IMPROVING JOB SCHEDULING ON A HETEROGENEOUS CLUSTER BY PREDICTING JOB EXECUTION TIMES USING HEURISTICS

Hannes Brandstätter-Müller[(a)], Bahram Parsapour[(b)], Andreas Hölzlwimmer[(c)], Gerald Lirk[(d)], Peter Kulczycki[(e)]

[(a, b, c, d, e)]Upper Austria University of Applied Sciences
School of Informatics, Communication, and Media
Department of Bioinformatics

[(a)]hannes.brandstaetter@... [(b)]bahram.parsapour@... [(c)]andreas.hoelzlwimmer@... [(d)]gerald.lirk@...
[(e)]peter.kulczycki@fh-hagenberg.at

ABSTRACT

In this paper, we propose the scheduling system for the *Bioinformatics Resource Facility Hagenberg (BiRFH)*. This system takes advantage of the fact that the facility offers tailored solutions for the customers, which includes having a limited amount of different programs available. Additionally, the BiRFH system provides access to different hardware platforms (standard CPU, GPGPU on NVIDIA Cuda, and IMB Cell on Sony Playstation machines) with multiple versions of the same algorithm optimized for these platforms. The BiRFH scheduling system takes these into account and uses knowledge about past runs and run times to predict the expected run time of a job. That leads to a better scheduling and resource usage. The prediction and scheduling use heuristic and artificial intelligence methods to achieve acceptable results.

The paper presents the proposed prediction method as well as an overview of the scheduling algorithm.

Keywords: algorithms, bioinformatics, high performance computing, molecular biology

1. INTRODUCTION

Scheduling and resource management are fundamental tasks when running a high performance computing system. Resource management and scheduling systems for different processor technologies and architectures in a single cluster are not very common although they offer great possibilities to the user. Our system software "Bioinformatics Resource Facility Hagenberg" (BiRFH) allows efficient control and management of the so-called "Meta-Heterogeneous Cluster". BiRFH allows one not only to drive classic heterogeneous clusters (i. e., systems that comprise nodes that vary just in CPU speed and RAM size), it allows one to integrate and operate different processor architectures simultaneously. Our resource facility currently consists of standard Intel CPUs (Intel 2005), NVIDIA GPUs (NVIDIA 2010), and IBM Cell Broadband Engines (IBM 2006). There are many strategies available for scheduling jobs on a cluster. We focus on the special case where jobs are based on a small number of algorithms with different input data. This allows runtime prediction using heuristics and therefore an improved deadline scheduling.

The "Bioinformatics Resource Facility Hagenberg" is a resource management and scheduling system targeting the needs of microbiology and bioinformatics related high performance computing. The main features are: (1) integration and management of different hardware platforms, (2) scheduling jobs so that the available hardware is used, and (3) making the management and job creation more accessible to non-technical users.

To fully enable the necessary features, and supported by the fact that the BiRFH service is designed to be used with a limited number of algorithms rather than allowing arbitrary code to be uploaded and executed like on many other standard compute clusters, BiRFH requires the algorithms to be adapted and to support some defined methods. The BiRFH framework seeks to mitigate the necessary development effort. The system is based on a framework that is to be included in each algorithm if possible, i. e., if the source code is available for modification. Including the framework directly into the source allows the use of more advanced features and more control over the program. Should the source for a program be unavailable, the framework also supports the creation of wrapper programs that can in turn execute the desired program. Regardless of how the framework is applied, it allows the use of the best available hardware for the selected algorithms with the trade-off of higher development effort to enable the algorithm on as many hardware platforms as possible.

As a further feature, BiRFH uses heuristic scheduling and resource management algorithms in order to optimize the cluster's throughput.

There are many scheduling systems and resource managers for high-performance cluster computing available (see Table 1). Most of them are designed for uniform hardware. Almost no system allows the coupling of compute platforms having different processor architectures in a way that e. g. allows the migration of a running algorithm from one type of platform to a different one. BiRFH offers this possibility for algorithms with available source

code by implementing a data exchange that can also be used for hibernation, i. e., the freeing of used system resources by writing the current state of the calculations to the hard drive.

