Embedded Systems Project Report

Connecting Konro with Slurm

Christian Dattola 10769164
Luca Di Pietro 10711026
Rashmi Di Michino 10728262

Supervisor: Gabriele Magnani

October 2, 2024

Contents

1 Introduction

This project deals with the deep understanding of two software applications: Slurm and Konro.
Slurm is a cluster management and job scheduling system for large and small Linux clusters: it allows to allocate exclusive and/or non exclusive access to resources to users for a certain amount of time during which they can perform some tasks, then it provides a framework to start, execute and monitor the jobs on the set of allocated nodes and finally, it manages the convention for resources by exploiting a queue of pending work.
Konro is a run-time resource manager implemented in user space. It allows the management of system resources in Linux-based systems across different architectures. Its flexible framework allows developers to create custom resource

allocation policies tailored to specific goals. It offers several policies for the allocation of resources, however for the scope of this project just one of them was considered, namely the "Drom Random Policy".
The aim of this project is to first understand how the two applications work and then open a communication between them so that they can cooperate during the allocation of the resources.
For the sake of our purposes, Slurm was always assumed to perform a static resource allocation, which means that it was configured to run a task on each core, while Konro dynamically allocates the resources (cores) based on the need of the application.
The core of the project consists in enabling Slurm to know the actual number of free cores from Konro each time a new task is launched. On the basis of this parameter, Slurm decides how to manage the execution of the new task: either running it on an existing node (if the resources are sufficient), pushing it in a queue or allocating other resources to run it.

The code-base of the two applications with the integration of the communication protocol can be found at the following GitHub repository.

2 Implementation

The main steps of the project implementation are:

1. Design of the communication architecture.
2. Inspection of the codebase of Slurm and Konro.
3. Implementation of the communication protocol.
4. Testing.
5. Detection of the errors and their origin.

2.1 Design of the communication architecture

In order to enable the communication, the Client-Server architecture was adopted. Specifically:

- Konro implements a Server listening continuously on the network. The Server starts as soon as Konro's policy is executed.
- Slurm implements a Client that, each time a new task is launched, establishes a connection with the Server and sends a request to obtain the number of free cores. Then, it waits for a reply from the Server and stores this value.

2.2 Code inspection

The next step was a deep inspection of the code to understand the software applications, Slurm and Konro, store the number of available cores and how it is used to actually allocate the resources. Starting from the debug messages displayed by the applications when running them, it was possible to determine which of the files were related to the problem.
The aim of the inspection was to find proper points of the code where it is possible to implement the architecture described above.

- In Konro, the ideal piece of code where we could open the Server is the one that is executed only once when Konro starts and from which it is possible to easily access the parameter of interest. The point in the code that fulfills these requests was found in the file "*\policymanager\policies\dromrandpolicy.cpp*" and specifically when the class *DromRandPolicy* is instantiated. This class, in fact, contains a reference to "*cpuSetControl*" object, which enables the access to the parameter "*freeCPU*", that is a queue containing information about the free CPUs. The size of this queue is the parameter of interest, that will be sent to Slurm.

- In Slurm, the ideal piece of code where the communication can be performed is the one that is executed each time a new task is launched and in the same scope of the functions that control the allocation of the resources. Specifically, the parameter containing the number of free cores has to be set before Slurm checks it to perform resource allocation. The point in the code that fulfills these requests was found in the file "*\plugins\select\cons_tres\job_test.c*" and specifically in the function "*allocate_sc*". This function performs several complex operations in order to determine which cores from a given node can be allocated to execute the current job. It returns a structure identifying the usable resources. The request to the Server will be implemented in this function and the parameter received as response will be overwritten in the structure that is returned, which contains the parameter *avail_cpus*, representing the number of free CPUs in the given node.

2.3 Communication protocol

The implementation of the communication between the two processes uses classical sockets in C and C++.

Server on Konro

At the creation of class *DromRandPolicy*, a new thread that manages incoming connections is created. In this new thread, the server creates a socket, sets socket options, configures the server address and port, binds the socket to the address and port and finally starts listening for incoming connections. To do

so, the server enters a loop in which waits for new connections and then, when receiving one, a new thread manages the single client request.
In this thread, upon receiving a message with the content "*GetFreeCPUs*", the parameter *freeCPUs* is retrieved and sent back to the client. If the content is different, the received message is simply ignored.

