Workflows and Scheduling

Seminararbeit

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

Vorgelegt von:	Frank Roeder
E-Mail-Adresse:	3roeder@informatik.uni-hamburg.de
Matrikelnummer:	6526113
Studiengang:	Informatik Bachelor
Betreuer:	Julian Kunkel

Hamburg, den 14.03.2016

Contents

1 Introduction

If we look back what happened to computer-science in general it is unbelievable how drastic the value of processing power and quantity of data had changed over the years. Not to mention what the progress did to social media and the internet in general. We want to take a closer look at science with its new method beside theory and experiments. Ressource hungry applications, like simulations, models and analyses are now a very important part of nowadays research. Slogans such as "Big Data" accentuate the dimension of information we need to get along with. Microphones, cameras and sensors in general are providing us with tons of data by improving their accuracy and resolution. As provision to prevent this "Drowning in Data and Starving for Knowledge", there is a whole branch of research which is about handling the flow of data by using supercomputers.
To represent such a flow of introductions related to the data, we use the mathematical approach of directed acycle graphs. The nodes represent the task and the edges between them the dependencies. What we obtain could be named as workflow - a flow of work must be done, through which we could walk step by step and apply the operations to our given data. These operation could differ in their origin and need for matching resources. Procedures could be preparation of the settings, visualization, calculation or input/output related. Tasks can only be executed in their topological order. Failures in a workflow are special events to be aware of. To get a good impression about a workflow build as an directed acycle graph ones could simulate itself with different tools. To use the supercomputer in an appropriated way there has to be a method which is allot the task to a resource in relation to the time it should start and finish. This method is called scheduling. Scheduling is the activity of mapping processes to resources at a certain time to optimize the allocation in relation to the priority and run time of a task. This paper is based on the given references and includes redrawn and copied figures.

2 Ressource hungry applications

Resource hungry applications are about the solving of grand challenges related to big amounts of data and a lot of processing capacity to handle. Through explicitly and implicitly parallelism within the application it is possible to gain knowledge in a acceptable time period. The job is not done by running our program blindly on every resource we have, but think about the most efficient management is a major part of it. We optimize the performance through development on smarter scheduling, programming, mathematical approaches and a well structured workflow. There are only three basic ways to improve the performance of a resource hungry application. The first step might be to optimize the intelligence at every given point. Algorithms, scheduling and the workflow are huge factors for the performance it can achieve. Another part of improving is to work harder - to push the hardware to its limit and overclock cpus are an example for that. A third idea is to get help and cooperate with other institutions who can supply us with process time. The following picture is based on big data analysis and high performance computing. Those are the key to reach such detailed results with the help of workflows and smart scheduling.

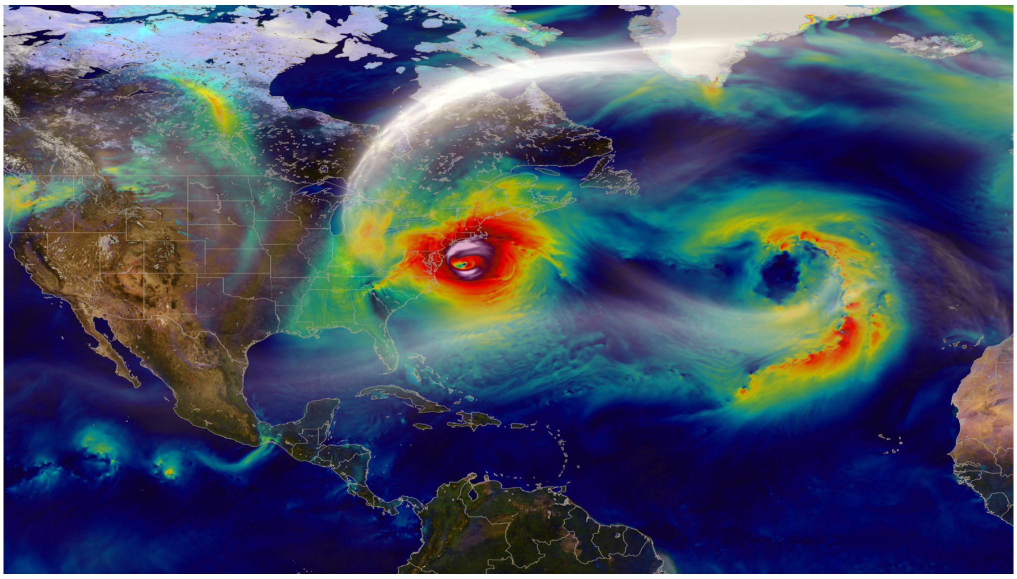


Figure 2.1: NASA Climate Sandy Windstorm [1]