There are some other approaches to enable heterogeneity for compute-intensive applications. These include OpenCL (Munshi 2011) and C++ Accelerated Massive Parallelism (AMP) (Sutter 2011; Moth 2011). These focus on enabling single applications flexible access to any available computing device rather than distributing instances of algorithms over the available hardware. The BiRFH approach focuses more on user guidance and supporting the storage of data on the remote compute system. It is notable, however, that Microsoft's C++ AMP moves the definition of heterogeneity more in the direction of heterogeneous platforms than the previously common heterogeneous systems, i. e., including different processor architectures.

BiRFH focuses on algorithms and computations for biomolecular and bioinformatics applications. The reasons for focusing especially on bioinformatics lie (1) in the near-exponential growth of available data in the currently booming field (Howe et al. 2008), (2) in the demands for making high performance computing available to non-technical users and (3) the availability of bioinformatics knowledge in the project team. Moreover, the BiRFH system supports a scheduling mechanism that is (1) based on the temporal behavior of algorithms and (2) also based on the size and the inner structure of input data to be processed.

Feature	**Condor**	**SGE**	**SLURM**	**Maui**	**MPI2**	**BiRFH**
Workload Manager	✓	✓	✓	✓		✓
Cycle Scavenging	✓					
Heterogeneous Platforms	✓		(✓)		(✓)	✓
Priority Based Scheduling	✓	✓	✓			✓
Hibernation Support	(✓)		✓			✓
Resource Based Scheduling		✓		✓		✓
Advanced Resource Reservation				✓		✓
Topology Awareness			✓			

Table 1: Some features of well-established resource managers and scheduling systems, in addition to the BiRFH system, from (Hoelzlwimmer 2010).

2. DATA MINING AND HEURISTICS

The terms *makespan* and *flowtime* (Pinedo 2008) are commonly used when classifying the success of the output of a scheduler. More advanced scheduling techniques, i. e., most scheduling methods beyond simple load balancing, require a knowledge of the expected time to complete a task. Some solutions (Xhafa and Abraham 2008) require an estimation by the user, which is sometimes too complicated a task for non-IT scientists, and on the other hand does not account for different versions of an algorithm optimized for the available heterogeneous hardware. Therefore, the *BiRFH* approach does not require run time estimations by the user.

Most algorithms exhibit some form of correlation between the input data, other given parameters and the time the program needs to run. The *BiRFH* system gathers performance data on the various algorithm implementations as they are executed. The collected data consists of the runtime measurements, i. e., how long the execution of the task took in real-time. Additionally, the consumed CPU time is also measured. Should the hardware be under-subscribed, algorithms can be executed during this idle time to generate additional measurements, especially for combinations of parameters that are expected to complete "holes" in the available data. This performance data, combined with data about the input and other parameters, is used by machine learning algorithms to predict the temporal behavior of subsequent runs, especially for new parameter combinations or input data.

As the parameter values, especially the input data, usually consist of file names or other non-numeric values, it can not be used directly for the prediction of the expected runtime. Therefore, the *BiRFH* framework calls a method that is expected to be provided for each algorithm. This method should provide a numeric value that represents each non-numeric parameter value. If, for example, one algorithm parameter is an input file name, the method produces a number signifying the "weight", or expected impact on the runtime, of the contained data. Simple file size is sometimes not sufficient to use when predicting the impact on run time, e. g. when processing a file containing sequences in FASTA format, sometimes the number of sequences, other times the average or maximum length of the sequences is more significant. The amount of computation time to produce this number should be kept within a reasonable time frame, but is left to the algorithm developer to decide. If a simple file size is not sufficient, then partial, sampling or even a total file analysis should be done. This is only appropriate if the computational run time is very long compared to the time needed to load the data from the file, and not just slightly longer than the full analysis of the input file itself would take. The next chapter, 2.1 Heuristics, contains an example dataset and shows how the prediction works.

2.1. Heuristics

The machine learning algorithms that provide the best results on the current training data sets are Artificial Neural

Networks (ANNs). These are very versatile and produce accurate predictions for the available data. Other evaluated machine learning algorithms are the regression algorithms offered by the Weka toolkit (Hall et al. 2009).

