Multicore workload scheduling in JUNO[1]

Xiaomei Zhang[1], *Kang* Li[2], *Xiang Hu* Zhao[1], *Tian* Yan[1], *Yong* Sun[2]
1 Institute of High Energy Physics, 19B Yuquan Road, Beijing 100049, P.R. China
2 Soochow University, 1 Shizi Street, Suzhou, JiangSu Province, 215006, P. R. China
E-mail: zhangxm@ihep.ac.cn

Abstract. The Jiangmen Underground Neutrino Observatory (JUNO) is going to apply parallel computing in its software to accelerate JUNO data processing and fully use capability of multi-core and manycore CPUs. Therefore, it is necessary for the JUNO distributed computing system to explore the way to support single-core and multi-core jobs in a consistent way. To support multi-core jobs, a series of changes to the job descriptions, scheduling, monitoring needs to be considered, in which the pilot-based scheduling for a hybrid of single-core and multi-core jobs is the most complicated part. Two scheduling modes and their efficiency are presented and compared in this paper, and also a way to optimize efficiency is provided.

1. Introduction

JUNO[1] is a multi-purpose neutrino experiment designed to measure the neutrino mass hierarchy and mixing parameters, which will be operational in 2021. The JUNO offline software system is being built on SNiPER[2]. With traditional single-core programming, JUNO data processing meets some limitations on memory and processing time. To improve the performances of JUNO data processing and fully use modern multi-core and manycore hardware, the framework SNiPER is introducing parallelization based on Threading Building Blocks (TBB) [3] to enable multi-thread and multi-process simulation and reconstruction. Event-level parallel processing managed by the TBB task scheduler is already in a prototype phase.

JUNO plans to take 2PB of raw data every year for 10 years. With a large worldwide collaboration, JUNO decided to provision resources for data processing using the distributed computing technology. The IHEP distributed computing system has been built on DIRAC[4][5] to integrate heterogeneous resources from collaborating institutes and commercial resource providers for data processing of IHEP experiments. Support for JUNO started in 2015. The system is currently able to integrate with various resource types including local cluster, grid and cloud. With more and more HPC (High Performance Computing) resources emerging and growing parallelism in the JUNO software, it is necessary for the system to consider full support of multi-core and manycore resources in current infrastructure.

In this paper, the support of multi-core JUNO jobs in current JUNO distributed computing infrastructure will be presented. Section 2 will describe the current workload management system working in the single-core mode, and the design of two new pilot modes to support both single-core and multi-core jobs in a common way. Section 3 will describe the implementation of multi-core support in configuration, scheduling and monitoring. Functionality and efficiency tests are presented in Section 4.

2. Design

In the current infrastructure based on the single-core pilot mode, the JUNO distributed computing is not able to accept and schedule multi-core jobs. In order to enable multi-core support，several issues have to be addressed. Sites itself need to be configured to support multi-core jobs and registered in DIRAC with multi-core information. Job interface has to be expanded to accept multi-core jobs so that user jobs can include multi-core requirements in their requests. The interface to resource in DIRAC also needs to be adjusted to allow sending of multi-core jobs to sites. Information on number of cores needs to be added to job monitoring for resource efficiency study.

Besides that, the most complicated changes are in the Workload Management System (WMS), so in this section we will mainly focus on the new workload scheduling strategy, which was introduced to send multi-core pilots and to deal with matching between multi-core jobs and pilots, as well as a new pilot mode which is needed to pull multi-core jobs. In the foreseeable future, single-core and multi-core JUNO jobs are expected to coexist so it is desirable to support both single-core and multi-core jobs in a consistent way. Also at the sites where resources are shared among different experiments scheduling policy can be adjusted to keep good resource efficiency and reduce side effects by introducing multi-core job support. Therefore, these two requirements are considered in the following design.

2.1. JUNO distributed computing WMS based on DIRAC

The WMS in the JUNO distributed computing framework is based on DIRAC using pilot strategy, currently working in single-core mode, as shown in Fig. 1. Pilots are light agents which can run as normal jobs on worker nodes, get user jobs from the DIRAC job repository and execute user jobs on worker nodes. Matching is divided into three stages. When user jobs arrive in the task queue of the DIRAC server, pilots will be sent out by DIRAC to the sites matching job requirements of user jobs. When pilots enter the queues in the local batch systems, the batch system will schedule the pilots to the proper worker nodes. Then when pilots are successfully running in one of the worker nodes, they will start to ask the DIRAC server for jobs and the server matcher will match and assign the proper user jobs to the pilots. When the jobs finish, the pilots will report the status back to the DIRAC server and start another cycle in its life time.