3 Workflows

Scientific workflows are the flow and order of work where conditions determine in which order each work step has to be done. They explain the computational order of tasks very well and visualize it for later analysis. The procedures in a workflow are for example the preparation of source code and scripts. Another node could be the data input and output at the beginning or at the end to store the results. Analyzing during progress to filter only the important data out of the experiment for the visualization as another node afterwards. Store some data while executing to verify the correctness with these pre results, otherwise we need to fix the input or whatever caused the wrong results and restart the the part of the workflow again, is essential for applications with a long run time. Dependencies between work steps are declared by requirements which have to be fulfilled. Without any input operation there is no data to calculate with and without an finished experiment there is no post processing. Graphs are the mathematical approach to represent a workflow in suitable way for the computer.

A **directed acyclic graph** (DAG) is a graph with directed edges where every edge $e \in E$ consists of a pair of vertices (u, v) pointing in the direction from u to v , but not back from v to u. There is no path p which starts at a vertex $v \in V$ through a set of edges $e \in E$ which leads to the vertex v back again. Each node is representing a computational activity and the edges the conditions between them.
One can make a difference between symmetric and asymmetric workflows that will need other handling.

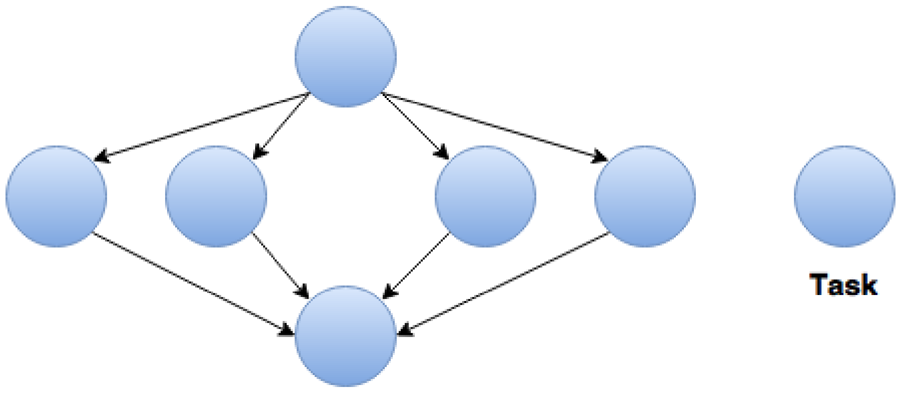


Figure 3.1: [6]

Similar tasks in a directed acyclic graph can be grouped as a job.

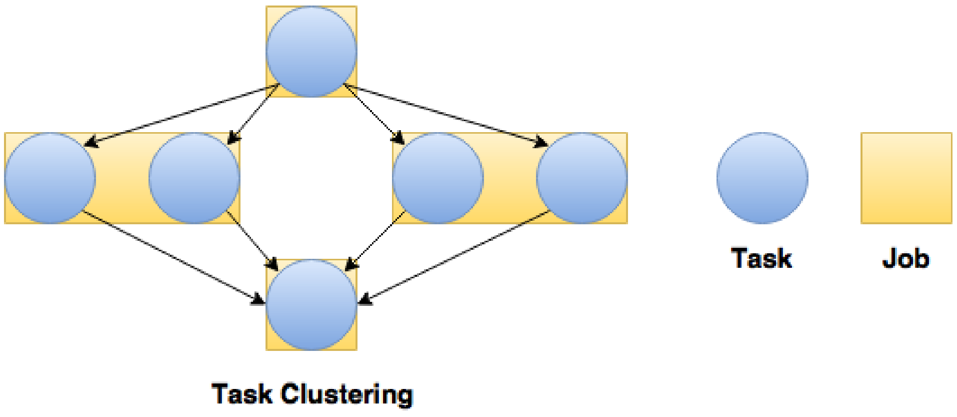


Figure 3.2: [6]

A job can be a sub-graph for parallel operations. Within those jobs individual tasks could fail which is named a task-failure. It is called a job-failure if all the tasks of a job fail. Failures in general have to be considered and observed. Because of failures taking impact on the performance a whole failure-monitoring of the workflow is the key for a stable system.

