% of the machine) and 24K (50% of the machine) partitions. In Figure 3, a pictorial depiction of the queues, prod-capability appears to be at 4096 rather than 8192. The line is at 4097, because if a job were to request 4097 nodes, Cobalt would have to allocate an 8K partition in order to accommodate it. For purposes of queue routing, a 4097 node job will get routed to the prod capability queue, but for purposes of the capability metric, only jobs that request $\geq$ 8K nodes qualify.

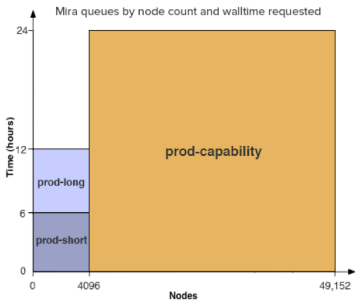


Fig. 3. Mira queues by node count and walltime requested.

While the job is being routed to its queue, a site-specific "filter" script sets an initial score for the job. This is based on the project that the job belongs to, with INCITE and ALCC projects getting a more favorable initial score, and the projects allocation status. Projects that have exceeded their allocation, in addition to being routed to backfill, are given a starting score that diminishes proportionally to how much time they have used beyond their initial allocation. Jobs in the backfill queue have their score capped to below that of any production queue job. After that, during each "scheduling iteration", which is currently 15 seconds, a "utility function" is executed for each job. This utility function calculates a priority increment that is added to the current job priority. The jobs are then ordered based on priority, and resource allocations are made to run the highest priority job first. The heart of the current utility function in use is:

$$\frac{(queued\_time - hold\_time)^2}{score\_wall\_time^3} * \frac{size}{MAX\_SIZE} * project\_weight \qquad (1)$$

Where

- $(queued\_time - hold\_time)$ is referred to as the eligible time; The length of the time that job has been in the queue and it was eligible to be considered to run.
- $score\_wall\_time$ is the requested wall time normalized to be a minimum of one hour and a maximum of 12 hours.
- $size$ is the number of nodes requested
- MAX_SIZE is the total number of nodes in the machine (48K or 49152 for Mira)
- $project\_weight$ is an arbitrary value that is based on the project / account being charged that defaults to 1.0, but which the facility can adjust if there is a reason to prefer or retard the score increment for that project.

The general concepts behind this algorithm are as follows:

- The fraction of the machine $size/MAX\_SIZE$ is a multiplier, so larger jobs will gain priority faster and will tend toward the top of the priority which meets the goal of enabling large jobs to run. This dominates the early part of score accrual and "stratifies" jobs.
- The longer the requested wall time, the longer the user should be willing to wait for it to run. This is handled in the first term of the equation:

$$\frac{(queued\_time - hold\_time)^2}{score\_wall\_time^3} \tag{2}$$

Once the jobs eligible wait time $(queued\_time - hold\_time)$ squared exceeds the requested wall time cubed, the priority growth increases rapidly. The assumption about longer wait times for longer wall times is psychology and arbitrary, but it has worked well. This time factor also provides an anti-starvation factor for jobs in the queue, ensuring that a job will eventually reach the front of the line and run.

4.1 Negative Accounts and Overburn

ALCF has a variety of policies regarding what to do when an allocation reaches zero. For Director's Discretionary accounts, they are blocked from running. If they want to continue running they must request another allocation. INCITE and ALCC projects may continue to run, but only in backfill. There is an exception referred to as "overburn". During the third quarter of the allocation year (INCITE is on the calendar year; ALCC is July 1st to June 30th), ALCF allows accounts to run in the prod-capability queue at up to 200% of their allocation before all of their jobs are forced to backfill.. Jobs that would go to prod-short or prod-long still get routed to backfill under this policy, even in an overburn period.

Overburn helps to smooth out the delivery time. We were finding that users were "hoarding" their allocations, causing them to wait until later in the year to perform their runs. As INCITE and ALCC allocations are "use or lose" at the end of their term, this would result in a rush at the end of the year and significant

increases in user wait times. This policy incentivizes both earlier running, as users that have not used their allocations do not benefit from the policy, as well as the use of capability size jobs. Part of this incentive is that these capability runs get normal capability priority, in this case. We disable overburn during the fourth quarter of an allocation cycle so that projects that were late to start or otherwise have large percentages of their allocations remaining do not have to compete for priority against projects that have completed their allocations.

4.2 Big Run Mondays

The ALCF typically engages in preventative maintenance on its resources once every two weeks on Monday. During these periods no jobs run due to highly disruptive work that may be occurring on the resource itself, or on supporting infrastructure of the facility. As this necessitates draining the full machines resources at the beginning of maintenance, we effectively open the machine for a full-machine job for "free". Before the machine is released back to normal operation, a set of capability-sized non-backfill, non-overburn jobs is selected and has their scores altered to be at the front of the queue. Scores are set such that the largest of the capability jobs run first, making the best use of this "free" drain of the systems resources.

