Energy Optimizer Documentation

Introduction

The node energy optimizer modifies the node setting based on the policies defined by cluster manager and hardware measurement from the node.

The node agent receives hardware counter measurements and passes them onto the energy optimizer which calculates an optimal value for the node. The new value is then applied to the node and the details are sent to the cluster manager. This method is repeated every few seconds until the job terminates. Once the job has terminated the node agent cleanup process is started which gathers the information sends it to the cluster manager and returns the node to the default state.

Development

Golang was chosen as the programming language used to develop the node energy optimizer. As it is the programming language used to develop the existing monitoring framework (ClusterCockpit) and the LIKWID performance tool suit, which have been developed by our partner in the EE-HPC project FAU and deployed by FAU and our partner DKRZ.

Energy Optimizer

The energy optimizer employs a version of the Golden Section Search (GSS) algorithm to dynamically guide workloads to the current minimum or maximum of our chosen metric. The current metric used is the energy delay product (EDP) of the node. However, we can apply the GSS optimizer to optimize other energy tuning metrics, for example, we can calculate the Energy Delay Squared Product (ED2P) metric and use that as a basis for our optimizations.
Choose 3 points x_1, x_3 and x_4 where ($x_1 > x_3 > x_4$) along the x axis along with the corresponding function values $f(x_1)$, $f(x_3)$, and $f(x_4)$. Since $f(x_1) > f(x_4)$ and $f(x_3) > f(x_1)$ the maximum lies between the points x_1 and x_3. As illustrated in figure 1.

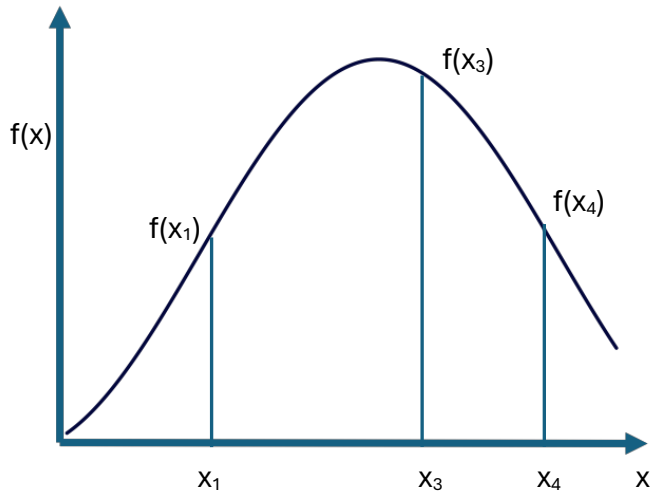

Figure 1 Intervals

Now add a fourth point x_2 between the larger interval of $[x_1, x_3]$ and $[x_3, x_4]$, in the case depicted in figure 1 x_2 with be inserted between the interval $[x_1, x_3]$.
If $f(x_2) > f(x_1)$ then the 3 points would be $(x_1 < x_2 < x_3)$; else if $f(x_2) < f(x_1)$ then the 3 points would be $(x_2 < x_3 < x_4)$. This process is continued until the distance between the outer points is sufficiently small, as shown in figure 2.

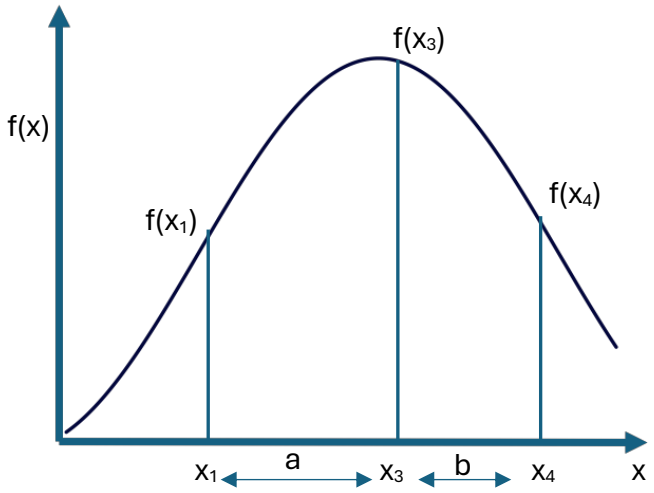

Figure 2 Determine the first intermediate point

To determine the intermediate points in the GSS we choose the first intermediate point x_1 to equalize the ratio of the lengths a and b as illustrated in figure 3 using the equation $\frac{a}{a+b} = \frac{b}{a}$ note that $a + b$ is equal to the distance between to lower and upper boundary points x_1 and x_4.

