Introduction to energy-efficient resource management with SLURM and plugin development

Seminar and project report

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

Vorgelegt von:	Tobias Weßeler
E-Mail-Adresse:	tobias.wesseler@posteo.de
Matrikelnummer:	6536232
Studiengang:	B.Sc. Informatik

Erstgutachter:	M. Reza Heidari
Zweitgutachter:	Dr. Julian Kunkel

Betreuer:	M. Reza Heidari

Hamburg, den 01.01.2015

Abstract

This report is serving two purposes. Firstly, it is an introduction to slurm, plugins and energy profiling. Secondly it documents the efforts to solve a problem, that prevents users of slurm 14.11 from profiling multiple metrics at once. The steps undertaken to resolving this issue include installation of a test environment, compiling and installing slurm from source, configuring and running slurm and modifying slurm plugin source code to profile dummy data to eliminate potential error sources. The results show that there have been big changes to the source code of plugins between versions 14.11 and 15.08 of slurm, especially regarding the hdf5 plugin. In version 15 the issue was fixed, but it is not apparent as to how this was achieved. As of writing this report the next viable step to solve this issue would be to backport the hdf5 plugin from slurm 15 to slurm 14.

Contents

1. Foreword

In this chapter, an overview of the contents of this report will be given. It is very likely that, if you read this you will not be interested in the whole range of information available in this report. Instead you should read the following introduction. It will help you to select which sections of this report might be of interest to you.

This report is a combined report of the results of my seminar with the topic *Resource management with Slurm* and documentation of my project *Slurm plugin development*. So the first half of the report elaborates on the topic resource management and energy efficiency, while the second half focuses on setting up slurm, configuring and running it and also on modifying it, trying to debug an issue with the profiling plugin and the results thereof.

The content of first half consists mostly of my research in the course of the Seminar *Newest Trends in HPC* during the winter semester 15/16. If you are very new to the field of HPC (High Performance Computing) and do not know what slurm is, or if the words resource management and energy-efficiency have little meaning to you or if you want to grasp the concept of plugins, then i recommend to read the first part of this report.

The second half of this report, can be seen as the documentation of my part of the project *Evaluation of parallel computers* at the DKRZ. The ultimate goal of the project was the development of an energy profiling plugin for slurm, which was later reduced to debugging some related issues in slurm 14.

If you already have some background, if you have ever successfully used a console or terminal and if you are interested in setting up a minimal installation of slurm and or to undertake first steps of plugin development, then the second part is right for you. Also those of you interested in the issue of profiling multiple metrics in slurm version 14.11 should definitely take a look at the latter part.

2. Introduction - Part 1 / Seminar

This first part of the report will elaborate on my findings during the seminar NTHR1516. The overall task was to report on new trends in HPC. I choose 'energy-efficient resource management' as sub-topic. Because a lot of the participants of the seminar, including myself, are rather new to the field of HPC - some of them have never heard about slurm and other things, a big section of this first part explains basic concepts. Among these are for example resource management and plugins and energy measuring. All of these topics are very relevant to HPC, as modern systems grow bigger and consume more energy. Thus HPC is having a large economical and ecological impact. The primary motivation for using the resources as energy-efficient as possible is the money. Without funding there is no research and funding is always limited. In the end there will be a brief outlook on new functionality that will be introduced with slurm version 16.05. This version is planned to be released in may 2016 and brings some new features that could be used for more efficient resource management.

3. Terminology

In this chapter the terms resource manager and job scheduler will be explained. Also the reader will be given a brief overview of slurm as an example implementation for these tasks. Finally the concept of plugins is introduced. There are many different definitions and explanations of the following terms out there. The ones in the next section depict my understanding of the concepts.

3.1. Resource management

Resource management is usually implemented as a piece of software that handles several tasks:

- **Monitoring resources (nodes, cpu, ram, ...)**
  The resource manager has to keep track of which nodes in the cluster are being used and to what extend. That means he also keeps track of the resources within a node. He knows how many processors are being used and how much ram has been reserved. Also he has oversight of all available resources.

- **monitoring power consumption**
  Monitoring power consumption is different from the aforementionend resources because the power is not directly part of the system. The other resources are dedicated pieces of hardware. Power is the rate at which energy is consumed to operate this hardware. To monitor power consumption we need special means, which we will discuss in the next chapter.

- **handling resources**
  Turning unused nodes off or switching them to standby, to save energy, as well as the allocation of all other resources is an important aspect of resource management software. Handling resources means all the active parts of resource management, as opposed to the rather passive monitoring.

3.2. Job scheduling

A HPC-system is usually used by many users simultaneously. Thus a lot of jobs are sent to the system, requesting resources for their execution. Often more resources are requested than there are available. That is why some jobs have to wait. To bring the jobs in an order which uses resources as efficient as possible while not letting users wait

for to long takes an algorithm. The simplest implementation of such an algorithm would be to just serve the requests in the order in which they arrive. That would mean using a naive fifo-queue (fifo = first in, first out). This is not very efficient. Appendix B has some graphics that illustrate unusable resources in this case and how a simple backfill algorithm, can ensure more efficient use of resource. In short, a backfill algorithm allows jobs that arrive later, to jump ahead in the queue, if they are not delaying jobs that arrived earlier. See appendix B for a graphical demonstration.
The job scheduling algorithm usually needs information provided by the resource manager (not in the inefficient, naive case).

3.3. SLURM

The Simple Linux Utility for Resource Management is widely used in HPC software stacks to handle the tasks of resource management and job scheduling. The software is open source and free, while mainly being developed by the company schedmd which offers commercial, professional support. Slurm supports the use of plugins and has many features beyond resource management and job scheduling.
While SLURM is just one example, there are other pieces of software available that incorparate the tasks of job scheduler, resource manager or both. But SLURM is the one used in the DKRZ and "As of the June 2015 Top 500 computer list, Slurm was performing workload management on six of the ten most powerful computers in the world including the number 1 system, Tianhe-2 with 3,120,000 computing cores." [Sche]

3.4. Plugin

The word plugin describes a modular piece of software. Something that can extend an existing program with additional functionality or change it's behavior. Plugins usually cannot be used by themselves and depend on the software they were created for. They need to be *plugged into software*. To do so, the software the plugin is written for also needs to be designed to accept plugins. Therefore, software that supports plugins must define an interface to the programmers.
An easily understandable example for a plugin would be the comparison plugin for the text editor Notepad++. It was written to enable highlighting the differences between to text documents, directly within the editor.
Of course embedding an interface for plugins when designing a software enlarges the surface that can be attacked. It creates new vulnerabilities, because some people might try to write and distribute plugins with unwanted side-effects. One must always consider that plugins can be written by someone else entirely, than the original author of the software.w

4. Architecture of slurm

In this chapter, we will go into the design of slurm itself. Even though slurm has been designed to be lightweight, as to not consume to much resources, it offers a wide range of functionalities and thus has an extensive code base. So in a manner slurm is also a very large program. Large, but light-weight. Because of this, slurm consists of multiple programs that take care of different tasks. These programs are the slurm daemons. Daemons in linux are what services are for windows. They are processes that, once launched, are running quietly in the background, carrying out tasks or awaiting instructions. The can communicate with each other or take input from the user.

4.1. Daemons of slurm

The following daemons constitute the substance of the slurm:
The *d* at the end of the names, denotes that we deal with daemons.)