4.3 Interactions with User Behavior

In some sense, scheduling algorithm and policy development are an ongoing "arms race" with the users of the system. Facilities are constantly receiving feedback from the users on issues they are seeing, new needs they might have, etc. This feedback may be direct via surveys or trouble tickets, or it may be indirect in how they interact with the scheduler. Everyone wants their results as fast as possible, and some users get extremely creative when trying to achieve that. Below we describe some of the user behaviors and how they were addressed.

4.3.1 Eligible Time vs. Queued Time

Originally the first term of the utility function was just $queued\_time^2$. However, we found that users were submitting dummy jobs and putting them on hold to let them accrue priority until they were ready to run the jobs. They would have all the right parameters, so the qalter fix above wouldn't discourage this, as exiting a hold does not cause a job to be requeued. The solution to that was to subtract the hold time so that we were using what we now refer to as eligible time in the first term.

4.3.2 Dependency Chaining

To actually complete the science, most science teams require many millions of core hours. To allow all projects to make incremental progress, maximum run times are established and the users must make their runs in pieces, writing out restart files, usually referred to as "check points", that the next job can use to pick up where the previous job left off.

Another common queue parameter is the "max queued" parameter. Even if thousands of jobs are required to complete an overall computation, only so many

can be in the queue at once. This keeps the size of the queue manageable, but also prevents a science team from queuing up an entire years worth of work and having them all gaining priority, which would effectively allow them to monopolize the queue. Sometimes the jobs are independent and can run in parallel and one job failure has no impact on the other jobs, but other times, the jobs are steps in an overall workflow and they must proceed in sequence and the next one can only proceed if the previous one completes successfully. For the latter situation, we provide "dependency chaining". When you submit a job, you can specify another job ID that it depends on which places that job in a "dep hold" state. The scheduler will ensure they run in sequence and if a job fails, the scheduler will place the dependent job into a "dep fail" state until the user can fix the problem and re-run the failed job. A scheduling issue for this is that because subsequent jobs are in a hold state, they cannot accrue priority and it extends the total time to solution. To ameliorate this problem, Cobalt assigns the starting score of a dependent job to a facility configurable percentage (currently 50%) of the priority that the preceding job had when it started running. For instance, if job B depends on job A, and Job A had a priority of 200 when it started running, when it completes, the starting priority of Job B will be set to 100 (50% of 200). An exception to this is that should the jobs current priority be higher than the newly calculated dependency priority, the current priority is maintained.

4.3.3 Using Qalter to Game the System

Early on, some users would submit a job so as to maximize its score function and accrue priority as rapidly as possible, and then qalter it to the real job parameters. For instance, they might submit a full machine job maximizing the $size/MAX\_SIZE$ term. Additionally, they might set a wall time of 5 minutes which means that the priority would begin to grow superlinearly (quadraticly) after only 11.2 minutes (square root of $5^3 = 11.2$; see the first term of the utility function described above). Then, just before it was ready to run, they would use the qalter command and modify the job parameters to what they really were, perhaps a 512 node job for 12 hours. To discourage this type of behavior, we modified Cobalt so that any qalter command that would result in a change of queue (for instance, from prod-capability to prod-long in the example above) results in the jobs priority being reset to the starting default as if it were a new job submission.

4.3.4 Adding a floor and ceiling on the wall time parameters in the utility function

In general, the behavior achieved with the current utility function accomplishes ALCF goals and usually behaves predictably. However, there were corner cases early on that required adjustments. One of them was regarding the *score_wall_time* parameter. The algorithm had been running stably, but utilization had dropped and ALCF began to get user complaints that the jobs start times were not behaving as expected. Investigation into this determined that mathematically, things were working as they should, but a recent change in max wall time from 12 to 24 hours for capability jobs had an impact we had not anticipated. This resulted in situations where a preferred capability job would

accrue score unusually slowly due to the very large denominator of the wait-time factor of the score function. Placing the 12 hour ceiling allowed for a good turnaround on these longer jobs without unduly favoring them. The one-hour floor was placed due to very similar scenarios where a large, short job would get to the front of the queue very rapidly. In instances where a user may be debugging at scale or running some other high node count, but short duration workload we would end up in a pattern where a user would submit a short job, run the job, and then submit another soon after the job ended, resulting in a very unfavorable "sawtooth" drain pattern, causing a severe impact to utilization. The ceiling and the floor corrected these two issues.

4.3.5 Risks of High Initial Scores

Very early in the life of Mira, there was a need to significantly increase the throughput of some projects that had trouble initially getting started on the new platform. A policy change was implemented such that a jobs initial score would depend not only on the queue, but the number of core hours left in the allocation. This would result in a very high initial score for any job in an "underburned" project. The following scenario would then happen:

1. A large, long job is the top priority and a set of resources is selected and begins to drain. This is hurting utilization, but that is expected and normal so far.
2. A small job is submitted from an unburned, favored, project.
3. After several hours of draining the machine for the large job, the small job enters ahead of, or soon passes the large job and becomes the top priority job. Because it is small and the scheduler had been draining, there is very likely an open resource, so the scheduler immediately starts the new top priority job (the small job). Once the small job is started, the large job is once again top priority, but now the partition is blocked by the small job, so a different partition is selected and it begins to drain.

