# RAPPORT DE PROJET

# Go Free !
# An extreme freecooling solution for a CIMENT Grid cluster

Baptiste MORIN
Alban MOREAU
RICM4

Project supervisor : Olivier RICHARD
Bruno BZEZNIK

# Table des matières

**1 Principe** **3**

**2 Système de contrôle** **5**
2.1 Equipements déjà en place . . . . . 5
2.2 Travail à réaliser . . . . . 6
2.2.1 Système de contrôle . . . . . 6
2.2.2 Sondes de températures . . . . . 6
2.2.3 Commande des servos . . . . . 6

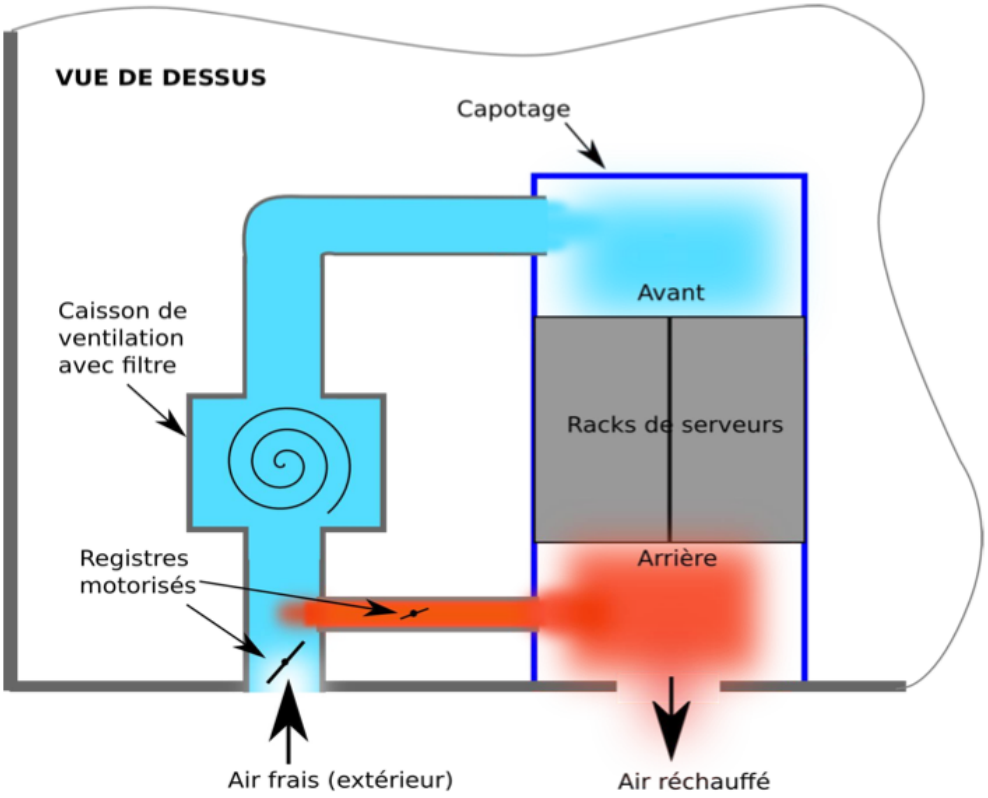
**3 Implémentation** **8**
3.1 L'algorithme de contrôle . . . . . 8
3.1.1 Principe . . . . . 8
3.1.2 Librairie . . . . . 9
3.2 Carte Arduino . . . . . 10
3.2.1 Librairies . . . . . 11

**4 Installation** **12**
4.1 Installation . . . . . 12
4.2 Tests . . . . . 12

CHAPITRE 1

# Principe

Le principe du freecooling est d'abord de bien confiner les zones froides (à l'avant des machines, là où elles aspirent l'air) et chaudes (à l'arrière, là où elles rejettent de l'air chaud).




Nous réalisons donc un capotage autour des 2 racks prévus (en gris sur le schéma) qui enveloppe complètement ces derniers. Un caisson de ventilation équipé d'un filtre aspire l'air extérieur et l'injecte en face avant des machines, dans la zone froide. L'air chaud est évacué par une ouverture vers l'extérieur à l'arrière des machines, dans la zone chaude, la différence de pression faisant sortir l'air naturellement à l'extérieur du bâtiment.