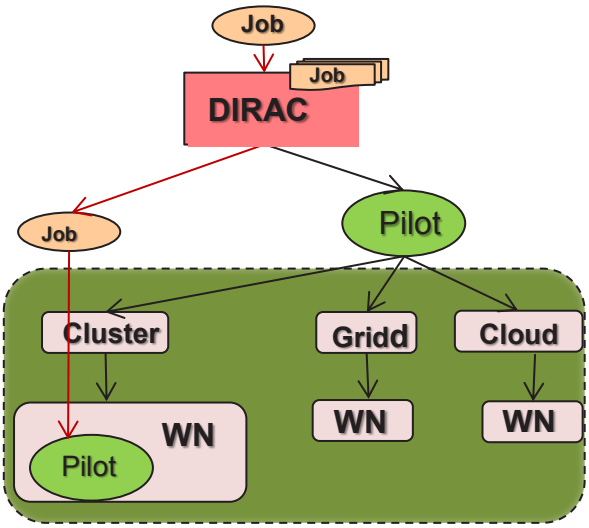

Fig. 1. Pilot scheme in current infrastructure

2.2. Multi-core scheduling strategies

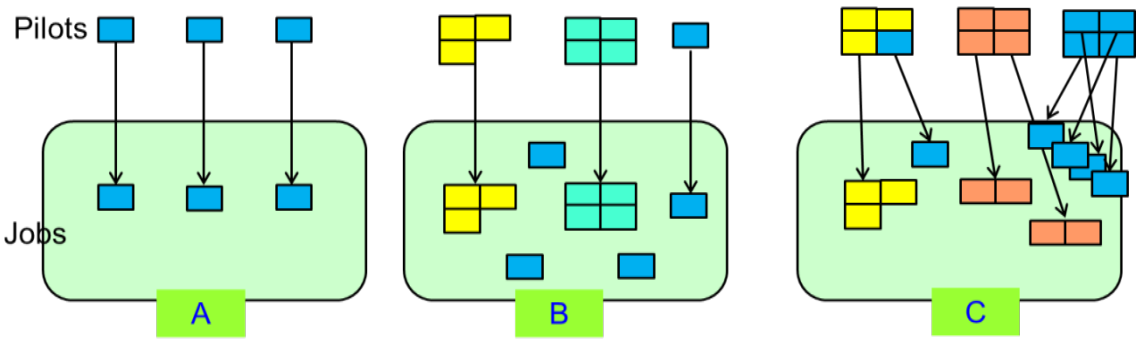

Fig. 2. Design of new pilot models to accept multi-core jobs (Different colours represent jobs or pilots with different cores)

In the single-core mode, each pilot takes one CPU core from worker nodes and pulls one single-core job from task queues to run in this core, as shown in the plot A of Fig. 2. With multi-core jobs added, as shown in plot B and plot C, scheduling strategy should be redesigned to accept both single-core and multi-core jobs. There are two points to consider when designing new scheduling strategies. First, the new pilot mode needs to obtain multi-core resource instead of the single core ones. Secondly, matching needs to deal with multi-core resource and multi-core jobs. In this section, two scheduling strategies which can meet these demands are described. One is based on customized pilots, as shown in the plot B of Fig. 2. The other is based on common and partitionable pilots, as shown in plot C of Fig. 2.

2.2.1. Customized pilots mode

In this mode, the size of pilots varies according to the number of cores requested by the user jobs. That is to say, M-core user jobs in the Task Queue will trigger submitting M-core pilots to sites. One pilot pulls one user job at the time. If M is equal to 1, single-core pilots will be sent out. When the pilots arrive to worker nodes, they will occupy M cores and ask for M-core jobs. Therefore, both single-core and multi-core user jobs can be pulled by corresponding pilots. In case of jobs of just one type, for example only whole-node jobs, the efficiency would remain the same as that with single-core jobs. But with a hybrid of various-core user jobs, efficiency would be lower, as the pilots in this mode can't be shared among jobs with different number of core requirements and many pilots would be wasted when there are still jobs in the Task Queue. The other reason is that when pilots with various number of cores arrive at the site, site batch systems could produce significant resource gaps without applying complicated multi-cores scheduling policies.