This resulted in a loss of utilization, which, at the time was considered acceptable in light of the INCITE time delivery goal of the ALCF. This, however, had the side effect of making large, capability-sized jobs, very unfavorable on the system, in addition to user complaints of very unexpected scheduling behavior. Due to these negative effects, this initial score policy was reverted after the end of the INCITE year.

5 Mira Log Analysis and Key Observations

This section presents the results of our analysis on Mira logs from April 9, 2013 to January 31, 2017. We conduct our analysis on Mira logs from several aspects, such as queues, users, exit codes, modes, co-analysis with RAS logs, cycles, and account. Particularly, we provide job distribution on queues, exit codes, modes. We find two frequent users in Mira logs and examine the effect of their behavior on Mira. Next, we look for the correlations between exit codes and modes. Further, we analyze the Mira job logs with RAS logs. We provide users weekly and daily submission cycles. Finally, we show statistics of overburn per month.

5.1 Queues

In this subsection, we explore jobs in prod-short, backfill, prod-capability, prod-long, and prod-1024-torus queues. We make the following observations.

Observation 1 *In Table 2, the number of jobs in prod-short queue is much more than other queues. The total core hours consumed by jobs in prod-capability is more than half of the core hours used by all jobs. On average, jobs in prod-long queue ran longest time (24907 seconds), while jobs in prod-capability queue used the most number of nodes (13530 nodes). In addition, jobs in prod-capability on average consume large core hours (656480).*

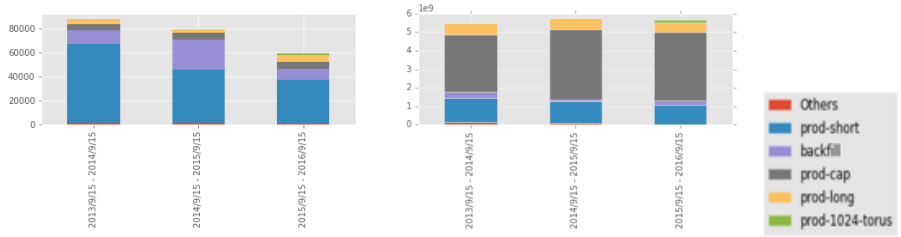
Observation 2 *In Table 2, jobs in prod-short queue have higher priority than jobs in backfill queue. Although jobs in backfill queue on average used less core hours than jobs in prod-short queue, the average wait time of the jobs in backfill queue is much higher than that of jobs in prod-short queue.*

Observation 3 *In Table 2, jobs in prod-capability queue have higher priority than jobs in prod-long queue. Jobs in prod-capability queue on average used more core hours than jobs in prod-long queue, their mean wait times are almost the same.*

Observation 4 *In Figure 4, the total number of jobs submitted in each year decreases, whereas the total core hours used in each year increases. From Sep. 15, 2013 to Sep. 15, 2014, we observe that a large decrease in the submission of short jobs, while the number of jobs in backfill queue increases.*

Table 2. Statistic of Queues

	Prod-short	Backfill	Prod-capability	Prod-long	Prod-1024-torus
Mean Used Nodes	1080	1043	13530	1253	1024
Mean Wait Seconds	45243	61279	278510	276839	17338
Mean Walltime Seconds	6754	4118	20151	39734	3499
Mean Runtime Seconds	4361	2789	12544	24907	14652
Mean Used Core Hours	23082	16794	656480	137328	66687
Mean Requested Core Hours	22406	16663	651597	33966	66687
Total Used Core Hours	4289056697	883159087	13126325662	2221830694	149446510



(a) Number of Jobs submitted (b) Total core hours used

Fig. 4. Number of jobs and total core hours used by queues in each year.

5.2 Users

In this subsection, we analyze the Mira logs based on users. We analyze two frequent users on Mira.

5.2.1 User 15303315089691

Observation 5 *User 15303315089691 submitted most jobs in Mira. In Figure 5, we can see from Aug. to Oct. in 2015, this user submitted more than 20,000 jobs in Mira. All the jobs submitted in this period of time queued in backfill queue and the job size is 512. In Figure 6(a), the submissions of user 15303315089691 lead to large increase in the total number of jobs submitted to Mira during that period. However, because these jobs are small jobs ran short times, they did not affect the total core hours used in Mira too much as shown in Figure 6(b).*

Observation 6 *In Figure 6(c), we can see that large increase in the number of jobs submitted to backfill queue only affect the average wait time of jobs in backfill queue. The average wait time of jobs in backfill queue increases significantly, but there is no evidence showing that other queues were affected by the surge.*