Une canalisation en zone chaude permet de reprendre de l'air chaud qui sera réinjecté si nécessaire vers l'entrée d'air du caisson afin de réguler la température. La régulation est contrôlée par un système d'asservissement électronique qui pilote les 2 registres (un pour l'entrée d'air principale et un autre vers le circuit de recyclage d'air chaud). Le registre qui est à l'entrée d'air froid n'est fermé que par grand froid et lorsque les machines ne chauffent pas assez pour réchauffer l'air via la reprise d'air chaud. En aucun cas les 2 registres ne doivent être fermés simultanément !

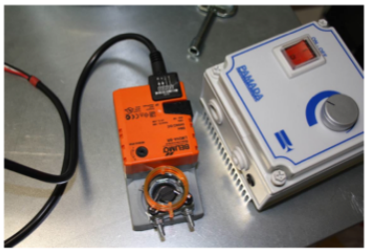
CHAPITRE 2

# Système de contrôle

Le système de contrôle électronique du dispositif fait l'objet d'une $2^{me}$ tranche d'équipement. Durant les premiers mois de fonctionnement, le dispositif a été contrôlé de manière pseudo-manuelle. Certains des équipements électroniques déjà en place sont à réutiliser dans cette seconde tranche.

## 2.1 Equipements déjà en place

- 2 servo-moteurs industriels (Belimo lM24ASR)
- 1 variateur de vitesse manuel pour moteur mono-phasé 220V (Pamada)
- 3 PDU manageables comportant 8 prises commandées chacune (on/off) par SNMP (commande d'arrêt d'urgence des noeuds de calcul ainsi que de la marche/arrêt du moteur du ventilateur)
- Un pressostat coupe l'alimentation du ventilateur en cas de trop forte dépression



Un des 2 servo est déjà installé, sur la vanne de recyclage d'air chaud, et se trouve piloté par un Arduino branché sur le serveur frontale de la machine de calcul. 4 sondes de température sont en place, mais ne sont pas à conserver en l'état, car elles sont pilotées par un appareil mobile qui n'est pas destiné à être installé en permanence. Les températures sont récupérées également par le serveur frontale, ce qui permet de réguler la température envoyée en face avant des machines, via la vanne de recyclage d'air chaud.

Le serveur frontale récupère aussi les températures fournies par les cartes mères des noeuds. C'est cette donnée (le min et le max sur l'ensemble des noeuds de calcul), qui permet à la frontale d'arrêter les noeuds s'il y a surchauffe.

## 2.2 Travail à réaliser

### *2.2.1 Système de contrôle*

Le système de contrôle sera composé d'un PC sous Linux et d'interfaces de pilotage permettant de commander les vannes (servo), le moteur (marche/arrêt et/ou variation de vitesse) et d'interroger les sondes de température. Le PC pourra aussi envoyer des commandes SNMP aux PDUs, via son interface réseau. Il fournira aussi un moyen d'accès aux valeurs des températures via le réseau.

Dans un premier temps, ce PC pourra être le serveur frontal lui-même, mais tout ce qui sera développé devra l'être de manière à pouvoir tourner sur un PC indépendant. La partie logicielle sera développée dans un langage de script Ruby avec éventuellement des parties en Bash. L'asservissement à mettre en place peut être décrit comme suit. Toutes les valeurs de consigne de température sont données à titre indicatif et devront être paramétrisables.

- En fonctionnement normal, c'est à dire au dessus de 20 degrés de température ambiante à l'extérieur environ, le moteur du ventilateur tourne, la vanne d'entrée d'air frais est ouverte et la vanne de recyclage d'air chaud est fermée. Si la température en sortie dépasse une certaine valeur (surchauffe), un signal est envoyé (un script est appelé ; ce script contrôle l'arrêt d'urgence des noeuds de calcul $\rightarrow$ envoi de commandes SNMP aux PDUS).
- Par temps frais (moins de 20 degrés), le système de régulation joue sur les vannes d'entrée d'air frais et de recyclage d'air chaud pour maintenir une température en face avant de 20 degrés. On ouvre d'abord la vanne de recyclage d'air chaud. Si la vanne est ouverte à fond et que la température ne peut plus être régulée, alors la vanne d'entrée d'air frais est fermée progressivement pour forcer la circulation en circuit fermé. Si la température ne peut plus être régulée et passe en dessous d'une certaine valeur (vanne en panne), alors un signal d'alarme est envoyé (un script est appelé). On veillera à éviter à tout prix la situation où les 2 vannes seraient fermées, ce qui causerai une dépression trop importante pouvant conduire à des dommages sur les gaines de ventilation.
- Eventuellement, l'état du pressostat devra être détecté. Cela revient à détecter la présence du courant 220V à l'entrée du régulateur de vitesse. Une alerte pourra être déclenchée si le pressostat se coupe : filtre bouché ou défaut de vanne (les 2 vannes bloquées fermées).
- Eventuellement, la vitesse du moteur est régulée pour fonctionner avec un minimum de consommation électrique en fonction du besoin de refroidissement. Le régulateur de vitesse manuel pourra être modifié en branchant un circuit à la place du potentiomètre si possible.