To evaluate the available regression algorithms, a sample data set with 2400 measurement instances was created. This data set consists of measurements of a simple algorithm that reads a file containing several FASTA formatted sequences. Then, some string operations are performed and an output file is written. This algorithm is executed with three different input files (one with 500 sequences, one with 1000 and another one with 1500) and also with different parameters influencing the string operations. The collected data can be presented as CSV:

```
wall,InSize,InCount,MaxMut,WinSize,GC-Cont
2288.74,89725,500,1,5,0
3279.26,89725,500,1,5,0.8
2540.13,181156,1000,1,5,0
4627.02,564117,1500,1,5,0.7
5089.33,564117,1500,5,16,0.6
...
```

Wall represents the wall clock time used to complete this algorithm. InSize is the size of the input file, InCount is the number of FASTA sequences, MaxMut, WinSize and GC-Cont are parameters that influence how the input is processed and may or may not have an influence on the run time. These is the data available after the algorithm runs have finished, and as mentioned in the chapter above, some of these data points are available before the real computation is started. In this case, everything except the wall clock time and the InCount is available before the algorithm is started for the calculation run. Getting the file size is not as compute intensive and can substitute the exact count of input sequences for this case.

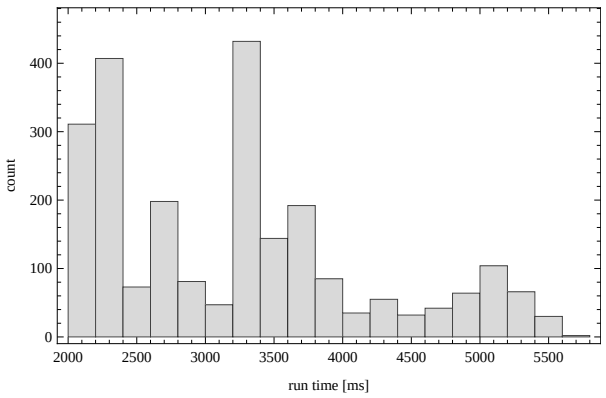

Figure 1: Run times of 2400 calls to the same algorithm with different input data and parameters

Looking at the run times gathered, see Figure 1, it is easy to spot three distinct peaks which indicate the run time effect of the two different input file sizes. There is also a third peak in the upper run time regions that is most likely caused by one of the parameters. The remaining variance due to other parameter variations. This reinforces the assumption that a regression method can most likely predict the run times accurately from this data set.

Weka yielded the following results with 10-fold cross validation:

Classifier	**CC**	**MAE**	**RMS**	**RAE**
MultilayerPerc.	0.9256	280.06	358.72	50.3 %
LinearRegr.	0.8714	385.02	464.75	69.7 %
IsotonicRegr.	0.7620	502.95	613.38	65.8 %
SimpleLinRegr.	0.6626	551.43	709.47	72.2 %
PACERegr.	0.8713	385.04	464.79	50.4 %
LibSVM	0.8452	403.02	507.48	52.7 %

Table 2: Results in the Weka Toolkit for various prediction methods: MultilayerPerceptron, LinearRegression, IsotonicRegression, SimpleLinearRegression, PACERegression, LibSVM (*CC*: Correlation Coefficient, *MAE*: Mean absolute error, *RMS*: Root mean squared error, *RAE*: Relative absolute error)

Taking one of the validation results to better visualize the resulting prediction quality.

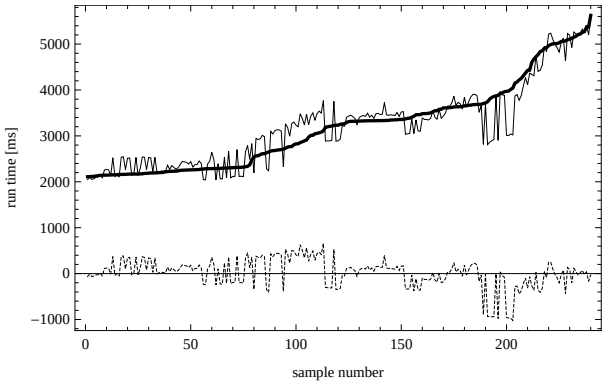

Figure 2: Actual and predicted values of a validation run using MultilayerPerceptron