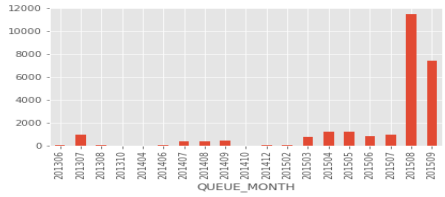
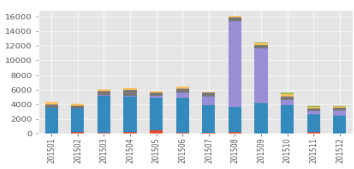
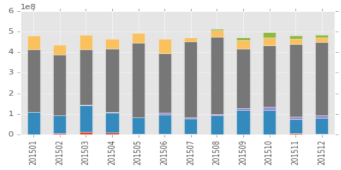


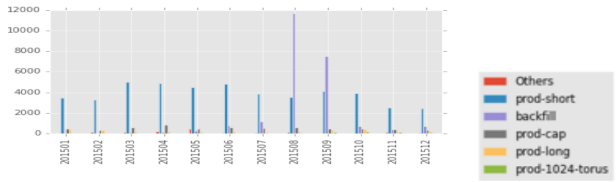
Fig. 5. Number of jobs submitted by user 15303315089691 in each month.



(a) Number of jobs submitted in each queue in each month



(b) Core hours used in each queue in each month



(c) Average wait time of jobs in each queue in each month

Fig. 6. The correlation between runtime, used nodes and queues.

5.2.2 User 32764951387776

Observation 7 *In Figure 7, user 32764951387776 submitted 48 capability jobs in April 2016. As shown in Figure 8 and 9, the large submission by this user leads to less total number of jobs submitted in April 2016, but the total core hours used in that month increases. Therefore, large jobs can boost system utilization.*

Observation 8 *In Figure 10, increasing number of jobs queued in prod-capability increases the average wait time of jobs in prod-capability, backfill, and prod-long queues. The increases in the average wait time of jobs in prod-long and backfill are more obvious. Therefore, job wait time of jobs in prod-long and backfill is prone to be affected by the number of job submissions in prod-capability queue.*

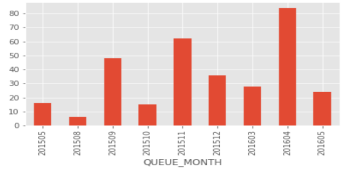
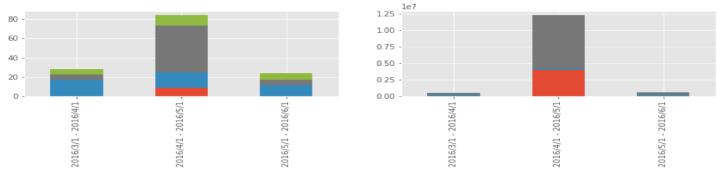
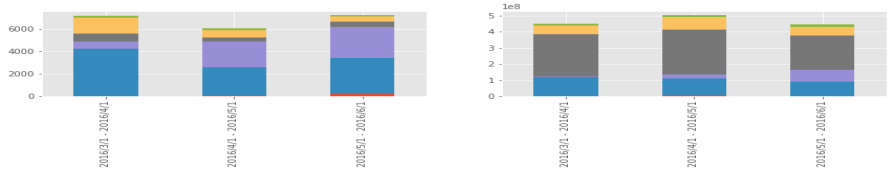


Fig. 7. Number of jobs submitted by User 32764951387776 in each month.



(a) Number of jobs submitted (b) Core hours used

Fig. 8. Number of jobs submitted and core hours used by User 32764951387776 in each month during March 1. 2016 to June 1. 2016.



(a) Number of jobs submitted (b) Core hours used

Fig. 9. Number of jobs submitted and core hours used by all users in each month during March 1. 2016 to June 1. 2016.

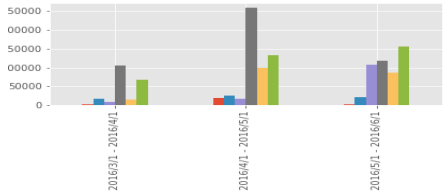
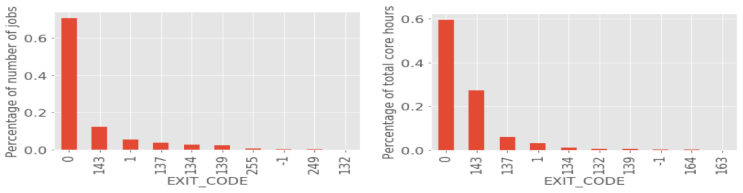


Fig. 10. Average job wait time of jobs in different queues in each month during March 1. 2016 to June 1. 2016.

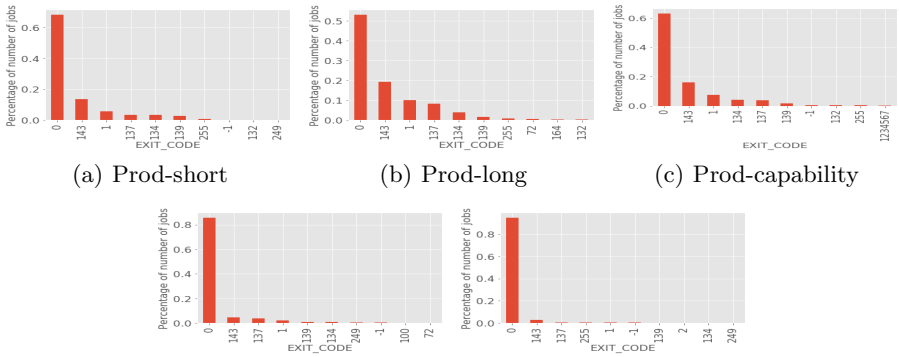
5.3 Exit Codes

The exit codes in Mira logs follow the standard exit codes in Linux. For example, 0 means a job exits Mira normally. 1 means general errors caused by users. In this subsection, we analyze the distribution of exit codes. Figure 11 presents the distribution of exit codes.