- **slurmd**
  The slurm daemon is the main daemon that is running on every node. It requires root privileges, because it needs to execute the (parallel) programs on behalf of the user.
- **slurmctld**
  The slurm control daemon is the daemon which the typical user interacts with mostly. It offers lot of user commands. Some of these are:
  - **sbatch** - schedule a job via script
  - **srun** - run a job interactively
  - **squeue** - show a list of waiting jobs
  - **scancel** - cancel a job
  - and more...

The other daemons also offer some user commands and some commands require more than one daemon running.
It is possible to have more than one slurm control daemon running at the same time. This way one can have a redundancy. The daemon was designed so that, if configured properly, the redundant daemon will act as an automatic fail over in case the first daemon becomes available.

- **slurmdbd**
  The slurm database daemon is an optional daemon. It manages the database which can hold configuration settings and accounting data. It can also be extended with a redundant daemon on a different machine which acts as an automatic fail over if properly configured.
  Another feature of the database daemon is, that it can attend multiple clusters, i.e. the slurm control daemons of entirely different clusters can have a shared database daemon. This is very interesting from an administrative perspective in a multi-cluster data facility.

- *slurmstepd*
  The slurm step daemon is kind of a special case, because unlike the other daemons, it is not always running. Instead it is spawned by the slurm daemon at the beginning of a step of a job and destroyed at the end.
  Among things it manages the Input and Output for job steps.

4.2. Slurm plugin architecture

Slurm supports plugins in different ways. One way is SPANK, the Slurm Plug-in Architecture for Node and job (K)control. Plugins written for SPANK (or just Spank-Plugins) work very similar to the above definition of plugin and to what one usually expects. But spank plugins were introduced to slurm later. In the context of this document, when we talk about plugins for slurm we are talking about a different kind of plugin.
These classic slurm plugins differ a little bit from plugins in the usual sense, because to develop them one needs part of the slurm source code to compile them. Also, more importantly, in some cases the slurm source code must be changed to accept and deal with these plugins. Therefore our plugins are not really separable pieces of software, but optional parts that can easily be turned on or off, when starting the daemons.
There are about 26 different categories for slurm plugins. Each category has its own interface that the developer of a plugin has to use [Schb]. The interfaces consist of sets of functions that need to be implemented by programmer. If we decide not to use a plugin for some categories in slurm, then we are acutally using a 'none' plugin. Those none plugins are skeleton plugins without functionality. Due to the strong entanglement of the plugin source code and the slurm source code, they are a necessity.

5. Energy-efficient resource management

This chapter deals with the topic of energy efficiency. It explain why it is a matter to be considered, introduces some basic concepts of energy measuring and takes a look at what means slurm provides to achieve energy-efficiency and where it can be improved.

5.1. Motivation

As computers get faster and clusters get bigger we are facing an obstacle - money. Computers require energy and energy costs money. Furthermore, consuming energy means consuming resources, which might have an environmental impact.
Modern high performance computing systems are consuming enormous amounts of energy. The average four person household is consuming 5200 kWh of electricity a year [Ene]. The DKRZ needs 17GWh. That is roughly 3000 times more. In 2011 the DKRZ had an electricity bill of nearly two million euros [DKR].
These two reasons should be more than enough motivation to try and optimise the energy-efficiency of our cluster and programs.

5.2. Measuring energy

Before we talk about measuring energy, i will list the measuring sizes and units that are being used.

Power - watts
The rate at which energy is consumed. It is measured in watts.
1 Watt = 1 VA (Volt × Ampere).

Energy - watt-hours
Energy is the product of power and time. If we are constantly consuming 1000 Watts over 1 hour, then we have consumed 1 kilowatthour of energy.

If we want to optimise, we need to analyse first. But before we can analyse our energy consumption we need a means of measuring how much energy we consume. A basic approach how to do that is described in the following:

We can use specialized hardware to measure the energy consumed by our computers. The simple way to do that, would be to have an analog power meters connected to the power cycle right before or behind the power supply units of our machines. Then we would need an analog-to-digital-converter as well as a physical connection to our hardware and an interface or **api** which we can read the data off of. For an conceptual illustration see figure ...

5.3. Profiling energy

What is profiling? It means to collect data and to analyse and interpret it, to gain additional information.

When we are doing an energy profile of running a certain program, we measure and record how fast energy is consumed at different points in time and try to match these points to the according passages in the source code as closely as possible. As a result we get an overview of how much certain parts of our software influence energy consumption. With the help of this data, we can then try and optimise our programs in terms of energy-efficiency. Another benefit is the ability to better plan ahead, when working together with power companies.

5.4. Plugins for slurm

Slurm already comes with a couple of plugins for collecting energy data, provided we have the required hardware and drivers installed. A third plugin is currently in development by (who/where again?) and the DKRZ.

5.4.1. IPMI

IPMI is short for Intelligent Platform Management Interface, a standardized Interface for remote monitoring and controlling computers. It is not to be confused with the *IPMI-plugin* for slurm. The slurm accounting and gathering energy plugin just adapted this abbreviation, even though it just utilizes part of the **IPMI**. In more detail: The *IPMI-plugin* retrieves power data from the **BMC** (Baseboard Management Controller) by utilizing the **IPMI** [Schf].

5.4.2. RAPL

RAPL is short for *Running Average Power Limit*. The name becomes clearer when we look at how RAPL works. "RAPL is not an analog power meter, but rather uses a software power model." [Zer] This quote alone gives some ideas on what is happening here. Short version is this: Hardware performance counters provide information at set intervals that is combined with I/O models to produce an estimation of consumed energy in the

time period in between. Test by the Intel Corporation have shown that the estimates are close to real values.

5.4.3. HDEEM (in development)

HDEEM is short for High Definition Energy Efficiecy Monitoring. The new hdeem-plugin will be using the HDEEM infrastructure to provide highly improved energy profiling capabilities.
At the moment, all available plugins for profiling the energy use, have a glaring limitation - the sampling frequency. In the slurm configuration files, the smallest possible interval for sampling energy consumption is one second. The problem with this interval is that millions of operations are processed by computers within this one second. So the resolution we have for sampling the energy is much lower than what we could use for profiling our software.
The HDEEM architecture promises to be able to give up to 1000 samples per second. That is still a low number compared to the operations per second but far better than what we have now. [**?**]

5.4.4. HDF5 - storing the collected data

This plugin is of a different category than those above. While the other plugins deal with gathering the actual data from sensors, the HDF5-plugin handles the storage of this data during profiling. It is currently the only profiling plugin, that comes with slurm.
HDF5 is a very efficient file format which supports parallel I/O as well as storing custom datatypes. It is optimised to store large amounts of complex data and is thus well suited for HPC. [**?**]

5.5. Outlook

HDEEM seems a promising concept in the field of energy profiling and more methods and techniques will follow. Furthermore in the upcoming version 16.05 of slurm there will be additional means provided that can be used to optimise resource usage.
Most notably are support for *asymmetric resource allocation* and *MPMD programming*. Asymmetric resource allocation allows the user to specify what kinds of hardware need to be allocated for his job in more detail than a certain number of nodes. It will be possible to specify the required amount CPUs, GPUs, RAM and more per node. It is unclear how well the MPMD approach will be recepted, but the fact that they are developing it itself, shows that there must be some interest. MPMD mean Multiple-Program-Multiple-Data. So we can run different executables that communicate with each other, contrary to the common SPMD-approach (Single-Program-Multiple-Data) where only one program is written. In SPMD conditional structures are usually used in conjuncture with ranks to determine which cpus or nodes are handling which tasks.
Figure ... shows a conceptual illustration of the mpmd-approach.

