Multi-core job submission and grid resource scheduling for ATLAS AthenaMP

D. Crooks[3], P. Calafiura[2], R. Harrington[3], M. Jha[4], T. Maeno[5], S. Purdie[3], H. Severini[6], S. Skipsey[1], V. Tsulaia[2], R. Walker[7], A. Washbrook[3] on behalf of the ATLAS collaboration

[1] Kelvin Building, Physics and Astronomy, University of Glasgow, Glasgow, G12 8QQ
[2] Lawrence Berkeley National Lab, 1 Cyclotron Road, Berkeley, CA 94720, USA
[3] School of Physics and Astronomy, The University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, Edinburgh, EH9 3JZ, United Kingdom
[4] INFN CNAF and INFN Bologna, V.Le Berti Pichat 6/2, IT-40127 Bologna, Italy
[5] Brookhaven National Laboratory, NY, USA
[6] University of Oklahoma, 660 Parrington Oval, Norman, OK 73019, USA
[7] Ludwig-Maximilians-Universitat Muchen, Munchen, Germany

E-mail: `awashbro@ph.ed.ac.uk`

Abstract. AthenaMP is the multi-core implementation of the ATLAS software framework and allows the efficient sharing of memory pages between multiple threads of execution. This has now been validated for production and delivers a significant reduction on overall memory footprint with negligible CPU overhead. Before AthenaMP can be routinely run on the LHC Computing Grid, it must be determined how the computing resources available to ATLAS can best exploit the notable improvements delivered by switching to this multi-process model. A study into the effectiveness and scalability of AthenaMP in a production environment will be presented. Best practices for configuring the main LRMS implementations currently used by Tier-2 sites will be identified in the context of multi-core job optimisation.

1. Introduction

ATLAS Monte Carlo simulation, data reprocessing and user analysis jobs are run successfully on computing resources at over 100 computing sites worldwide on a variety of grid infrastructures. As the number of CPU cores on worker nodes at these sites has increased it has been preferable to allocate one job per core in order to maximise resources. Although the number of cores per worker node has increased the ratio of device memory to number of cores per worker node has remained constant. A value of 2 to 3 GB per CPU core is typical and is lower for sites taking advantage of hyper-threading technology to double their effective core count per node.

The memory footprint of ATLAS pileup reconstruction jobs is expected to exceed the 2GB per core due to the higher amount of pileup events expected in current and future data taking conditions. The assumption of running one ATLAS job per core will be difficult to sustain without memory limits on the worker node being reached and less jobs will have to be allocated per worker node unless memory pressure can be mitigated. To address this issue, AthenaMP - the multiprocess implementation of ATLAS Athena software framework - provides a method of enabling maximum memory sharing between multiple Athena worker processes. Almost 80% memory sharing can be achieved with negligible CPU overhead [1].

Although the advantages of using AthenaMP are clear, effort is now required to define the best approach to reconfigure local resource management systems and scheduler software at grid sites providing resources to ATLAS to run multicore jobs in a timely manner. In particular, some care is needed to avoid scheduling contention for jobs requiring different CPU and memory resources to run.

This note will look at how the increasing need for AthenaMP can be incorporated into the ATLAS production system. The implementation and the main features of AthenaMP will be outlined in section 2. In section 3 some of the issues faced for sites accepting multicore jobs from ATLAS will be discussed. Section 4 will cover some of the possible multicore job scheduling scenarios in further detail by modelling scheduling behaviour in a controlled environment. The current status of multicore readiness in the production system will be described in Section 5. Recommendations for wider deployment across grid infrastructures and future prospects in this area will be outlined in section 6.

2. AthenaMP

AthenaMP [2] is an extension of the Athena software framework which provides a method of maximum memory sharing between multiple Athena worker processes whilst retaining event based parallelism. The underlying process creation and communication management to worker processes is provided by the Python multiprocessing module [3]. By incorporating all multi-process semantics into the existing Athena/Gaudi framework [4] there is no need for changes to client code using AthenaMP.

2.1. Implementation and workflow

The multi-process mode of Athena is activated by either a command line switch or an environment variable indicating the amount of cores to use (a setting of -1 utilises all the cores on the host). A schematic of the job workflow for Athena and AthenaMP is shown in Figure 1. The allocated number of worker processes is created prior to the main event loop. The `fork()` routine then clones and shares the address space of the parent process with the worker processes and the Linux kernel Copy On Write mechanism (CoW) ensures an efficient use of memory with only the differences in memory between the worker and the master processes using up additional memory space.