(a) Percentage of jobs (b) Percentage of used core hours

Fig. 11. Distribution of exit codes.



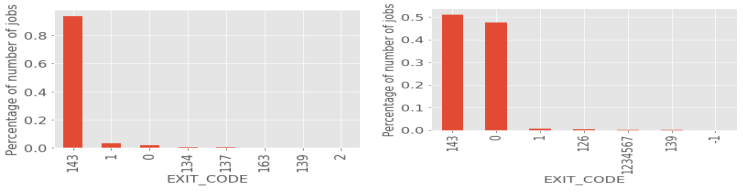
(a) Prod-short (b) Prod-long (c) Prod-capability

(d) Backfill (e) Prod-torus

Fig. 12. Distribution of exit codes based on queues.

Observation 9 *In Figure 12, Jobs in prod-short, backfill, and prod-torus queues have the higher probability of exiting system normally.*

Observation 10 *In Figure 13, users have different distributions of exit codes. For example, 90% of jobs of user 8445397395848 exit system normally, whereas more than half of jobs of user 41675626785343 exit system with code 143.*

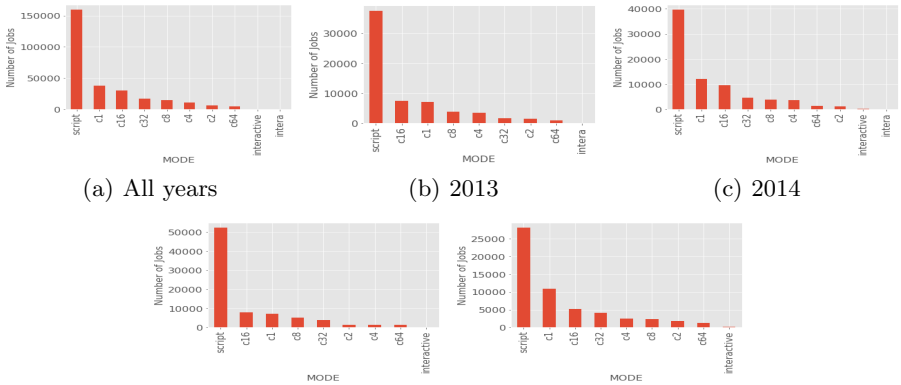


(a) User 8445397395848 (b) User 41675626785343

Fig. 13. Distribution of exit codes based on users.

5.4 Modes

Mira support three types of job submission: basic job submission, script submission, and interactive job submission. Basic job submission allows users to submit an executable. Basic job submissions are further divided into several modes (i.e. c1, c2, c4, c8, c32, and c64) based on the number of ranks per nodes. For example, c2 means 2 ranks per node. Script submission enables users submit a single Cobalt job script and conduct multiple runs within a script, if jobs in a script all require the same size partition. The script mode is more flexible and a user need to wait only once to run all jobs in a script. Interactive submission allocates partitions and gives a user a shell prompt when the interactive session start. Users can submit jobs and debug jobs in interactive mode.



(a) All years (b) 2013 (c) 2014

(d) 2015 (e) 2016

Fig. 14. Mode Distribution

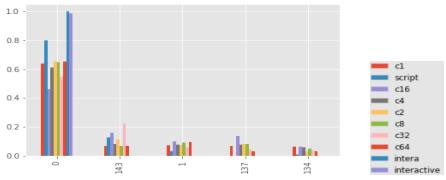


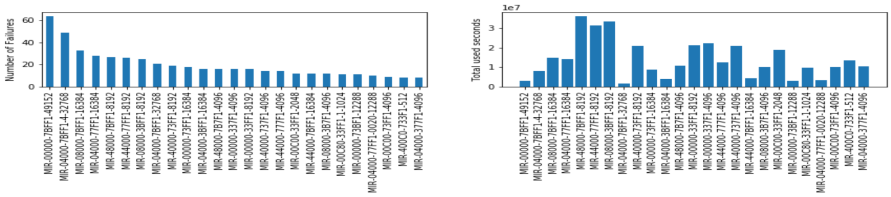
Fig. 15. Percentage of jobs in Combination of Exit codes and Modes.