6. Conclusion - Part 1 / Seminar

It seems that slurm will be a good choice for the HPC software stack for the near future. Apart from the already acquired expertise of the DKRZ personnel the software is widely used and recognized by top HPC centers and actively being developed. New promising features are introduced with each version. In the future the will be more possibilities in assigning resources to programs and jobs and with the hdeem plugin energy profiling will yield better results.
It is important that the people working with the software read up on the available features, try to keep their software up-to-date and to utilize the offered features to encounter the financial and environmental challenges of high energy-consumption.

7. Introduction - Part 2 / Project

This second part of the report is the report of my bachelor's project here at the dkrz. It started out with the goal of helping to implement the HDEEM plugin. Soon it became clear that this goal was to vague and also some issues came up, that needed to be dealt with first.
One of these issues is, that in slurm version 14.11, the version currently being used on mistral, data recorded into hdf5-files is corrupted. So the new target became to tackle this issue and to eliminate possible error sources.
Thus, this part describes and documents my efforts in helping with this issue. Because i had nearly no experience with linux beforehand, the resulting documentation, also contains a quick tutorial to setting up debian and a minimal installation of slurm.

The configuration data of the test machine can be found in appendix A.

8. Setting up the test environment

In this chapter, I will demonstrate the steps i had to undertake to install debian jessie (debian 8.3) which served as the operating system under which i conducted all the following experiments.
Afterwards i will describe how to acquire, compile, install, configure and run slurm.

8.1. Installing the operating system

Installing the operating system is very straightforward. Modern installation routines offer simple, graphical user interfaces with acceptable default options already set. The steps i used to install debian jessie were the following:

- **Download Debian**
  There are several options to download and install slurm. My setup was the following: The bare machine to install debian on, with a working ethernet, internet connection, as well as my notebook with an installation of MS Windows. I used my notebook to download a **small installation image** of debian jessie from *https://www.debian.org/distrib/* [Deb] using the link labeled "64-bit PC netinst iso".

- **Download Win32 Disk Imager**
  Because i was working with windows, i had to find a means to create a bootable device with the image downloaded during the first step, to deploy my linux environment. I choose to download the utility named **Win32 Disk Imager** from *https://sourceforge.net/projects/win32diskimager/* [Sou], because it does exactly that, in a very simple way. The user interface is absolutely self explanatory.

- **Creating the image and installing Debian**
  The following was just a matter of clicking and waiting. Firstly, i used the tool to put the image on a flash drive and make it bootable at the same time. Secondly, i plugged it into my bare machine, on bootup i pressed the appropriate key to choose the boot medium (F2 in this case, the key is shown at the startup screen of the motherboard manufacturer) and selected the usb drive. Once the installation routine opened up, i choose the *Automated GUI Installation*. For user name and password of the test machine, please see appendix a.

8.2. Installing slurm

Once the operating system was set up, i continued with the installation of slurm. The next few subsections document that process.

8.2.1. Prerequisites

Before we deal with the installation of slurm itself, we need to make sure we have all the prerequistes or dependencies.

sudo - optional, recommended

For a lot of the installation process we need root priviliges. The options are twofold: 1. we could just use the root account, or 2. we install the sudo command which enables our non-root users to be added to the sudo group and to execute certain commands with elevated privileges, when needed. I recommend the second method.

Munge

Slurm has one major dependency which is the need of an authentication plugin. The default case is that munge is being used for that. Munge in turn has a dependency on Libgcrypt or OpenSSL. Since we have a bare system, we need to install all of it. Just run the following commands, and hopefully, you should be fine:

- sudo apt-get install libgcrypt11-dev

  or

- sudo apt-get install libgcrypt libgcrypt-devel gcc-c++

  and

- sudo apt-get install libmunge-dev

- sudo apt-get install munge
  Also one should secure the munge installation as described in the offical guide [mun].

- Create a secret key
  $ dd if=/dev/random bs=1 count=1024 >/etc/munge/munge.key

- Startup the munge daemon (some errors that can be encountered, might be overridden by the argument *–force*)
  $ /usr/sbin/munged –force

autotools - optional

Installing the autotoolchain is useful when adding new kinds of plugins to the source code, because it makes the difficult process of amending the makefiles of the slurm source code very easy by allowing you to generate them through the autogen.sh-script.
The steps required can be found here *http://slurm.schedmd.com/add.html* [Scha].
Later i reverted to just changing existing plugins, because it saved a lot of time and the existing plugins weren't needed for my experiments.

8.2.2. Downloading

The current stable version of slurm can be downloaded here:
http://www.schedmd.com/#repos [Schc]
Older Versions are available on a subpage:
http://www.schedmd.com/#archives [Schd]
To install either of the versions, first download the source code and then extract it to an appropriate directory where you will manage it. It is recommended to use some versioning system when modifying the source, so think about where to put your source code and consider initialising a git repository in said directory.

8.2.3. Compiling and installing

Once you are set with your source code, before you can install slurm you need to compile the source to create the binaries. The installation process just copies the compiled binaries to appropriate folders on the operating system to integrate the software. To compile and install execute the following commands:

- $ cd /...path.to.source.../
- $ sudo ./configure –prefix=/usr/local –sysconfdir=/usr/local/etc –enable-debug
  You might want to change these, if you are planning to install multiple versions of slurm simultaneously. The prefix directory is where the binaries will be placed and the sysconfdir is where you will put your slurm config files. I just like having the option to debug as much as possible, which is why i added the 3rd parameter.
- $ sudo make
- $ sudo make install
- If you are getting any errors or warnings, or something does not work, the config.log (generated by ./configure ...) is always a good place to start looking. Check it with for example with:
  cat config.log | grep error

8.3. Configuring slurm

Now that we have actually installed slurm on our system, we will not be able to run it just yet. First we have to add a slurm.conf to the sysconfdir. Thankfully, the slurm developers have provided an html-template that will generate such a file for us, with very little information needed on our part.
Navigate to the slurm source code directory and open one of the follwing files with your browser:

- $ firefox doc/html/configurator.html

  or

- $ firefox doc/html/configurator.easy.html

The second choice will query much less parameters and use more default values. It is the one i used. The config values i used, are the following:

- ControlMachine: [name.of.my.machine] //you can get that with the hostname command

- ControlAddr: 127.0.0.1
- NodeName: [name.of.my.machine]
- NodeAddr: 127.0.0.1
- CPUs, Sockets, CoresPerSocket, ThreadsPerCode //got that from the command lscpu
- For the rest i just left default values for now. I later edited the config files manually when setting up the profiling.

After exporting the config file, you will need to move it to the sysconfdir specified during the installation of slurm. Default is: /usr/local/etc/. Also look into the file and check the two or three bottom lines with the configuration of the node. Since for now we are having a minimal cluster with only one node, we only need to check this file. Make sure the machine name is there and not followed by an unnecessary index. You might also need to edit the default location of your mail program. Otherwise the slurm daemons might throw some errors. But you can just leave it for now, and take care of it, when it happens. It really depends on your operating system.

8.4. Running slurm

Now that we have set everything up, we need to startup all the daemons. Afterwards we can test our installation by running a simple interactive job.