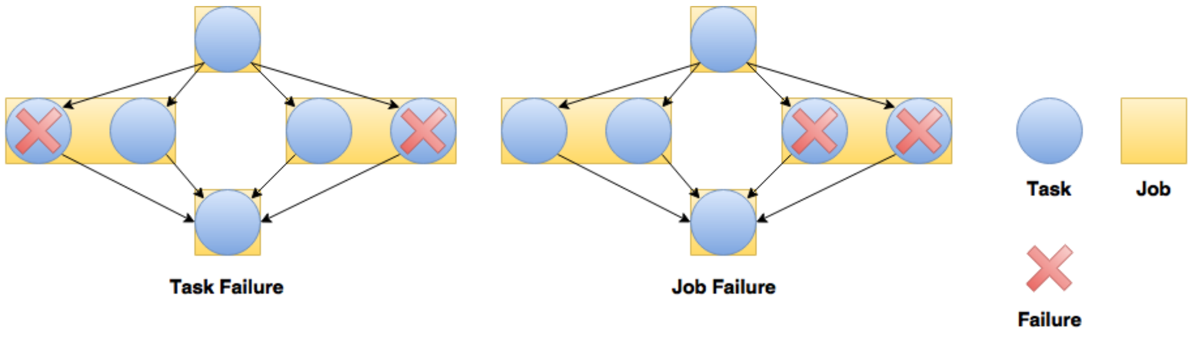


Figure 3.3: [6]

What are the reasons of failures?
The cause of a failure can be very different. Having a mistake in the program itself like wrong code, storage access, communication between processes and so on might be the root of the problem. Also hardware problems could lead to unmeant performance reducing. High Performance Computers have a inescapable fault rate because of the massive amount of computer components. There is a need for involving the "fault tolerance" into the workflow at every given point. How is it possible to make a guess about stochastic probabilities of something going wrong in our workflow graph - this is a question of simulating our workflow itself with tools like pegasus[2].

A small **example of an workflow** in context and how it could be arranged can be seen in the figure 3.4. The preprocessed data take its way through the I/O into the network of the high performance computer to be available to all resources for calculation. While the data is available on the cluster we could decide to share it to other scientists for similar calculations. At the next work step we run a set of experiments which can be part of a job or a task itself. While executing there is a need for verification, especially when a experiments takes days of calculation time. Therefore a visualization of a pre result could be made. At the end only the important results will be analyzed and stored.