Observation 11 *In Figure 14, script is the most popular mode in all years. In addition, there is a trend towards using c1 mode. A node in Mira can run at most 64 MPI ranks. The probable reason for not fully use 64 ranks is some resources in a node are not sufficient to run a job. For example, memory may not be sufficient to share between 64 MPI ranks, hence users may request to use less ranks in one node.*

Observation 12 *In Figure 15, jobs in script modes have higher probability of exiting system normally. 80% of jobs in script mode exit system normally, whereas less than 50% of jobs in c16 mode exit system normally. 22% of jobs in c32 mode exit system with code 143.*

5.5 Co-analysis on RAS and Job Logs

Reliability, Availability, and Serviceability (RAS) logs are the primary source of information that a system administrator can use to understand failures [21]. Co-analysis on RAS and job logs can reveal the job related failures.



(a) Number of Failures of Top 25 partitions

(b) Number of seconds used on different partitions

Fig. 16. Failures and jobs on partitions.

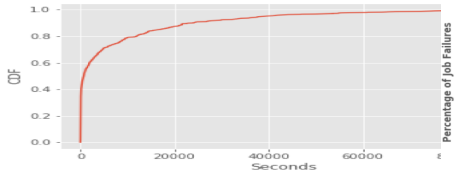


Fig. 17. CDF of Failure happened after jobs started.

Fig. 18. Percentage of jobs having fatal failures in different queues.

Observation 13 *From Figure 16, we can see bigger jobs and partitions have the higher failure probability. Jobs using all the nodes have the highest failure counts. In addition, the total failure counts (Figure 16(a)) is related to total used node hours of the partitions (Figure 16(b)).*

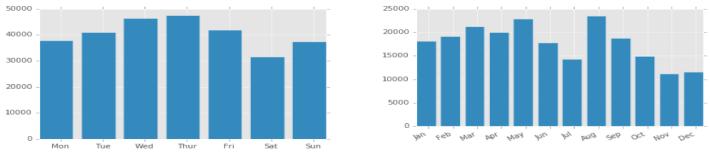
Observation 14 *From Figure 17, we can see that more than half of the failures happened at the very beginning of job execution.*

Observation 15 *From Figure 18, jobs in prod-capability have the highest probability of failures. This is probably because of jobs in this queue are bigger than jobs in other queues, hence, they have higher probability of failure.*

5.6 Cycles

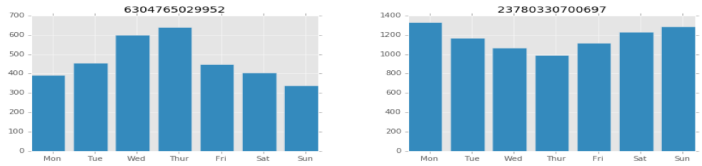
In this subsection, we focus on analyzing user behavior and their submission cycles.

Observation 16 *From Figure 19 and 20, more jobs were submitted on Wednesday and Thursday. More jobs were submitted in May and August. Users have their own submission pattern. Some users are more active on weekdays, while others are more active on weekends.*



(a) Week cycle (b) Month Cycle

Fig. 19. Job submission cycles



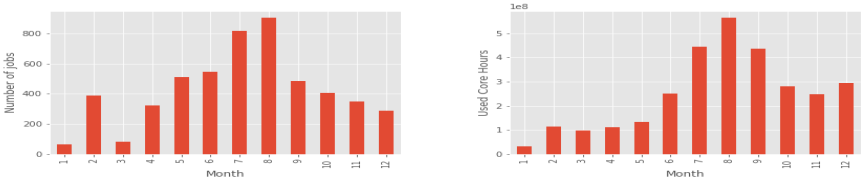
(a) User 6304765029952 (b) User 23780330700697

Fig. 20. Users Week Cycle

5.7 Account

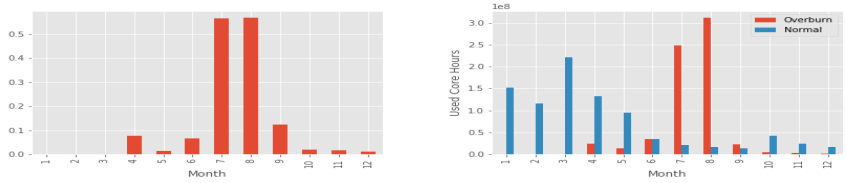
In this subsection, we analyze INCITE and ALCC accounts and the overburn effects of these accounts.

Observation 17 *Overburn jobs consists of 13% of the total jobs. Approximately 20% of core hours are overburn. In Figure 21, overburn is highly utilized from July to September.*



(a) Percentage of overburn jobs. (b) Core hours used on overburn jobs.

Fig. 21. Distribution of overburn jobs per month.



(a) Percentage of overburn jobs. (b) Core hours used on overburn jobs.

Fig. 22. Distribution of overburn jobs per month of project 902672531503.

Observation 18 *In Figure 22, project 902672531503 is the largest project in Mira. The analysis on overburn shows that in July and August, approximate 57% of jobs of this project were overburn, which makes up 80% of total core hours used by this project.*

6 Upcoming Challenges at Argonne

For many years DOE facilities have been on the bleeding edge of parallel computing and managing large volumes of data. For somewhere on the order of 30 years, the basic workload has been massively parallel jobs, batch scheduled, with relatively long run times and few jobs.