1. Start out by ensuring that the munge daemon is running. You can use the command *munge -n* to do so. If the output is a hex-key, then munge is working fine - if not, use system commands to see whether the daemon is running or not. If not try starting it by calling the executable *munged*. When encountering error, check if all the directory permissions were set correctly, or - for the sake of testing - just use the -f parameter. It will force munge to treat some errors just as warnings.

2. Next start up the *slurmd* and *slurmctld* daemons, by calling those executables. You can also use the -D parameter, to have them running in the foreground and up to 3 or 4 -v parameters to set the level of verbosity. The -c parameter can be used to make a clean start of the daemons in case it gets stuck or something.

3. Once you have all the daemons up and running use the srun command to schedule an interactive job.
   Just try: *srun hostname* for instance. It should take a second and then the name of your machine should be shown on the console. If it does, it means the command was successfully scheduled and run by slurm - Congratulations. If not - do not despair. Instead start the slurm daemons with *-Dvvvvc* and use the information given to figure out whats wrong.
   Again - another good place to look is the config.log in your slurm source directories. Occasionally it happens that some dependencies are missing, when compiling the source code.

9. Profiling data

In this chapter we will talk about how to use the profiling plugins that come with slurm and how to modify them to write dummy data, in case we do not have the required hardware. We will also deal with the issue encountered at the DKRZ which is closely related to profiling data.

9.1. The problem

The data that we want to collect needs to be stored somewhere. For that purpose, a plugin with the name hdf exists, that constitutes the interface between slurm and hdf-data-files. Unfortunately, the installation of slurm 14.11. on the cluster Mistral at the DKRZ does not seem to be working correct in this manner. When we configure slurm (via the config file) to gather energy data or other data, then only the first sample is written into the hdf-file. For all subsequent samples there are null-, zero- or uninstantiated values. See Appendix D for some corrupted example files. Another problem, which was not confirmed yet, is that in slurm 15.08 the lustre/filesystem data is missing, when using the option *–profile=all*. While not being confirmed, we do have a suspicion towards the cause of this. Refer to the section *Results of profiling dummy data* for more information.

9.2. The approach

So before we could approach the implementation of the HDEEM plugin, we have to solve the issue regarding the hdf5-plugin. Our approach was the following: Modify some existing plugins, by taking out the interfaces to hardware and drivers and replacing them with hardcoded dummy data. This way we can find out wether the problem lies with the hdf5 plugin. If it does not, we would at least have ruled out some potential causes for the errors. By using dummy data we can see wether the hdf plugins writes, what it is supposed to.

9.3. Modification of plugins

For profiling dummy data, we needed at least two plugins that were also used in the production environment, to be able to analyse the different cases:

- profile data from a single plugin

- profile data from multiple plugins, listed one by one (–profile=energy,lustre)
- profile data from all (configured) plugins (–profile=all)

9.4. Results of profiling dummy data

The modification i made to the plugins, so that they would write dummy data, are shown in appendix C. The config files for the runs of the follwing table are also listed in appendix C. There results of the profiling are as follows:

	Slurm 14.11	Slurm 15.08
Production machine		
#sbatch –profile=energy	data: corrupted	???
#sbatch –profile=network	data: corrupted	???
#sbatch –profile=energy,network	data: corrupted	???
Test machine		
#srun –profile=energy,lustre sleep 181	data: corrupted	data: ok
#srun –profile=all sleep 181	data: corrupted	data: ok
#srun –profile=energy sleep 181	data: ok	data: ok
#srun –profile=lustre sleep 181	data: ok	data: ok

So when i tried to collect data of multiple metrics simultaneuously in Slurm version 14.11 the collected data became corrupted on its way to the hdf-file. This suggests a bug in the hdf-plugin or in hdf5 itself. The next step would be to confirm or rule out, wether the problem lies with the plugin. This can be done by debugging the hdf-plugin or by trying to backport the hdf-plugin from slurm version 15.11 where the problem does not appear.

Trying to figure out the issue with the hdf-plugin, i also had a look at some of the source files and noticed some changes, when comparing the slurm 14 with the slurm 15 files. The listed files (plus the according header-files) might be worth taking a look at,

when trying to debug the issue or backporting the plugin.

Also worth noting: In slurm 15 the multiple metrics are recorded correctly even with different sampling frequencies.

src/common/slurm_acct_gather_profile.c
Some changes regarding the handling of datasets.

src/common/slurm_jobacct_gather.c
Some changes; not sure if important to profiling or only accounting

src/plugins/acct_gather_profile/hdf5/hdf5_api.c
Much shorter (62kB $\rightarrow$ 6kB) and more generic approach to handling the storage of data samples in an hdf5 file. Makes adding new metrics to the plugin-code easier and results in more space efficient, and clearer hdf5-files.

src/plugins/acct_gather_profile/hdf5/acct_gather_profile_hdf5.c
Some changes to account for the changes in hdf5_api.c

There are probably more files that need to be modified for a successful backport of the hdf5-plugin from slurm 15 to slurm 14, but these are the ones i identiefied so far, that i think are crucial to the process.
While studying the source code i also stumbled upon the following *semantical bug*:

src/plugins/acct_gather_filesystem/lustre/acct_gather_filesystem_lustre.c
In line 308 the dataset *NETWORK* is created where it should actually be named *FILESYSTEM* or *LUSTRE*. This could cause some irritation when studying the hdf5-dumps while looking for the data set. Also it might cause problems when trying to profile filesystem data as well as network data, when the plugin tries to put the same label on both data sets, although i have not confirmed this, yet.
Furthermore i noted that there is an inconsistency in the naming. Each plugin is activated by its category name (e.g. rapl-plugin is an energy-plugin, thus being used by –profile=energy), except for the filesystem plugin lustre (i.e. the plugin lustre is a filesystem-plugin and should be made available via –profile=filesystem, not as it is the case via –profile=lustre).

10. Conclusion - Part 2 / Project

The development of the HDEEM plugin is hindered by the issues regarding the hdf5-plugin, which need to be resolved first. This second part of the report documented the setup of the testmachine, that was used to eliminate some potential error sources and laid the ground work for additional research.
It has been shown that there are some bugs in the source code of slurm version 14.11 and some semantical bugs that were carried over to slurm 15.11 which might also turn out to be real bugs. Maintaining a code base as large as the one of slurm is not a light task, therefore it is no surprise that bugs are being found.
Still the biggest part of the software is structured clearly and the more one delves into the source code, the easier it gets to find the right files and to understand how things are done.
It would probably pay off to add some more documentation to some of the source files.
Regarding the plugin, there is hope that the hdf5 issue can be resolved in the future and that the hdeem plugin can be implemented.

Bibliography

[Deb] Debian. https://www.debian.org/distrib/. Accessed: 2016-04-18.

[DKR] Steigerung der energie-effizienz am dkrz. https://www.dkrz.de/about/kontakt/presse/aktuell/archiv-2013/energieeffizienz. Accessed: 2016-04-14.

[Ene] Energieheld.de. http://www.energieheld.de/blog/energieverbrauch-eines-wohnhauses/. Accessed: 2016-04-14.

[mun] Munge github. https://github.com/dun/munge/wiki/Installation-Guide. Accessed: 2016-04-18.

[Scha] Adding files or plugins to slurm. http://slurm.schedmd.com/add.html. Accessed: 2016-04-18.

[Schb] Documentation of version 15.08. http://slurm.schedmd.com/documentation.html. Accessed: 2016-04-14.