A bootstrap function handles I/O reinitialisation to allow each process to run in a separate working directory with no communication required between processes. Input events are then allocated via a shared queue and the master process remains idle until all events are processed. Output files generated by worker processes are then merged before the finalisation step is run. More details on the design of AthenaMP are provided in [2].

2.2. Running considerations

It is important to note that the performance does not scale linearly with N processes and it is evident from Amdahl's law [5] that any serialisation steps during execution will significantly affect scaling ability. For AthenaMP, the main areas of serialisation are in the job initialisation and file merging steps. Therefore undue latency or inefficiencies in these areas needs to be minimised whilst retaining significant memory sharing ability.

The timing of the worker process `fork()` is crucial in order to enable the maximum amount of memory to be shared. If the `fork()` call is made before data common to worker processes (such as detector conditions data) is allocated into memory then an unnecessary duplication of memory pages will result. Conversely, a late `fork()` call will result in a larger proportion of the overall execution time to be run in serial mode. The optimal approach is found by processing

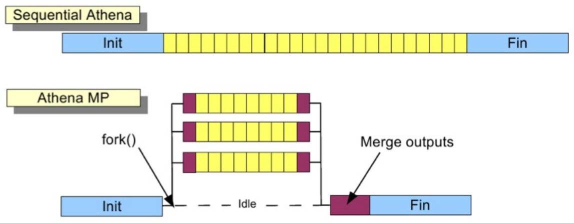

Figure 1. Job flow for serial and multi-process implementations of the ATLAS Athena framework

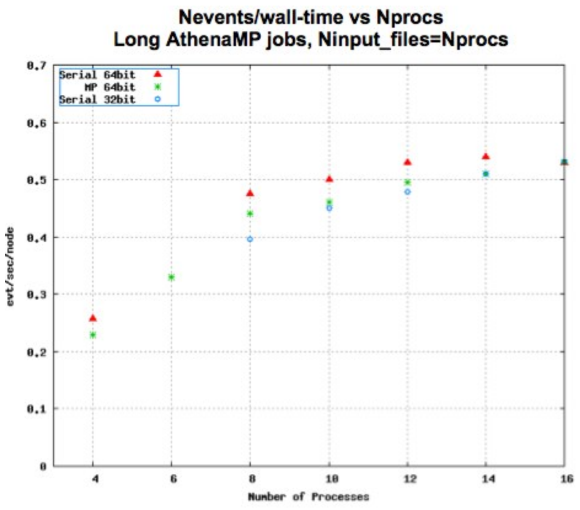

Figure 2. Job event throughput for an increasing number of worker processes. The number of input files is equal to the number of processes.

the first input event before the `fork()` is called. A large amount of memory sharing is then enabled with only a small serialisation penalty incurred.

In order to increase the proportion of the job in parallel mode it is desirable to increase the length of the event loop. A common approach is to match the number of input files with the number of processes. Figure 2 shows the event throughput (the number of events processed per second per node) of a typical reconstruction job for a increasing number of worker processes. As the number of files (and processes) increases the overall event throughput is comparable to running N jobs in serial mode.

Although a longer event loop (and a higher number of input files) is preferable this will result in an increase in processing time in the finalisation stage due to file merging. This effect can be mitigated by choosing a faster merge algorithm which concatenates event data and metadata files rather than full event validation.

In addition to the running modes described above there are a number of AthenaMP job options that can affect execution time. Instead of an event queue it is possible to define a fixed allocation of events for each worker process. However, due to large deviations in event processing times causing an imbalance in worker process lifetimes this is now generally discarded in favour of the event queue model. Worker processes can also be pinned to a predefined list of cores rather than be allocated dynamically by the Linux scheduler. This is not well used at present but could be potentially useful to mitigate undesirable NUMA effects.

3. AthenaMP on the ATLAS PanDA system

A recent series of simulation exercises at the ATLAS Tier-0 computing centre has validated AthenaMP for production use. The next step in deployment is to ensure that AthenaMP can run successfully and in a timely manner on the existing ATLAS computing infrastructure.

3.1. ATLAS PanDA System

A key component of the ATLAS distributed computing operations is the ATLAS Production and Distributed Analysis system (PanDA) which provides robust workload management for Monte

Carlo simulation, data reprocessing and user analysis. A brokerage module in PanDA prioritises and assigns tasks to site queues based on a number of factors such as CPU availability and input data locality. The level of workload received by each grid site is managed by a pilot system [6]. One or more pilot factories installed in each regional cloud send jobs directly to grid computing sites using Condor-G [7]. Once pilots are executed any available jobs from the PanDA server brokered to the site are retrieved and the job payload is executed and monitored by the pilot. The amount of pilot jobs sent to a site queue is determined by the number of current queued and running pilots on the site queue. This provides an efficient self-regulating method of utilising available ATLAS computing resources. More information on the PanDA system can be found in [8].