2.2.2. Shared partitionable pilot mode

In this mode, the pilots remain the same for jobs with different number of core requested, but can be defined separately in the DIRAC Configuration Service (CS) for each site according to the site resource status. For example, for 24-core worker nodes at sites, the pilot can be chosen to be whole-node, 2-core, 4-core, 8-core, 12-core. Each pilot of the shared partitionable pilot mode can pull more than one jobs. That is to say, M-core pilots can pull several N-core jobs (N<=M) until internal slots of pilots are used up. The pilots in this case need to behave like a mini "cluster", taking care of matching and assigning user jobs to the slots they own. While provisioning common-size pilots, sites only need simple scheduling policy similar to that of single-core. But the pilot of the shared partitionable pilot mode will introduce resource gaps themselves when taking care of internal scheduling and the central matcher will need to apply "smarter" scheduling policy to reduce these resource gaps. In this mode pilots can be shared among jobs with different number of core requested. Therefore with a hybrid of various-core jobs, it is expected to be more efficient than the customized pilot mode, and less efficient than the single-core pilot mode.

3. Implementation

In the DIRAC infrastructure as shown in Fig. 3, Task Queue Database, Matcher, Pilot and Pilot Director are four important components taking care of user job matching and scheduling. When user jobs are registered in the Task Queue, the Matcher will do the matching using site information from the DIRAC CS and job information from the Task Queue, and decide which sites pilots need to be sent out. For grid or local cluster sites, the pilot Director will send out proper pilots to get slots from worker nodes in sites. For cloud sites, instead of submitting pilots proper virtual machines (VM) will be started by the VM director to run pilots inside. When pilots are ready on worker nodes or VMs, they will contact the Matcher to pull proper user jobs and run them in worker nodes. Matching is done among sites, jobs and pilots. This is the implementation of scheduling in the single-core mode. In the multi-core mode, the procedure still remains the same, but every component needs to be adjusted to accept multi-core parameters. Tags are introduced to help the matching among multi-core parameters from sites and user jobs. More details are described in this section.
Besides changes in the central DIRAC server, the sites also need to be configured to accept multi-core jobs. For local cluster or grid sites, a multi-processor queue or whole node queue need to be created to accept multi-core pilot jobs. For cloud sites, multi-core virtual machines should be allowed to be created. For example, OpenNebula[6] need to have multi-core templates ready for creating multi-core VMs.

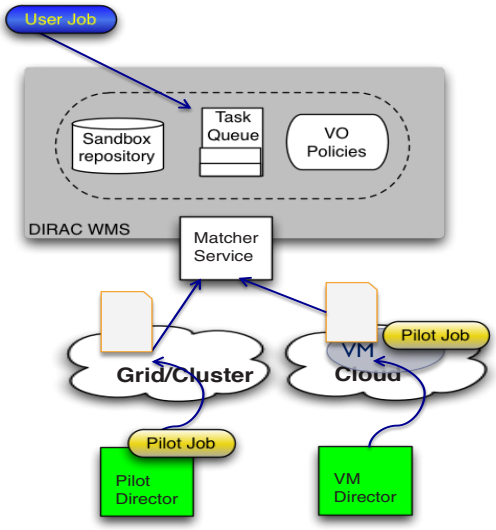

Fig. 3. DIRAC Workload Management System Implementation

3.1 Multi-core Pilot Director and pilots

The Pilot Director has to be changed in order to send out multi-core pilots besides single-core ones. In the customized pilot mode, a new Pilot Director is developed to submit multi-core pilots whose number of cores varies with job requirements registered in the Task Queue. The pilots in this mode are working in the similar way as that of original single-core pilots, but they occupy more than one core and pull jobs with the same number of cores.

In the shared partitionable pilot mode, the Pilot Director needs to be adjusted to submit pilots with similar number of cores equal to what is defined as one of site tags in the DIRAC CS. Different from the single and customized pilot mode where one pilot only accepts one job, the new pilot working mode introduced a need to accept more than one job type, which is the key part in the design. In the new model, the pilots take care of more steps than the pilots in the single-core and customized pilot mode:

(1) First detect available cores which can be provided to user jobs
(2) Ask the matcher for a proper user job which must require less cores than current available cores
(3) Start the user job
(4) Do step (1) (2) (3) again until the number of available cores are equal to 0

A process management tool in DIRAC has been used to handle multiple job processes in a job pool where jobs are managed with a queue, arranged to run in parallel, and get out of queue when finished. The job status is reported periodically to the pilot for adjusting number of available cores.