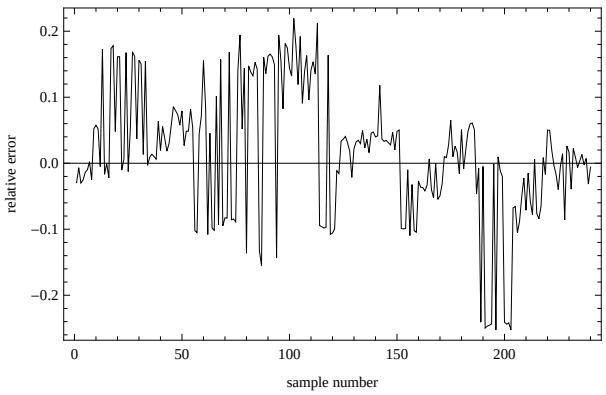

Figure 3: Relative errors of the validation run using MultilayerPerceptron

Using libSVM, the results are similar but with a higher margin of error. Figures 4 and 5 show the best result achieved by tweaking the libSVM parameters "cost", "gamma" and "epsilon" with *epsilon-SVR* in a similar fashion as above with *MultilayerPerceptron*.

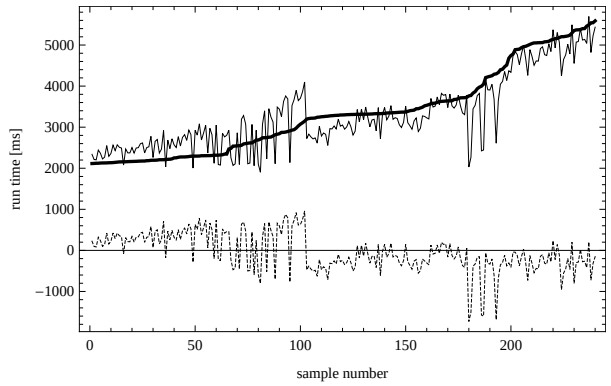

Figure 4: Actual and predicted values of a validation run using libSVM

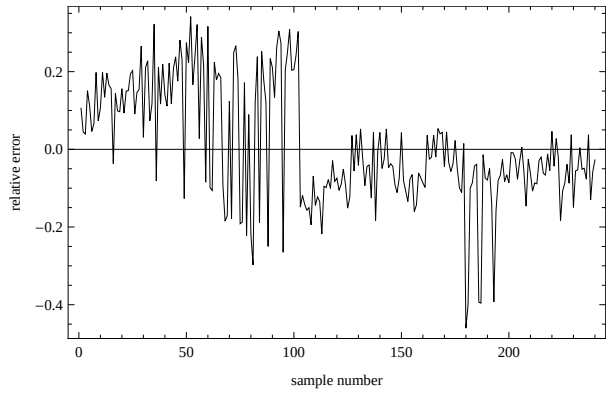

Figure 5: Relative errors of the validation run using libSVM

Standard ANN training algorithms usually require the basic structure to be predefined, i. e., not only the number of inputs and outputs, but also the number of internal layers, the nodes per layer and how the nodes are connected. This would prevent maximal flexibility of the training for the expected diversity of the dataset, as the complexity is an important factor in the success of the neural network training: too little complexity, and the solution quality suffers, but too much complexity would lead to over-fitting during the training. To work around this limitation and make the ANN approach viable for a multitude of different algorithms, training algorithms that start with an empty network and add complexity as needed to reach a neural network for the given training set are used in the BiRFH system. The FANN library (Nissen 2003), which was chosen as ANN implementation, provides such a training algorithm called Cascade2 (Nissen 2007).

3. SCHEDULING

The task of scheduling and resource management working together is to ensure that computation tasks are completed in a timely manner and that the available hardware is used as efficient as possible. There are many strategies available to fulfill these requirements.

As mentioned before in chapter 2, makespan and flowtime are used as indicators for the quality of a scheduling attempt. Having a reliable prediction for the expected run time enables more advanced scheduling features. *Earliest Deadline First (EDF)* is one of the most popular and widely used scheduling strategies, and has some parts in common to the BiRFH approach. The *EDF* strategy requires tasks to have a deadline, usually either the latest possible start time or the latest acceptable finishing time. The scheduler then orders the tasks by their deadlines and executes those with the earliest deadline first. Usually resource managers can also interrupt running tasks should a new task with an earlier deadline become available (Kruk et al. 2007).