3.2. Multicore site configuration

The use of AthenaMP will be incorporated into production operations as the memory footprint of Athena reconstruction jobs increases. It will be necessary to continue running single core jobs with an gradual introduction of AthenaMP tasks added to the system. For sites wishing to pledge multicore resources to PanDA system there are two main issues to address:

(i) Should multicore jobs reserve all the cores available on a worker node (i.e. wholenode execution)?

(ii) Should a dedicated set of resources be provided to new multicore queues?

For (i), wholenode execution allows runtime inspection of the worker node hardware to define the number of cores for execution rather than relying on information from Panda configuration or external information systems. The reservation of a entire worker node also guarantees that memory and CPU resources will be dedicated to the AthenaMP job and not shared with other jobs that may impact on execution. Furthermore, a wholenode job can still retain the option to not use all cores if memory limitations are observed.

Although this appears to the simplest approach there are still some issues to address. The number of cores is an important factor in determining optimum job length and so core count may need to be included as part of any AthenaMP-based task definition. In addition, an increasing amount of cores per node (with 64 core worker nodes already in production) reduces job scheduling flexibility for the grid site.

For (ii), sites could partition a small but dedicated amount of resources for exclusive multicore use. This modification requires minimal configuration changes to the existing PanDA model and pilot jobs sent to these multicore queues can be scheduled and run in exactly the same manner as for single core jobs.

An important issue arises when determining how many resources will need be reserved for participating sites. It is clear that too conservative an estimate will result in a large waiting time but arguably a larger problem occurs when the allocated resources are too generous. In low usage periods, worker nodes attributed to a dedicated queue will be left unoccupied and cannot be reassigned for single core jobs, or for any other users without continual intervention by site administrators. This will impact on overall site throughput and has led candidate sites to be initially conservative with multicore resource allocation.

3.3. Dynamic Resource Allocation

An alternative option to dedicated multicore resource allocation is to allow a site batch system scheduler to dynamically assign resources by taking batch system load into account. This can provide an efficient automated mechanism to enable the regulation of multicore use without loss of overall job throughput. Although this is a more flexible approach than static allocation of multicore resources there is the potential for scheduling contention which can cause undue latency not evident in current production system.

The scheduling of jobs with different CPU and memory resource requirements is a well defined problem which leading scheduler implementations already address successfully. In particular, this scenario is well covered for efficient simultaneous execution of MPI jobs across multiple cluster nodes. However additional factors have to be taken into account when considering multicore job submission from the ATLAS production system:

- The job submission rate is dependent on batch system load
- The job lifetime depends upon external brokerage which in turn is decided in part by batch system load
- Grid Job queues are not (in general) exclusive to ATLAS

The scheduling conditions anticipated from this new setup were modelled in a controlled environment taking the above factors into consideration.

4. Multicore Scheduling Simulation

A testbed was used to model the submission of single core and multicore jobs of a site accepting workload from the ATLAS PanDA system. The components of the testbed and workflow are shown in Figure 3.

4.1. Testbed Configuration

The batch system and scheduler chosen for this simulation was Torque and Maui which is a common resource manager and scheduling choice for many grid sites available to ATLAS. A feature useful in Torque was the ability for a single node to host multiple batch client instances [9]. This enabled a scaling up of computing resources from the original testbed size to any appropriate test setting. Furthermore, the number of slots per node could be set to any value. A testbed of 100 8-core “virtual” nodes was chosen to capture scheduling conditions expected at a small Tier-2 site.

Although tools exist to evaluate scheduler response without the need for job submission [10] it was necessary to include an approximation of job brokerage and pilot factory submission rates to model realistic ATLAS PanDA submission patterns. Job submission and timing, pilot activation, job brokerage and queue performance monitoring were controlled by a suite of steering scripts created for this study.

Jobs submitted to the testbed batch queue were stored on a job list that was populated from input files at the start of the test and then from ”pilot factories” throughout the test lifetime. These approximations of pilot factories used the same submission algorithm currently used in production. To emulate job brokerage it was not necessary to model the entire PanDA brokerage system. Instead, a tally of the number of jobs available for processing was stored. If this tally was non-zero then any pilots queued in the job list were switched from “idle” to “active” by adjusting the job length to a nominal running time representative of job running in production. The available job tally could be either refreshed at regular time intervals or boosted at discrete points during the test run.

Scheduling patterns only become apparent after a number of hours into the test so it was preferable to speed up the simulation by a global scaling factor. A $10x$ speedup was used for all tests.