While that workload will continue to be a major part of the mission for the foreseeable future, the science teams and DOE are asking the DOE facilities to support a broader range of workloads which are going to raise challenges across the board including scheduling. Here is a brief description of the workloads and the potential scheduling challenges associated with them.

6.1 Multi-scheduling

Changes in architecture are requiring multiple resources to be considered "first class citizens" from a scheduling perspective. One that is already appearing is power management [22][23]. Power consumption of the largest machines may exceed what the data center has available and the scheduler will need to consider job power consumption to avoid exceeding power limits. Some architectures can also gain performance improvements if the power is managed on the node. Another that is already here is the "burst buffer" [26]. This is a very fast, generally very close, storage cache. In the future we will need to also consider space available in the burst buffer. Similarly, Argonne has a research project to investigate making RAM a manageable, allocable network resource, which would also need to be scheduled. Network bandwidth and storage space are also possibilities that have already been explored.

6.2 On-demand or Deadline Sensitive Computing

Many of the experimental "instruments" such as light sources like the Argonne Advanced Photon Source[16], or fusion tokamaks like the current DIII-D[17] or the future ITER[18] require, or at the very least could greatly benefit by having, computational capabilities that could be available on demand (as soon as a data acquisition is complete) that can complete in a short period of time so that the results can be used to adjust parameters for the next run. This is almost in direct conflict with the traditional workload where a single job can consume the entire machine for long (12-24 hours) periods of time. How can both demands be accommodated? Here are some potential solutions:

- A fraction of the machine is reserved during periods when the instruments are running, assuming they are fairly regular and scheduled. This hurts utilization since the reserved nodes sit idle while the run is in progress.
- Pre-emption: With the right job mix and the right machine characteristics this is a potentially very good solution. However, some machines, the Blue Gene in particular, are not very amenable to this. The Blue Gene reboots the nodes between every job, and for a full machine job that could take 5 minutes. It also takes time to kill a job and do appropriate cleanup. If the total allowable time is 15 minutes, a number often suggested for the fusion community, the overhead to do pre-emption is approaching 50%. There is

also the loss of science from the last check point till the job was killed. This can easily be several hours and that is a non-trivial loss on a very important and expensive resource.
- What if we could design a hardware / system software system that enabled rapid, efficient, and low cost "task switches", resembling more what an operating system does on a multi-core processor? This would enable us to then simply "suspend" the job running, task switch to the time constrained job, possibly backfilling if the size of the suspended job is much bigger than the needs of the time-constrained job. How would you schedule this? How would you do the accounting for it?

6.3 High Throughput Computing (HTC) workloads

HTC workloads tend to be the polar opposite of big parallel jobs. They tend to require little or no communication between them. They tend to be small, often single core, or single node. They tend to be short, a few minutes, or even a few seconds. There tend to be many of them, potentially tens of thousands or even millions. Typical batch scheduler may not scale well for that number of jobs, and the startup and shutdown of some large supercomputers would fail under that sort of load. For instance, the Blue Gene records every hardware event that occurs in a DB2 relational database. The kind of job thrashing that an HTC workload brings could run the database out of resources and crash the machine. We had it happen early on until we built throttles in to prevent it, but that means the HTC workloads cannot practically run. From a pure architecture point of view, HTC jobs dont fit on a big parallel machine. They have no need of the very expensive, generally proprietary interconnect, and they are a perfect fit for distributed computing, which is what the High Energy Physics Community has been doing for years. So why the push for HTC jobs on supercomputers? They represent a very sizeable investment. They can potentially provide better time to solution.

- Deal with queue depths of millions
- Possibly batch job starter (ala Condor)

6.4 Complex Domain Specific Software Stacks

Typically, the environment on a batch compute cluster is a "take it or leave it" affair. There is a single base OS and a set of supported packages. Your application must either be able to run in that environment or you must be able to build any required ancillary packages yourself in that environment. At the very least, this can require significant effort on the part of the application scientists / developers to build and support multiple environments. In the worst case, they simply cannot make use of the resource because their software stack cant run in the environment. Perhaps there are libraries they dont control that cant build, or there are vetted algorithms that cannot be altered or wont run. Perhaps you could build your software, but it requires root access, something most facilities are not willing to provide.

Virtual machines and particularly containerization are offering a paradigm that can potentially support these complex software stacks. The applications sci-

entist can build complete end-to-end images of their software stack and execute those on the system. There are projects like Shifter [19] and Singularity [20] that are attempting to enable root access inside the container while ameliorating the security concerns that come along with that for the facility.

6.5 Coordinated Services and Access to Remote Data

Another paradigm that is becoming more common are computational tasks that either require the compute nodes to have internet access and/or they require another application / service to be up and running while the computational job is running. Many facilities, ours included, put the compute nodes on an internal non-routable network. This is primarily done as a security measure, but it can also have a cost factor. If there are other services required to run, there are several options:

- Provide resources for "always on" long running services; But this brings several issues: who installs and maintains it? If the user, what about root access? Who deals with the issues if it goes down? What about differing hardware and software requirements? For instance, if this service is a local database it might require very substantial core, RAM, and I/O resources.
- Provide resources that can be co-scheduled and used "on-demand" and spin the services up and down with the job. A virtual machine facility either internal or something like Amazon Web Services could be used for this, but those can take several minutes to spin up. How do you avoid your computational resource from sitting idle while that happens?