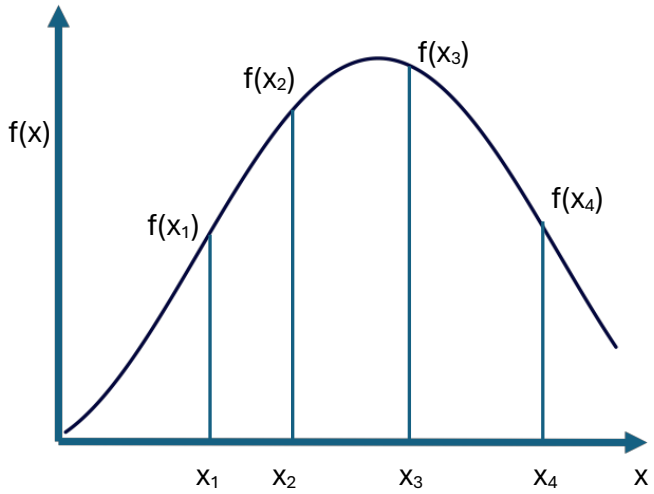

Figure 3 Add the intermediate point X_2

The second intermediate point x_2 in the same way in the interval a which satisfies the following ratio $\frac{a-b}{b} = \frac{b}{a}$ where the distances a and b are depicted in figure 4

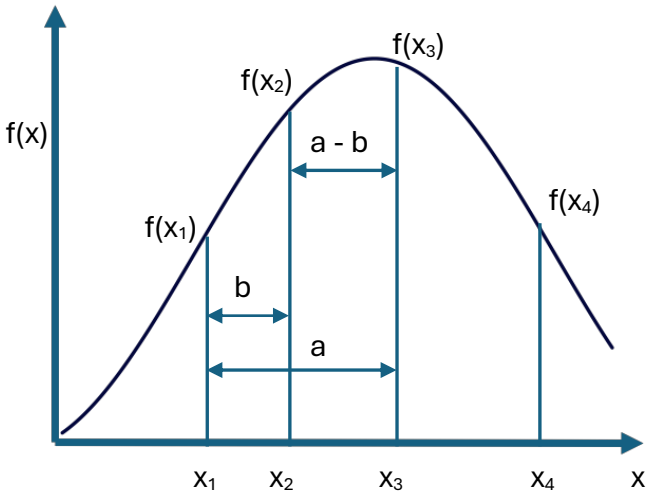

Figure 4 Determine the second intermediate point

The ratios of the equations $\frac{a}{a+b} = \frac{b}{a}$ and $\frac{a-b}{b} = \frac{b}{a}$ are equal and is known as the Golden Ratio. To determine the value of the Golden Ratio let $R = \frac{a}{b}$ then the equation $\frac{a}{a+b} = \frac{b}{a}$ can be written as $R + 1 = \frac{1}{R}$ or $R^2 + R - 1 = 0$ Then solve the quadratic formula to get the value of the Golden Ratio $\frac{1+\sqrt{5}}{2}$ and $\frac{1-\sqrt{5}}{2}$. The intermediate point x1 and x_2 are chosen so that the ratio of the distances from these points to the boundaries of the search region is equal to the Golden Ratio, as shown in figure 5.