4.2. Observations

A number of simulations were run to identify common scheduling scenarios that could be observed at sites that accept multicore pilot jobs running on the same resources as single core jobs. In each test, a number of job and queue-based metrics were collected to evaluate relative performance. Job utilisation, queue utilisation, average job wait, average pilot wait and a

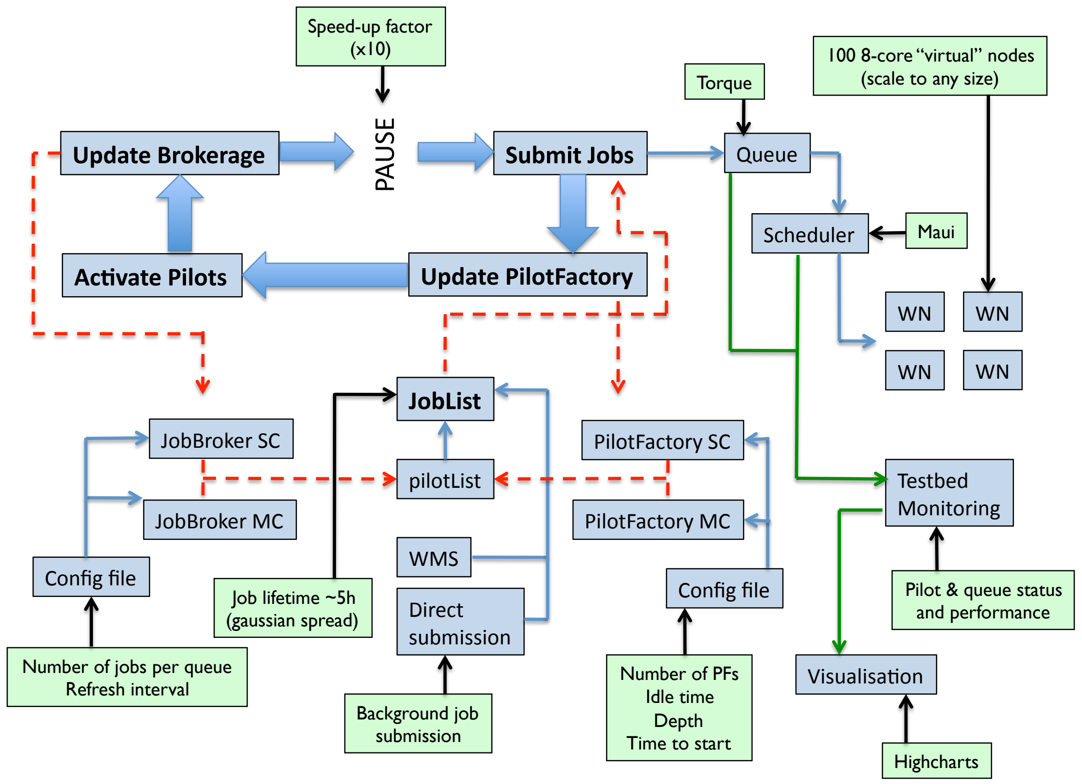

Figure 3. Scheduler Simulation steering script components

derived brokerage value were able to show how the testbed scheduler responded to different job submission scenarios.

The average job wait measured the time jobs spent on the queue before being run on a worker node. Note that this metric in isolation was not enough to fully describe the queue performance due to the variation in pilot jobs submitted by a pilot factory at any given time. To complement this metric the average pilot wait was used to measure the time interval for *any* new pilot job to start running. Both wait times were calculated as averages over a 12 hour period. In addition, wait times for smaller time periods were collected to show short term scheduler response patterns.

The brokerage value was a reflection of the CPU availability algorithm used to determine brokerage decisions by the PanDA Server. A higher value denotes an increase in brokerage weight which will consequently attract more jobs to be sent to a queue. This is evaluated by considering the number of running and idle pilots and is averaged over a 3 hour period.

4.2.1. Scenario 1: Single Core Pilots The first simulation, shown in Figure 4, was a baseline test with one single core pilot factory. For the first two hours of the test the queue was populated with non-pilot jobs ("background jobs") to fill the queue. This was in order to avoid edge effects in the results due to synchronised job start and completion. Pilot jobs were included gradually into the queue as slots were freed from completed background jobs. Once these pilot jobs fully occupy the available slots it is seen that the number of queued jobs is regulated by the pilot factory to a predefined depth. As the number of queued job falls below a set level it is then replenished by another batch of jobs which then keeps the queue fully utilised. This submission pattern continues through the remainder of the test.