### *2.2.2 Sondes de températures*

Des sondes de température doivent être installées à plusieurs emplacements stratégiques. Ces sondes remplaceront le système provisoire actuel :

- Au moins 3 sondes placées sur la hauteur du rack en face avant (haut, milieu et bas)
- Au moins 3 sondes placées sur la hauteur du rack en face arrière (haut milieu et bas)
- Une sonde située au niveau de l'entrée d'air frais (avant le registre)
- Une sonde située au niveau de la reprise d'air chaud (avant le registre)
- Une sonde située dans la salle hébergeant le bloc freecooling, mais à l'extérieur de ce dernier (température ambiante de la salle)

Les valeurs de ces températures pourront être utilisées par le système de régulation, mais elles devront également être stockées et graphées pour information (graphes sur la dernière heure, le dernier jour, la dernière semaine, le dernier mois, la dernière année). Un outil comme CACTI pourra être utilisé pour les graphes.

### *2.2.3 Commande des servos*

Les servos prennent une tension d'entrée entre 0 et 10V comme consigne. Un circuit a déjà été réalisé autour d'un Arduino pour le pilotage par le PC de contrôle. Il commande actuellement la vanne de recyclage d'air chaud qui permet, lorsqu'elle est ouverte, de réchauffer l'air qui est injecté en face avant des machines.

Cela permet d'éviter d'envoyer de l'air trop froid sur les machines. Le système actuel ne fonctionne pas lorsque les températures sont trop basses car l'air chaud recyclé ne suffit pas à réchauffer l'air très froid qui peut venir de dehors. Il faut donc mettre en place un 2ème servo qui contrôle la fermeture de la vanne d'entrée d'air afin de réduire l'arrivée d'air frais et forcer plus d'air chaud à circuler en boucle.Le système provisoire actuel comporte aussi un autre défaut : il commande uniquement l'ouverture ou la fermeture totale de la vanne en fonction d'une consigne de température basse de 20 degrés, ce qui provoque des fluctuations de température de plus de 2 degrés. Il conviendra de réaliser une régulation plus fine avec une variation de température de moins d'un degré.

CHAPITRE 3

# Implémentation

Ici sera présenté l'implémentation globale du système mais aussi les explications techniques destinées à aider les éventuels développeur en vue de l'accroissement du nombre d'objets de contrôle et/ou de clusters.

## 3.1 L'algorithme de contrôle

Comme indiqué dans le début de ce document, l'algorithme de contrôle a été réalisé dans le langage Ruby.

### *3.1.1 Principe*

L'algorithme regroupe quatre parties de contrôle, une pour chaque entité : le cluster, le ventilateur et les deux vannes. L'algorithme en fonctionnement est une boucle infinie, générant une pause de 5 secondes (valeur qui peut être adaptée) à la fin de chaque tour. Chaque partie n'a pas besoin d'être contrôlée à la même fréquence que les autres, c'est pourquoi nous avons introduit des variables de contrôle pour chaque partie. Ces variables correspondent à une durée et définiront, par exemple, le fait que le ventilateur soit contrôlé toutes les $x$ secondes et le cluster toutes les $y$ secondes... Une pseudo variable de temps est incrémenté à chaque tour de boucle et sert à savoir quelles parties doivent être contrôlées sur un tour donné.

Voici un exemple qui permettra de mieux comprendre, les températures du ventilateur doivent être contrôlé toutes les 5 secondes, celles des valves toutes les 10 secondes et celles du cluster toutes les 30 secondes :

- Au premier tour toutes les valeurs de températures sont contrôlées, la variable de temps est à zéro.
- A la fin du tour la variable de temps est incrémentée de 5, et l'algorithme fait une pause de 5 secondes.
- Comme nous l'avons définit, au tour suivant seules les températures du ventilateur seront contrôlées (à intervalles de 5 secondes).
- Au troisième tour les températures du ventilateur et des valves seront récupérées (intervalles de 5 secondes par rapport au dernier contrôle et de 10 secondes par rapport au début) etc...