[Schc] Download the latest stable version of slurm. http://www.schedmd.com/#repos. Accessed: 2016-04-18.

[Schd] Here is a list of past versions of slurm. http://www.schedmd.com/#archives. Accessed: 2016-04-18.

[Sche] Slurm commercial support and development. http://www.schedmd.com/#index. Accessed: 2016-04-14.

[Schf] Slurm energy accounting plugin api. http://slurm.schedmd.com/acct_gather_energy_plugins.html. Accessed: 2016-04-14.

[Sou] Sourceforge. https://sourceforge.net/projects/win32diskimager/. Accessed: 2016-04-18.

[Zer] Rapl use cases. https://01.org/blogs/tlcounts/2014/running-average-power-limit-%E2%80%93-rapl. Accessed: 2016-04-14.

Appendices

A. Test Environment Access Information

Test machine login information for debian operating system:
username - password
root - bapple
overlord - bapple

Slurm config files:
slurm.conf

```
1  # slurm.conf file generated by configurator easy.html.
2  # Put this file on all nodes of your cluster.
3  # See the slurm.conf man page for more information.
4
5  ControlMachine=nerge-desktop
6
7  MailProg=/usr/bin/mail
8  MpiDefault=none
9  #MpiParams=ports=#-#
10 ProctrackType=proctrack/pgid
11 ReturnToService=1
12 SlurmctldPidFile=/var/run/slurmctld14.pid
13 SlurmctldPort=1417
14 SlurmdPidFile=/var/run/slurmd14.pid
15 SlurmdPort=1418
16 SlurmdSpoolDir=/home/overlord/slurm-14/slurmd
17 SlurmUser=overlord
18 StateSaveLocation=/home/overlord/slurm-14/slurmctld
19 SwitchType=switch/none
20 TaskPlugin=task/none
21
22 # SCHEDULING
23 FastSchedule=1
24 SchedulerType=sched/backfill
25 SelectType=select/linear
26
27 ClusterName=cluster14
```

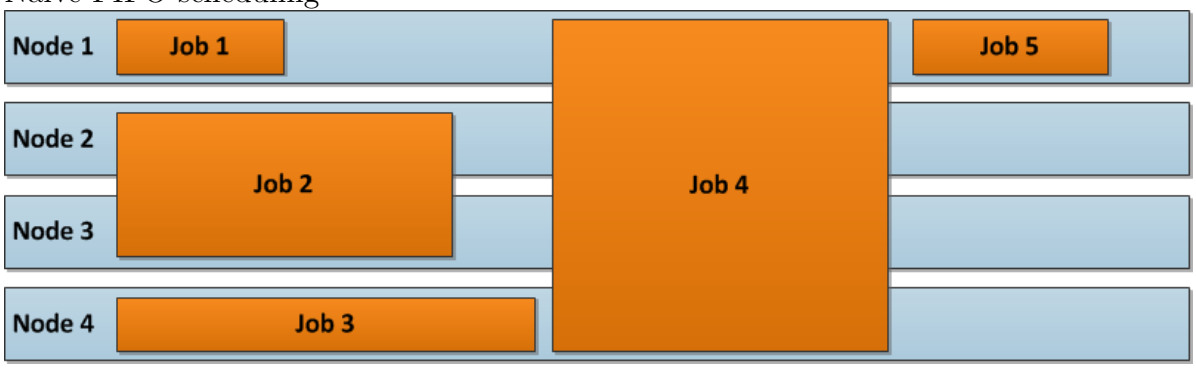
```
28
29 JobAcctGatherFrequency=energy=30,task=30,network=30,filesystem=30
   ↪ #set desired accounting frequencies
30 AcctGatherFilesystemType=acct_gather_filesystem/lustre
   ↪ #comment this line out if desired
31 AcctGatherEnergyType=acct_gather_energy/rapl #or ipmi
   ↪ #comment this line out if desired
32 AcctGatherNodeFreq=30
33 AcctGatherProfileType=acct_gather_profile/hdf5
34
35 # COMPUTE NODES
36 NodeName=nerge-desktop CPUs=1 Sockets=1 CoresPerSocket=4
   ↪ ThreadsPerCore=2 State=UNKNOWN
37 PartitionName=debug Nodes=nerge-desktop Default=YES
   ↪ MaxTime=INFINITE State=UP
```

acct_gather.conf

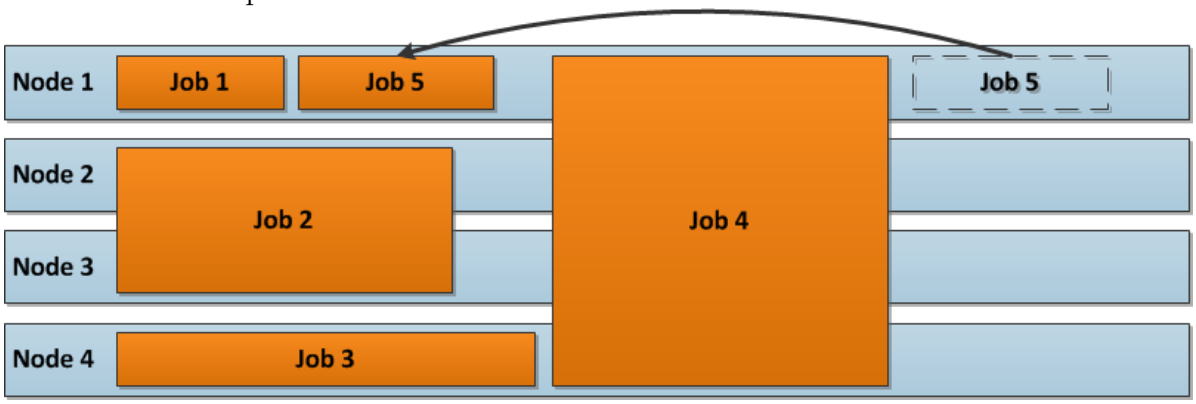
```
1 ProfileHDF5Dir=/home/overlord/slurm-14/profile_data/ #path
   ↪ where the profiled data is recorded, needs to be on a
   ↪ shared directory accessible to all nodes
2 ProfileHDF5Default=None
```