6.6 Workflows

The work required to achieve scientific output is much more than just running the computational jobs. Much of that work is repetitive and mundane, particularly when the science involves multiple facilities, which is becoming the norm rather than the exception. You have to sequence the jobs in the right order to satisfy data dependencies (the output of one job is the input to another), you may have to move data from one facility to another, archive / backup results, etc. Many science projects are moving towards using automated workflow tools to manage this complexity. This brings at least two issues to the facility:

- Remote job launch: You may no longer have a user sitting on your login node submitting jobs, so how do you enable your scheduler to support this and do so in a way that avoids the application developers from having to support a different mechanism at every facility?
- Security: Not only is the user not logged into the system, a user can't be sitting by ready to type their password when the workflow system decides to submit a batch of jobs at 3AM on Sunday morning. How do we balance supporting the science workflows with securing our facilities so we don't end up on the front page of the New York Times because our supercomputer launched a Denial-Of-Service attack against the White House?
- Impacts of having the WAN as a critical component: If the data is distributed and needs to be moved, as opposed to moving the compute to the data,

scheduling becomes much more difficult because WAN transfer times are notoriously difficult to predict, and if the data movement is happening while the job is running there is a high probability that the compute resources will go idle while waiting for data due to WAN performance variability.

7 Conclusions

In summary, we have described the specific batch scheduling goals and constraints at ALCF. We have also analyzed workload traces collected from the production petascale supercomputer Mira and made eighteen key observations at various perspectives. Finally, we have discussed the upcoming scheduling challenges at Argonne and potential solutions to some of the challenges.

Acknowledgement

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. Zhiling Lan is supported in part by US National Science Foundation grants CNS-1320125 and CCF-1422009.

References

1. Argonne National Laboratory, http://www.anl.gov/
2. Argonne National Laboratory User Facilities, http://www.anl.gov/user-facilities
3. Argonne Leadership Computing Facility, http://www.alcf.anl.gov/
4. Top500, https://www.top500.org/
5. Oak Ridge Leadership Computing Facility, https://www.olcf.ornl.gov/
6. Innovative and Novel Computational Impact on Theory and Experiment (INCITE) Program, http://www.doeleadershipcomputing.org/incite-program/
7. Advanced Scientific Computing Research (ASCR) Leadership Computing Challenge (ALCC) Program, https://science.energy.gov/ascr/facilities/accessing-ascr-facilities/alcc/
8. The Directors Discretionary (DD) program, https://www.alcf.anl.gov/dd-program
9. IBM Blue Gene, https://en.wikipedia.org/wiki/Blue_Gene
10. Plan9, https://en.wikipedia.org/wiki/Plan_9_from_Bell_Labs
11. ZeptoOS, http://www.mcs.anl.gov/research/projects/zeptoos/
12. http://trac.mcs.anl.gov/projects/cobalt/
13. SciDAC Scalable Systems Software ISIC, http://www.scidac.gov/ASCR/ASCR_SSS.html
14. Intrepid, https://www.alcf.anl.gov/intrepid
15. Mira, https://www.alcf.anl.gov/mira
16. Argonne Advanced Photon Source, https://www1.aps.anl.gov/
17. DIII-D, https://en.wikipedia.org/wiki/DIII-D_(fusion_reactor)
18. ITER, https://www.iter.org/
19. Shifter, https://github.com/NERSC/shifter
20. Singularity, http://singularity.lbl.gov/
21. Z. Zheng, L. Yu, W. Tang, and Z. Lan, "Co-analysis of RAS Log and Job Log on Blue Gene/P", Proc. of IPDPS, 2011.
22. X. Yang, Z. Zhou, S. Wallace, Z. Lan, W. Tang, S. Coghlan, and M. Papka, "Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems", Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2013.

23. S. Wallace, X. Yang, V. Vishwanath, W. Allcock, S. Coghlan, M. Papka, and Z. Lan, "A Data Driven Scheduling Approach for Power Management on HPC Systems", Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2016.
24. Z. Zhou, X. Yang, Z. Lan, P. Rich, W. Tang, V. Morozov, and N. Desai, "Improving batch scheduling on Blue Gene/Q by relaxing 5D torus network allocation constraints", Proc. of IEEE IPDPS, 2015.
25. Z. Zhou, X. Yang, D. Zhao, P. Rich, W. Tang, J. Wang, and Z. Lan, "I/O–Aware Batch Scheduling for Petascale Computing Systems", Proc. of IEEE Cluster, 2015.
26. J. Yan, X. Yang, D. Jin and Z. Lan, "Cerberus: A Three–Phase Burst-Buffer-Aware Batch Scheduler for High Performance Computing", Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), poster session, 2016.