For cloud, VM Director instead of Pilot Director needs to create multi-core VMs and run multi-core pilots mentioned above according to the mode chosen.

3.2 Tags for matching

In the multi-core case, the number of cores is one of the important factors for matching resources and jobs. Therefore, tags are introduced to mark number of cores, which jobs, sites and pilots are configured with. In job JDL (Job Description Language) files, the tag “NumberOfProcessors” as one of job parameters marks the number of cores the jobs required. Jobs are registered into the Task Queue Database (TQDB) with these tags for later matching. For sites, two tags are added in the DIRAC Configuration Service to describe what kind of multi-core pilots and jobs are suitable for their resource. One tag is “MaxProcessors”, which defines how many cores at most can be occupied for one pilot. Normally this value is decided based on the type of worker nodes at the sites. For example, 12-core worker node on sites had better ask for 4-core pilots. The Pilot Director submits pilot jobs to the site batch system asking for the number of cores which is equal to the value of “MaxProcessor”. The other tag is “RequiredTag” to define the number of cores for real jobs which can be accepted by the pilots in this site. The default value of this tag is the range of 1 to MaxProcessors. This tag will be configured in pilots when landing on sites. When the pilots ask for jobs, the matcher will match the tag information “NumberOfProcessors” from TaskQueue and “RequiredTag” from pilots to find proper jobs to send to the pilots.

3.3 Monitoring

There are two kinds of job monitoring in DIRAC. One is user job's monitoring showing information and status of user jobs. The other is pilot jobs monitoring showing the status of pilot jobs and the links to the jobs it has pulled. In multi-core case, since both user jobs and pilot jobs are multi-core, the monitoring needs to add core counts to show how many cores are used by jobs or occupied by pilots. In the shared partitionable pilot mode, besides showing the total cores occupied, it is necessary for the pilot job monitoring to show how many cores have been in use or are free at the moment.
Differently from single-core case, the shared pilots are not easily filled up by user jobs. Therefore, scheduling efficiency of pilots is a factor requiring attention. A plot showing changes of number of cores for each pilot and the jobs running inside over times has been developed. The cores used by jobs are called job cores, and the cores reserved by pilot are called pilot cores in this paper. The difference between number of job cores and pilot cores shows the scheduling efficiency of pilots. In Fig. 4, the scheduling efficiency of the chosen pilot has been shown. From the plot we can see that cores of pilots are not fully utilised by user jobs in its life cycle.

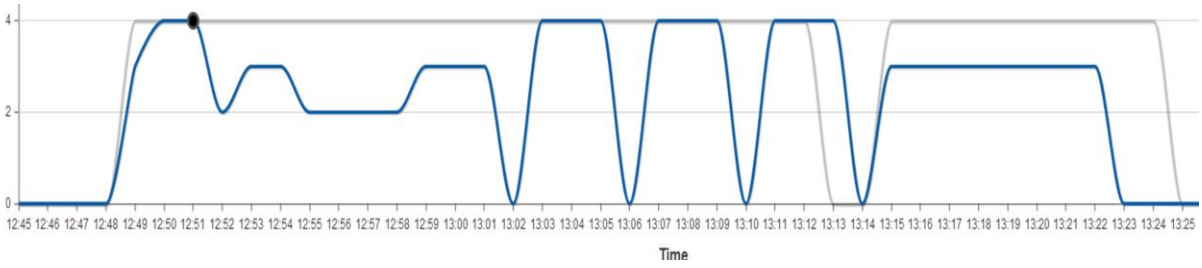

Fig. 4. Scheduling efficiency for one multi-core pilot (X: Time, Y: Cores, gray line shows available cores in pilots, blue line shows cores used by jobs)

4. Tests and scheduling efficiency study

4.1 Tests

Tests sites with SLURM[7] and HTCondor[8] batch systems have been set up and registered in the DIRAC multi-core testbed. Multi-core test jobs are submitted through DIRAC and run in these two sites. Multi-core and whole-node queues were built up in the SLURM cluster. HTCondor nodes are configured to support whole-node or multi-core. The paralleled JUNO simulation software based on the Geant4.10 was used in the tests. The total resources are 216 CPU cores, and each worker nodes have either 12 or 24 CPU cores. Three user job types are used: single-core, whole-node, mixture of single-core and multi-core jobs, and three working modes are used: original single-core, customized multi-core, shared partitionable multi-core. Monitoring and accounting were done with ElasticSearch[9]. Test results showed that three modes are working well in the testbed with SLURM and HTCondor sites, and scheduling efficiency varies with different job types. Scheduling efficiency (**e**) here was defined as resource utilization rate of user jobs using resources provided by pilots, which is described by Equation (1).

