New Multi-resource Fairshare Prioritization Mechanisms for Heterogeneous Computing Platforms

Dalibor Klusáček[1,2], Hana Rudová[2]

[1] CESNET, Prague, Czech Republic
[2] Faculty of Informatics, Masaryk University, Brno, Czech Republic
emails: klusacek@cesnet.cz, hanka@fi.muni.cz

Keywords: multi resource fairness, fairshare, scheduling, user, CPU time, RAM

1. Introduction

This work describes newly developed multi-resource fairshare mechanism which is currently implemented in the Czech National Grid MetaCentrum. MetaCentrum provides computational resources to various users and research groups. Here, it is very important to guarantee that computational resources are shared in a fair fashion with respect to different users. For such a purpose, fairshare-based prioritization of users is commonly applied.

As discussed in [1,2], common fairshare mechanisms only deal with a *single resource* situation. It means that the fairshare is computed with respect to a single resource only, e.g., CPU time. Sadly, such an approach is unfair as both jobs and nodes in the system can be highly heterogeneous by means of requested and provided resources. In a real life, some users utilize very few CPUs but require lots of memory (and vice versa). Then, remaining CPUs cannot be utilized by remaining users since there is no free RAM left. Clearly, classical fairshare mechanism based on consumed CPU time is absolutely unacceptable as the users with high RAM requirements are not adequately penalized with respect to users who only need (a lot of) CPUs.

2. Related Work and Proposed Solution

There are few works that consider the use of multi resource-based fairness. For example, *Dominant Resource Factor (DRF)* [1] schedules jobs according to so called *dominant user's share*, which is the *maximum share* that the user has been allocated of any resource. Sadly, DRF is not very suitable as it assumes that all jobs and resources are infinitely divisible [1], which is rather unrealistic in Grids. Moreover, DRF only focuses on current users' allocations and does not consider their previous resource usage. A more suitable is so called *Processor Equivalent (PE)* [3] which takes the *maximum* of relative consumptions of each resource and multiplies it by the number of available CPUs. If CPU and RAM requests are considered, then for a given job *j* its PE_j is computed using Formula 1.

$$PE_j = \max(reqCPUs_j / availCPUs, reqRAM_j / availRAM) \cdot availCPUs \quad (1)$$

PE is then used within fairshare to measure users' resource usage. However, PE is not suitable for heterogeneous, Grid-like environments. Clearly, as soon as machines are heterogeneous, different values of PE_j appear for machines that have different CPU and RAM parameters. This is unfair as the resulting PE_j (affecting user's priority) then highly depends on a given scheduler's decision, which a user cannot fully control. Therefore, in the following text we present a *complex extension of PE mechanism for heterogeneous platforms*.

We propose a new, multi resource-aware mechanism that determines job's penalty based on its CPU and RAM requirements. Further resources like HDD or GPUs can be also

included. The job's penalty is then used in the overall fairshare mechanism to determine user's priority. The computation works as follows. At first, we compute $PE_{j,i}$ for every machine *i* suitable to run job *j* (see Formula 2). This must be done since different suitable machines may produce different $PE_{j,i}$. Next, a job's penalty is determined by Formula 3.

$$PE_{j,i} = \max(reqCPUs_j / availCPUs_i, req\ RAM_j / availRAM_i) \cdot availCPUs_i \quad (2)$$

$$penalty_j = \min(PE_{j,1}, .., PE_{j,m}) \cdot walltime_j \cdot SPEC_{target} \cdot queue\_cost \quad (3)$$

The key step is that we choose the minimum of all $PE_{j,i}$ values. This guarantees that our users will pay the smallest available "price" for their jobs. It is worth noticing that the actual *target* machine(s) where a job is executed may be different from the "cheapest" one, as the allocation is done by a scheduler which uses different criteria. Finally, the minimal $PE_{j,i}$ is multiplied by a job's walltime which is normalized according to a speed of the target machine (see $SPEC_{target}$) which is established using a SPEC benchmark (Standard Performance Evaluation Corporation). This normalization is necessary in order to capture the speedup effect of new, fast machines that may significantly reduce job's walltime compared to old and slow machines. Without this normalization, jobs being executed on slow machines (with low SPEC) would be highly penalized (via their long walltimes) compared to those executed on fast machines. Last but not least, if a job requests a (priority) queue with a given fixed price (*queue_cost*), it is used as well to further adjust $penalty_j$.

3. Conclusion and Future Work

This paper addresses so called multi resource fairness which is an important issue in current heterogeneous computing platforms. We have proposed a new metric that allows computation of job's penalty which is later used in fairshare prioritization process. The main contribution is that our solution—unlike traditional techniques—considers multiple consumed resources. Moreover, it is designed for heterogeneous environments. As the solution accounts grid users with the smallest possible price, it is fair as the resulting "price" is only related to the user's requests (job specification) and is therefore independent from actual scheduler's decisions, which is highly desirable.

The solution presented in this paper is currently implemented within MetaCentrum's scheduler. Prior deployment, we will perform an analysis to what extent is the walltime's normalization by SPEC reasonable. Furthermore, in the near future we plan to extend $PE_{j,i}$ metric by considering other important (and restricting) resources like GPUs and HDD. This extension will require some modification of $PE_{j,i}$ calculation. The reason is that while CPUs and RAM are required by all jobs, this is not the case for, e.g., GPUs. Clearly, a machine with fully consumed GPUs may still execute other jobs that do not require GPUs.

Acknowledgements. This work is kindly supported by the LM2010005 project funded by the Ministry of Education, Youth, and Sports of the Czech Republic and by the grant No. P202/12/0306 funded by the Grant Agency of the Czech Republic.

References

1. A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, I. Stoica. "Dominant resource fairness: fair allocation of multiple resource types". In Proc. of the 8th USENIX conference, 2011.
2. D. Klusáček, H. Rudová, M. Jaroš. "Multi Resource Fairness: Problems and Challenges". In Proc. of 17th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), 2013.
3. D. Jackson, Q. Snell, M. Clement. "Core algorithms of the Maui scheduler". In Proc. of 7th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), 2001.