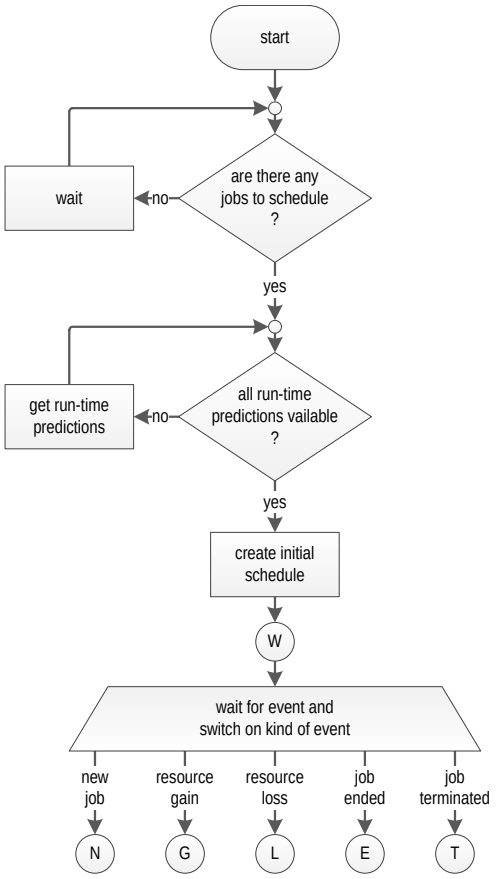

Figure 6: A simplified flowchart of the scheduling system

The basic program flow in the scheduling logic is displayed in figure 6. After an initial check for pending jobs, and calculating the run time estimation for these, the scheduler distributes these jobs among the available resources according to the scheduling policy. Then, the scheduler waits for one of the following events to occur:

(N) new jobs are submitted to be scheduled and executed, (G) an additional machine comes online and is available to execute jobs, (L) a currently available machine goes offline, (E) a currently running job ends or a running job terminates unexpected (T). The scheduler then handles these events accordingly and returns to the waiting state. Some paths lead to a rescheduling, others do not require a reordering. Depending on the circumstances, the reordering can affect the currently running processes as well.

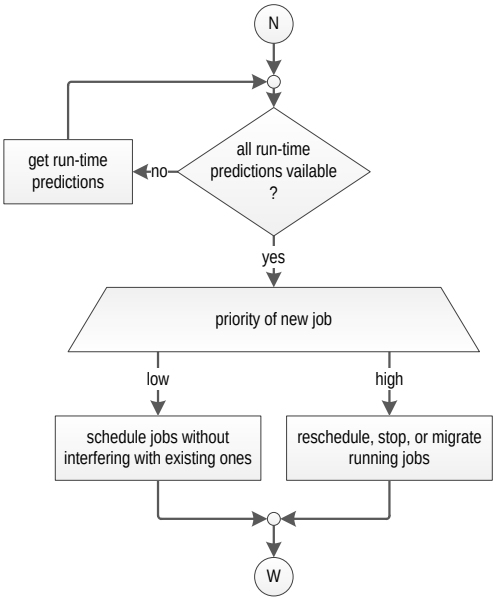

Figure 7: Scheduling for newly submitted jobs

In the path (N) new jobs are submitted, these are again run through the prediction. After the predictions are available for all new tasks, these are scheduled according to their priority. If their priority is default normal, then no special additional steps have to be performed and the tasks are scheduled without interfering with currently running tasks. Should the new tasks have high priority, the currently running tasks are included in the rescheduling, which could lead to low priority tasks to be paused, migrated or even aborted and restarted at a later point.

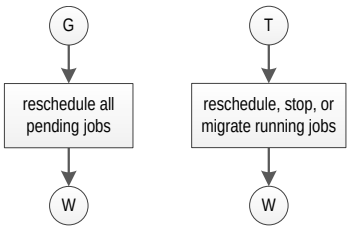

Figure 8: Scheduling for the case of the gain of a resource (left) or an erroneous job termination (right)

Path (G) (Figure 8, left) is caused by new hardware resources coming available for the system. The special case where these resource has been known to the system already is handled in the third path. This path assumes that the resource is new or has been offline long enough that it can be regarded as new. All pending jobs are rescheduled, as are all currently running jobs if there is a significant reduction in overall time to completion.