4.2.2. Scenario 2: Single Core and Multicore Pilots Figure 5 outlines the simulation when a multi core pilot factory is included 7 hours into the test. It is observed that submitted multicore pilots reside on the queue for a number of hours before a whole node is allocated. In this time the overall queue utilisation drops considerably. This is due to multicore jobs blocking resources for lower priority single core jobs whilst a whole node becomes fully available. Although the utilisation recovers after this initial phase the multicore utilisation is far greater than for single core despite equal submission rates and scheduler weighting. In this scenario the multi core pilot submission rate would have to be throttled to balance resource allocation.

4.2.3. Scenario 3: Multicore Idle Pilots In the scenario shown in Figure 6, an overall queue utilisation drop is also observed even if there are no multicore jobs to be run on the queue. This is because a pilot factory will still submit pilot jobs regardless of defined workload. Although these jobs are short lived (approximately 5 minutes) they still have to potential to disrupt normal utilisation patterns.

4.2.4. Scenario 4: Backfilling Figure 7 shows the effect of scheduling configuration tuning away from a simple fifo model. In this case, backfilling has been enabled to allow single core jobs to preempt higher priority multi core jobs where slots are available. This is in contrast to scenario 2 where single core jobs are forced to wait for higher priority jobs to be cleared from the queue. Indeed, this approach allows for a higher utilisation of single core jobs as a result.

Although 100% utilisation is not observed this is still a marked improvement and further tuning using more detailed backfilling methods could yield improved results. In particular, the a priori knowledge of job lifetime can be used directly by the scheduler to determine whether a single core job can run within the timeframe of a node being drained for multicore use. At present, job lifetime is not accurately provided through the pilot mechanism or from grid middleware and so could be a significant factor to enable single core and multi core jobs to run on the same resources.

5. Current Production Status

A number of grid sites have already pledged multicore resources to allow the testing of AthenaMP jobs in the ATLAS PanDA system. A core count parameter for each new queue is used to determine the number of worker processes that can created by an AthenaMP job. At this time it is assumed that the core count value refers to the total number of cores on the worker node.

Multicore queues available to ATLAS reside on grid sites with a variety of hardware, resource managers, scheduler implementations and queue definitions. The resources allocated for AthenaMP operations has been decided by each site and is shown in Table 1.

In most cases a small amount of worker nodes has been partitioned for exclusive use with the anticipation of an future increase given demand. The queues at GLASGOW and ECDF use the same resources as advertised in single core queues. At ECDF, no additional batch queue has been created. Instead, jobs submitted to this queue are prepended with appropriate batch system flags to inform the scheduler that a whole node is required for execution.

6. Discussion

The requirements for successfully scheduling and running multicore jobs at grid sites is not unique to ATLAS and is relevant for other LHC experiments and for other Virtual Organisations (VOs), and common approaches in multicore job specification and resource advertising can be captured by modifying existing middleware frameworks. The LCG Technical Evolution Group