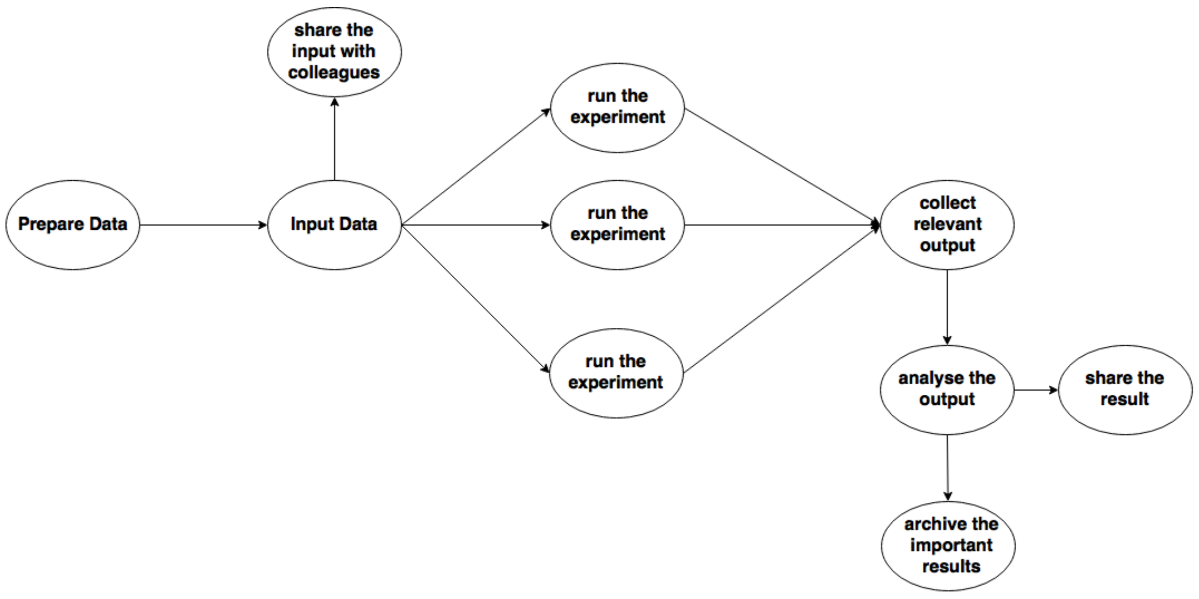


Figure 3.4: [5]

There are **symmetric** and **asymmetric** workflows. Both of them need to be handled in a different way. Unbalanced workflows can be justified by having a parallel-for loop which takes much longer than the other loops. In other words a branch of the DAG will take much longer to execute than others. This unbalanced workflows also require full-graph analysis to be scheduled properly. As a possible solution of such tasks with longer executing time and a longer branch following there is a need of rating them with higher priority than those with a smaller following branches.

With **workflow partitioning and Pegasus** it is possible to divide the workflow into sub-workflows which should be scheduled sequentially one after another. One approach is the 1-layer partitioning where each layer of a workflow is combined to one new workflow step as can be seen in the figure 3.5 below. Through the amount of nodes which are combined in each layer there are more tasks in a new workflow step a just-in-time scheduler can handle. It will consider a lot of more tasks than before. This sequences can be executed by one after another. So the workflow partitioning strategy is a way to use different algorithm than those with full-graph strategy.

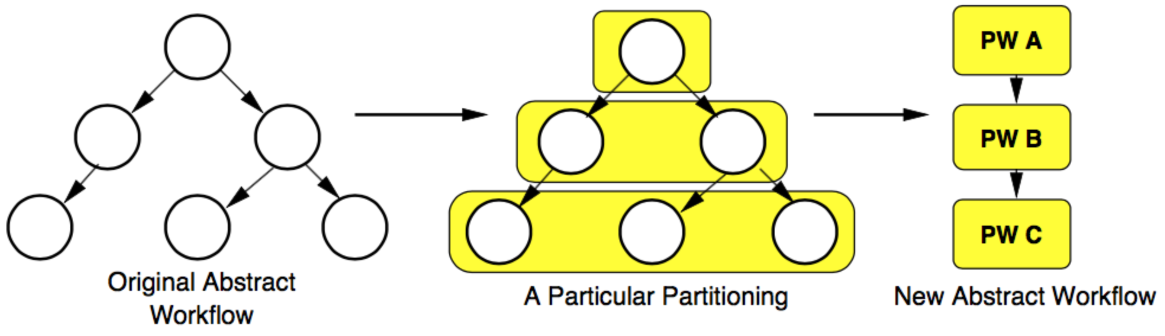


Figure 3.5: [4]

4 Scheduling

The intelligence behind every decision of mapping a certain task to a fitting resource when and in which order is a very important part of the whole progress with a lot of potential in increasing performance and work smarter with the available funds. Scheduling is a key to achieve performance. We can divide scheduling in two approaches which work very different before and while execution. There is a difference to be mentioned between execution time and scheduling time. Scheduling time is the period where the scheduler is preparing the schedule to be run. Because there are two varieties, the moment of scheduling distinguishes either it is static or dynamic.
The **static** idea is it to predict the runtime of a task and run it on the resource without any possibility to interrupt or reschedule the whole work. There is a strict list and order of executions in the schedule which will never change, even a unit has nothing to do. As already mentioned failures will need such thing as a restart of a task or job. The possibility to reschedule will cope with those problems much better than leaving the resource unused and the scheduling untouched.
The **dynamic** approach has the possibility to reschedule while runtime to optimize the resource allocation. Therefore the schedule control is provided with realtime information. If there is a job which is running longer than expected or a failure will take a task down, the system can make decisions based on the information. The look-ahead about what will happen in the next steps is an important factor the schedule control system might be aware of.
Some different scenarios which will have a big impact on working smarter with our scheduling system leads to optimization at every given instance. There is a approach as already mentioned in the chapter before to divide the workflow into sub-workflows which can be scheduled sequentially. Another clue is to treat the whole workflow and if there is a reason for rescheduling it will take impact at whole graph. There are two different examples of algorithm given to get a good clue of the computing steps.