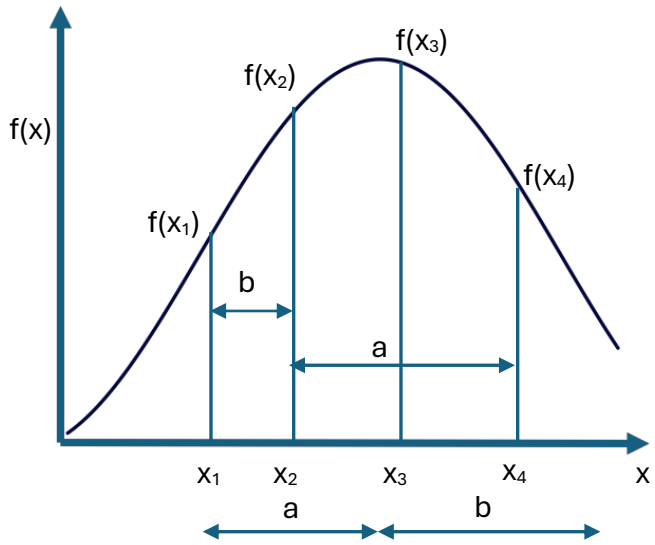

Figure 5 Intermediate points and their relation to the boundary points

Next, we determine a new and smaller interval where the maximum value of the function lies in. We know that the new interval is either [x_1 , x_2 , x_3] or [x_2 , x_3 , x_4], To determine which of these intervals will be considered in the next iteration, the function is evaluated at the intermediate points x_2 and x_3 . If $f(x_2) > f(x_1)$, the new region of interest will be [x_1 , x_2 , x_3]; Else if $f(x_2) < f(x_1)$, then the new region of interest will be [x_2 , x_3 , x_4]. The boundaries of the new smaller region are now determined by x_1 and x_3, and we already have one of the intermediate points, x_2 located at a point where the ratio of the distance to the boundaries is the Golden Ratio. All that is left to do is to determine the location of the second intermediate point. This process of determining a new smaller region of interest and a new intermediate point will continue until the distance between the boundary points are sufficiently small.

Golden Section Search Algorithm

The following algorithm is used to calculate the maximum of the function f(x).

Initialization Step:

Select the points x_1 and x_4 which contains the maximum of the function f(x).

Step 1:

Select the two intermediate points x_2 and x_3 so that $x_2 = x_4 - d$ and $x_3 = 1 + d$ where $d = \frac{1-\sqrt{5}}{2}(x_4 - x_1)$

Step 2:

Evaluate $f(x_2)$ and $f(x_3)$. If $f(x_2) < f(x_3)$ then determine the new [x_1 , x_2 , x_3, x_4] ;

$$x_1 = x_2,$$
$$x_2 = x_3,$$
$$x_4 = x_4,$$
$$x_3 = x_1 + \frac{1-\sqrt{5}}{2}(x_4 - x_1)$$

Else if $f(x_2) < f(x_3)$:

$$x_1 = x_1,$$
$$x_4 = x_3,$$
$$x_3 = x_2,$$
$$x_2 = x_4 - \frac{1-\sqrt{5}}{2}(x_4 - x_1)$$

Step 3:

If $x_4 - x_1 < \varepsilon$ where ε is a sufficiently small number, the maximum is located at the position $\frac{x_4 + x_1}{2}$ and stop. Else go to Step 2.

Implementation

As mentioned earlier the node energy optimizer is developed in the Golang programming language to help facilitate easier integration into the existing ClusterCockpit framework and LIKWID performance tool suite by FAU. Go also provides existing functionality for interfacing with the NATS messaging system, which is the messaging system proposed for the EE-HPC project.

Configuration & Installation

First install LIKWID with the likwid-sysFeatures component enabled on the system using the following commands:

```
$ git clone <likwid repo>
$ cd likwid
$ vi config.mk
# - change PREFIX if needed
# - enable sysfeatures component by setting BUILD_SYSFEATURES=true
$ make
$ sudo make install
```

If you don't have elevated privileges required for sudo, ask someone from the system support team to install it for you.

The next install the Go programming environment on the system or in your local user environment.
Linux:

- Remove any existing Go installations

```
$ rm -rf /usr/local/go && tar -C /usr/local -xzf go1.23.0.linux-amd64.tar.gz
```

- Unpack the tar file
- Add the location of the bin directory to your PATH and the lib directory to your LD_LIBRARY_PATH. For example,

```
$ export PATH=$PATH:/usr/local/go/bin
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/go/lib
```

- Verify that you've installed Go by typing the following command:

$ go version
Confirm that the installed version is printed to the screen.

The next install the NATS messaging system, which can be installed as a container. For details on how to install the container see the instructions on the website (https://docs.nats.io/running-a-nats-service/nats_docker).

//Once the NATS server has been installed and started

In addition to being installed within the EE-HPC software framework and ClusterCockpit. The energy optimizer can be deployed as a stand-alone node agent for testing and evaluation purposes, and this requires LIKWID configured with likwid-sysFeatures enabled to be installed. To avoid writing your own communicator functionality use the cc-metric-collector to connect the node energy optimizer with LIKWID. The configurations files for the cc-metric-collector:

liwid.json file:

```
"eventsets": [
      {
        "events" : {
          "FIXC0": "INST_RETIRED_ANY",
          "PWR0": "PWR_PKG_ENERGY"
        },
        "metrics" : [
          {
            "name": "instructions",
            "calc": "FIXC0",
            "publish": true,
            "type": "hwthread"
          },
          {
            "name": "cpu_power",
            "calc": "PWR0/time",
            "publish": true,
            "type": "socket"
          }
        ]
      }
    ]
```