The next path, (L), deals with the sudden, unplanned loss of a resource: all currently running and pending jobs of this resource are rescheduled over the remaining resources. Should the resource come back online in time, i. e., before the "replacement job" is completed, and the job on that resource is healthy (i. e., was not broken by the loss of connectivity), the now duplicate backup job is canceled and the pending jobs are rescheduled.

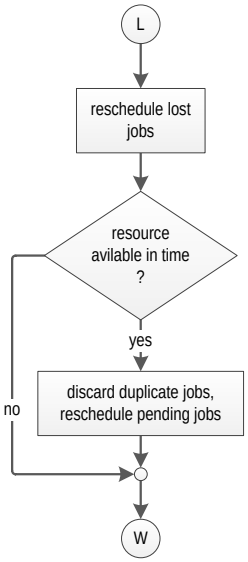

Figure 9: The scheduler handles the unexpected loss of a resource

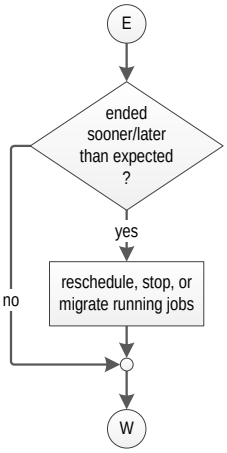

Figure 10: Scheduling when a job ends without error

The remaining two paths are triggered every time a job ends. If the job ends cleanly (E), the run time is compared to the estimate, if the difference is below the

threshold, no further scheduling is needed. If the difference exceeds the threshold, a scheduling run is performed to ensure optimal resource usage. If the job did not terminate cleanly (T) (see Figure 8), subsequent dependent runs are removed from the queue and an error notification is sent to the originating user. The remaining jobs, including the running ones, are then rescheduled again. Under one special condition, in the case where two algorithms are running simultaneously and one algorithm produces output that is immediately used by the second algorithm—called streaming read/write—the consuming job has to be terminated if the producing job experiences an error and terminates.

The scheduling algorithm itself is designed to be fully modular. It reads the information about the pending jobs as well as the current system status from a database, and after reordering the jobs according to the scheduling rules and policies, writes the new orders back to the database, where the resource manager reads them and relays the orders to the compute nodes. That way, different scheduling strategies and optimizations can be evaluated without big changes in the whole BiRFH system.

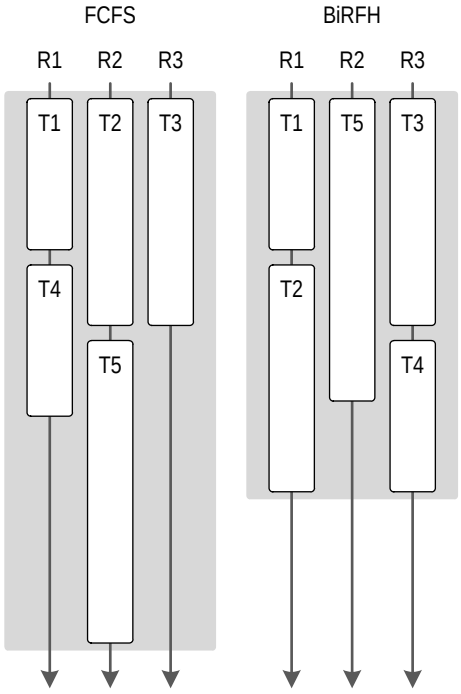

Figure 11: A simple scenario where knowledge of run times yield better scheduling than a basic First Come First Serve

Knowing about the expected runtime can improve the utilization and therefore reduce the overall time needed to complete several tasks. Figure 11 displays a hypothetical comparison of a simple EDF variant, where the tasks are prioritized in the order that they become available, i. e., a First Come First Serve scheduling. In this scenario, 5 tasks with different runtimes are available: Tasks T1 and T4 need 2 time units, tasks T2 and T3 take 3 time units to finish and task T5 is the longest with 4 time units. The simple FCFS scheduling on the left produces an execution sequence that would take 7 time units. Knowledge of the expected run times can rearrange the tasks in such a way that the total execution time would be 5 time units.