B. Scheduling example

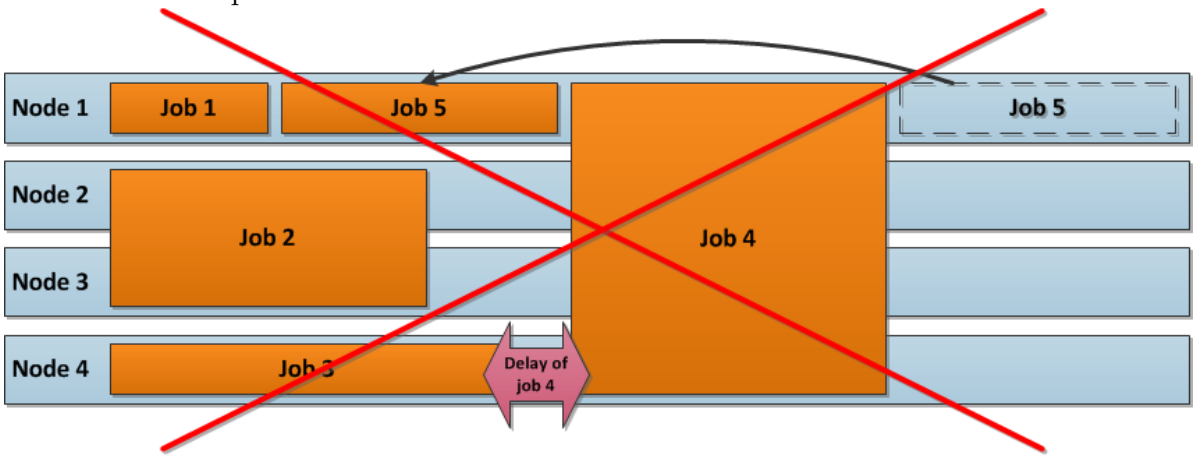
Naive FIFO-scheduling



Allowed backfill operation



Invalid backfill operation



C. Modification of plugins

src/plugins/acct_gather_energy/rapl/acct_gather_energy_rapl.c

```
1  static int counter = 1; //added this for dummy data
   ↪ generation
2
3  extern int fini(void)
4  {
5      int i;
6
7      if (!_run_in_daemon())
8          return SLURM_SUCCESS;
9
10     //for (i = 0; i < nb_pkg; i++) {
11     //  if (pkg_fd[i] != -1) {
12     //      close(pkg_fd[i]);
13     //      pkg_fd[i] = -1;
14     //  }
15     //}
16
17     acct_gather_energy_destroy(local_energy);
18     local_energy = NULL;
19     return SLURM_SUCCESS;
20 }
21
22 extern void acct_gather_energy_p_conf_set(s_p_hashtbl_t
   ↪ *tbl)
23 {
24     int i;
25     uint64_t result;
26     if (!_run_in_daemon())
27         return;
28     //_hardware();
29     //for (i = 0; i < nb_pkg; i++)
30     //  pkg_fd[i] = _open_msr(pkg2cpu[i]);
31     local_energy = acct_gather_energy_alloc();
32     result = 42;//_read_msr(pkg_fd[0], MSR_RAPL_POWER_UNIT);
33     if (result == 0)
```

```
34          local_energy->current_watts = NO_VAL;
35      debug("%s loaded", plugin_name);
36      return;
37  }
38
39  static void _get_joules_task(acct_gather_energy_t *energy)
40  {
41      int i;
42      double energy_units;
43      uint64_t result;
44      double ret;
45
46      ret = counter;
47      if (energy->consumed_energy) {
48          uint16_t node_freq;
49          //MAKE UP SOME DUMMY DATA THAT IS RECOGNIZABLE
50          energy->consumed_energy =
51              (uint64_t)ret - energy->base_consumed_energy;
52          energy->current_watts =
53              (uint32_t)ret -
                  ↪ energy->previous_consumed_energy;
54          node_freq = 0;//(double)1000;// + 100 *
                ↪ (random()%5+1);
55          if (node_freq)  /* Prevent divide by zero */
56              energy->current_watts /= (float)node_freq;
57      } else {
58          energy->consumed_energy = 1;
59          energy->base_consumed_energy = (uint64_t)ret;
60      }
61      energy->previous_consumed_energy = (uint64_t)ret;
62      counter *= 2;
63      energy->consumed_energy = (uint64_t)ret;
64      energy->current_watts = (uint64_t)ret*10;
65      energy->poll_time = time(NULL);
66      return;
67      //...
68  }
```

src/plugins/acct_gather_energy/rapl/acct_gather_filesystem_rapl.c

```
1   static int _read_lustre_counters(void)
2   {
3       static bool first2 = true;
4       if (first2) {
5           first2 = false;
```

```
6           lustre_se.all_lustre_write_bytes = 0;
7           lustre_se.all_lustre_read_bytes = 0;
8           lustre_se.all_lustre_nb_writes = 0;
9           lustre_se.all_lustre_nb_reads = 0;
10      }else{
11          lustre_se.all_lustre_write_bytes += 101; //DUMMY
              ↪ DATA
12          lustre_se.all_lustre_read_bytes += 2002;
13          lustre_se.all_lustre_nb_writes += 30003;
14          lustre_se.all_lustre_nb_reads += 400004;
15          debug3("%s: all_lustre_write_bytes %"PRIu64" "
16                 "all_lustre_read_bytes %"PRIu64"",
17                 __func__, lustre_se.all_lustre_write_bytes,
18                 lustre_se.all_lustre_read_bytes);
19          debug3("%s: all_lustre_nb_writes %"PRIu64" "
20                 "all_lustre_nb_reads %"PRIu64"",
21                 __func__, lustre_se.all_lustre_nb_writes,
22                 lustre_se.all_lustre_nb_reads);
23      }
24      lustre_se.last_update_time = lustre_se.update_time;
25      lustre_se.update_time = time(NULL);
26      return SLURM_SUCCESS;
27  }
28
29  static int _update_node_filesystem(void)
30  {
31      static acct_filesystem_data_t fls;
32      static acct_filesystem_data_t current;
33      static acct_filesystem_data_t previous;
34      static bool first = true;
35      int cc;
36
37      slurm_mutex_lock(&lustre_lock);
38      cc = _read_lustre_counters();
39      if (cc != SLURM_SUCCESS) {
40          error("%s: Cannot read lustre counters", __func__);
41          slurm_mutex_unlock(&lustre_lock);
42          return SLURM_FAILURE;
43      }
44      if (first) {
45          /* First time initialize the counters and return.
46           */
47          previous.reads = lustre_se.all_lustre_nb_reads;
48          previous.writes = lustre_se.all_lustre_nb_writes;
```

```
49          previous.read_size
50              = (double)lustre_se.all_lustre_read_bytes;
51          previous.write_size
52              = (double)lustre_se.all_lustre_write_bytes;
53
54          first = false;
55          memset(&lustre_se, 0, sizeof(lustre_sens_t));
56          slurm_mutex_unlock(&lustre_lock);
57
58          return SLURM_SUCCESS;
59      }
60
61      /* Compute the current values read from all lustre-xxxx
62       * directories
63       */
64      current.reads = lustre_se.all_lustre_nb_reads;
65      current.writes = lustre_se.all_lustre_nb_writes;
66      current.read_size =
          ↪ (double)lustre_se.all_lustre_read_bytes;
67      current.write_size =
          ↪ (double)lustre_se.all_lustre_write_bytes;
68
69      /* record sample */
70      fls.reads = current.reads - previous.reads;
71      fls.writes = (current.read_size -
          ↪ previous.read_size);// /      (1 << 20);
72      fls.read_size = current.writes - previous.writes;
73      fls.write_size = (current.write_size -
          ↪ previous.write_size);// /        (1 << 20);
74
75      acct_gather_profile_g_add_sample_data(ACCT_GATHER_PROFILE_LUSTRE,
          ↪ &fls);
76      /* Save current as previous and clean up the working
77       * data structure.
78       */
79      memcpy(&previous, &current,
          ↪ sizeof(acct_filesystem_data_t));
80      //memset(&lustre_se, 0, sizeof(lustre_sens_t));
81      info("%s: num reads %"PRIu64" nums write %"PRIu64" "
82           "read %f MB wrote %f MB",
83           __func__, fls.reads, fls.writes, fls.read_size,
               ↪ fls.write_size);
84      slurm_mutex_unlock(&lustre_lock);
85      return SLURM_SUCCESS;
```

```
86  }
87  extern int acct_gather_filesystem_p_node_update(void)
88  {
89      //if (_run_in_daemon() && (_check_lustre_fs() ==
        ↪ SLURM_SUCCESS)) //No real lustre calls wanted
90          _update_node_filesystem();
91      return SLURM_SUCCESS;
92  }
93
94
95  extern void acct_gather_filesystem_p_conf_set(s_p_hashtbl_t
    ↪ *tbl)
96  {
97      //if (!_run_in_daemon())
98      //  return;
99      debug("%s loaded", plugin_name);
100 }
```