Les contrôles ne sont réalisés que si une opération conditionnelle sur un calcul de modulo est un succès (autrement dit pour nous est égal à zéro). La pseudo variable de temps est incrémenté à la fin de chaque tour de boucle et nous permet de déclencher ces contrôles ou non. Par exemple, on définit les intervalles de contrôles avec les mêmes valeurs que pour l'exemple précédent. Les contrôles pour le ventilateur ne seront réalisé uniquement si la pseudo variable de temps modulo 5 (durée entre chaque contrôle) vaut zéro, idem pour les valves et le cluster mais respectivement modulo 10 et 30. Dans le cadre de cet exemple les températures pour le ventilateur seront contrôlées à chaque tour (pause de 5 secondes), les températures pour les valves tous les 2 tours et tous les 6 tours pour le cluster.

Avant de générer des actions, les contrôle devront être fait plusieurs fois (définit en fonction des besoins). Ce n'est donc que lorsqu'une température trop élevée pour le ventilateur aura été vu 3 fois (par exemple) qu'une action sera déclenchée (arrêt/marche). Ces actions déclenchent des appels système qui peuvent parfois échouer... Ce cas "d'échec" est prévu et géré grâce aux `timeout` gérés avec Ruby. Nous créons pour chaque action un processus fils qui exécutera la commande spécifiée, le processus père attend normalement qu'il se termine. Mais dans le cas d'un échec lors de l'exécution d'une commande celui-ci peut rester bloqué, empêchant ainsi la poursuite de l'algorithme. Pour contrer cela des `timeout` ont été mis en place sur l'exécution des commandes. Si au bout de $x$ secondes la processus fils n'a pas terminé correctement, il est tué et une variable de warning est mise à jour. A la fin de chaque boucle les warning sont comptés et si le nombre est trop important, une alerte est ajoutée au fichier de log. En fonctionnement réel il sera plus judicieux de générer l'envoi d'une alerte à la personne en charge du système ou d'arrêter complètement le cluster.

Lors de l'exécution de l'algorithme les intervalles sont grossièrement respectés, cela est dû à la durée d'exécution que peuvent prendre les différents contrôles et l'attente lors de déclenchement de commandes. Cela n'est pas gênant puisque la température est un facteur physique qui varie relativement lentement.

### *3.1.2 Librairie*

Pour pouvoir communiquer avec le port série de la carte, nous avons choisi d'utiliser la librairie `ruby-serialport` qui est relativement simple d'utilisation mais n'est plus soutenue depuis 2006. Vous trouverez néanmoins les sources à l'adresse http ://ruby-serialport.rubyforge.org/. La manière la plus simple de l'obtenir est de passer par `rubygems`. La démarche à suivre est la suivante :

```
sudo apt-get install rubygems
sudo apt-get install gem
sudo gem install ruby-serialport
```

Une fois que l'installation s'est correctement déroulée, il suffit de mettre en en-tête :

```
require 'rubygems'
require 'serialport'
```

## 3.2 Carte Arduino

Le système actuel dispose d'une carte Arduino Uno reliée par USB à l'ordinateur de contrôle. Cette carte est reliée au PIN numéro 10 qui va récupérer la température grâce au capteur de température DALLAS DS18B20 mais elle sert également à piloter les 2 servos moteurs relatif aux entrée et sorties d'air. Le capteur digital fournit une température Celsius entre 9 et 12 bits grâce à une liaison "One Wire" qui, par définition, demande seulement une ligne de données. La documentation constructeur est disponible à l'adresse http ://datasheets.maxim-ic.com/en/ds/DS18B20.pdf, voici le schéma du capteur :

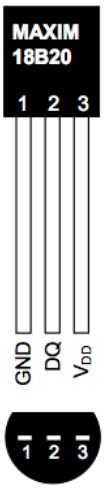



FIGURE 3.1 – Schéma du capteur de température

Le montage nécessite également la présence d'une résistance en série pour éviter le grillage du capteur. Actuellement, la résistance montée a pour valeur $10k\Omega \pm 5\%$. Voici une photo du montage avec le capteur relié à la carte Arduino :