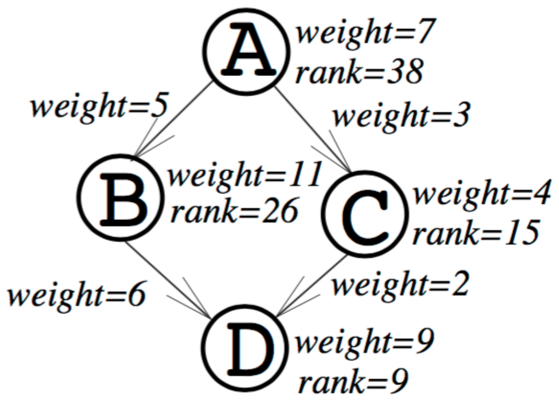
4.1 HEFT Algorithm

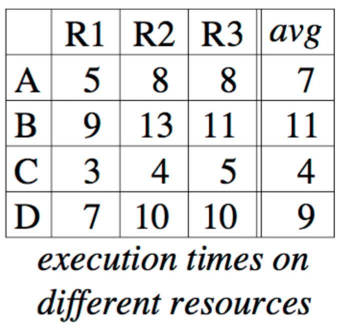
```
T - set of all tasks in the workflow

E - set of all dependencies

R - set of all available resources

(t1,t2) - dependence between task t1 and t2

time(t,r) - execution time of task t on resource r

time(e,r1,r2) - data transfer time of data between r1
   ↪ and r2

//Weight phase
for each t ∈ T do
    w(t) = $\frac{\sum r \in R \; time(e,r1,r2)}{R}$

for each e ∈ E do
    w(e) = $\frac{\sum r1,r2 \in R, r1 \neq r2 \; time(e,r1,r2)}{R(R-1)}$

//Ranking phase
take the max of sum (w(t),w(e)) from bottom to the top
ranking = sort(T,rank)

//Mapping phase
for i ranking downto 1 do
    t = ranking[i]
Find resource r ∈ R - min(finish_time(t,r))
Schedule t to r
Mark r as reserved until finish_time(t,r)
```

Listing 4.1: pseudocode 4

In this example the **HEFT Algorithm** starts with a set of all Tasks T, the set E as set of all dependencies and set R as set of all available resources. With a pair of tasks $t1, t2 \in T$ the dependencies are declared. Now there are two functions which are estimating the time of execution with task t on resource r and the time for data transfer from $r1$ to $r2$ with an link $e \in E$. The weight phase will do this for all tasks and dependencies. What we get are two tables where the average costs of time are the key for our later steps. In this tables it is easy to see which tasks fits best for which resource and how good the transfer rate from $t1$ to $t2$ shows up while booth of them are on different resources.

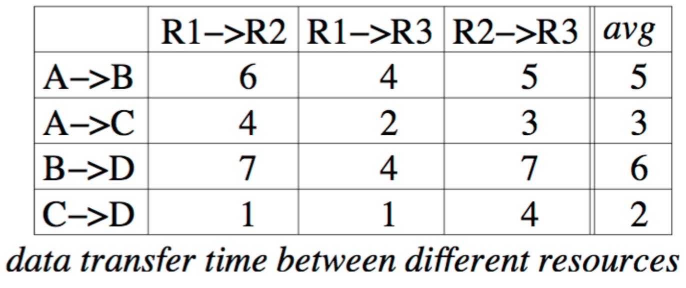
The ranking phase will take its way through the workflow from bottom to the top, while have a look at each node and edge. At each point we will take the average value and sum it up with the past points we already spectated. If there are two possible paths it will take the greater one. This max sum of weights the algorithm gets for every t and e will be the rank it has to set for every node.
Now it sorts the tasks t related to their rank. The mapping phase as last phase will take the tasks (sorted by their ranks) from top to the bottom, find a resource for them with the smallest finish time and schedule it to it TODO. The resource now will be marked as reserved until the task reaches its finishing time. This will lead to schedule the required ranks first and work through the procedure while using the best resources available for a given task. With full-graph scheduling HEFT is a pretty good algorithm for unbalanced workflows. HEFT will work pretty well for heterogenous environments while observe different kind of resources. The following example in figure 4.1 shows how it works.