collectors.json file:

```
{
  "likwid": {
    "force_overwrite" : true,
    "invalid_to_zero" : true,
    "access_mode" : "accessdaemon",
    "accessdaemon_path" : "/usr/local/lib/sbin",
    "liblikwid_path": "/usr/local/lib/liblikwid.so",
    "eventsets": [
      {
```

```
        "events": {
          "FIXC0": "INSTR_RETIRED_ANY"
        },
        "metrics": [
          {
            "calc": "FIXC0",
            "name": "inst_retired",
            "publish": true,
            "type": "hwthread"
          }
        ]
      }
    ]
  }
}
```

You will need to modify the paths to the directory where LIKWID AccessDaemon and the LIKWID shared libraries are located on your system.
For the cc-energy-manager you will need to modify the receivers.json configuration file to ensure that we subscribe to the correct data stream or messaging system.

receivers.json file:

```
{
    "testnats" : {
        "type": "nats",
        "address" : "127.0.0.1",
        "port" : "4222",
        "subject" : "ee-hpc-nats"
    }
}
```

Recommendations

Depending on the system constraints, policies and execution time of the jobs we can initially set the frequency at which the optimizer runs on the node. This frequency can then be decreased or increased depending on how the optimizer impacts the operational mode of the system and the execution time of the job. Examples include:

- The number of messages generated and sent start to cause network congestion, which leads to performance degradation and system instability. If this is the case the frequency of the node energy optimizer needs to be decreased to reduce the number of messages sent on the network. For example, increased MPI latency due to excessive network traffic.
- The frequency at which the node optimizer executes negatively impacts the stability and performance of the job running on the node, by using compute resources that would otherwise be used by the job. If this occurs, the frequency of the node energy optimizer needs to be reduced or even disabled to ensure that the executing job is running in a performant manner.

- The execution time of the job in some cases determines the frequency at which the node energy optimizer executes. For example, if the job takes 5 minutes to complete, we might need to increase the frequency at which the node energy manager executes to every 2 seconds to ensure that the optimal energy setting can be calculated and applied to the node(s) before the job terminates.
- Different execution blocks or code regions within an application can have significantly different energy profiles and it might be useful to modify the frequency at which the node energy manager executes to better analyze and optimize the node(s) energy/power settings based on this information. For example, an application contains code regions that have high and low energy profiles, and the node energy optimizer can potentially allow the node to run in a high-power mode when executing the regions that use more energy intense instructions and reduce the power setting of the node(s) when the application is executing less energy intense instructions. While maintaining the same overall job energy consumption and power cap. This feature can be triggered by the EEHPC library.

Understanding the behaviour of the application and its phases is important in configuring the policies of the node energy optimizer for jobs. For example, for compute bound applications and execution phases we want the job to finish as quickly as possible, commonly known as race-to-finish, as it will use less total energy. However, memory bound applications and phases where increasing the time it takes for the job to finish will save more total energy. That means the node energy optimizer should set the level higher for compute bound jobs and lower for memory bound jobs whilst maintaining the overall system power policy.
Recommendations for the initial configuration of node energy optimizer would be to set the execution frequency to 2 minutes and verify that the node energy optimizer is correctly calculating the new power values and successfully setting the node powercap to the new value. Then run a set of applications and workflows that are commonly used on the system and finely tune the execution frequency of the node energy optimizer to ensure that the node power is optimized quickly, but also ensure that the operational mode of the system is not compromised.

Gotchas

A list of things you should check if something isn't working.

- Is the NATS server running.
    - If the NATS server is running within a container, check that the container is correctly configured and test by manually subscribing and publishing to a subject. using the command line terminal commands `nats sub test` and `nats pub test "testing NATS"` this should print out the line `testing NATS`.
- Are the configurations correctly set, check the relevant json files and modify if needed.
- Check that the subject parameter in the configuration for the sender and receiver components are the same.
- Is there a flush command in the sender configuration (check the sinks.json file).
- If the application that uses NATS is deployed in a container, verfiy that you can execute the application from within the container and include debug parameters if possible. For

example, `singularity exec --bind $(pwd)/cc-metric-collector-config:/etc/cc-metric-collector cc-metric-collector.sif /usr/bin/cc-metric-collector -config "/etc/cc-metric-collector/config.json" -debug`