D. Corrupted hdf5-dump example

Provided by Hendryk Bockelmann. This excerpt is the result of a batch job in slurm 14.11 on mistral. Only the first sample in each profiling-file contains data. Only four samples are taken from the file, because otherwise it would clog this document.

```
435             GROUP "Energy_0000000001" {
436                ATTRIBUTE "Data Type" {
437                   DATATYPE  H5T_STRING {
438                      STRSIZE 6;
439                      STRPAD H5T_STR_NULLTERM;
440                      CSET H5T_CSET_ASCII;
441                      CTYPE H5T_C_S1;
442                   }
443                   DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
444                   DATA {
445                   (0): "Energy"
446                   }
447                }
448                ATTRIBUTE "Subdata Type" {
449                   DATATYPE  H5T_STRING {
450                      STRSIZE 6;
451                      STRPAD H5T_STR_NULLTERM;
452                      CSET H5T_CSET_ASCII;
453                      CTYPE H5T_C_S1;
454                   }
455                   DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
456                   DATA {
457                   (0): "Sample"
458                   }
459                }
460                DATASET "Energy_0000000001 Data" {
461                   DATATYPE  H5T_COMPOUND {
462                      H5T_STRING {
463                         STRSIZE 24;
464                         STRPAD H5T_STR_NULLTERM;
465                         CSET H5T_CSET_ASCII;
466                         CTYPE H5T_C_S1;
```

```
                    } "Date_Time";
                    H5T_STD_U64LE "Time";
                    H5T_STD_U64LE "Power";
                    H5T_STD_U64LE "CPU_Frequency";
                }
                DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
                DATA {
                (0): {
                      "",
                      1455807780,
                      158,
                      1
                    }
                }
             }
          }
          GROUP "Energy_0000000004" {
             ATTRIBUTE "Data Type" {
                DATATYPE  H5T_STRING {
                   STRSIZE 6;
                   STRPAD H5T_STR_NULLTERM;
                   CSET H5T_CSET_ASCII;
                   CTYPE H5T_C_S1;
                }
                DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
                DATA {
                (0): "Energy"
                }
             }
             ATTRIBUTE "Subdata Type" {
                DATATYPE  H5T_STRING {
                   STRSIZE 6;
                   STRPAD H5T_STR_NULLTERM;
                   CSET H5T_CSET_ASCII;
                   CTYPE H5T_C_S1;
                }
                DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
                DATA {
                (0): "Sample"
                }
             }
             DATASET "Energy_0000000004 Data" {
                DATATYPE  H5T_COMPOUND {
                   H5T_STRING {
```

```
                        STRSIZE 24;
                        STRPAD H5T_STR_NULLTERM;
                        CSET H5T_CSET_ASCII;
                        CTYPE H5T_C_S1;
                     } "Date_Time";
                     H5T_STD_U64LE "Time";
                     H5T_STD_U64LE "Power";
                     H5T_STD_U64LE "CPU_Frequency";
                  }
                  DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
                  DATA {
                  (0): {
                        "",
                        0,
                        0,
                        0
                     }
                  }
               }
            }
            GROUP "Energy_0000000006" {
               ATTRIBUTE "Data Type" {
                  DATATYPE  H5T_STRING {
                     STRSIZE 6;
                     STRPAD H5T_STR_NULLTERM;
                     CSET H5T_CSET_ASCII;
                     CTYPE H5T_C_S1;
                  }
                  DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
                  DATA {
                  (0): "Energy"
                  }
               }
               ATTRIBUTE "Subdata Type" {
                  DATATYPE  H5T_STRING {
                     STRSIZE 6;
                     STRPAD H5T_STR_NULLTERM;
                     CSET H5T_CSET_ASCII;
                     CTYPE H5T_C_S1;
                  }
                  DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
                  DATA {
                  (0): "Sample"
                  }
```

```
555                }
556                DATASET "Energy_0000000006 Data" {
557                   DATATYPE  H5T_COMPOUND {
558                      H5T_STRING {
559                         STRSIZE 24;
560                         STRPAD H5T_STR_NULLTERM;
561                         CSET H5T_CSET_ASCII;
562                         CTYPE H5T_C_S1;
563                      } "Date_Time";
564                      H5T_STD_U64LE "Time";
565                      H5T_STD_U64LE "Power";
566                      H5T_STD_U64LE "CPU_Frequency";
567                   }
568                   DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
569                   DATA {
570                   (0): {
571                         "",
572                         0,
573                         0,
574                         0
575                      }
576                   }
577                }
578             }
```