	R1	R2	R3	*avg*
A	5	8	8	7
B	9	13	11	11
C	3	4	5	4
D	7	10	10	9

execution times on different resources



	R1–>R2	R1–>R3	R2–>R3	*avg*
A–>B	6	4	5	5
A–>C	4	2	3	3
B–>D	7	4	7	6
C–>D	1	1	4	2

data transfer time between different resources

Figure 4.1: [4]

4.2 Myopic Algorithm

A different and very inexpensive approach is the Myopic-algorithm makes only the decision for the local optima. The algorithm takes the set of all tasks T at each time instance and schedules the ones with the earliest start time first to a resource r with a minimum in finishing time. This will lead us to local optimized decisions while not watching the steps afterwards. In other words it will schedule the task to the best resource available which is declared as just-in-time strategy.

```
T - set of all tasks in the workflow
NT = T
while NT $\neq \emptyset$ do

    Find task t $\in$ NT with min(earliest_starting_time(t))

    Find resource r $\in$ R : min(finish_time(t,r))

    Schedule t to r

    Mark r as reserved until finish_time(t,r)

    NT = NT \ {t}
end while
```

Listing 4.2: pseudocode 4

One can say that there are expensive and cheaper algorithms and those who analyze the graph more than others. Myopic will work very good if a partitioning of the workflow has happened. HEFT will take a full graph (even a unbalanced one) and will create a result with good performance. Taken from the referred paper there are some interesting results which focus on the execution time prediction in the following figure 4.2.

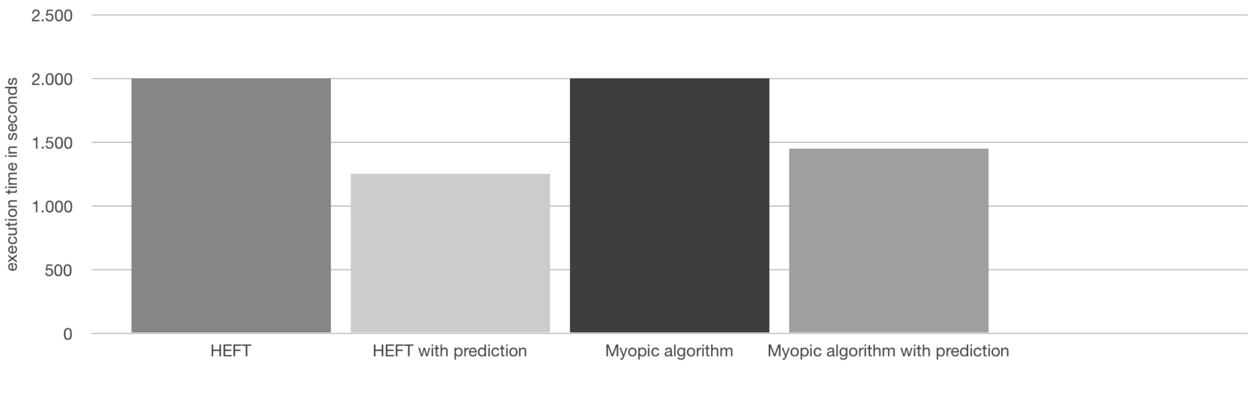


Figure 4.2: [4]

5 Conclusion

To fulfill the need for performance and save money through efficiency, modern research is forced to optimize the intelligence of algorithms, software, hardware, the mathematical approach and the best representation of the science subject the computer can work with. Workflows will give us a very good visual basis of understanding the steps and importance of of certain tasks to handle. Scheduling will give its best to get the finest benefit out of the resources. High performance computing is already a very important part of achieving knowledge in this digital era.

Sources

- Link: http://www.nas.nasa.gov/SC13/assets/images [1]
- Link: http://pegasus.isi.edu [2]
- Link: http://www.workflowsim.org [3]
- Link: http://www.sigmod.org/publications/sigmod-record [4]
- Link: https://wikis.nyu.edu/display [5]
- Link: http://de.slideshare.net/WeiweiChen/workflowsim-escience12-14674703 [6]
- Link: http://citeseerx.ist.psu.edu [7]