$$\mathbf{e} = \frac{\sum_{i=1}^{n} \mathbf{Jc}_i * t_i}{N * \mathrm{time}} \tag{1}$$

time: total time used by all user jobs run during the test
N: total CPU cores provided by all pilot jobs
n: number of user jobs
Jc_i: number of cores job i used
t_i: running time of job i

4.2 Efficiency study

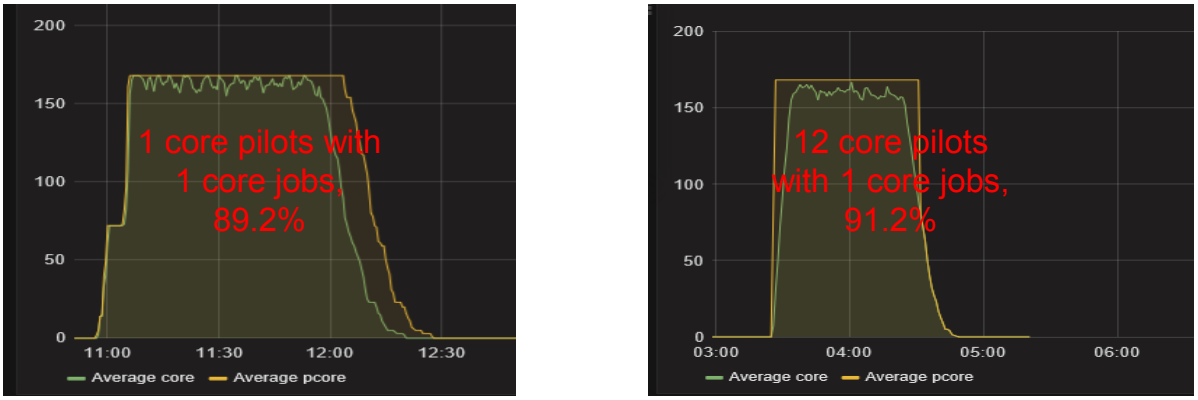

Fig. 5. Scheduling efficiency with single-core jobs running in single-core mode and shared partitionable mode (X: Time, Y: CPU cores. yellow lines show number of CPU cores occupied by pilots and green lines show number of CPU cores used by user jobs)

There are only slight differences in scheduling efficiency between single-core jobs running in three modes. An example is shown in Fig. 5. With same input of job type and number (170 CPU cores, 1000 jobs)， single-core jobs running in the single-core mode got 89.2% efficiency, while 91.2% in the shared partitionable mode. In this case waist of efficiency visible in the plot as difference of number of pilot cores vs number of job cores is caused by the pilot overhead (pilot start, pulling user job, running without jobs inside).
Tests were also done with a hybrid of various-core jobs for two multi-core pilot modes. Small scale tests showed that the scheduling efficiency of customized pilots is 48%, much worse than that of the shared partitionable pilots 81%. It is expected, since there were more idle pilots in the customized pilot mode than in the shared partitionable one due to its one-to-one matching policy. Also tests showed that the scheduling efficiency of the shared partitionable pilot mode is not as good as that of single-core mode, where user jobs can't fill the whole resource with pilots, as shown in Fig. 6. In the shared partitionable pilot mode, 4-core pilots have been used in the tests. The scheduling efficiency of the single-core mode is 89.6% and that of that partitionable mode is 77.1%

Fig. 6. Comparison of scheduling efficiency between single-core pilot mode and shared partitionable pilot mode, the left plot shows the scheduling efficiency (total number of cores used by jobs vs. total number of cores reserved by pilots) of two modes independently of their job type in terms of number of cores, the right plot shows number of cores used by every type of jobs individually over time. The left part of each plot is single-job running in single-core pilot mode, The right part is various-core jobs running in shared partitionable pilot mode (green line is for single-core jobs, yellow line for two-core jobs, blue line for four-core jobs)

Let's look deeply into the default way the partitionable pilots do the scheduling. 12-core pilots were used for tests with 1-core, 2-core, 4-core, 8-core jobs. The number of jobs of each type was identical. Results have shown that the scheduling efficiency for this case is 75%. As can be seen from Fig. 7, the jobs with less cores are selected at the beginning and 8-core jobs finish at the end, while 4-core pilots are idle since no other jobs left at this time. Therefore, one of main efficiency losses is caused by the inefficient scheduling strategy.