The above example is based on the assumption of three completely identical compute nodes. Our scheduling approach aims to improve this strategy for *Meta-Heterogeneous Clusters*, i. e., clusters consisting of different hardware architectures like CPU, GPUs and others, by considering multiple versions on the different hardware platforms and deferring some algorithms expecting a more powerful platform to become available. This is enabled by the run time predictions of the heuristic analysis.

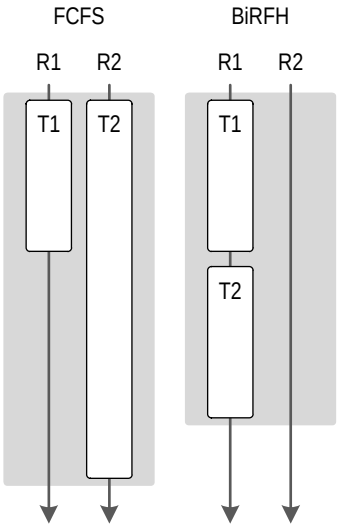

Figure 12: Advantage of knowledge of runtimes with two different hardware platforms

A simple scenario (see Figure 12) where this would be of benefit: Task T2 is a task that finishes fastest on the first system, e. g. a GPU, but the GPU is currently occupied by task T1. Using a standard deadline first algorithm, this task would now run on the second available system, e. g.the CPU. If the R1 node is expected to be ready before the task on the R2 system would be finished, and the sum of the expected calculation for both tasks on the R1 system would put the finishing time earlier, the task would be put on hold instead of using the free CPU.

Even if not all the tasks are known from the beginning, knowledge of the runtimes enables better scheduling in many cases. Figure 13 shows an example: the compute nodes R1 and R2 are again different hardware platforms, which results in different runtimes for the tasks. The FCFS version also displays a possible variant of the basic FCFS strategy, where tasks only are scheduled to hardware platforms where they take the least time to finish. This is shown with T3 (transparent on the lower left). Naively, this should yield better results, but even in this example, using all the available resources still provides better results. The BiRFH result on the right again shows optimized resource usage with reduced overall runtime.

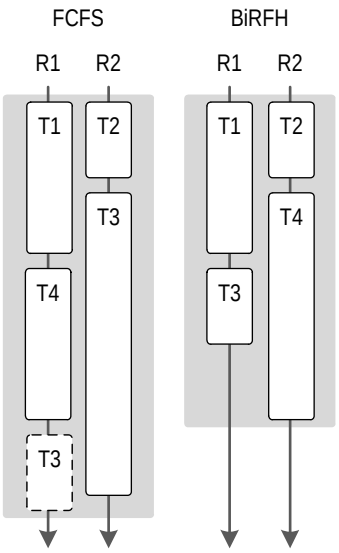

Figure 13: Prioritizing algorithm versions on heterogeneous platforms with minimal makespan in the FCFS strategy does not yield better results

Some scheduling scenarios require a complete rearrangement of the tasks, even running tasks could be blocking an optimal arrangement. In these cases, there has to be an assessment if the running task should be canceled and moved to a different hardware or if it would be better to wait for the task to finish. Knowledge of the expected finishing time can be very helpful in these cases. Figure 14 visualizes a scenario where a new task T3 is added after the tasks T1 and T2 have already been started. This task is only available on the hardware platform of R2, but this system is being used by T2. Therefore, T2 is canceled and restarted on node R1 after T1 has finished. Even though some computational effort is lost, the overall time is significantly shorter than waiting for T2 to finish.

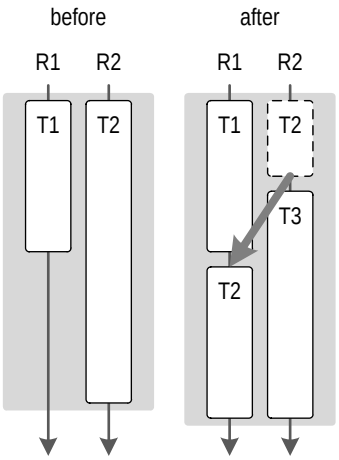

Figure 14: Canceling and rescheduling a task.