E. hdf5-dumps of slurm 14 compared to slurm 15

Generated by me with the h5dump executable. The storage of data has gotten much more space efficient from slurm 14 to 15. Where slurm 14 consumes 7kB for 3 samples of data, slurm 15 only needs 3 kB for 9 samples:
Slurm 14 - 3 samples of energy data

```
HDF5 "32_0_nerge-desktop.h5" {
GROUP "/" {
   GROUP "Node_nerge-desktop" {
      ATTRIBUTE "Node Name" {
         DATATYPE  H5T_STRING {
            STRSIZE 13;
            STRPAD H5T_STR_NULLTERM;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
         DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
         DATA {
         (0): "nerge-desktop"
         }
      }
      ATTRIBUTE "Number of Tasks" {
         DATATYPE  H5T_STD_I32LE
         DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
         DATA {
         (0): 1
         }
      }
      ATTRIBUTE "Start Time" {
         DATATYPE  H5T_STRING {
            STRSIZE 24;
            STRPAD H5T_STR_NULLTERM;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
```

```
30          DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
31          DATA {
32          (0): "Fri Apr 01 15:47:11 2016"
33          }
34       }
35       GROUP "Tasks" {
36          GROUP "Task_0" {
37             ATTRIBUTE "CPUs per Task" {
38                DATATYPE  H5T_STD_I32LE
39                DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
40                DATA {
41                (0): 1
42                }
43             }
44             ATTRIBUTE "Task Id" {
45                DATATYPE  H5T_STD_I32LE
46                DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
47                DATA {
48                (0): 0
49                }
50             }
51          }
52       }
53       GROUP "Time Series" {
54          GROUP "Energy" {
55             ATTRIBUTE "Data Type" {
56                DATATYPE  H5T_STRING {
57                   STRSIZE 6;
58                   STRPAD H5T_STR_NULLTERM;
59                   CSET H5T_CSET_ASCII;
60                   CTYPE H5T_C_S1;
61                }
62                DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
63                DATA {
64                (0): "Energy"
65                }
66             }
67             GROUP "Energy_0000000001" {
68                ATTRIBUTE "Data Type" {
69                   DATATYPE  H5T_STRING {
70                      STRSIZE 6;
71                      STRPAD H5T_STR_NULLTERM;
72                      CSET H5T_CSET_ASCII;
73                      CTYPE H5T_C_S1;
```

```
74                   }
75                   DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
76                   DATA {
77                   (0): "Energy"
78                   }
79                }
80                ATTRIBUTE "Subdata Type" {
81                   DATATYPE  H5T_STRING {
82                      STRSIZE 6;
83                      STRPAD H5T_STR_NULLTERM;
84                      CSET H5T_CSET_ASCII;
85                      CTYPE H5T_C_S1;
86                   }
87                   DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
88                   DATA {
89                   (0): "Sample"
90                   }
91                }
92                DATASET "Energy_0000000001 Data" {
93                   DATATYPE  H5T_COMPOUND {
94                      H5T_STRING {
95                         STRSIZE 24;
96                         STRPAD H5T_STR_NULLTERM;
97                         CSET H5T_CSET_ASCII;
98                         CTYPE H5T_C_S1;
99                      } "Date_Time";
100                     H5T_STD_U64LE "Time";
101                     H5T_STD_U64LE "Power";
102                     H5T_STD_U64LE "CPU_Frequency";
103                  }
104                  DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
105                  DATA {
106                  (0): {
107                        "",
108                        1459518461,
109                        20,
110                        1
111                     }
112                  }
113               }
114            }
115            GROUP "Energy_0000000002" {
116               ATTRIBUTE "Data Type" {
117                  DATATYPE  H5T_STRING {
```

```
                        STRSIZE 6;
                        STRPAD H5T_STR_NULLTERM;
                        CSET H5T_CSET_ASCII;
                        CTYPE H5T_C_S1;
                     }
                     DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
                     DATA {
                     (0): "Energy"
                     }
                  }
                  ATTRIBUTE "Subdata Type" {
                     DATATYPE  H5T_STRING {
                        STRSIZE 6;
                        STRPAD H5T_STR_NULLTERM;
                        CSET H5T_CSET_ASCII;
                        CTYPE H5T_C_S1;
                     }
                     DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
                     DATA {
                     (0): "Sample"
                     }
                  }
                  DATASET "Energy_0000000002 Data" {
                     DATATYPE  H5T_COMPOUND {
                        H5T_STRING {
                           STRSIZE 24;
                           STRPAD H5T_STR_NULLTERM;
                           CSET H5T_CSET_ASCII;
                           CTYPE H5T_C_S1;
                        } "Date_Time";
                        H5T_STD_U64LE "Time";
                        H5T_STD_U64LE "Power";
                        H5T_STD_U64LE "CPU_Frequency";
                     }
                     DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
                     DATA {
                     (0): {
                           "",
                           1459518491,
                           40,
                           1
                        }
                     }
                  }
```

```
162             }
163             GROUP "Energy_0000000003" {
164                ATTRIBUTE "Data Type" {
165                   DATATYPE  H5T_STRING {
166                      STRSIZE 6;
167                      STRPAD H5T_STR_NULLTERM;
168                      CSET H5T_CSET_ASCII;
169                      CTYPE H5T_C_S1;
170                   }
171                   DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
172                   DATA {
173                   (0): "Energy"
174                   }
175                }
176                ATTRIBUTE "Subdata Type" {
177                   DATATYPE  H5T_STRING {
178                      STRSIZE 6;
179                      STRPAD H5T_STR_NULLTERM;
180                      CSET H5T_CSET_ASCII;
181                      CTYPE H5T_C_S1;
182                   }
183                   DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
184                   DATA {
185                   (0): "Sample"
186                   }
187                }
188                DATASET "Energy_0000000003 Data" {
189                   DATATYPE  H5T_COMPOUND {
190                      H5T_STRING {
191                         STRSIZE 24;
192                         STRPAD H5T_STR_NULLTERM;
193                         CSET H5T_CSET_ASCII;
194                         CTYPE H5T_C_S1;
195                      } "Date_Time";
196                      H5T_STD_U64LE "Time";
197                      H5T_STD_U64LE "Power";
198                      H5T_STD_U64LE "CPU_Frequency";
199                   }
200                   DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
201                   DATA {
202                   (0): {
203                         "",
204                         0,
205                         0,
```

```
                        0
                    }
                }
            }
        }
    }
}
}
```

Slurm 15 - 6 samples of energy data and 3 samples of filesystem data

```
HDF5 "83_0_nerge-desktop.h5" {
GROUP "/" {
   GROUP "nerge-desktop" {
      ATTRIBUTE "CPUs per Task" {
         DATATYPE  H5T_STD_I32LE
         DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
         DATA {
         (0): 1
         }
      }
      ATTRIBUTE "Node Name" {
         DATATYPE  H5T_STRING {
            STRSIZE 13;
            STRPAD H5T_STR_NULLTERM;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
         DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
         DATA {
         (0): "nerge-desktop"
         }
      }
      ATTRIBUTE "Number of Tasks" {
         DATATYPE  H5T_STD_I32LE
         DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
         DATA {
         (0): 1
         }
      }
      ATTRIBUTE "Start Time" {
         DATATYPE  H5T_STRING {
```

```
32              STRSIZE 24;
33              STRPAD H5T_STR_NULLTERM;
34              CSET H5T_CSET_ASCII;
35              CTYPE H5T_C_S1;
36           }
37           DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
38           DATA {
39           (0): "Fri Apr 01 14:21:21 2016"
40           }
41        }
42        DATASET "Energy" {
43           DATATYPE  H5T_COMPOUND {
44              H5T_STD_U64LE "ElapsedTime";
45              H5T_STD_U64LE "EpochTime";
46              H5T_STD_U64LE "Power";
47           }
48           DATASPACE  SIMPLE { ( 7 ) / ( H5S_UNLIMITED ) }
49           DATA {
50           (0): {
51                 0,
52                 1459513281,
53                 10
54              },
55           (1): {
56                 15,
57                 1459513296,
58                 20
59              },
60           (2): {
61                 30,
62                 1459513311,
63                 40
64              },
65           (3): {
66                 45,
67                 1459513326,
68                 80
69              },
70           (4): {
71                 60,
72                 1459513341,
73                 160
74              },
75           (5): {
```

```
                75,
                1459513356,
                320
            },
         (6): {
                90,
                1459513371,
                640
            }
         }
      }
      DATASET "Filesystem" {
         DATATYPE  H5T_COMPOUND {
            H5T_STD_U64LE "ElapsedTime";
            H5T_STD_U64LE "EpochTime";
            H5T_STD_U64LE "Reads";
            H5T_IEEE_F64LE "ReadMB";
            H5T_STD_U64LE "Writes";
            H5T_IEEE_F64LE "WriteMB";
         }
         DATASPACE  SIMPLE { ( 3 ) / ( H5S_UNLIMITED ) }
         DATA {
         (0): {
                0,
                1459513281,
                0,
                0,
                0,
                0
            },
         (1): {
                45,
                1459513326,
                400004,
                2002,
                30003,
                101
            },
         (2): {
                90,
                1459513371,
                400004,
                2002,
                30003,
```

```
                    101
                }
            }
        }
    }
}
}
```

List of Figures

Appendix B contains three figures created by me, depicting my understanding of naive scheduling and backfill scheduling.

List of Listings

There are two listings in appendix a: They are config files generated by configurator.easy.html and myself. There are two listings in appendix c: They are the slurm source code files, which i modified (including the modifications). There is one listings in appendix d: An excerpt from an corrupted hdf5-file provided by Hendryk Bockelmann. There are two listings in appendix e: They are hdf5-files dumped by me, to show how the structure has improved between slurm 14 and 15.

List of Tables

There is only one table, i made up myself. It is depicting the results of my experiments.