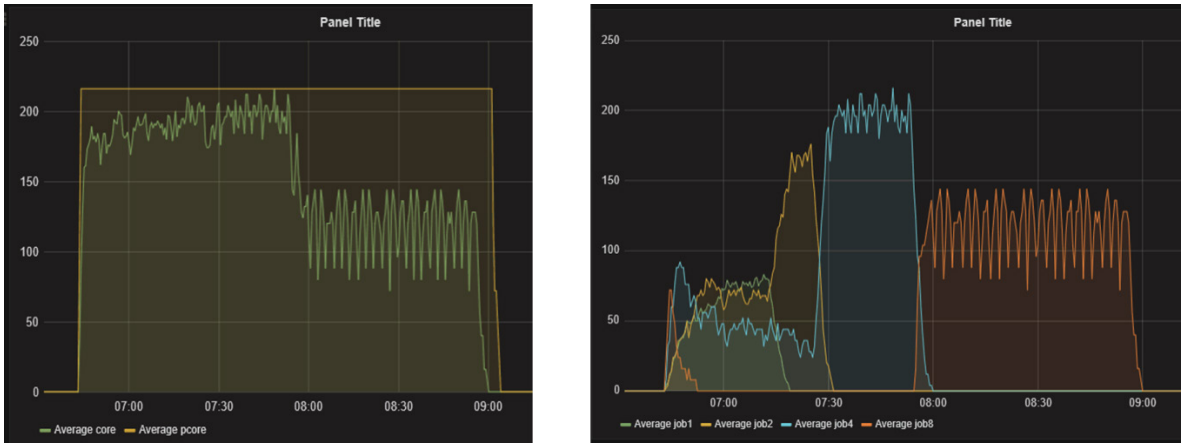

Fig. 7. Scheduling efficiency for 12-core pilots shown on the left plot, the corresponding CPU cores used by user jobs over time shown on the right plot (green line is for 1-core jobs, yellow line for 2-core jobs, blue line for 4-core jobs and orange line for 8-core jobs)

4.3 Scheduling efficiency optimization

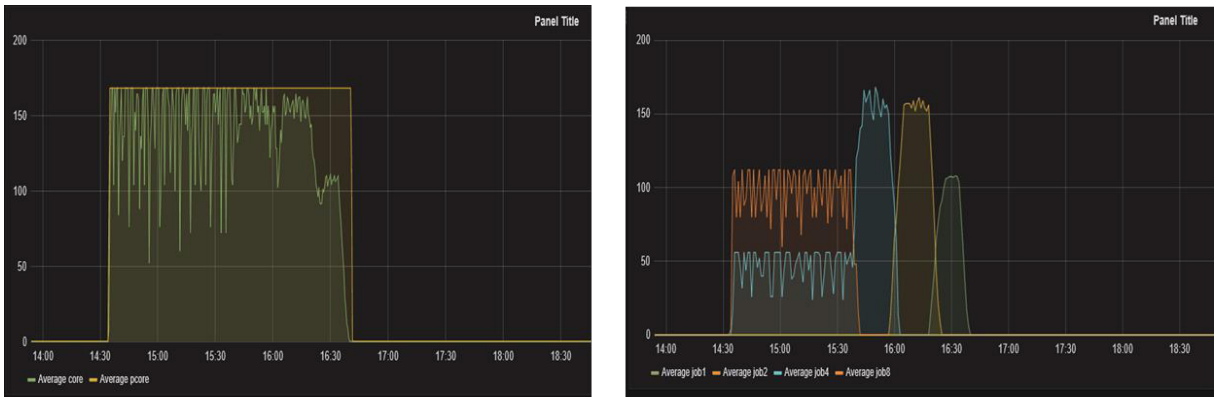

Fig. 8. Scheduling efficiency for 12-core pilots with the new scheduling policy (green line is for 1-core jobs, yellow line for 2-core jobs, blue line for 4-core jobs and orange line for 8-core jobs)