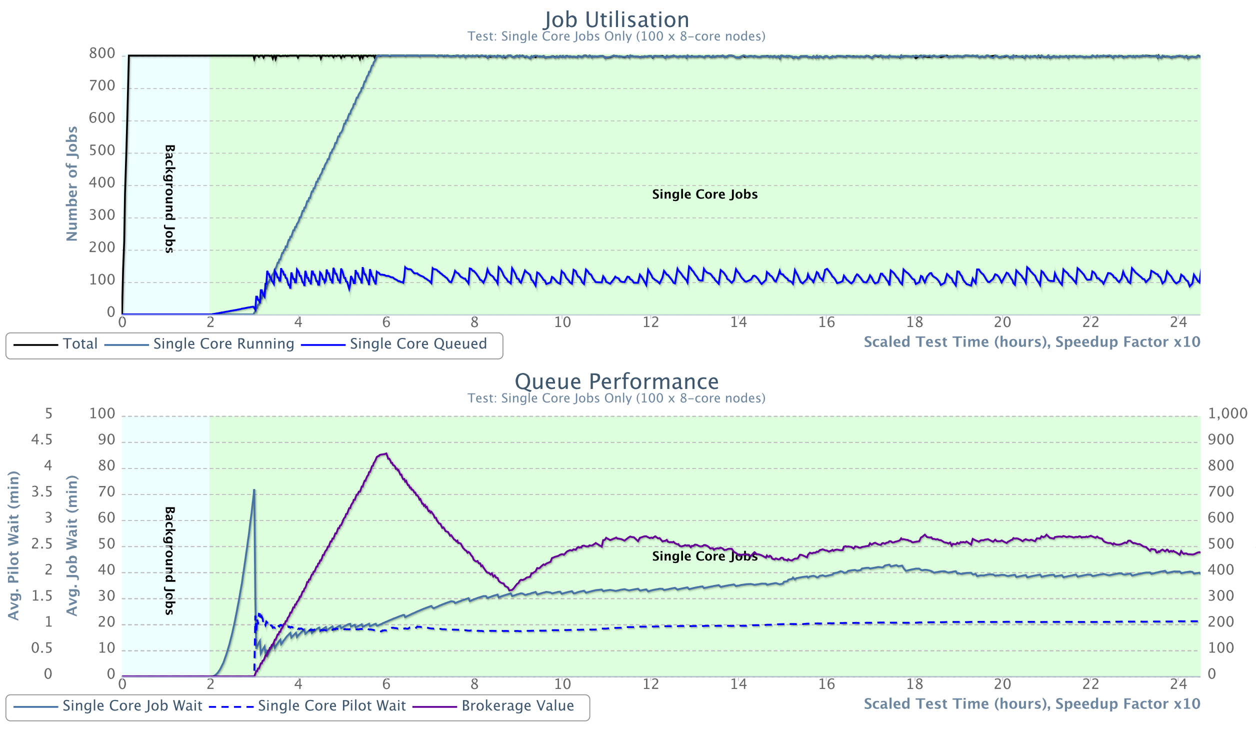

Figure 4. Job utilisation and queue performance for single core pilots.

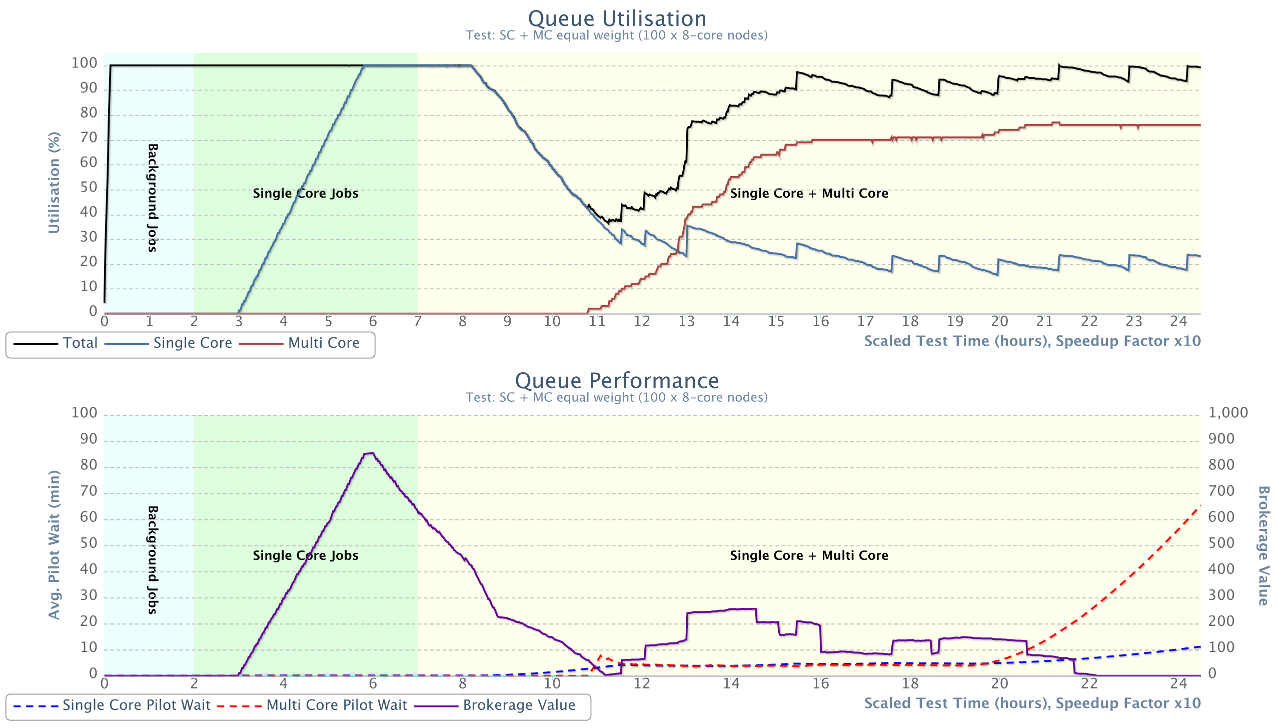

Figure 5. Queue utilisation and queue performance for single core and multi core pilots.