Instead of losing the computational progress so far it would be preferable to keep it while migrating to a different hardware platform. This can be enabled when the source code of the algorithm is available. If the algorithm can be adapted to some kind of stepwise progress, the framework defines the methods to pause and serialize the current memory content to the hard disk for transfer. The counterpart implementation on the target hardware then can read the progress and continue with the calculations.

All the above examples target the least "real world" run time, or minimal makespan, for optimization. Other possible targets, especially when using heterogeneous hardware, could be the conservation of power. If a deadline is set and multiple hardware platforms are able to execute the task, the job could run on a platform that consumes less power during the calculations. The power consumption factor, combined with the run time, can be added to the optimization parameters.

As the overall design of the system allows for the individual models to be easily interchangeable, the prediction model and scheduling methods can gradually be replaced by new ones that yield better results. The underlying database offers a simple interface that enables this modularity. Therefore, future development also encompasses the implementation and evaluation of different prediction models and scheduling strategies.

4. CONCLUSION AND FUTURE WORK

The use of heuristics can improve the scheduling quality given some special circumstances, like, in our case, the limited number of different algorithms and the degree of heterogeneity of the hardware.

Future work includes the implementation and evaluation of other computational intelligence strategies as well as the inclusion of the *pipeline* concept, i. e., the consideration of follow-up calculations or calculations using different algorithms based on the same input data, which are required by follow-up calculations.

The research project "Bioinformatics Resource Facility Hagenberg" is supported by grant #821037 from the Austrian Research Promotion Agency (FFG).

REFERENCES

Hall, Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. "The WEKA Data Mining Software: An Update." *SIGKDD Explorations* 11:10–18.

Hoelzlwimmer, Andreas. 2010, September. "A Scheduling Framework Prototype for Heterogeneous Platform High Performance Computing." Master's thesis, University of Applied Sciences, Hagenberg.

Howe, Doug, Maria Costanzo, Petra Fey, Takashi Gojobori, Linda Hannick, Winston Hide, David P. Hill, Renate Kania, Mary Schaeffer, Susan St Pierre, Simon Twigger, Owen White, and Seung Yon Rhee. 2008. "Big data: The future of biocuration." *Nature* 455:47–50.

IBM. 2006. "Cell Broadband Engine Programming Handbook." Technical Report, IBM.

Intel. 2005, November. Intel Pentium 4. http://www.intel.com/support/processors/

pentium4/sb/cs-007993.htm. last visit: July 28, 2010.

Kruk, Lukasz, John Lehoczky, Kavita Ramanan, and Steven Shreve. 2007, December. Heavy Traffic Analysis for EDF Queues with Reneging.

Moth, Daniel. 2011, June. Blazing-fast code using GPUs and more, with C++ AMP. Session at AMD Fusion Developer Conference 2011, http://ecn.channel9.msdn.com/content/DanielMoth_CppAMP_Intro.pdf.

Munshi, Aaftab. 2011, January. The OpenCL Specification, Version 1.1.

Nissen, S. 2003, October. "Implementation of a Fast Artificial Neural Network Library (FANN)." Technical Report, Department of Computer Science, University of Copenhagen.

Nissen, Steffen. 2007, October. "Large Scale Reinforcement Learning using Q-SARSA(λ) and Cascading Neural Networks." Master's thesis, University of Copenhagen.

NVIDIA. 2010, June. NVIDIA CUDA Reference Manual Version 3.1. http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/CudaReferenceManual.pdf. last visit: June 28, 2010.

Pinedo, Michael L. 2008. *Scheduling: Theory, Algorithms, and Systems*. 3rd Edition. Springer.

Sutter, Herb. 2011, June. Heterogeneous Parallelism at Microsoft. Keynote at AMD Fusion Developer Summit 2011, http://developer.amd.com/documentation/presentations/assets/4-Sutter_Microsoft-FINAL.pdf.

Xhafa, Fatos, and Ajith Abraham. 2008. Chapter Metaheuristics for Grid Scheduling Problems of *Metaheuristics for Scheduling in Distributed Computing Environments*, edited by Fatos Xhafa and Ajith Abraham, 1–38. Springer.