The default scheduling policy in the single-core mode is that jobs matched are randomly chosen to run inside pilots. With this default policy, just as the test described in section 4.2, jobs with less cores are chosen first which results in resource gaps in case of jobs requiring more cores. As a result, a new scheduling policy was proposed and tested to improve scheduling efficiency. In this policy, jobs with high priority are chosen to run first. The priority of each job is based on factors like job waiting time, current available number of cores inside pilots and number of cores requested by jobs, etc. The equation (2) is defined to count job priority in JUNO case, where *Pi* is priority value for job_i, more factors can be added according to different use cases.

$$P_{\mathrm{i}} = ae^{k(\mathrm{v}-c_i)^2} + b(w_i + r_i)/r_i \tag{2}$$

There are two parts in the equation. The parameters a、b、k can be tuned according to the requirements of experiment use cases. The first part of the equation is based on the exponential distribution function, where e is natural constant or Euler number[10]. The function tends to give "big" jobs high priority to reduce resource gap. When the number of pilot cores is the same as the number of job cores, the value of the first part is a, the maximum. The smaller difference in core number between pilot（v）and the job（c）, the higher the priority the job gets. The second part of the equation is to avoid "starving" of "small" jobs. The higher waiting time (w) above average waiting time (r), the higher is priority. The tests with this new policy showed that the efficiency can be improved by 15%, as shown in Fig. 8. 8-core jobs are matched first, and 1-core jobs can fill the remaining gaps at the end.

5. Conclusions and future work

The prototype of multi-core job support in the DIRAC-based JUNO distributed computing platform was set up and was working properly with two multi-core pilot modes. Tests with multi-core JUNO simulation jobs showed that scheduling efficiency of multi-core jobs is not as good as that of single-core. Small scale efficiency study showed that the shared partitionable pilot mode is more promising with a hybrid of various-core jobs than the customized pilot mode. An optimization method has been provided to improve the scheduling policy in the shared partitionable pilot mode and tests has shown that 15% improvement can be gained with this method. Larger scale tests and further work is required to further improve scheduling efficiency with real JUNO use cases.

Acknowledgments

The authors would like to thank JUNO offline software team for their help with the JUNO software, Andrei Tsaregorodtsev for his help with the DIRAC workload management system, and colleagues at the IHEP computing centre for their support. This work was funded by the National Natural Science Foundation of China (NSFC) under grant no. 11775246.

References

[1] F.P. An, *Neutrino Physics with JUNO*, J. Phys. G **43** 030401 (2016)

[2] J.H. Zou, *SNiPER: an offline software framework for non-collider physics experiments*, J. Phys.: Conf. Ser. **664** 072053 (2015)

[3] J. Reinders, *Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism* , Sebastopol: O'Reilly Media, ISBN **978-0-596-51480-8** (2007)

[4] A. Casajus1, R. Graciani, A. Tsaregorodtsev et al. *DIRAC pilot framework and the DIRAC Workload Management System*, J. Phys: Conference Series **219** (6) (2010)

[5] F. Stagni, A. Tsaregorodtsev, P. Charpentier, K. D. Ciba, Z. Mathe, ... L. Arrabito. (2018, October 8). *DIRACGrid/DIRAC: v6r20p15* (Version v6r20p15). Zenodo. http://doi.org/10.5281/zenodo.1451647

[6] R. Moreno-Vozmediano, R. S. Montero, I. M. Llorente. *IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures*, IEEE Computer, vol. **45**, pp. 65-72, Dec. (2012)

[7] A.B. Yoo, M.A. Jette, M. Grondona, *SLURM: Simple Linux Utility for Resource Management*. In: D. Feitelson, L. Rudolph, U. Schwiegelshohn (eds) *Job Scheduling Strategies for Parallel Processing*. JSSPP 2003. Lecture Notes in Computer Science, vol. **2862**. Springer, Berlin, Heidelberg (2003)

[8] J. Basney, M. Livny, T. Tannenbaum: *High Throughput Computing with Condor*. HPCU news **1**(2) (June 1997)

[9] G. Clinton , T. Zachary, *Elasticsearch: The Definitive Guide*, O'Reilly Media, Inc. (2015)

[10] M. Hazewinkel, ed. (2001) [1994], "Exponential distribution", Encyclopedia of Mathematics, Springer Science+Business Media B.V. / Kluwer Academic Publishers, ISBN 978-1-55608-010-4