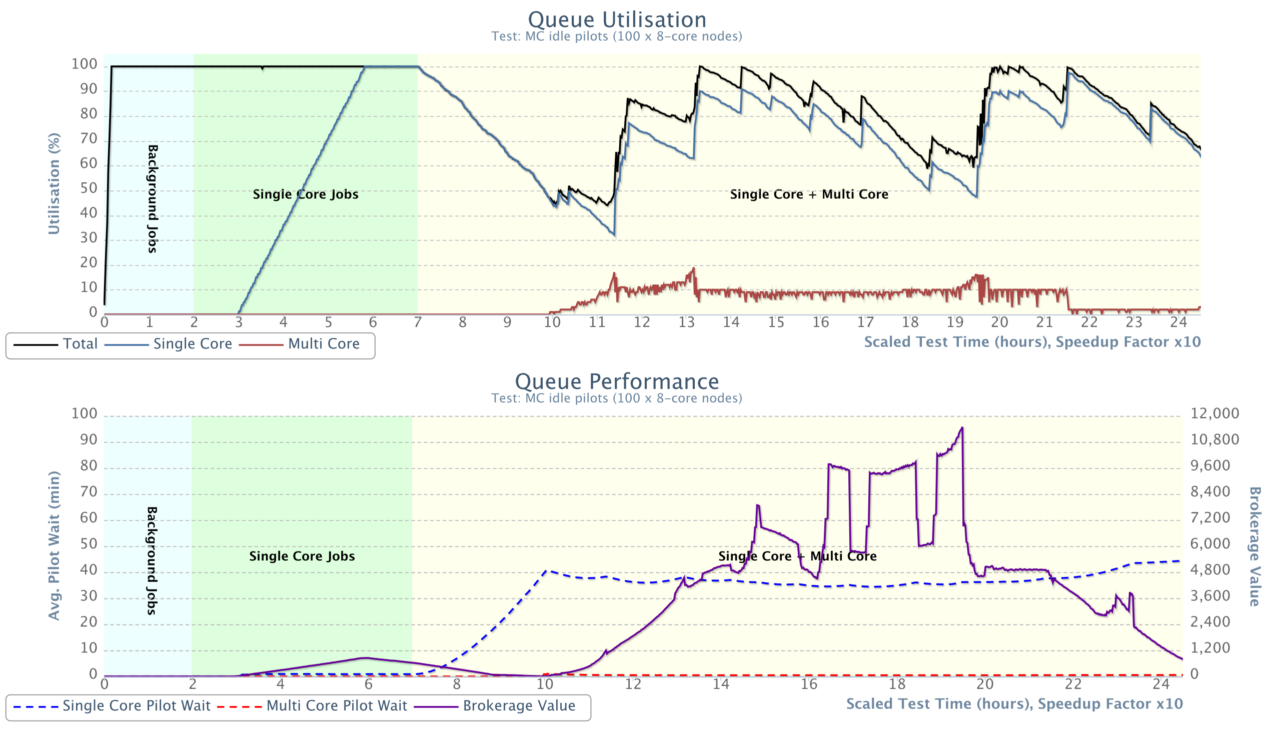

Figure 6. Queue utilisation and queue performance for single core and idle multi core pilots.