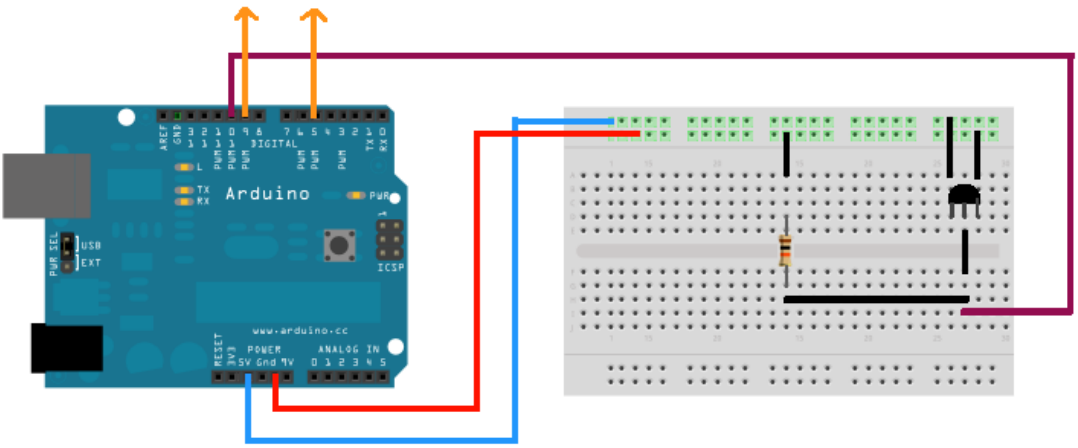



FIGURE 3.2 – Montage du capteur avec la carte

Pour ce qui est du pilotage des servos moteurs, la carte nous sert ici de convertisseur de signal PWN (Pulse Width Modulation) en signal DC (Direct Current). Typiquement cela consiste à récupérer sur le port série (par exemple `/dev/ttyACM0` sur Ubuntu) les quatre octets de contrôle envoyés depuis le PC du

système de contrôle puis, après analyse de cette entrée, on envoie sur le bon port une chaîne de caractère en hexadecimal. Selon les valeurs de ces quatre octets, l'ouverture et la fermeture des vannes seront actionnées. Le format des octets en entrée est le suivant :

- `aP1C` fermeture de la vanne de recyclage d'air chaud
- `bP1C` fermeture de la vanne extérieure (entrée air froid généralement)
- `aPA0` ouverture de la vanne de recyclage d'air chaud
- `bPA0` ouverture de la vanne extérieure (entrée air froid généralement)

### *3.2.1 Librairies*

Le code implémenté dans la carte fait appel à la librairie `OneWire` qui permet de contrôler entre autres, une famille de dispositifs DALLAS par exemple. Le principe du OneWire est qu'un seul "maître" communique avec un ou plusieurs "esclaves" sur un ligne de données unique, qui sert également d'alimentation pour les esclaves. Cette librairie est disponible à l'adresse http ://www.pjrc.com/teensy/td_libs_OneWire.html. Grâce à cette structure OneWire, on se laisse la possibilité d'ajouter plusieurs capteurs sur la carte. Pour se servir de la librairie, il suffit de télécharger et extraire l'archive `zip` dans le dossier `libraries` du dossier source Arduino. Ensuite, il faut simplement l'importer avec l'interface Arduino puis redémarrer le logiciel. Un code de démonstration dont nous nous sommes inspiré est disponible à l'adresse
http ://www.arduino.cc/playground/Learning/OneWire.

Une librairie `DallasTemperature` est aussi disponible et apparaît très simple d'utilisation, mais par soucis d'évolutivité nous avons préféré choisir la liaison OneWire dans le cas ou des capteurs autres que DALLAS soient ajoutés.

CHAPITRE 4

# Installation

## 4.1 Installation

L'installation du système est prévu pour fin Septembre, il est prévu que nous venions tester et installer le système avec Burno Bzeznik, ingénieur à l'INRIA chargé du projet sur place. Pour l'instant, les résultats obtenus avec les fichiers logs ont permis de valider le système, mais il faut encore affiner notamment la gestion du temps par rapport au temps que mettent les servos moteurs à se fermer/s'ouvrir. Voici un échantillon de photos représentant la salle :

FIGURE 4.1 – Vanne de recylage d'air chaud

FIGURE 4.2 – Caisson de ventilation

## 4.2 Tests

Pour s'assurer du bon fonctionnement du système, toutes les traces des différentes actions sont gardées pour qu'on puisse les analyser. En partant de fichiers contenant des températures critiques pour tester toutes les différentes actions, on observe que le système répond pleinement aux attentes. Les fichiers créé pour tester ne correspondent pas à des conditions réelles d'utilisation, une ouverture/fermeture pour les valves, et un démarrage/arrêt pour le cluster et le ventilateur s'enchaîne successsivement et permettent de tester les réactions de l'algorithme dans les cas spécifiques.

FIGURE 4.3 – Capotage autour du cluster