Client on Slurm

Inside the function mentioned above, Slurm tries to connect to the Konro Server and sends it a message with content "*GetFreeCPUs*". Slurm now blocks until it gets a response from Konro or continues its execution if a response does not arrive within 5 seconds. In the first case, the value returned by Konro is later overwritten in the proper structure. In the latter case, an error message is displayed and Slurm keeps executing as if this communication never existed.

2.4 Testing

The system was tested in a configuration with a single node having 4 CPUs; instances of a task performing complex mathematical calculations were launched on the node. This task requires, in some cases, more than one CPU.
The behavior of the system, before the code modifications, was as follows: out of the 4 CPUs, only 3 could be allocated to the tasks to run; Slurm allowed the execution of a maximum of 3 tasks in any case, regardless of the actual CPU usage. However, the resource usage monitoring performed by Konro showed that the tasks launched in some cases occupied more than one CPU per task, so the execution of the third task could force the use of a CPU that was already previously occupied.

After the implementation of communication between Slurm and Konro, the system was tested by launching instances of the same task mentioned above. The implementation also includes debug messages that allow the visualization of information about the connection between Slurm and Konro and the number of free CPUs that Konro communicates to Slurm at the time of an *srun* invocation. The observed behavior is as follows: tasks are normally started on the node as long as Konro reports a number of free CPUs greater than 0; if Konro reports that the number of available resources on the node is null, the task execution is interrupted, displaying the following error message.

On Slurm controller terminal:

```
slurmctld: _slurm_rpc_allocate_resources: Requested
node configuration is not available
```

On the terminal where *srun* was launched:

```
srun: error: Unable to allocate resources: Requested
node configuration is not available
```

From the testing conducted, it is possible to observe that tasks requiring resources already in use are not executed. This ensures that the node's resources are utilized optimally, while also taking into account their usage at runtime, preventing new tasks from taking resources already in use by other tasks.

2.5 Detection of errors

With the aim of providing a starting point for future improvements, an effort was made to identify the source of the error.
The detected error could indicate the presence of a mismatch between internal parameters in Slurm. Specifically, there are other parameters relevant to Slurm that likely have not been updated properly, although the value of free CPUs in the struct mentioned in Section 2.2 has been updated, and the effect of this update is visible from the fact that Slurm blocks the execution of a new task. If the specific information regarding the occupancy of the node in use is displayed on screen via the command "*scontrol show node <node_name>*", it can be observed that the "*CPUAlloc*" and "*AllocTRES*" parameters, which represent the number of CPUs allocated by Slurm, are not consistent with the number dynamically communicated by Konro. It is possible that, at some point during the execution of the controller, a consistency check is performed between these parameters and the one we modified, and Slurm detects a mismatch that causes the error.

3 Future Improvements

The project described in this document presents several possible extensions and improvements such as:

- **queuing of tasks**: when a task is launched and there are no available cores on a given node, it would be preferable for it to be added to a queue that allows the task to run as soon as resources become available, instead of stopping its execution. An initial approach could be understanding which parameters are involved in the queue implementation (already present in Slurm) and update them consistently;
- **multiple node testing**: the implementation was tested only on a single node, while Slurm supports also resource management on multiple nodes. The preferred behaviour would be that, when the dynamic number of available resources is insufficient on a node, Slurm moved that task to another available node;
- **dynamic IP addresses**: the current implementation was tested only for one node with a given IP address, which is set directly in the code. A better implementation would consist in either setting the addresses of the nodes in a configuration file or retrieving them at run-time at the moment of the setup of the connection with Konro.

4 Conclusions

This project allowed the communication between the two software applications, Slurm and Konro, with the aim of exchanging information about the number of occupied resources on a node dynamically. This implementation allows Slurm to receive this information but the way it uses it should be inspected more deeply.

During the development of this assignment, we encountered some adversities. First of all, setting up the environment on which the two software applications run, including dependencies installation, environment variables setting and configuration files definition. Furthermore, we found particularly challenging dealing with such a large code-base like Slurm's, a software used in real systems and with poorly detailed code documentation.
On the other hand, we think that the project allowed us to practice with important tools such as Linux command line, to work with an already existing software, update it properly and to work with sockets in C and C++.

Overall, this project shows a possible solution to implement a collaborative resource management in a complex computing system, leading to the development of more efficient and adaptive system architectures.