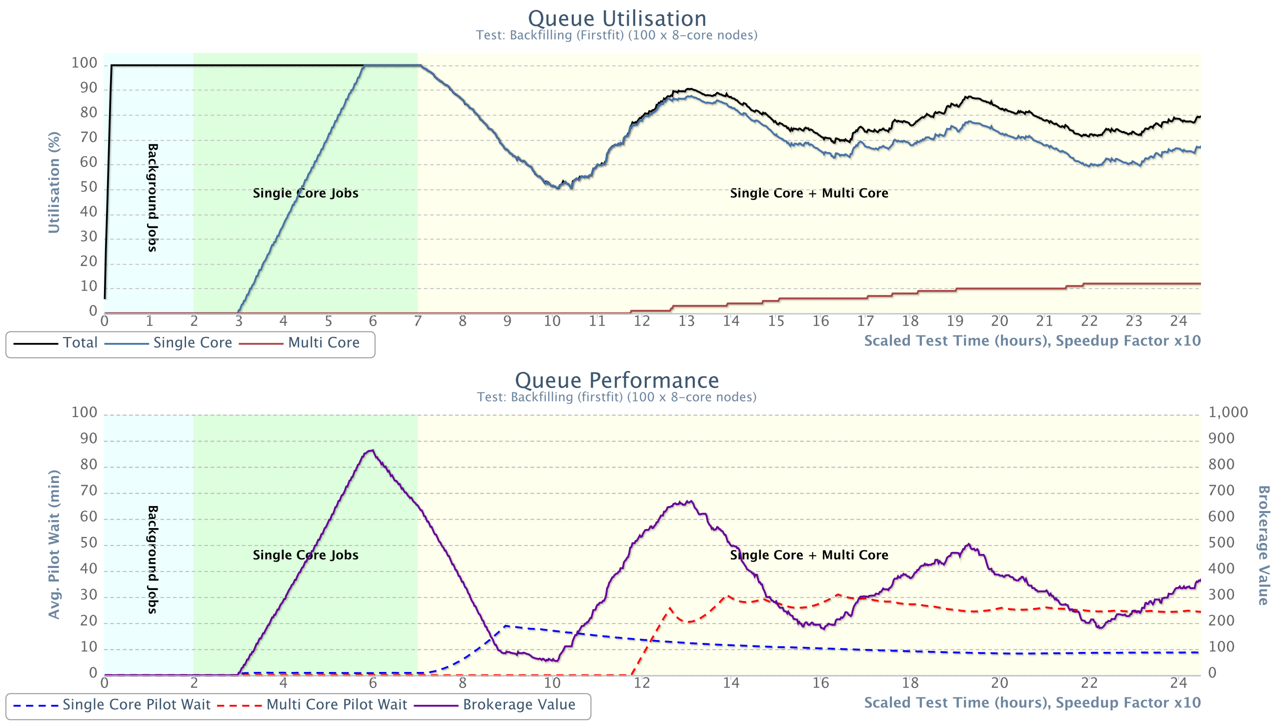

Figure 7. Queue utilisation and queue performance for single core and multi core pilots with scheduler backfilling enabled.

Site Name	**Cores/Node**	**LRMS**	**Grid Excl.**	**Mcore Queue**	**Pledge**
BNL (US)	8/24	Condor	No	Dedicated	50
ECDF (UK)	8/12	SGE	No	Shared	N/A
Glasgow (UK)	8/12/64	Torque/Maui	Yes	Shared	N/A
INFN-T1 (IT)	4	LSF	Yes	Dedicated	8
Lancaster (UK)	8	Torque/Maui	Yes	Dedicated	8
OSCER (US)	8	LSF	No	Shared	N/A
RAL (UK)	8	Torque/Maui	No	Dedicated	15

Table 1. List of Grid sites used for ATLAS multicore testing. Sites that allow access to resources through Grid submission only is shown in the *Grid Excl.* column. The *Mcore Queue* column denotes which sites have partitioned resources for exclusive multicore use.

(TEG) has recommended that additional parameters are available in Job Description Languages (JDLs) in each middleware stack. The number of requested cores, the total memory for the job (or memory per core), wholenode availability, and the minimum and maximum number of cores should be included. Most of these specifications are already available as part of MPI support and could immediately be used for multicore job specification.

An additional recommendation is for sites to be able to advertise multicore queue status through grid information systems such as AGIS. For example, if a site can accept wholenode jobs and the maximum number of cores supported. Information publishing could also be extended to include dynamic information indicating multicore readiness based on load conditions. For example, an availability metric could indicate if load conditions are favourable for multicore job submission. Advanced queue status and performance (as shown in Figures 4 to 7) could potentially be useful especially for central job brokerage.

An approximation of job lifetime is a useful parameter to provide more efficient job scheduling by backfilling single core jobs around potentially longer lived multicore jobs. At present a realistic evaluation of job lifetime is not delivered as part of grid job submission. This functionality has not been required to successfully process jobs which require identical resources to execute but may have to be considered if the rate of multicore job submission increases.

In addition, grid accounting software will have to be validated to ensure that CPU efficiency, CPU time and wallclock time for a multicore job is averaged over N cores. The assumption of one core per job could lead sites to under-report usage which has important implications for the CPU pledges sites make to VOs.

References

[1] ACAT 2011 Multicore in Production: Advantages and Limits of the Multi- Process Approach

[2] Binet S, Calafiura P, Snyder S, Wiedenmann W and Winklmeier F 2010 Harnessing multicores: strategies and implementations in ATLAS J. Phys.: Conf. Ser. 219 042002

[3] The multiprocessing module, http://docs.python.org/library/multiprocessing.html

[4] Mato P 1998 Gaudi - architecture design document Tech. Rep. LHCb-98-064 Geneva

[5] Amdahl G 1967 Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities AFIPS Conference Proceedings (30) pp. 483-485

[6] The ATLAS PanDA Pilot in Operation, P. Nilsson, CHEP 2010, Taiwan, October 2010

[7] The Condor Project. http://www.cs.wisc.edu/condor

[8] Overview of ATLAS PanDA Workload Management, T. Maeno, CHEP 2010, Taiwan, October 2010

[9] Torque Multi-MOM http://www.clusterresources.com/torquedocs21/1.8multimom.shtml

[10] http://www.adaptivecomputing.com/resources/docs/maui/mauiadmin.php